Journal of Internet Services
and Applications

RESEARCH Open Access

A comprehensive survey on machine learning for networking: evolution, applications and research opportunities

Raouf Boutaba[1*], Mohammad A. Salahuddin[1], Noura Limam[1], Sara Ayoubi[1], Nashid Shahriar[1], Felipe Estrada-Solano[1,2] and Oscar M. Caicedo[2]

Abstract

Machine Learning (ML) has been enjoying an unprecedented surge in applications that solve problems and enable automation in diverse domains. Primarily, this is due to the explosion in the availability of data, significant improvements in ML techniques, and advancement in computing capabilities. Undoubtedly, ML has been applied to various mundane and complex problems arising in network operation and management. There are various surveys on ML for specific areas in networking or for specific network technologies. This survey is original, since it jointly presents the application of diverse ML techniques in various key areas of networking across different network technologies. In this way, readers will benefit from a comprehensive discussion on the different learning paradigms and ML techniques applied to fundamental problems in networking, including traffic prediction, routing and classification, congestion control, resource and fault management, QoS and QoE management, and network security. Furthermore, this survey delineates the limitations, give insights, research challenges and future opportunities to advance ML in networking. Therefore, this is a timely contribution of the implications of ML for networking, that is pushing the barriers of autonomic network operation and management.

Keywords: Machine learning, Traffic prediction, Traffic classification, Traffic routing, Congestion control, Resource management, Fault management, QoS and QoE management, Network security

1 Introduction

Machine learning (ML) enables a system to scrutinize data and deduce knowledge. It goes beyond simply learning or extracting knowledge, to utilizing and improving knowledge over time and with experience. In essence, the goal of ML is to identify and exploit hidden patterns in "training" data. The patterns learnt are used to analyze unknown data, such that it can be grouped together or mapped to the known groups. This instigates a shift in the traditional programming paradigm, where programs are written to automate tasks. ML *creates* the program (i.e. *model*) that fits the data. Recently, ML is enjoying renewed interest. Early ML techniques were rigid and incapable of tolerating any variations from the training data [134].

Recent advances in ML have made these techniques flexible and resilient in their applicability to various real-world scenarios, ranging from extraordinary to mundane. For instance, ML in health care has greatly improved the areas of medical imaging and computer-aided diagnosis. Ordinarily, we often use technological tools that are founded upon ML. For example, search engines extensively use ML for non-trivial tasks, such as query suggestions, spell correction, web indexing and page ranking. Evidently, as we look forward to automating more aspects of our lives, ranging from home automation to autonomous vehicles, ML techniques will become an increasingly important facet in various systems that aid in decision making, analysis, and automation.

Apart from the advances in ML techniques, various other factors contribute to its revival. Most importantly, the success of ML techniques relies heavily on data [77]. Undoubtedly, there is a colossal amount of data in todays' networks, which is bound to grow further with emerging

*Correspondence: rboutaba@uwaterloo.ca
[1] David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
Full list of author information is available at the end of the article

networks, such as the Internet of Things (IoT) and its billions of connected devices [162]. This encourages the application of ML that not only identifies hidden and unexpected patterns, but can also be applied to learn and understand the processes that generate the data.

Recent advances in computing offer storage and processing capabilities required for training and testing ML models for the voluminous data. For instance, Cloud Computing offers seemingly infinite compute and storage resources, while Graphics Processing Units [342] (GPUs) and Tensor Processing Units [170] (TPUs) provide accelerated training and inference for voluminous data. It is important to note that a trained ML model can be deployed for inference on less capable devices e.g. smartphones. Despite these advances, network operations and management still remains cumbersome, and network faults are prevalent primarily due to human error [291]. Network faults lead to financial liability and defamation in reputation of network providers. Therefore, there is immense interest in building autonomic (i.e. self-configuring, self-healing, self-optimizing and self-protecting) networks [28] that are highly resilient.

Though, there is a dire need for cognitive control in network operation and management [28], it poses a unique set of challenges for ML. First, each network is unique and there is a lack of enforcement of standards to attain uniformity across networks. For instance, the enterprise network from one organization is diverse and disparate from another. Therefore, the patterns proven to work in one network may not be feasible for another network of the same kind. Second, the network is continually evolving and the dynamics inhibit the application of a fixed set of patterns that aid in network operation and management. It is almost impossible to manually keep up with network administration, due to the continuous growth in the number of applications running in the network and the kinds of devices connected to the network.

Key technological advances in networking, such as network programmability via Software-Defined Networking (SDN), promote the applicability of ML in networking. Though, ML has been extensively applied to problems in pattern recognition, speech synthesis, and outlier detection, its successful deployment for network operations and management has been limited. The main obstacles include what data can be collected from and what control actions can be exercised on legacy network devices. The ability to program the network by leveraging SDN alleviates these obstacles. The cognition from ML can be used to aid in the automation of network operation and management tasks. Therefore, it is exciting and non-trivial to apply ML techniques for such diverse and complex problems in networking. This makes ML in networking an interesting research area, and requires an understanding of the ML techniques and the problems in networking.

In this paper, we discuss the advances made in the application of ML in networking. We focus on traffic engineering, performance optimization and network security. In traffic engineering, we discuss traffic prediction, classification and routing that are fundamental in providing differentiated and prioritized services. In performance optimization, we discuss application of ML techniques in the context of congestion control, QoS/QoE correlation, and resource and fault management. Undoubtedly, security is a cornerstone in networking and in this regard, we highlight existing efforts that use ML techniques for network security.

The primary objective of this survey is to provide a comprehensive body of knowledge on ML techniques in support of networking. Furthermore, we complement the discussion with key insights into the techniques employed, their benefits, limitations and their feasibility to real-world networking scenarios. Our contributions are summarized as follows:

- *A comprehensive view of ML techniques in networking.* We review literature published in peer-reviewed venues over the past two decades that have high impact and have been well received by peers. The works selected and discussed in this survey are comprehensive in the advances made for networking. The key criteria used in the selection is a combination of the year of publication, citation count and merit. For example, consider two papers A and B published in the same year with citation counts x and y, respectively. If x is significantly larger than y, A would be selected for discussion. However, upon evaluating B, if it is evidenced that it presents original ideas, critical insights or lessons learnt, then it is also selected for discussion due to its merit, despite the lower citation count.
- *A purposeful discussion on the feasibility of the ML techniques for networking.* We explore ML techniques in networking, including their benefits and limitations. It is important to realize that our coverage of networking aspects are not limited to a specific network technology (e.g. cellular network, wireless sensor network (WSN), mobile ad hoc network (MANET), cognitive radio network (CRN)). This gives readers a broad view of the possible solutions to networking problems across network technologies.
- *Identification of key challenges and future research opportunities.* The presented discussion on ML-based techniques in networking uncovers fundamental research challenges that confront networking and inhibit ultimate cognition in network operation and management. A discussion of these opportunities will motivate future work and push the boundaries of networking.

Though there are various surveys on ML in networking [18, 61, 82, 142, 246, 339], this survey is purposefully different. Primarily, this is due to its timeliness, the comprehensiveness of ML techniques covered, and the various aspects of networking discussed, irrespective of the network technology. For instance, Nguyen and Armitage [339], though impactful, is now dated and only addresses traffic classification in networking. Whereas, Fadlullah et al. [142] and Buczak et al. [82], both state-of-the-art surveys, have a specialized treatment of ML to specific problems in networking. On the other hand, Klaine et al. [246], Bkassiny et al. [61] and Alsheikh et al. [18], though comprehensive in their coverage of ML techniques in networking, are specialized to specific network technology i.e. cellular network, CRN and WSN, respectively. Therefore, our survey provides a holistic view of the applicability, challenges and limitations of ML techniques in networking.

We organize the remainder of this paper as follows. In Section 2, we provide a primer on ML, which discusses different categories of ML-based techniques, their essential constituents and their evolution. Sections 3, 4 and 5 discuss the application of the various ML-based techniques for traffic prediction, classification and routing, respectively. We present the ML-based advances in performance management, with respect to congestion control, resource management, fault management, and QoS/QoE management for networking in Sections 6, 7, 8 and 9. In Section 10, we examine the benefits of ML for anomaly and misuse detection for intrusion detection in networking. Finally, we delineate the lessons learned, and future research challenges and opportunities for ML in networking in Section 11. We conclude in Section 12 with a brief overview of our contributions. To facilitate reading, Fig. 1 presents a conceptual map of the survey, and Table 1 provides the list of acronyms and definitions for ML.

2 Machine learning for networking—a primer

In 1959, Arthur Samuel coined the term "Machine Learning", as *"the field of study that gives computers the ability to learn without being explicitly programmed"* [369]. There are four broad categories of problems that can leverage ML, namely, *clustering*, *classification*, *regression* and *rule extraction* [79]. In clustering problems, the objective is to group similar data together, while increasing the gap between the groups. Whereas, in classification and regression problems, the goal is to map a set of new input data to a set of discrete or continuous valued output, respectively. Rule extraction problems are intrinsically different, where the goal is to identify statistical relationships in data.

ML techniques have been applied to various problem domains. A closely related domain consists of data analysis for large databases, called data mining [16]. Though, ML techniques can be applied to aid in data mining, the goal of data mining problems is to critically and meticulously analyze data—its features, variables, invariants, temporal granularity, probability distributions and their transformations. However, ML goes beyond data mining to predict future events or sequence of events.

Generally, ML is ideal for inferring solutions to problems that have a large representative dataset. In this way, as illustrated in Fig. 2, ML techniques are designed to identify and exploit hidden patterns in data for (i) describing the outcome as a grouping of data for clustering problems, (ii) predicting the outcome of future events for classification and regression problems, and (iii) evaluating the outcome of a sequence of data points for rule extraction problems. Though, the figure illustrates data and outcome in a two-dimensional plane, the discussion holds for multi-dimensional datasets and outcome functions. For instance, in the case of clustering, the outcome can be a non-linear function in a hyperplane that discriminates between groups of data. Networking problems can be formulated as one of these problems that can leverage ML. For example, a classification problem in networking can be formulated to predict the kind of security attack: Denial-of-Service (DoS), User-to-Root (U2R), Root-to-Local (R2L), or probing, given network conditions. Whereas, a regression problem can be formulated to predict of when a future failure will transpire.

Though there are different categories of problems that enjoy the benefits of ML, there is a generic approach to building ML-based solutions. Figure 3 illustrates the key constituents in designing ML-based solutions for networking. *Data collection* pertains to gathering, generating and, or defining the set of data and the set of classes of interest. *Feature engineering* is used to reduce dimensionality in data and identify discriminating features that reduce computational overhead and increase accuracy. Finally, ML techniques carefully analyze the complex inter- and intra-relationships in data and learn a model for the outcome.

For instance, consider an example of Gold values over time, as illustrated in Fig. 2c. Naïvely, a linear regression model, shown as a best-fit line through the historical data, can facilitate in predicting future values of Gold. Therefore, once the ML model is built, it can be deployed to deduce outcomes from new data. However, the outcomes are periodically validated, since they can drift over time, known as concept drifting. This can be used as an indicator for incremental learning and re-training of the ML model. In the following subsections, we discuss each of the components in Fig. 3.

2.1 Learning paradigms

There are four learning paradigms in ML, *supervised*, *unsupervised*, *semi-supervised* and *reinforcement learning*. These paradigms influence data collection, feature

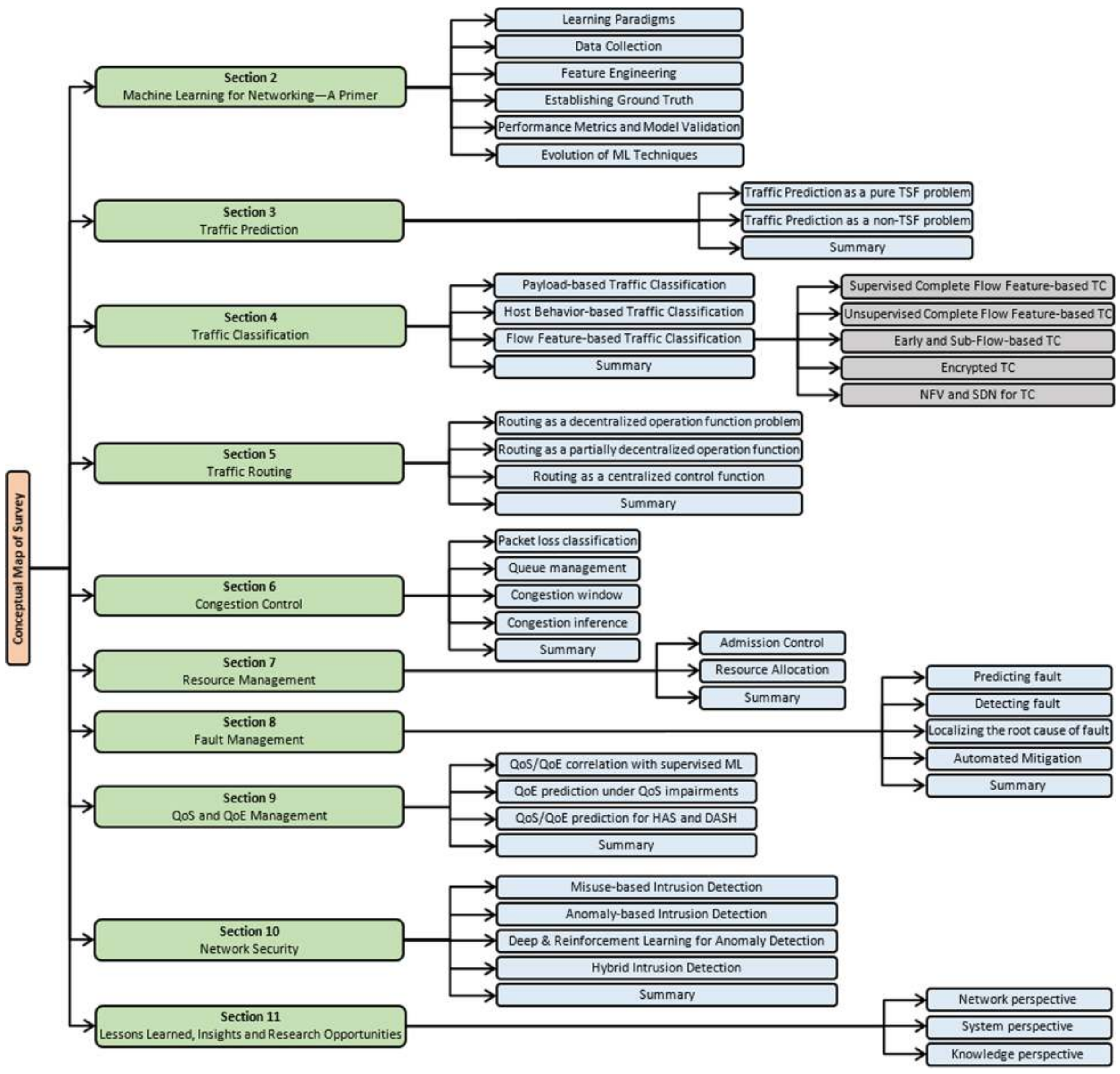

Conceptual Map of Survey
Section 2
Machine Learning for Networking—A Primer
Learning Paradigms
Data Collection
Feature Engineering
Establishing Ground Truth
Performance Metrics and Model Validation
Evolution of ML Techniques
Section 3
Traffic Prediction
Traffic Prediction as a pure TSF problem
Traffic Prediction as a non-TSF problem
Summary
Section 4
Traffic Classification
Payload-based Traffic Classification
Host Behavior-based Traffic Classification
Flow Feature-based Traffic Classification
Summary
Supervised Complete Flow Feature-based TC
Unsupervised Complete Flow Feature-based TC
Early and Sub-Flow-based TC
Encrypted TC
NFV and SDN for TC
Section 5
Traffic Routing
Routing as a decentralized operation function problem
Routing as a partially decentralized operation function
Routing as a centralized control function
Summary
Section 6
Congestion Control
Packet loss classification
Queue management
Congestion window
Congestion inference
Summary
Section 7
Resource Management
Admission Control
Resource Allocation
Summary
Section 8
Fault Management
Predicting fault
Detecting fault
Localizing the root cause of fault
Automated Mitigation
Summary
Section 9
QoS and QoE Management
QoS/QoE correlation with supervised ML
QoE prediction under QoS impairments
QoS/QoE prediction for HAS and DASH
Summary
Section 10
Network Security
Misuse-based Intrusion Detection
Anomaly-based Intrusion Detection
Deep & Reinforcement Learning for Anomaly Detection
Hybrid Intrusion Detection
Summary
Section 11
Lessons Learned, Insights and Research Opportunities
Network perspective
System perspective
Knowledge perspective

Fig. 1 Conceptual map of the survey

engineering, and establishing ground truth. Recall, the objective is to infer an outcome, given some dataset. The dataset used in constructing the ML model is often denoted as training data and labels are associated with training data if the user is aware of the description of the data. The outcome is often perceived as the identification of membership to a class of interest.

There are two schools of thought on the methodology for learning; *generative* and *discriminative* [333]. The basis for the learning methodologies is rooted in the famous Bayes' theorem for conditional probability and the fundamental rule that relates joint probability to conditional probability. Bayes' theorem is stated as follows. Given two events A and B, the conditional probability is defined as

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)},$$

which is also stated as

$$posterior = \frac{likelihood \times prior}{evidence}.$$

Table 1 List of acronyms for machine learning

AdaBoost	Adaptive Boosting
AIWPSO	Adaptive Inertia Weight Particle Swarm Optimization
BN	Bayesian Network
BNN	Bayesian Neural Network
BP	BackPropagation
CALA	Continuous Action-set Learning Automata
CART	Classification and Regression Tree
CMAC	Cerebellar Model Articulation Controller
DBN	Deep belief Network
DBSCAN	Density-based Spatial Clustering of Applications with Noise
DE	Differential Evolution
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q-Network
DT	Decision Tree
EM	Expectation Maximization
EMD	Entropy Minimization Discretization
FALA	Finite Action-set Learning Automata
FCM	Fuzzy C Means
FNN	Feedforward Neural Network
GD	Gradient Descent
HCA	Hierarchical Clustering Analysis
HMM	Hidden Markov Model
HNN	Hopfield Neural Network
ID3	Iterative Dichotomiser 3
k-NN	k-Nearest Neighbor
KDE	Kernel Density Estimation
LDA	Linear Discriminant Analysis
LSTM	Long Short-term Memory
LVQ	Learning Vector Quantization
MART	Multiple Additive Regression Tree
MaxEnt	Maximum Entropy
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-layer Perceptron
NB	Naïve Bayes
NBKE	Naïve Bayes with Kernel Estimation
NN	Neural Network
OLS	Ordinary Least Squares
PCA	Principal Component Analysis
PNN	Probabilistic Neural Network
POMDP	Partially Observable Markov Decision Process
RandNN	Random Neural Network
RBF	Radial Basis Function
RBFNN	Radial Basis Function Neural Network
RBM	Restricted Boltzman Machines
REPTree	Reduced Error Pruning Tree
RF	Random Forest
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SARSA	State-Action-Reward-State-Action
SGBoost	Stochastic Gradient Boosting
SHLLE	Supervised Hessian Locally Linear Embedding
SLP	Single-Layer Perceptron
SOM	Self-Organizing Map
STL	Selt-Taught Learning
SVM	Support Vector Machine
SVR	Support Vector Regression
TD	Temporal Difference
THAID	THeta Automatic Interaction Detection
TLFN	Time-Lagged Feedforward Neural Network
WMA	Weighted Majority Algorithm
XGBoost	eXtreme Gradient Boosting

The joint probability $P(A, B)$ of events A and B is $P(A \cap B) = P(B \mid A) \times P(A)$, and the conditional probability is the normalized joint probability. The generative methodology aims at modeling the joint probability $P(A, B)$ by predicting the conditional probability. On the other hand, in the discriminative methodology a function is learned for the conditional probability.

Supervised learning uses labeled training datasets to create models. There are various methods for labeling datasets known as ground truth (cf., Section 2.4). This learning technique is employed to "learn" to identify patterns or behaviors in the "known" training datasets. Typically, this approach is used to solve *classification* and *regression* problems that pertain to predicting discrete or continuous valued outcomes, respectively. On the other hand, it is possible to employ semi-supervised ML techniques in the face of partial knowledge. That is, having incomplete labels for training data or missing labels. Unsupervised learning uses unlabeled training datasets to create models that can discriminate between patterns in the data. This approach is most suited for *clustering* problems. For instance, outliers detection and density estimation problems in networking, can pertain to grouping different instances of attacks based on their similarities.

Reinforcement learning (RL) is an agent-based iterative process for modeling problems for decision making.

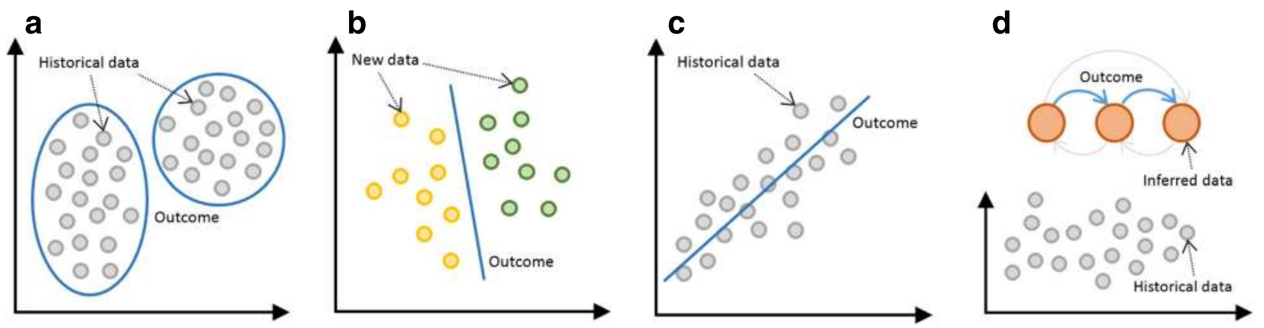


Fig. 2 Problem categories that benefit from machine learning. **a** Clustering. **b** Classification. **c** Regression. **d** Rule extraction

Generally, learning is based on exemplars from training datasets. However, in RL there is an agent that interacts with the external world, and instead of being taught by exemplars, it learns by exploring the environment and exploiting the knowledge. The actions are rewarded or penalized. Therefore, the training data in RL constitutes a set of state-action pairs and rewards (or penalties). The agent uses feedback from the environment to learn the best sequence of actions or "policy" to optimize a cumulative reward. For example, *rule extraction* from the data that is statistically supported and not predicted. Unlike, generative and discriminative approaches that are myopic in nature, RL may sacrifice immediate gains for long-term rewards. Hence, RL is best suited for making cognitive choices, such as decision making, planning and scheduling [441].

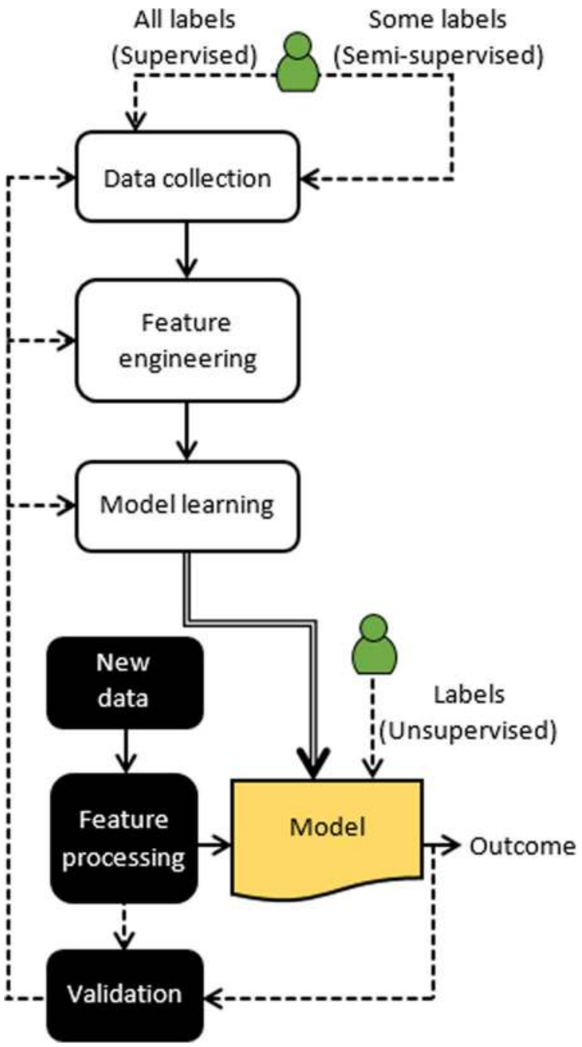


Fig. 3 The constituents of ML-based solutions

It is important to note that there is a strong relationship between the training data, problem and the learning paradigm. For instance, it is possible that due to lack of knowledge about the training data, supervised learning cannot be employed and other learning paradigms have to be employed for model construction.

2.2 Data collection

ML techniques require representative data, possibly without bias, to build an effective ML model for a given networking problem. Data collection is an important step, since representative datasets vary not only from one problem to another but also from one time period to the next. In general, data collection can be achieved in two phases—offline and online [460]. Offline data collection allows to gather a large amount of historical data that can be used for model training and testing. Whereas, real-time network data collected in the online phase can be used as feedback to the model, or as input for re-training the model. Offline data can also be obtained from various repositories, given it is relevant to the networking problem being studied. Examples of these repositories include Waikato Internet Traffic Storage (WITS) [457], UCI Knowledge Discovery in Databases (KDD) Archive [450], Measurement and Analysis on the WIDE Internet (MAWI) Working Group Traffic Archive [474], and Information Marketplace for Policy and Analysis of Cyber-risk & Trust (IMPACT) Archive [202].

An effective way to collect both offline and online data is by using monitoring and measurement tools. These tools provide greater control in various aspects of data collection, such as data sampling rate, monitoring duration and location (e.g. network core *vs.* network edge). They often use network monitoring protocols, such as Simple Network Management Protocol (SNMP) [208], Cisco NetFlow [100], and IP Flow Information Export (IPFIX) [209]. However, monitoring can be active or passive [152]. Active monitoring injects measurement traffic, such as probe packets in the network and collects relevant data from this traffic. In contrast, passive monitoring collects data by observing the actual network traffic. Evidently, active monitoring introduces additional overhead due to bandwidth consumption from injected traffic. Whereas, passive monitoring eliminates this overhead, at the expense of additional devices that analyze the network traffic to gather relevant information.

Once data is collected, it is decomposed into training, validation (also called development set or the "dev set"), and test datasets. The training set is leveraged to find the ideal parameters (e.g. weights of connections between neurons in a Neural Network (NN)) of a ML model. Whereas, the validation set is used to choose the suitable architecture (e.g. the number of hidden layers in a NN) of a ML model, or choose a model from a pool of ML models. Note, if a ML model and its architecture are pre-selected, there is no need for a validation set. Finally, test set is used to assess the unbiased performance of the selected model. Note, validation and testing can be performed using one of two methods—holdout or k-fold cross-validation. In the holdout method, part of the available dataset is set aside and used as a validation (or testing) set. Whereas, in the k-fold cross-validation, the available dataset is randomly divided into k equal subsets. Validation (or testing) process is repeated k times, with $k - 1$ unique subsets for training and the remaining subset for validating (or testing) the model, and the outcomes are averaged over the rounds.

A common decomposition of the dataset can conform to 60/20/20% among training, validation, and test datasets, or 70/30% in case validation is not required. These rule-of-thumb decompositions are reasonable for datasets that are not very large. However, in the era of big data, where a dataset can have millions of entries, other extreme decompositions, such as 98/1/1% or 99/0.4/0.1%, are also valid. However, it is important to avoid skewness in the training datasets, with respect to the classes of interest. This inhibits the learning and generalization of the outcome, leading to model over- and/or under-fitting. In addition, both validation and testing datasets should be independent of the training dataset and follow the same probability distribution as the training dataset.

Temporal and spatial robustness of ML model can be evaluated by exposing the model to training and validation datasets that are temporally and spatially distant. For instance, a model that performs well when evaluated with datasets collected a year after being trained or from a different network, exhibits temporal and spatial stability, respectively.

2.3 Feature engineering

The collected raw data may be noisy or incomplete. Before using the data for learning, it must go through a preprocessing phase to clean the data. Another important step prior to learning, or training a model, is feature extraction. These features act as discriminators for learning and inference. In networking, there are many choices of features to choose from. Broadly, they can be categorized based on the level of granularity.

At the finest level of granularity, packet-level features are simplistically extracted or derived from collected packets, e.g. statistics of packet size, including mean, root mean square (RMS) and variance, and time series information, such as hurst. The key advantage of packet-level statistics is their insensitivity to packet sampling that is often employed for data collection and interferes with feature characteristics [390]. On the other hand, Flow-level features are derived using simple statistics, such as mean flow duration, mean number of packets per flow, and mean number of bytes per flow [390]. Whereas, connection-level features from the transport layer are exploited to infer connection oriented details. In addition to the flow-level features, transport layer details, such as throughput and advertised window size in TCP connection headers, can be employed. Though these features generate high quality data, they incur computational overhead and are highly susceptible to sampling and routing asymmetries [390].

Feature engineering is a critical aspect in ML that includes feature selection and extraction. It is used to reduce dimensionality in voluminous data and to identify discriminating features that reduce computational overhead and increase accuracy of ML models. Feature selection is the removal of features that are irrelevant or redundant [321]. Irrelevant features increase computational overhead with marginal to no gain in accuracy, while redundant features promote over-fitting. Feature extraction is often a computationally intensive process of deriving extended or new features from existing features, using techniques, such as entropy, Fourier transform and principal component analysis (PCA).

Features selection and extraction can be performed using tools, such as NetMate [21] and WEKA [288]. However, in this case, the extraction and selection techniques are limited by the capability of the tool employed. Therefore, often specialized filter, embedded, and wrapper-based methods are employed for feature selection. Filtering prunes out the training data after carefully analyzing the dataset for identifying the irrelevant and redundant features. In contrast, wrapper-based techniques take an iterative approach, using a different subset of features in every iteration to identify the optimal subset. Whereas, embedded methods combine the benefits of filter and wrapper-based methods, and perform feature selection during model creation. Examples of feature selection techniques include, colored traffic activity graphs (TAG) [221], breadth-first search (BFS) [496], L1 Regularization [259], backward greedy feature selection (BGFS) [137], consistency-based (CON) and correlation-based feature selection (CFS) [321, 476]. It is crucial to carefully select an ideal set of features that strikes a balance between exploiting correlation and reducing/eliminating over-fitting for higher accuracy and lower computational overhead.

Furthermore, it is important to consider the characteristics of the task we are addressing while performing feature engineering. To better illustrate this, consider the following scenario from network traffic classification. One variant of the problem entails the identification of a streaming application (e.g. Netflix) from network traces. Intuitively, average packet-size and packet inter-arrival times are representative features, as they play a dominant role in traffic classification. Average packet size is fairly constant in nature [492] and packet inter-arrival times are a good discriminator for bulk data transfer (e.g. FTP) and streaming applications [390]. However, average packet size can be skewed by intermediate fragmentation and encryption, and packet inter-arrival times and their distributions are affected by queuing in routers [492]. Furthermore, streaming applications often behave similar to bulk data transfer applications [390]. Therefore, it is imperative to consider the classes of interest i.e. applications, before selecting the features for this network traffic classification problem.

Finally, It is also essential to select features that do not contradict underlying assumptions in the context of the problem. For example, in traffic classification, features that are extracted from multi-modal application classes (e.g. WWW) tend to show a non-Gaussian behavior [321]. These relationships not only become irrelevant and redundant, they contradict widely held assumptions in traffic classification, such as feature distributions being independent and following a Gaussian distribution. Therefore, careful feature extraction and selection is crucial for the performance of ML models [77].

2.4 Establishing ground truth

Establishing the ground truth pertains to giving a formal description (i.e. labels) to the classes of interest. There are various methods for labeling datasets using the features of a class. Primarily, it requires hand-labeling by domain experts, with aid from deep packet inspection (DPI) [462, 496], pattern matching (e.g. application signatures) or unsupervised ML techniques (e.g. AutoClass using EM) [136].

For instance, in traffic classification, establishing ground truth for application classes in the training dataset can be achieved using application signature pattern matching [140]. Application signatures are built using features, such as average packet size, flow duration, bytes per flow, packets per flow, root mean square packet size and IP traffic packet payload [176, 390]. Average packet size and flow duration have been shown to be good discriminators [390]. Application signatures for encrypted traffic (e.g. SSH, HTTPS) extract the signature from unencrypted handshakes. However, these application signatures must be kept up-to-date and adapted to the application dynamics [176].

Alternatively, it is possible to design and rely on statistical and structural content models for describing the datasets and infer the classes of interest. For instance, these models can be used to classify a protocol based on the label of a single instance of that protocol and correlations can be derived from unlabeled training data [286]. On the other hand, common substring graphs capture structural information about the training data [286]. These models are good at inferring discriminators for binary, textual and structural content [286].

Inadvertently, the ground truth drives the accuracy of ML models. There is also an inherent mutual dependency on the size of the training data of one class of interest on another, impacting model performance [417]. The imbalance in the number of training data across classes, is a violation of the assumptions maintained by many ML techniques, that is, the data is independent and identically distributed. Therefore, typically there is a need to combat class imbalance by applying under-, over-, joint-, or ensemble-sampling techniques [267]. For example, uniform weighted threshold under-sampling creates smaller balanced training sets [222].

2.5 Performance metrics and model validation

Once an ML model has been built and the ground truth has been ascertained, it is crucial to gauge the performance of the ML model that will describe, predict, or evaluate outcomes. However, it is important to realize that there is no way to distinguish a learning algorithm as the “best” and it is not fair to compare error rates across a whole variety of applications [16]. The performance metrics can be used to measure the different aspects of the model, such as reliability, robustness, accuracy, and complexity. In this section, we discuss the validation of the ML models with respect to accuracy (cf., Table 2), which is a critical aspect in the applicability of the model for networking problems. Moreover, the accuracy is often used as a feedback for incremental learning [389], to increase model robustness and resilience in a dynamic environment.

Table 2 Performance metrics for accuracy validation

Metric	Description
Mean Absolute Error (MAE)	Average of the absolute error between the actual and predicted values. Facilitates error interpretability.
Mean Squared Error (MSE)	Average of the squares of the error between the actual and predicted values. Heavily penalizes large errors.
Mean Absolute Prediction Error (MAPE)	Percentage of the error between the actual and predicted values. Not reliable for zero values or low-scale data.
Root MSE (RMSE)	Squared root of MSE. Represents the standard deviation of the error between the actual and predicted values.
Normalized RMSE (NRMSE)	Normalized RMSE. Facilitates comparing different models independently of their working scale.
Cross-entropy	Metric based on the logistic function that measures the error between the actual and predicted values.
Accuracy	Proportion of correct predictions among the total number of predictions. Not reliable for skewed class-wise data.
True Positive Rate (TPR) or recall	Proportion of actual positives that are correctly predicted. Represents the sensitivity or detection rate (DR) of a model.
False Positive Rate (FPR)	Proportion of actual negatives predicted as positives. Represents the significance level of a model.
True Negative Rate (TNR)	Proportion of actual negatives that are correctly predicted. Represents the specificity of a model.
False Negative Rate (FNR)	Proportion of actual positives predicted as negatives. Inversely proportional to the statistical power of a model.
Received Operating Characteristic (ROC)	Curve that plots TPR versus FPR at different parameter settings. Facilitates analyzing the cost-benefit of possibly optimal models.
Area Under the ROC Curve (AUC)	Probability of confidence in a model to accurately predict positive outcomes for actual positive instances.
Precision	Proportion of positive predictions that are correctly predicted.
F-measure	Harmonic mean of precision and recall. Facilitates analyzing the trade-off between these metrics.
Coefficient of Variation (CV)	Intra-cluster similarity to measure the accuracy of unsupervised classification models based on clusters.

Let us consider the accuracy validation of ML models for prediction. Usually, this accuracy validation undergoes an error analysis that computes the difference between the actual and predicted values. Recall, a prediction is an outcome of ML models for classification and regression problems. In classification, the common metrics for error analysis are based on the logistic function, such as binary and categorical cross-entropy—for binary and multi-class classification, respectively. In regression, the conventional error metrics are Mean Absolute Error (MAE) and Mean Squared Error (MSE). Both regression error metrics disregard the direction of under- and over-estimations in the predictions. MAE is simpler and easier to interpret than MSE, though MSE is more useful for heavily penalizing large errors.

The above error metrics are commonly used to compute the cost function of ML-based classification and regression models. Computing the cost function of the training and validation datasets (cf., Section 2.2) allow diagnosing performance problems due to high bias or high variance. High bias refers to a simple ML model that poorly maps the relations between features and outcomes (under-fitting). High variance implies an ML model that fits the training data but does not generalize well to predict new data (over-fitting). Depending on the diagnosed problem, the ML model can be improved by going back to one of the following design constituents (cf., Fig. 3): (i) data collection, for getting more training data (only for high variance), (ii) feature engineering, for increasing or reducing the set of features, and (iii) model learning, for building a simpler or more complex model, or for adjusting a regularization parameter.

After tuning the ML model for the training and validation datasets, the accuracy metrics for the test dataset are reported as the performance validation of the model. Regression models often use MAE or MSE (i.e. error metrics) to report the performance results. Other error metrics commonly used in the literature to gauge the accuracy of regression models include Mean Absolute Prediction Error (MAPE), Root MSE (RMSE), and Normalized RMSE

(NRMSE). MAPE states the prediction error as a percentage, while RMSE expresses the standard deviation of the error. Whereas, NRMSE allows comparing between models working on different scales, unlike the other error metrics described.

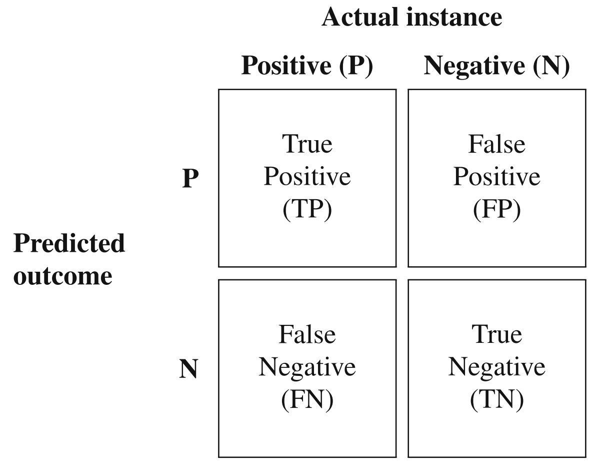
In classification, the conventional metric to report the performance of an ML model is the accuracy. The accuracy metric is defined as the proportion of true predictions T for each class C_i $\forall i = 1...N$ among the total number of predictions, as follows:

$$Accuracy = \frac{\sum_{i=1}^{N} T_{C_i}}{Total\ Predictions}$$

For example, let us consider a classification model that predicts whether an email should go to the spam, inbox, or promotion folder (i.e. multi-class classification). In this case, the accuracy is the sum of emails correctly predicted as spam, inbox, and promotion, divided by the total number of predicted emails. However, the accuracy metric is not reliable when the data is skewed with respect to the classes. For example, if the actual number of spam and promotion emails is very small compared to inbox emails, a simple classification model that predicts *every* email as inbox will still achieve a high accuracy.

To tackle this limitation, it is recommended to use the metrics derived from a confusion matrix, as illustrated in Fig. 4. Consider that each row in the confusion matrix represents a predicted outcome and each column represents the actual instance. In this manner, True Positive (TP) is the intersection between correctly predicted outcomes for the actual positive instances. Similarly, True Negative (TN) is when the classification model correctly predicts an actual negative instance. Whereas, False Positive (FP) and False Negative (FN) describe incorrect predictions for negative and positive actual instances, respectively. Note, that TP and TN correspond to the true predictions for the positive and negative classes, respectively. Therefore, the accuracy metric can also be defined in terms of the confusion matrix:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$



		Actual instance	
		Positive (P)	**Negative (N)**
Predicted outcome	**P**	True Positive (TP)	False Positive (FP)
	N	False Negative (FN)	True Negative (TN)

Fig. 4 Confusion matrix for binary classification

The confusion matrix in Fig. 4 works for a binary classification model. Therefore, it can be used in multi-class classification by building the confusion matrix for a specific class. This is achieved by transforming the multi-class classification problem into multiple binary classification subproblems, using the *one-vs-rest* strategy. For example, in the email multi-class classification, the confusion matrix for the spam class sets the positive class as spam and the negative class as the rest of the email classes (i.e. inbox and promotion), obtaining a binary classification model for spam and not spam email.

Consequentially, the True Positive Rate (TPR) describing the number of correct predictions is inferred from the confusion matrix as:

$$TPR\ (Recall) = \frac{TP}{TP + FN}$$

The converse, False Positive Rate (FPR) is the ratio of incorrect predictions and is defined as:

$$FPR = \frac{FP}{FP + TN}$$

Similarly, True Negative Rate (TNR) and False Negative Rate (FNR) are used to deduce the number of correct and incorrect negative predictions, respectively. The terms *recall*, *sensitivity*, and *detection rate (DR)* are often used to refer to TPR. In contrast, a comparison of the TPR versus FPR is depicted in a Received Operating Characteristics (ROC) graph. In a ROC graph, where TPR is on the y-axis and FPR is on the x-axis, a good classification model will yield a ROC curve that has a highly positive gradient. This implies high TP for a small change in FP. As the gradient gets closer to 1, the prediction of the ML model deteriorates, as the number of correct and incorrect predictions is approximately the same. It should be noted that a classification model with a negative gradient in the ROC curve can be easily improved by flipping the predictions from the model [16] or swapping the labels of the actual instances.

In this way, multiple classification models for the same problem can be compared and can give insight into the different conditions under which one model outperforms another. Though the ROC aids in visual analysis, the area under the ROC curve (AUC), ideally 1, is a measure of the probability of confidence in the model to accurately predict positive outcomes for actual positive instances. Therefore, the precision of the ML model can be formally defined as the frequency of correct predictions for actual positive instances:

$$Precision = \frac{TP}{TP + FP}$$

The trade-off between recall and precision values allows to tune the parameters of the classification models and achieve the desired results. For example, in the binary spam classifier, a higher recall would avoid missing too many spam emails (lower FN), but would incorrectly predict more emails as spam (higher FP). Whereas, a higher precision would reduce the number of incorrect predictions of emails as spam (lower FP), but would miss more spam emails (higher FN). F-measure allows to analyze the trade-off between recall and precision by providing the harmonic average, ideally 1, of these metrics:

$$F{-}measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

The Coefficient of Variation (CV) is another accuracy metric, particularly used for reporting the performance of unsupervised models that conduct classification using clusters (or states). CV is a standardized measure of dispersion that represents the intra-cluster (or intra-state) similarity. A lower CV implies a small variability of each outcome in relation to the mean of the corresponding cluster. This represents a higher intra-cluster similarity and a higher classification accuracy.

2.6 Evolution of machine learning techniques

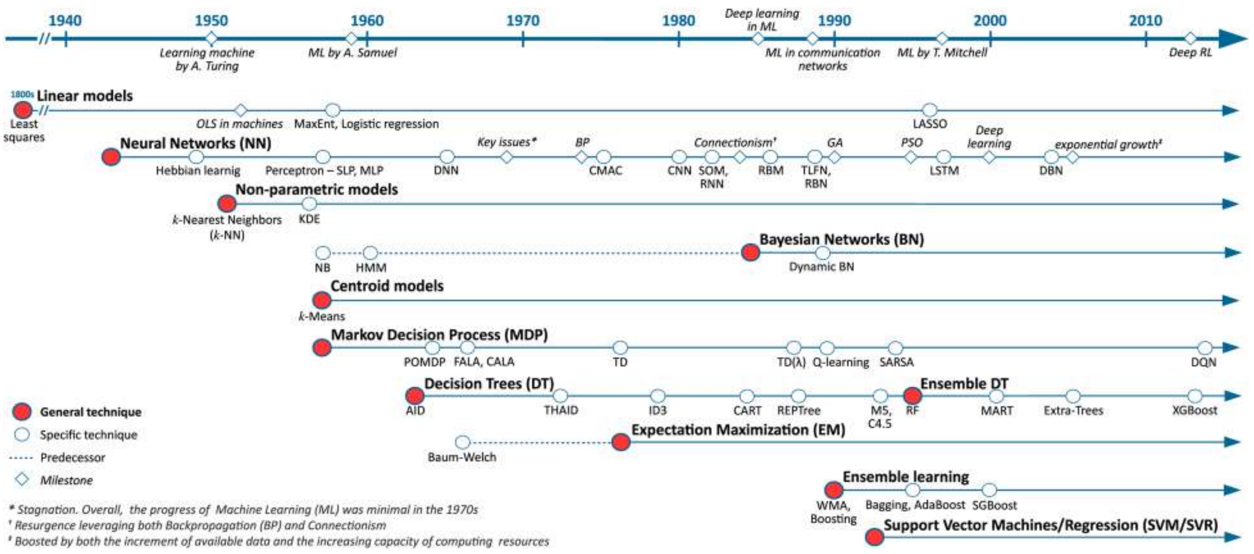
Machine learning is a branch of artificial intelligence whose foundational concepts were acquired over the years from contributions in the areas of computer science, mathematics, philosophy, economics, neuroscience, psychology, control theory, and more [397]. Research efforts during the last 75 years have given rise to a plethora of ML techniques [15, 169, 397, 435]. In this section, we provide a brief history of ML focusing on the techniques that have been particularly applied in the area of computer networks (cf., Fig. 5).

The beginning of ML dates back to 1943, when the first mathematical model of NNs for computers was proposed by McCulloch [302]. This model introduced a basic unit called artificial neuron that has been at the center of NN development to this day. However, this early model required to manually establish the correct weights of the connections between neurons. This limitation was addressed in 1949 by Hebbian learning [184], a simple rule-based algorithm for updating the connection weights of the early NN model. Like the neuron unit, Hebbian learning greatly influenced the progress of NN. These two concepts led to the construction of the first NN computer in 1950, called SNARC (Stochastic Neural Analog Reinforcement Computer) [397]. In the same year, Alan Turing proposed a test –where a computer tries to fool a human into believing it is also human– to determine if a computer is capable of showing intelligent behavior. He described the challenges underlying his idea of a *"learning machine"* in [449]. These developments encouraged many researchers to work on similar approaches, resulting in two decades of enthusiastic and prolific research in the ML area.

In the 1950s, the simplest linear regression model called Ordinary Least Squares (OLS) –derived from the least squares method [266, 423] developed around the 1800s– was used to calculate linear regressions in electro-mechanical desk calculators [168]. To the best of our knowledge, this is the first evidence of using OLS in computing machines. Following this trend, two linear models for conducting classification were introduced: Maximum Entropy (MaxEnt) [215, 216] and logistic regression [105]. A different research trend centered on pattern recognition exposed two non-parametric models (i.e. not restricted to a bounded set of parameters) capable of performing regression and classification: k-Nearest Neighbors (k-NN) [147, 420] and Kernel Density Estimation (KDE) [388], also known as Parzen density [349]. The former uses a distance metric to analyze the data, while the latter applies a kernel function (usually, Gaussian) to estimate the probability density function of the data.

The 1950s also witnessed the first applications of the Naïve Bayes (NB) classifier in the fields of pattern recognition [97] and information retrieval [297]. NB, whose foundations date back to the 18th and 19th centuries [43, 261], is a simple probabilistic classifier that applies Bayes' theorem on features with strong independence assumptions. NB was later generalized using KDE, also known as NB with Kernel Estimation (NBKE), to estimate the conditional probabilities of the features. In the area of clustering, Steinhaus [422] was the first to propose a continuous version of the to be called k-Means algorithm [290], to partition a heterogeneous solid with a given internal mass distribution into k subsets. The proposed centroid model employs a distance metric to partition the data into clusters where the distance to the centroid is minimized.

In addition, the Markov model [159, 296] (elaborated 50 years earlier) was leveraged to construct a process based on discrete-time state transitions and action rewards, named Markov Decision Process (MDP), which formalizes sequential decision-making problems in a fully observable, controlled environment [46]. MDP has been essential for the development of prevailing RL techniques [435]. Research efforts building on the initial NN model flourished too: the modern concept of perceptron was introduced as the first NN model that could learn the weights from input examples [387]. This model describes two NN classes according to the number of layers: Single-Layer Perceptron (SLP), an NN with one input layer



* Stagnation. Overall, the progress of Machine Learning (ML) was minimal in the 1970s
† Resurgence leveraging both Backpropagation (BP) and Connectionism
‡ Boosted by both the increment of available data and the increasing capacity of computing resources

Fig. 5 The evolution of machine learning techniques with key milestones

and one output layer, and Multi-Layer Perceptron (MLP), an NN with one or more hidden layers between the input and the output layers. The perceptron model is also known as Feedforward NN (FNN) since the nodes from each layer exhibit directed connections only to the nodes of the next layer. In the remainder of the paper, MLP-NNs and NNs in general, will be denoted by the tuple $(input\_nodes, hidden\_layer\_nodes^{+}, output\_nodes)$, for instance a $(106, 60, 40, 1)$ MLP-NN has a 160-node input layer, two hidden layers of 60 and 40 nodes respectively, and a single node output layer.

By the end of the 1950s, the term *"Machine Learning"* was coined and defined for the first time by Arthur Samuel (cf., Section 2), who also developed a checkers-playing game that is recognized as the earliest self-learning program [401]. ML research continued to flourish in the 1960s, giving rise to a novel statistical class of the Markov model, named Hidden Markov Model (HMM) [426]. An HMM describes the conditional probabilities between hidden states and visible outputs in a partially observable, autonomous environment. The Baum-Welch algorithm [41] was proposed in the mi-1960s to learn those conditional probabilities. At the same time, MDP continued to instigate various research efforts. The partially observable Markov decision process (POMDP) approach to finding optimal or near-optimal control strategies for partially observable stochastic environments, given a complete model of the environment, was first proposed by Cassandra et al. [25] in 1965, while the algorithm to find the optimal solution was only devised 5 years later [416]. Another development in MDP was the learning automata –officially published in 1973 [448]–, a Reinforcement Learning (RL) technique that continuously updates the probabilities of taking actions in an observed environment, according to given rewards. Depending on the nature of the action set, the learning automata is classified as Finite Action-set Learning Automata (FALA) or Continuous Action-set Learning Automata (CALA) [330].

In 1963, Morgan and Sonquis published Automatic Interaction Detection (AID) [323], the first regression tree algorithm that seeks sequential partitioning of an observation set into a series of mutually exclusive subsets, whose means reduces the error in predicting the dependent variable. AID marked the beginning of the first generation of Decision Trees (DT). However, the application of DTs to classification problems was only initiated a decade later by Morgan and Messenger's Theta AID (THAID) [305] algorithm.

In the meantime, the first algorithm for training MLP-NNs with many layers –also known as Deep NN (DNN) in today's jargon– was published by Ivakhnenko and Lapa in 1965 [210]. This algorithm marked the commencement of the Deep Learning (DL) discipline, though the term only started to be used in the 1980s in the general context of ML, and in the year 2000 in the specific context of NNs [9]. By the end of the 1960s, Minsky and Papertkey's *Perceptrons* book [315] drew the limitations of perceptrons-based NN through mathematical analysis, marking a historical turn in AI and ML in particular, and significantly reducing the research interest for this area over the next several years [397].

Although ML research was progressing slower than projected in the 1970s [397], the 1970s were marked by milestones that greatly shaped the evolution of ML, and contributed to its success in the following years. These include the Backpropagation (BP) algorithm, the

Cerebellar Model Articulation Controller (CMAC) NN model [11], the Expectation Maximization (EM) algorithm [115], the to-be-referred-to as Temporal Difference (TD) learning [478], and the Iterative Dichotomiser 3 (ID3) algorithm [373].

Werbos's application of BP –originally a control theory algorithm from the 1960s [80, 81, 233]– to train NNs [472] resurrected the research in the area. BP is to date the most popular NN training algorithm, and comes in different variants such as Gradient Descent (GD), Conjugate Gradient (CG), One Step Secant (SS), Levenberg-Marquardt (LM), and Resilient backpropagation (Rp). Though, BP is widely used in training NNs, its efficiency depends on the choice of initial weights. In particular, BP has been shown to have slower speed of convergence and to fall into local optima. Over the years, global optimization methods have been proposed to replace BP, including Genetic Algorithms (GA), Simulated Annealing (SA), and Ant Colony (AC) algorithm [500]. In 1975, Albus proposed CMAC, a new type of NN as an alternative to MLP [11]. Although CMAC was primarily designed as a function modeler for robotic controllers, it has been extensively used in RL and classification problems for its faster learning compared to MLP.

In 1977, in the area of statistical learning, Dempster et al. proposed EM, a generalization of the previous iterative, unsupervised methods, such as the Baum-Welch algorithm, for learning the unknown parameters of statistical HMM models [115]. At the same time, Witten developed an RL approach to solve MDPs, inspired by animal behavior and learning theories [478], that was later referred to as Temporal Difference (TD) in Sutton's work [433, 434]. In this approach the learning process is driven by the changes, or differences, in predictions over successive time steps, such that the prediction at any given time step is updated to bring it closer to the prediction of the same quantity at the next time step.

Towards the end of the 1970s, the second generation of DTs emerged as the Iterative Dichotomiser 3 (ID3) algorithm was released. The algorithm, developed by Quinlan [373], relies on a novel concept for attribute selection based on entropy maximization. ID3 is a precursor to the popular and widely used C4.5 and C5.0 algorithms.

The 1980s witnessed a renewed interest in ML research, and in particular in NNs. In the early 1980s, three new classes of NNs emerged, namely Convolutional Neural Network (CNN) [157], Self-Organizing Map (SOM) [249], and Hopfield network [195]. CNN is a feedforward NN specifically designed to be applied to visual imagery analysis and classification, and thus require minimal image preprocessing. Connectivity between neurons in CNN is inspired by the organization of the animal visual cortex –modeled by Hubel in the 1960s [200, 201]–, where the visual field is divided between neurons, each responding to stimuli only in its corresponding region. Similarly to CNN, SOM was also designed for a specific application domain; dimensionality reduction [249]. SOMs employ an unsupervised competitive learning approach, unlike traditional NNs that apply error-correction learning (such as BP with gradient descent).

In 1982, the first form of Recurrent Neural Network (RNN) was introduced by Hopfield. Named after the inventor, Hopfield network is an RNN where the weights connecting the neurons are bidirectional. The modern definition of RNN, as a network where connections between neurons exhibit one or more than one cycle, was introduced by Jordan in 1986 [226]. Cycles provide a structure for internal states or memory allowing RNNs to process arbitrary sequences of inputs. As such, RNNs are found particularly useful in Time Series Forecasting (TSF), handwriting recognition and speech recognition.

Several key concepts emerged from the 1980s' *connectionism movement*, one of which is the concept of *distributed representation* [187]. Introduced by Hinton in 1986, this concept supports the idea that a system should be represented by many features and that each feature may have different values. Distributed representation establishes a many-to-many relationship between neurons and *(feature,value)* pairs for improved efficiency, such that a *(feature,value)* input is represented by a pattern of activity across neurons as opposed to being locally represented by a single neuron. The second half of 1980s also witnessed the increase in popularity of the BP algorithm and its successful application in training DNNs [263, 394], as well as the emergence of new classes of NNs, such as Restricted Boltzmann Machines (RBM) [413], Time-Lagged Feedforward Network (TLFN) [260], and Radial Basis Function Neural Network (RBFNN) [260].

Originally named Harmonium by Smolensky, RBM is a variant of Boltzmann machines [2] with the restriction that there are no connections within any of the network layers, whether it is visible or hidden. Therefor, neurons in RBMs form a bipartite graph. This restriction allows for more efficient and simpler learning compared to traditional Boltzmann machines. RBMs are found useful in a variety of application domains such as dimensionality reduction, feature learning, and classification, as they can be trained in both supervised and unsupervised ways. The popularity of RBMs and the extent of their applicability significantly increased after the mid-2000s as Hinton introduced in 2006 a faster learning method for Boltzmann machines called Contrastive Divergence [186] making RBMs even more attractive for deep learning [399]. Interestingly, although the use of the term "deep learning" in the ML community dates back to 1986 [111], it did not apply to NNs at that time.

Towards the end of 1980s, TLFN –an MLP that incorporates the time dimension into the model for conducting TSF [260]–, and RBFNN –an NN with a weighted set of Radial Basis Function (RBF) kernels that can be trained in unsupervised and supervised ways [78]– joined the growing list of NN classes. Indeed any of the above NNs can be employed in a DL architecture, either by implementing a larger number of hidden layers or stacking multiple simple NNs.

In addition to NNs, several other ML techniques thrived during the 1980s. Among these techniques, Bayesian Network (BN) arose as a Directed Acyclic Graph (DAG) representation model for the statistical models in use [352], such as NB and HMM –the latter considered as the simplest dynamic BN [107, 110]–. Two DT learning algorithms, similar to ID3 but developed independently, referred to as Classification And Regression Trees (CART) [76], were proposed to model classification and regression problems. Another DT algorithm, under the name of Reduced Error Pruning Tree (REPTree), was also introduced for classification. REPTree aimed at building faster and simpler tree models using information gain for splitting, along with reduced-error pruning [374].

Towards the end of 1980s, two TD approaches were proposed for reinforcement learning: TD(λ) [433] and Q-learning [471]. TD(λ) adds a discount factor ($0 \leq \lambda \leq 1$) that determines to what extent estimates of previous state-values are eligible for updating based on current errors, in the policy evaluation process. For example, TD(0) only updates the estimate of the value of the state preceding the current state. Q-learning, however, replaces the traditional state-value function of TD by an action-value function (i.e. Q-value) that estimates the utility of taking a specific action in specific states. As of today, Q-learning is the most well-studied and widely-used model-free RL algorithm. By the end of the decade, the application domains of ML started expending to the operation and management of communication networks [57, 217, 289].

In the 1990s, significant advances were realized in ML research, focusing primarily on NNs and DTs. Bio-inspired optimization algorithms, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), received increasing attention and were used to train NNs for improved performance over the traditional BP-based learning [234, 319]. Probably one of the most important achievements in NNs was the work on Long Short-Term Memory (LSTM), an RNN capable of learning long-term dependencies for solving DL tasks that involve long input sequences [192]. Today, LSTM is widely used in speech recognition as well as natural language processing. In DT research, Quinlan published the M5 algorithm in 1992 [375] to construct tree-based multivariate linear models analogous to piecewise linear functions. One well-known variant of the M5 algorithm is M5P, which aims at building trees for regression models. A year later, Quinlan published C4.5 [376], that builds on and extends ID3 to address most of its practical shortcomings, including data overfitting and training with missing values. C4.5 is to date one of the most important and widely used algorithms in ML and data mining.

Several techniques other than NN and DT also prospered in the 1990s. Research on regression analysis propounded the Least Absolute Selection and Shrinkage Operator (LASSO), which performs variable selection and regularization for higher prediction accuracy [445]. Another well-known ML technique introduced in the 1990s was Support Vector Machines (SVM). SVM enables plugging different kernel functions (e.g. linear, polynomial, RBF) to find the optimal solution in higher-dimensional feature spaces. SVM-based classifiers find a hyperplane to discriminate between categories. A single-class SVM is a binary classifier that deduces the hyperplane to differentiate between the data belonging to the class against the rest of the data, that is, *one-vs-rest*. A multi-class approach in SVM can be formulated as a series of single class classifiers, where the data is assigned to the class that maximizes an output function. SVM has been widely used primarily for classification, although a regression variant exists, known as Support Vector Regression (SVR) [70].

In the area of RL, SARSA (State-Action-Reward-State-Action) was introduced as a more realistic, however less practical, Q-learning variation [395]. Unlike Q-learning, SARSA does not update the Q-value of an action based on the maximum action-value of the next state, but instead it uses the Q-value of the action chosen in the next state.

A new emerging concept called *ensemble learning* demonstrated that the predictive performance of a single learning model can be be improved when combined with other models [397]. As a result, the poor performance of a single predictor or classifier can be compensated with ensemble learning at the price of (significantly) extra computation. Indeed the results from ensemble learning must be aggregated, and a variety of techniques have been proposed in this matter. The first instances of ensemble learning include Weighted Majority Algorithm (WMA) [279], boosting [403], bootstrap aggregating (or bagging) [75], and Random Forests (RF) [191]. RF focused explicitly on tree models and marked the beginning of a new generation of ensemble DT. In addition, some variants of the original boosting algorithm were also developed, such as Adaptive Boosting (AdaBoost) [153] and Stochastic Gradient Boosting (SGBoost) [155].

These advances in ML facilitated the successful deployment of major use cases in the 1990s, particularly, handwriting recognition [419] and data mining [3]. The latter represented a great shift to data-driven ML, and since then it has been applied in many areas (e.g., retail, finance, manufacturing, medicine, science) for processing huge amounts of data to build models with valuable use [169]. Furthermore, from a conceptual perspective, Tom Mitchell formally defined ML: "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*" [317].

The 21st century began with a new wave of increasing interest in SVM and ensemble learning, and in particular ensemble DT. Research efforts in the field generated some of the the most widely used implementations of ensemble DT as of today: Multiple Additive Regression Trees (MART) [154], extra-trees [164], and eXtreme Gradient Boosting (XGBoost) [93]. MART and XGBoost are respectively a commercial and open source implementation of Friedman's Gradient Boosting Decision Tree (GBDT) algorithm; an ensemble DT algorithm based on gradient boosting [154, 155]. Extra-trees stands for *extremely randomized trees*, an ensemble DT algorithm that builds random trees based on *k* randomly chosen features. However instead to computing an optimal split-point for each one of the *k* features at each node as in RF, extra-trees selects a split-point randomly for reduced computational complexity.

At the same time, the popularity of DL increased significantly after the term "deep learning" was first introduced in the context of NNs in 2000 [9]. However, the attractiveness of DNN started decreasing shortly after due to the experienced difficulty of training DNNs using BP (e.g. vanishing gradient problem), in addition to the increasing competitiveness of other ML techniques (e.g. SVM) [169]. Hinton's work on Deep Belief Networks (DBN), published in 2006 [188], gave a new breath and strength to research in DNNs. DBN introduced an efficient training strategy for deep learning models, which was further used successfully in different classes of DNNs [49, 381]. The development in ML –particularly, in DNNs– grew exponentially with advances in storage capacity and large-scale data processing (i.e. Big Data) [169]. This wave of popularity in deep learning has continued to this day, yielding major research advances over the years. One approach that is currently receiving tremendous attention is Deep RL, which incorporates deep learning models into RL for solving complex problems. For example, Deep Q-Networks (DQN) –a combination of DNN and Q-learning– was proposed for mastering video games [318]. Although the term Deep RL was coined recently, this concept was already discussed and applied 25 years ago [275, 440].

It is important to mention that the evolution in ML research has enabled improved learning capabilities which were found useful in several application domains, ranging from games, image and speech recognition, network operation and management, to self-driving cars [120].

3 Traffic prediction

Network traffic prediction plays a key role in network operations and management for today's increasingly complex and diverse networks. It entails forecasting future traffic and traditionally has been addressed via time series forecasting (TSF). The objective in TSF is to construct a regression model capable of drawing accurate correlation between future traffic volume and previously observed traffic volumes.

Existing TSF models for traffic prediction can be broadly decomposed into statistical analysis models and supervised ML models. Statistical analysis models are typically built upon the generalized autoregressive integrated moving average (ARIMA) model, while majority of learning for traffic prediction is achieved via supervised NNs. Generally, the ARIMA model is a popular approach for TSF, where autoregressive (AR) and moving average (MA) models are applied in tandem to perform auto-regression on the differenced and "stationarized" data. However, with the rapid growth of networks and increasing complexity of network traffic, traditional TSF models are seemingly compromised, giving rise to more advanced ML models. More recently, efforts have been made to reduce overhead and, or improve accuracy in traffic prediction by employing features from flows, other than traffic volume. In the following subsections, we discuss the various traffic prediction techniques that leverage ML and summarize them in Table 3.

3.1 Traffic prediction as a pure TSF problem

To the best of our knowledge, Yu et al. [489] were the first to apply ML in traffic prediction using MLP-NN. Their primary motive was to improve *accuracy* over traditional AR methods. This was supported by rigorous independent mathematical proofs published in the late eighties and the early nineties by Cybenko [106], Hornik [196], and Funahashi [158]. These proofs showed that SLP-NN approximators, which employed sufficient number of neurons of continuous sigmoidal activation type (a constraint introduced by Cybenko and relaxed by Hornik), were universal approximators, capable of approximating any continuous function to any desired accuracy.

In the last decade, different types of NNs (SLP, MLP, RNN, etc.) and other supervised techniques have been employed for TSF of network traffic. Eswaradass et al. [141] propose a MLP-NN-based bandwidth prediction system for Grid environments and compare it to the Network Weather Service (NWS) [480] bandwidth forecasting AR models for traffic monitoring and measurement. The goal of the system is to forecast the available bandwidth on a given path by feeding the NN with the minimum, maximum and average number of bits per second used on that path in the last epoch (ranging from 10 to 30 s). Experiments on the *dotresearch.org* network and the

Table 3 Summary of TSF and non-TSF-based traffic prediction

Ref.	ML Technique	Application *(approach)*	Dataset *(availability)*	Features	Output *(training)*	Evaluation	
						Settings	Results[ab]
NBP [141]	Supervised: · MLP-NN *(offline)*	End-to-end path bandwidth availability prediction *(TSF)*	NSF TeraGrid dataset *(N/A)*	Max, Min, Avg load observed in past 10 s ∼ 30 s	Available bandwidth on a end-to-end path in future epoch	Number of features= 3 MLP-NN: · *(N/A)*	MSE = 8%
Cortez et al. [104]	Supervised: · NNE trained with Rp *(offline)*	Link load and traffic volume prediction in ISP networks *(TSF)*	SNMP traffic data from 2 ISP nets, · traffic on a transatlantic link · aggregated traffic in the ISP backbone *(N/A)*	Traffic volume observed in past few minutes∼several days	Expected traffic volume	Number of features= 6 ∼ 9 5 NNs NNE: · all SLPs for dataset1 · 1 hidden layer MLPs with 6 ∼ 8 neurons for dataset2	1h lookahead: · MAPE = 1.43% ∼ 5.23% 1h ∼ 24h lookahead: · MAPE = 6.34% ∼ 23.48%
Bermolen et al. [52]	Supervised: · SVR *(offline)*	Link load prediction in ISP networks *(TSF)*	Internet traffic collected at the POP of an ISP network *(N/A)*	Link load observed at τ time scale	Expected link load	Number of features= d samples with $d = 1..30$ Number of support vectors: · varies with d (e.g. ∼320 for $d = 10$)	RMSE < 2 for $\tau = 1ms$ and $d = 5$ · ≈ AR ·10% less than MA
Chabaa et al. [86]	Supervised: MLP-NN with different training algorithms (GD, CG, SS, LM, Rp) *(offline)*	Network traffic prediction *(TSF)*	1000 points dataset *(N/A)*	Past measurements	Expected traffic volume	Number of features *(N/A)* MLP-NN: · 1 hidden layer	LM: · RMSE= 0.0019 RPE = 0.0230% Rp: · RMSE= 0.0031 RPE= 0.0371%
Zhu et al. [500]	Supervised: MLP-NN with PSO-ABC *(offline)*	Network traffic prediction *(TSF)*	2-week hourly traffic measurements *(N/A)*	N past days hourly traffic volume	Expected next-day hourly traffic volume	Number of features= 5 MLP-NN *(5, 11, 1)* PSO-ABC: · *30 particles of length=66*	MSE = 0.006 on normalized data 50% less than BP
Li et al. [274]	Supervised: MLP-NN *(offline)*	Traffic volume prediction on an inter-DC link *(Regression)*	6-week inter-DC traffic dataset from Baidu · SNMP counters data collected every 30 s · Top-5 applications traffic data collected every 5 min *(N/A)*	Level-N wavelet transform used to extract time and frequency features from total and elephant traffic volumes time series	$k \times$ 30-s ahead expected traffic volume	Number of wavelets: · $N = 10$ Number of features= $k \times 120$ for $N = 10$ 1 hidden layer MLP-NN	RRMSE= 4% ∼ 10% for $k = 1 \sim 40$
Chen et al. [94]	Supervised: · KBR · LSTM-RNN *(offline)*	Inferring future traffic volume based on flow statistics *(regression)*	Network traffic volume and flow count collected every 5 min over a 24-week period *(public)*	Flow count	Expected traffic volume	Number of features: · 1 feature (past sample) LSTM-RNN: · *(N/A)*	RNN · MSE > 0.3 on normalized data · 0.05 higher than KBR · twice as much as RNN fed with traffic volume time series
Poupart et al. [365]	Supervised: · GPR · oBMM · MLP-NN *(offline)*	Early flow-size prediction and elephant flow detection *(classification)*	3 university and academic networks datasets with over three million flows each *(public)*	· source IP · destination IP · source port · destination port · protocol · server vs. client · size of 3 first packets	Flow size class; elephant vs. non-elephant	Number of features: · 7 features MLP-NN: · (106,60,40,1)	GPR: · TPR> 80% · TNR> 80% oBMM: · TPR and TNR ≈ 100% on one dataset · TPR< 50% on other datasets MLP-NN: · TPR> 80% · lowest TNR< 80%

[a] Average values. Results vary according to experimental settings

[b] Accuracy metrics: mean square error (MSE), relative prediction error (RPE), mean absolute prediction error (MAPE), average root mean square error (RMSE)

40 gigabit/s NSF TeraGrid network datasets show that the NN outperforms the NWS bandwidth forecasting models with an error rate of up to 8 and 121.9% for MLP-NN and NWS, respectively. Indeed the proposed NN-based forecasting system shows better learning ability than NWS's. However, no details are provided for the characteristics of the MLP employed in the study, nor the time complexity of the system compared to NWS.

Cortez et al. [104] choose to use a NN ensemble (NNE) of five MLP-NN with one hidden layer each. Resilient backpropagation (Rp) training is used on SNMP traffic data collected from two different ISP networks. The first data represents the traffic on a transatlantic link, while the second represents the aggregated traffic in the ISP backbone. Linear interpolation is used to complete missing SNMP data. The NNE is tested for *real-time* forecasting (online forecasting on a few-minute sample), *short-term* (one-hour to several-hours sample), and *mid-term* forecasting (one-day to several-days sample). The NNE is compared against AR models of traditional Holt-Winters, double Holt-Winters seasonal variant to identify repetitions in patterns at fixed time periods, and ARIMA. The comparison amongst the TSF methods show that in general the NNE produces the lowest MAPE for both datasets. It also shows that in terms of time and computational complexity, NNE outperforms the other methods with an order of magnitude, and is well suited for real-time forecasting.

The applicability of NNs in traffic prediction instigated various other efforts [86, 500] to compare and contrast various training algorithms for network traffic prediction. Chabaa et al. [86] evaluate the performance of various BP training algorithms to adjust the weights in the MLP-NN, when applied to Internet traffic time series. They show superior performance, with respect to RMSE and RPE, of the Levenberg-Marquardt (LM) and the Resilient backpropagation (Rp) algorithms over other BP algorithms.

In contrast to the local optimization in BP, Zhu et al. [500] propose a hybrid training algorithm that is based on global optimization, the PSO-ABC technique [98]. It is an artificial bee colony (ABC) algorithm employing particle swarm optimization (PSO), an evolutionary search algorithm. The training algorithm is implemented with a $(5, 11, 1)$ MLP-NN. Experiments on a 2 weeks hourly traffic measurement dataset show that PSO-ABC has higher prediction accuracy than BP, with an MSE of 0.006 and 0.011, respectively, on normalized data and has stable prediction performance. Furthermore, the hybrid PSO-ABC has a faster training time than ABC or PSO.

On the other hand, SVM has a low computational overhead and is more robust to parameter variations (e.g. time scale, number of samples) in general. However, they are usually applied to classification rather than TSF. SVM and its regression variant, SVR, are scrutinized for their applicability to traffic prediction in [52]. Bermolen et al. [52] consider the prospect of applying SVR for link load forecasting. The SVR model is tested on heterogeneous Internet traffic collected at the POP of an ISP network. At 1sec timescale, the SVR model shows a slight improvement over an AR model in terms of RMSE. A more significant 10% improvement is achieved over a MA model. Most importantly, SVR is able to achieve 9000 forecast per sec with 10 input samples, and shows the potential for real-time operation.

3.2 Traffic prediction as a non-TSF problem

In contrast to TSF methods, network traffic can be predicted leveraging other methods and features. For instance, Li et al. [274] propose a frequency domain based method for network traffic flows, instead of just traffic volume. The focus is on predicting incoming and outgoing traffic volume on an inter-data center link dominated by elephant flows. Their models incorporate FNN, trained with BP using simple gradient descent and wavelet transform to capture both the time and frequency features of the traffic time series. Elephant flows are added as separate feature dimensions in the prediction. However, collecting all elephant flows at high frequencies is more expensive than byte count for traffic volume. Therefore, elephant flow information is collected at lower frequencies and interpolated to fill in the missing values, overcoming the overhead for elephant flow collection.

The dataset contains the total incoming and outgoing traffic collected in 30 s intervals using SNMP counters on the data center (DC) edge routers and inter-DC link at Baidu over a six-week period. The top 5 applications account for 80% of the total incoming and outgoing traffic data, which is collected every 5 min and interpolated to estimate missing values at the 30 s scale. The time series is decomposed using level 10 wavelet transform, leading to 120 features per timestamp.

Thus, k-step ahead predictions, feed $k \times 120$ features into the NN and show a relative RMSE (RRMSE) ranging from 4 to 10% for the NN-Wavelet transformation model as the prediction horizon varies from 30 s to 20 min. Evidently, wavelet transformation reduces the average prediction errors for different prediction horizons by 5.4 and 2.9% for incoming and outgoing traffic, respectively. In contrast, the linear ARIMA model has prediction error of approximately 8.5 and 6.9% for incoming and outgoing traffic, respectively. The combined NN and wavelet transform model reduces the peak inter-DC link utilization, i.e. the ISP's billed utilization, by about 9%. However, the model does not seem to be fully considering the features related to the elephant flow, which may explain the inexplicable good performance of the 0-interpolation, a simple method that fills zeros for all unknown points.

Chen et al. [94], investigate the possibility of reducing cost of monitoring and collecting traffic volume, by inferring future traffic volume based on flow count only. They propose a HMM to describe the relationship between the flow count, flow volume and their temporal dynamic behavior. The Kernel Bayes Rule (KBR) and RNN with LSTM unit is used to predict future traffic volume based on current flow count. A normalized dataset, with mean=0 and standard deviation=1, consists of network traffic volumes and corresponding flow counts collected every 5 min over a 24-week period [391]. The RNN shows a prediction MSE of 0.3 at best, 0.05 higher than KBR and twice as much as the prediction error of an RNN fed with traffic volume instead of flow count. Therefore, though the motive was to promote flow count based traffic prediction to overcome the cost of monitoring traffic volume, the performance is compromised.

Poupart et al. [365] explore the use of different ML techniques for flow size prediction and elephant flow detection. These techniques include gaussian processes regression (GPR), online bayesian moment matching (oBMM) and a (106, 60, 40, 1) MLP-NN. Seven features are considered for each flow, including source IP, destination IP, source port, destination port, protocol, server versus client (if protocol is TCP), and the size of the first three data packets after the protocol/synchronization packets.

The datasets consist of three public datasets from two university networks [50] and an academic building at Dartmouth College [251] with over three million flows each. Elephant flow detection is based on a flow size threshold that varies from 10KB to 1MB. The experiments show noticeable discrepancies in the performance of the approaches with varying datasets. Although oBMM outperforms all other approaches in one dataset with an average TPR and TNR very close to 100%, it fails miserably in the other datasets with an average TPR below 50% for one dataset. In the latter dataset, oBMM seems to suffer the most from class imbalance. As the detection flow size threshold increases, less flows are tagged as elephant flows, creating class imbalance in the training dataset and leading to lower TPR. However, it is worth noting that oBMM outperforms all other approaches in terms of average TNR in all 3 datasets. On the other hand, NN and GPR, have an average TPR consistently above 80%. Although NN outperforms GPR in terms of robustness to class imbalance by looking at the consistency of its TPR with varying flow size threshold, it has the lowest average TNR of below 80% in all datasets.

The motive for flow size prediction in [365], is to discriminate elephant flows from mice flows in routing to speed up elephant flow completion time. Presumably, mice flows are routed through Equal-cost multi-path routing, while elephant flows are routed through the least congested path. The performance of the routing policy combined with GPR and oBMM for elephant flow prediction is tested with a varying subset of features. According to the authors, GPR improves the completion time by 6.6% in average for 99% of elephant flows when only the first packet header information is considered. A 14% improvement is observed when the size of the three first packets is used along with the header information. It is also noticed that considering the size of the first three packets alone leads to over 13.5% improvement, regardless of whether GPR or oBMM is used, with a very slight impact on the completion time of mice flows.

3.3 Summary

Supervised NNs (including MLP and RNN) have been successfully applied to traffic prediction, as shown in Table 9. TSF approaches, such as [52, 86, 104, 141, 500], where NNs are used to infer traffic volumes from past measured volumes, show high long-term and short-term prediction accuracy at low complexity with limited number of features and limited number of layers and neurons.

Unfortunately, TSF approaches are restrictive in general. In fact they are only possible if past observations on the prediction variable are made. For instance, in order to predict the traffic for a particular flow *f* on link *l*, there must be a counter on link *l* actively measuring the traffic for that particular flow *f*, which can be challenging on very high speed links. Because it might not be possible to have the appropriate counter in place, or because it might be technically difficult to conduct measurements at the required speed or granularity, non-TSF approaches can be useful.

Non-TSF approaches were investigated in [94, 274, 365] to infer traffic volumes from flow count and packet header fields. Although higher prediction error rates are experienced, these rates remain relatively low not only for NNs but also for other supervised learning techniques, such as GPR and oBMM. According to [365] a more complex MLP-NN (in terms of number of layers and neurons) might be required to achieve better accuracy in a non-TSF setting, and the performance of supervised learning techniques varies when tested on different datasets. This motivates the need for ensemble learning.

It is envisaged that as the applicability of ML techniques for traffic prediction increases, traffic prediction will improve with respect to computational overhead and accuracy. Furthermore, learning will enable automation of various network operation and management activities, such as network planing, resource provisioning, routing optimization, and SLA/QoS management.

4 Traffic classification

Traffic classification is quintessential for network operators to perform a wide range of network operation and management activities. These include capacity planning, security and intrusion detection, QoS and service differentiation, performance monitoring, and resource provisioning, to name a few. For example, an operator of an enterprise network may want to prioritize traffic for business critical applications, identify unknown traffic for anomaly detection, or perform workload characterization for designing efficient resource management schemes that satisfy diverse applications performance and resource requirements.

Traffic classification requires the ability to accurately associate network traffic to pre-defined classes of interest. These classes of interest can be classes of applications (e.g. HTTP, FTP, WWW, DNS and P2P), applications (e.g. Skype [310], YouTube [488] and Netflix [331]), or class of service [390]. A class of service, for instance based on QoS, encompasses all applications or classes of applications that have the same QoS requirements. Therefore, it is possible that applications that apparently behave differently, belong to the same class of service [462].

Generally, network traffic classification methodologies can be decomposed into four broad categories that leverage port number, packet payload, host behavior or flow features [31, 244]. The classical approach to traffic classification simply associates Internet Assigned Numbers Authority (IANA) [207] registered port numbers to applications. However, since it is no longer the de facto, nor, does it lend itself to learning due to trivial lookup, it is not in the scope of this survey. Furthermore, relying solely on port numbers has been shown to be ineffective [125, 228, 320], largely due to the use of dynamic port negotiation, tunneling and misuse of port numbers assigned to well-known applications for obfuscating traffic and avoiding firewalls [54, 109, 176, 286]. Nevertheless, various classifiers leverage port numbers in conjunction with other techniques [31, 56, 244, 417] to improve the performance of the traffic classifiers. In the following subsections, we discuss the various traffic classification techniques that leverage ML and summarize them in Tables 4, 5, 6, 7 and 8.

4.1 Payload-based traffic classification

Payload-based traffic classification is an alternate to port-based traffic classification. However, since it searches through the payload for known application signatures, it incurs higher computation and storage costs. Also, it is cumbersome to manually maintain and adapt the signatures to the ever growing number of applications and their dynamics [138]. Furthermore, with the rise in security and privacy concerns, payload is often encrypted and its access is prohibited due to privacy laws. This makes it non-trivial to infer a signature for an application class using payload [54, 138].

Haffner et al. [176] reduce the computational overhead by using only the first few bytes of unidirectional, unencrypted TCP flows as binary feature vectors. For SSH and HTTPS encrypted traffic, they extract features from the unencrypted handshake that negotiate the encryption parameters of the TCP connection. They use NB, AdaBoost and MaxEnt for traffic classification. AdaBoost outperforms NB and MaxEnt, and yields an overall precision of 99% with an error rate within 0.5%.

Table 4 Summary of Payload⋆ and Host Behavior†-based Traffic Classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Haffner et al. [176]⋆	Supervised NB, AdaBoost, MaxEnt	Proprietary	Discrete byte encoding for first n bytes of unidirectional flow	FTP, SMTP, POP3, IMAP, HTTPS, HTTP, SSH	$n = 64 - 256$ bytes	Overall error rate <0.51%, precision > 99%, recall > 94%
Ma et al. [286]⋆	Unsupervised HCA	Proprietary: U. Cambridge, UCSD	Discrete byte encoding for first n bytes of unidirectional flow	FTP, SMTP, HTTP, HTTPS, DNS, NTP, NetBIOS, SrvLoc	$n = 64$ bytes, distance metric: PD = 250, MP = 150, CSG = 12%	Error rate: PD ≤ 4.15%, MP ≤ 9.97%, CSG ≤ 6.19%
Finamore et al. [146]⋆	Supervised SVM	Tstat [439]; NAPA-WINE [268]; Proprietary: ISP network	Statistical characterization of first N bytes of each packet a window of size C, divided into G groups of b consecutive bits	eMule, BitTorrent, RTP, RTCP, DNS, P2P-TV (PPLive, Joost, SopCast, TVAnts), Skype, Background	$C = 80, N = 12, G = 24, b = 4$	Average TP = 99.6%, FP < 1%
Schatzmann et al. [404]†	Supervised SVM	Proprietary: ISP network	Service proximity, activity profiles, session duration, periodicity	Mail, Non-Mail	N/A	Average accuracy = 93.2%, precision = 79.2%
Bermolan et al. [53]†	Supervised SVM	Proprietary: campus network, ISP network	Packet count exchanged between peers in duration ΔT	PPLive, TVAnts, SopCast, Joost	$\Delta T = 5$ s, SVM distance metric $R = 0.5$	Worst-case TPR ≈ 95%, FPR < 0.1%

N/A: Not available

Table 5 Summary of supervised flow feature-based traffic classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Roughan et al. [390]	Supervised *k*-NN	Proprietary: univ. networks, streaming service	Packet-level and flow-level features	Telnet, FTP-data, Kazaa, RealMedia Streaming, DNS, HTTPS	$k = 3$, number of QoS classes $= 3, 4, 7$	Error rate: 5.1% (4), 2.5% (3), 9.4% (7); (#): number of QoS Classes
Moore and Zuev [321]	Supervised NBKE	Proprietary: campus network	Baseline and derivative packet-level features	BULK, WWW, MAIL, SERVICES, DB, P2P, ATTACK, MULTIMEDIA	N/A	Accuracy upto 95%, TPR upto 99%
Jiang et al. [218]	Supervised NBKE	Proprietary: campus network	Baseline and derivative flow-level features	WWW, email, bulk, attack, P2P, multimedia, service, database, interaction, games	N/A	Average accuracy ≈ 91%
Park et al. [347]	Supervised REPTree, REPTree-Bagging	NLANR [457]	Packet-level, flow-level and connection-level features	WWW, Telnet, Messenger, FTP, P2P, Multimedia, SMTP, POP, IMAP, DNS, Services	Burst packet threshold $=$ 0.007s	Accuracy $\geq 90\%$ (features ≥ 7)
Zhang et al. [496]	Supervised BoF-NB	WIDE [474], proprietary: ISP network	Packet-level and flow-level features from unidirectional flows	BT, DNS, FTP, HTTP, IMAP, MSN, POP3, SMTP, SSH, SSL, XMPP	Aggregation rule $=$ *sum*, BoF size	Accuracy 87-94%, F-measure $=$ 80%
Zhang et al. [497]	Supervised RF, Unsupervised *k*-Means (BoF-based, RTC)	KEIO [474], WIDE [474], proprietary: ISP network	Packet-level and flow-level features from unidirectional flows	FTP, HTTP, IMAP, POP3, RAZOR, SSH, SSL, UNKNOWN / ZERO-DAY (BT, DNS, SMTP)	N/A	RTC upto 15% and 10% better in flow and byte accuracy, respectively, than second best F-measure $=$ 0.91 (before update), 0.94 (after update)
Auld et al. [26]	Supervised BNN	Proprietary	Packet-level and flow-level features	ATTACK, BULK, DB, MAIL, P2P, SERVICE, WWW	Number of features $= 246$, hidden layers $= 0$-1, 0-30 nodes in the hidden layer, output $= 10$	Accuracy $> 99\%$, 95% with temporally distant training and testing datasets
Sun et al. [431]	Supervised PNN	Proprietary: campus networks	Packet-level and flow-level features	P2P, WEB, OTHERS	Number of features $= 22$	Accuracy $= 87.99\%$; P2P: TPR $= 91.25\%$, FPR $= 1.36\%$; WEB: TPR $= 98.74\%$, FPR $= 27.7\%$
Este et al. [140]	Supervised SVM	LBNL [262], CAIDA [451], proprietary: campus network	Packet payload size	HTTP, SMTP, POP3, HTTPS, IMAPS, BitTorrent, FTP, MSN, eDonkey, SSL, SMB, Kazaa, Gnutella, NNTP, DNS, LDAP, SSH	Number of support vectors cf., [140]	TP $> 90\%$ for most classes
Jing et al. [223]	Supervised FT-SVM	Proprietary [270, 321]	A subset of 12 from 248 features [321]	BULK, INTERACTIVE, WWW, MAIL, SERVICES, P2P, ATTACK, GAME, MULTIMEDIA, OTHER	SVM parameters automatically chosen	Accuracy up to 96%, error ratio ↓ 2.35 times, avg. computation cost ↓ 7.65 times
Wang et al. [464]	Supervised multi-class SVM, unbalance d binary SVM	Proprietary: univ. network	Flow-level and connection-level features	BitTorrent, eDonkey, Kazaa, pplive	N/A	Accuracy 75-99%

N/A: Not available

Table 6 Summary of unsupervised flow feature-based traffic classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Liu et al. [283]	Unsupervised *k*-Means	Proprietary: campus network	Packet-level and flow-level features	WWW, MAIL, P2P, FTP (CONTROL, PASV, DATA), ATTACK, DATABASE, SERVICES, INTERACTIVE, MULTIMEDIA, GAMES	$k = 80$	Average accuracy ≈ 90%, minimum recall = 70%
Zander et al. [492]	Unsupervised AutoClass	NLANR [457]	Packet-level and flow-level features	AOL Messenger, Napster, Half-Life, FTP, Telnet, SMTP, DNS, HTTP	Intra-class homogeneity (*H*)	Mean accuracy = 86.5%
Erman et al. [136]	Unsupervised AutoClass	Univ. Auckland [457]	Packet-level and flow-level features	HTTP, SMTP, DNS, SOCKS, IRC, FTP (control, data), POP3, LIMEWIRE, FTP	N/A	Accuracy = 91.2%
Erman et al. [135]	Unsupervised DBSCAN	Univ. Auckland [457], proprietary: Univ. Calgary	Packet-level and flow-level features	HTTP, P2P, SMTP, IMAP, POP3, MSSQL, OTHER	*eps* = 0.03, *minPts* = 3, number of clusters = 190	Overall accuracy = 75.6%, average precision > 95% (7/9 classes)
Erman et al. [138]	Unsupervised *k*-Means	Proprietary: univ. network	Packet-level and flow-level features from unidirectional flows	Web, EMAIL, DB, P2P, OTHER, CHAT, FTP, STREAMING	$k = 400$	Server-to-client: Avg. flow accuracy = 95%, Avg. byte accuracy = 79%; Web: precision = 97%, recall = 97%; P2P: precision = 82%, recall = 77%

N/A: Not available

Their ML models are scalable and robust due to the use of partial payloads, and unidirectional flows and diverse usage patterns, respectively. The unidirectional flows circumvent the challenges due to asymmetric routing. In comparison to campus or enterprise networks, residential network data offer an increased diversity, with respect to, social group, age and interest with less spatial and temporal correlation in usage patterns. Unfortunately, performance of AdaBoost traffic classifier deteriorates with noisy data [176] and their approach requires a priori knowledge about the protocols in the application classes.

Ma et al. [286] show that payload-based traffic classification can be performed without any a priori knowledge of the application classes using unsupervised clustering. They train their classifiers based on the label of a single instance of a protocol and a list of partially correlated protocols, where a protocol is modeled as a distribution of sessions. Each session is a pair of unidirectional flow distributions, one from the source to the destination and another from the destination to the source. For tractability, the sessions are assumed to be finite and a protocol model is derived as a distribution on *n* byte flows, rather than pair of flows.

In product distribution (PD) protocol model, the *n* byte flow distribution is statistically represented as a product of *n* independent byte distributions, each describing the distribution of bytes at a particular offset in the flow. Similarly, in the Markov process (MP) protocol model, nodes are labeled with unique byte values and the edges are weighted with a transition probability, such that the sum of all egress transition probabilities from a node is one. A random walk through the directed graph identify discriminator strings that are not tied to a fixed offset. In contrast, the common substring graphs (CSG) capture structural information about the flows using longest common subsequence. A subsequence in a series of common substrings that capture commonalities including the fixed offsets in statistical protocol modeling.

Finally, the authors perform agglomerative (bottom-up) hierarchical clustering analysis (HCA) to group the observed protocols and distinguish between the classes of interest. They employ weighted relative entropy for PD and MP, and approximate graph similarity for CSG, as the distance metric. In evaluation, the PD-based protocol models resulted in the lowest total misclassification error, under 5%. Thus, there is a high invariance at fixed

Table 7 Summary of Early*, Sub-flow†-based and Encrypted‡ flow feature-based traffic classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Bernaille et al. [55]*	Unsupervised *k*-Means	Proprietary: univ. network	Packet size and direction of first *P* packets in a flow	eDonkey, FTP, HTTP, Kazaa, NNTP, POP3, SMTP, SSH, HTTPS, POP3S	$P = 5, k = 50$	Accuracy > 80%
TIE [108, 121]*	Supervised J48 DT, *k*-NN, Random Tree, RIPPER, MLP, NB	Proprietary: Univ. Napoli campus network	Payload size stats and inter-packet time stats of first *N* packets, bidirectional flow duration and size, transport protocol	BitTorrent, SMTP, Skype2Skype, POP, HTTP, SOULSEEK, NBNS, QQ, DNS,SSL RTP, EDONKEY	$N = 1 ... 10$	Overall accuracy = 98.4% with BKS (J48, Random Tree, RIPPER, PL) combiner, $N = 10$
Nguyen et al. [337]†	Supervised NB, C4.5 DT	Proprietary: home network, univ. network, game server	Inter-packet arrival time statistics, inter-packet length variation statistics, IP packet length statistics of *N* consecutive packets	Enemy Territory (online game), VoIP, Other	$N = 25$	C4.5 DT: Enemy Territory - recall* = 99.3%, prec.* = 97%; VoIP - recall* = 95.7%, precision* = 99.2% NB: Enemy Territory - recall* = 98.9%, prec.* = 87%, VoIP - recall* = 99.6%, precision* = 95.4% * median
Erman et al. [137]*	Semi-supervised *k*-Means	Proprietary: Univ. Calgary	Number of packets, average packet size, total bytes, total header bytes, total payload bytes (caller to callee and *vice versa*)	P2P, HTTP, CHAT, EMAIL, FTP, STREAMING, OTHER	$k = 400$, 13 layers, packet milestones (number of packets) in layers are separated exponentially $(8, 16, 32, \ldots)$	Flow accuracy > 94%, byte accuracy 70-90%
Li et al. [270]*	Supervised C4.5 DT, C4.5 DT with AdaBoost, NBKE	Proprietary	A subset of 12 from 248 features [321] of first *N* packets	WEB, MAIL, BULK, Attack, P2P, DB, Service, Interactive	$N = 5$	C4.5 DT: Accuracy >99%; Attack is an exception with moderate-high recall
Jin et al.[222]*	Supervised AdaBoost	Proprietary: ISP network, labeled as in [176]	Lowsrcport, highsrcport, duration, mean packet size, mean packet rate, toscount, tcpflags, dstinnet, lowdstport, highdstport, packet, byte, tos, numtosbytes, srcinnet	Business, chat, DNS, FileSharing, FTP, Games, Mail, Multimedia, NetNews, SecurityThreat, VoIP, Web	Number of binary classifiers (*k*): TCP = 12, UDP = 8	Error rate: TCP = 3%, UDP = 0.4%
Bonfiglio et al. [69]‡	Supervised NB, Pearson's χ^2 test	Proprietary: univ. network, ISP network	Message size, average inter-packet gap	Skype	NB decision threshold $B_{min} = -5$, $\chi^2(Thr) = 150$	NB$\wedge\chi^2$: UDP – E2E - FP = 0.01%, FN = 29.98% E2O - FP = 0.0%, FN = 9.82% (univ. dataset); E2E - FP = 0.01%, FN = 24.62% E2O - FP = 0.11%, FN = 2.40% (ISP dataset) TCP – negligible FP
Alshammari et al. [17]‡	Supervised AdaBoost, SVM, NB, RIPPER, C4.5 DT	AMP [457], MAWI [474], DARPA99 [278], proprietary from Univ. Dalhousie	Packet size, packet inter-arrival time, number of packets, number of bytes, flow duration, protocol (forward and backward direction)	SSH, Skype	N/A	C4.5 DT: SSH – DR = 95.9%, FPR = 2.8% (Dalhousie), DR = 97.2%, FPR = 0.8% (AMP), DR = 82.9%, FPR = 0.5% (MAWI) Skype – DR = 98.4%, FPR = 7.8% (Dalhousie)

Table 7 Summary of Early⋆, Sub-flow†-based and Encrypted‡ flow feature-based traffic classification *(Continued)*

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Shbair et al. [409]‡	Supervised C4.5 DT, RF	Synthetic trace	Statistical features from encrypted payload and [253] (client to server and *vice versa*)	Service Provider (number of services): Uni-lorraine.fr (15), Google.com (29), akamihd.net (6), Googlevideo.com (1), Twitter.com (3), Youtube.com (1), Facebook.com (4), Yahoo.com (19), Cloudfront.com (1)	N/A	RF (service provider): precision = 92.6%, recall = 92.8%, F-measure = 92.6% RF (service): accuracy in 95-100% for majority of service providers > 100 connections per HTTPS service

N/A: Not available

offsets in binary and textual protocols, such as DNS and HTTP, respectively. Though, the CSG resulted in a higher misclassification error, approximately 7%, it performed best for SSH encrypted traffic. However, it is important to realize that encryption often introduces randomness in the payload. Hence, techniques such as in Ma et al. [286] that search for keywords at fixed offsets will suffer in performance.

Techniques that rely on capturing the beginning of flows [176, 286] are infeasible for links with high data rates where sampling is often employed. Finamore et al. [146] overcome this limitation by extracting signatures

Table 8 Summary of NFV⋆ and SDN†-based traffic classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
He et al. [182]⋆	Supervised *k*-NN, Linear-SVM, Radial-SVM, DT, RF, Extended Tree, AdaBoost, Gradient-AdaBoost, NB, MLP	KDD [42]	Protocol, network service, source bytes, destination bytes, login status, error rate, connection counts, connection percentages (different services among the same host, different hosts among the same service)	Attack types from [450]	Dynamic selection of classifier and features to collect	Accuracy = 95.6%
Amaral et al. [19]†	Supervised RF, SGBoost, XGBoost	Proprietary: enterprise network	Packet size (1 to *N* packets), packet timestamp (1 to *N* packets), inter-arrival time (*N* packets), source/destination MAC, source/destination IP, source/destination port, flow duration, packet count byte count	BitTorrent, Dropbox, Facebook, Web Browsing (HTTP), LinkedIn, Skype, Vimeo, YouTube	$N = 5$	RF: Accuracy 73.6-96.0% SGBoost: Accuracy 71.2-93.6% XGBoost: Accuracy 73.6-95.2%
Wang et al. [462]†	Semi-supervised Laplacian-SVM	Proprietary: univ. network	Entropy of packet length, average packet length (source to destination and *vice versa*), source port, destination port, packets to respond from source to destination, minimum length of packets from destination to source, packet inactivity degree from source to destination, median of packet length from source to destination for the first *N* packets	Voice/video conference, streaming, bulk data transfer, interactive	$N = 20$, Laplacian-SVM parameters $\lambda = 0.00001 - 0.0001$, $\sigma = 0.21 - 0.23$	Accuracy > 90%

N/A: Not available

from any point in a flow. In light of the rise in streaming applications, they focus on analyzing packet payload to extract signature of applications over long-lived UDP traffic. In essence, to extract application signatures, the authors employ Pearsons's Chi-square (χ^2) test to capture the level of randomness in the first N bytes of each packet divided into G groups of b consecutive bits within a window of C packets. The randomness is evaluated based on the distance between the observed values and a reference uniform distribution. The signatures are then used to train a SVM classifier that distinguishes between the classes of interest with an average TP of 99.6%. However, FP are more sensitive to the window size, and reduce below 5% only for window sizes over 80.

Despite the disadvantages of payload-based classification techniques, the payload-based classifiers achieve high accuracy and are often employed to establish ground truth [55].

4.2 Host behavior-based traffic classification

This technique leverages the inherent behavioral characteristics of hosts on the network to predict the classes of interest. It overcomes the limitations of unregistered or misused port numbers and encrypted packet payload, by moving the observation point to the edge of the network and examining traffic between hosts (e.g. how many hosts are contacted, by which transport protocol, how many different ports are involved). These classifiers rely on the notion that applications generate different communication patterns. For example, a P2P host may contact several different peers using a different port number for each peer. While, a webserver may be contacted by different clients on the same port.

Schatzmann et al. [404] exploit correlations across protocols and time, to identify webmail traffic over HTTPS. They exploit the following features: (i) service proximity—webmail servers tend to reside in the same domain or subnet as SMTP, IMAP, and POP servers, that are identifiable using port numbers [243], (ii) activity profiles—irrespective of the protocol (i.e. IMAP, POP, webmail), users of a mail server share distinct daily and weekly usage habits, (iii) session duration—users of a webmail service spend more time on emails than other web pages and tend to keep the web client open for incoming messages, and (iv) periodicity—webmail traffic exhibit periodic patterns due to application timers (e.g. asynchronous checking for new message from AJAX-based clients). The authors show that these features act as good discriminators for a SVM classifier to differentiate between webmail and non-webmail traffic. Using 5-fold cross validation, the classifier achieves an average accuracy of 93.2% and a precision of 79.2%. The higher FN is attributed to the inability of the classifier to distinguish between VPN and webmail servers.

The data exchanged amongst P2P applications is highly discriminative [53]. For example, a P2P application may establish long flows to download video content from a few peers. Whereas, another P2P application may prefer to use short flows to download fixed size video chunks from many peers in parallel. Therefore, Bermolan et al. [53] leverage this behavior to derive P2P application signatures from the packets and bytes exchanged between peers in small time windows. Formally, the application signature is the probability mass function (PMF) of the number of peers that send a given number of packets and bytes to a peer in the time interval $\triangle T$.

These signatures are used to train a SVM classifier with a Gaussian kernel function and exponential binning strategy, with a rejection threshold (distance metric) of 0.5, to discriminate between applications belonging to the P2P-TV class (i.e. PPLive, TVAnts, SopCast, Joost). The authors evaluate the sensitivity of parameters to optimize their settings in order to guarantee the best performance, that is higher TPR and lower FPR. The classifier results in a worst-case TPR of about 95%, with FPR well below 0.1%. Also, temporal and spatial portability of signatures is validated with marginal degradation in performance.

However, the accuracy of the host behavior-based traffic classification strongly depends on the location of the monitoring system, especially since the observed communication pattern may be affected by routing asymmetries in the network core [229].

4.3 Flow Feature-based traffic classification

In contrast to payload-based and host behavior-based traffic classifiers, flow feature-based classifiers have a different perspective. They step back and consider a communication session, which consists of a pair of complete flows. A complete flow is a unidirectional exchange of consecutive packets on the network between a port at an IP address and another port at a different IP address using a particular application protocol [100]. It is identified with the quintuple $\langle srcIP, destIP, srcPort, destPort, protocol \rangle$. For example, a complete flow in an online game session would consist of all sequential packets sent from source s to destination d (e.g. host to game server). Therefore, a complete flow includes all packets pertaining to session setup, data exchange and session tear-down. A sub-flow is a subset of a complete flow and can be collected over a time window in an on-going session. A feature is an attribute representing unique characteristic of a flow, such as packet length, packet inter-arrival time, flow duration, and number of packets in a flow. Flow feature-based technique uses flow features as discriminators to map flows to classes of interest.

In essence, flow feature-based traffic classification exploits the diversity and distinguishable characteristics of the traffic footprint generated by different applications

[31, 467]. It has the potential to overcome numerous limitations of other techniques, such as unregistered port numbers, encrypted packet payload, routing asymmetries, high storage and computational overhead [55, 138, 176, 347]. However, it remains to be evaluated if flow feature-based classifiers can achieve the accuracy of payload-based classifiers [176, 286]. The corresponding traffic classification problem can be defined as follows: given a set of flows $X = \{x_1, x_2, x_3, \ldots, x_{|X|}\}$, such that X consists of either complete or sub-flows, and a set of classes of interest $Y = \{y_1, y_2, y_3, \ldots, y_{|Y|}\}$, find the mapping $g(X) \rightarrow Y$. This mapping can be used to classify previously unseen flows. ML is an ideal tool for finding this mapping automatically.

4.3.1 Supervised complete flow feature-based traffic classification

One of the earliest works in network traffic classification using ML is from Roughan et al. [390]. They employ k-NN and Linear Discriminant Analysis (LDA) to map network traffic into different classes of interest based on QoS requirements. Their traffic classification framework uses statistics that are insensitive to application protocol. The authors employ both packet-level and flow-level features. However, they observe that the average packet size and flow duration act as good discriminators, hence used these in their preliminary evaluation.

In their evaluation, k-NN outperforms LDA with the lowest error rate of 5.1 and 9.4% for four and seven class classification, respectively. They notice that often streaming applications behave very similar to bulk data transfer applications. Therefore, either a prioritization rule is necessary to break the tie, or extended/derivative features must be employed to act as good disciminators. In their extended evaluation, the authors employ inter-arrival variability to distinguish between streaming and bulk data transfer applications.

On the other hand, flow features were also leveraged in Moore and Zuev [321] that extend NB with Kernel Estimation (NBKE) to overcome the limitations that make it impractical for network traffic classification. Though, NB classifiers have been commonly used in classification, they have two fundamental assumptions, (i) probability of occurrence of each feature being independent from the occurrence of another feature, and (ii) probability distribution of a feature following a Gaussian distribution. Both of these assumptions are unrealistic for traffic classification and lead to poor accuracy. However, NBKE is a feasible alternate that generalizes NB and overcomes the Gaussian distribution approximation assumption.

Features are extracted from the header of packets in TCP flows using Fast Correlation-Based Filter (FCBF) to address the first assumption. In this way, NBKE with FCBF achieves a classification accuracy of upto 95% and TPR of upto 99%. It also achieves temporal stability by classifying new flows collected twelve months later with an accuracy of 93% and TPR of 98%. Moreover, it also outperforms NB with respect to training time. However, it incurs increased inference time, especially for classifying unknown flows [347].

Realizing the need for lightweight traffic classification, Jiang et al. [218] further reduce the complexity of KE by employing symmetric uncertainty and correlation measures for feature selection, derived from flows rather than packets. In this manner, NBKE can still be used to classify flows with an accuracy of 91.4%. Though, the classification accuracy is lower than the NBKE in [321], the techniques of varying sampling and application aware feature selection increases its applicability for online classification. Generally, when training time is important, NBKE classifiers are preferred over tree-based approaches, such as C4.5 DT and NB tree [347, 476]. However, DT performs better than NBKE, with respect to execution time and space in memory [347]. Unfortunately, DT suffers from overfitting with noisy data, which deteriorates performance [347].

It is not always possible to collect bidirectional flows due to routing asymmetries [138, 347]. However, it is possible to derive components of the feature vector for an application class given a priori knowledge [347], or estimate the missing statistics [138]. In addition, filtering can be leveraged to reduce the dimensionality of the feature space and the training time. Park et al. [347] employ supervised tree-based classifiers on unidirectional flows and compare them against NBKE using WEKA [288]. They exploit the faster classification time and low memory storage requirements of DT to employ Reduced Error Pruning Tree (REPTree) for classification. REPTree finds a sub-optimal tree that minimizes classification error. In addition, the Bagging ensemble is used to classify flows using majority rule to aggregate multiple REPTree predictions. Recall, that P2P bulk data transfer and streaming applications often behave similar to each other [390]. Therefore, the authors in [347] employ a burst feature to better discriminate between such classes. The burst feature is based on packet inter-arrival statistics and a predetermined threshold that dictates whether packets are exhibiting "bursty" behavior. Evidently, bulk data transfer applications exhibit higher burst than streaming applications.

Though, it was presumed that Bagging will outperform REPTree, both classifiers exhibited similar performance. REPTree achieves over 90% accuracy in classification of unidirectional flows and plateaus at seven features. This is in contrast to NBKE, where the classification accuracy deteriorates dramatically with increasing number of features. Though, the accuracy of REPTree is sensitive to packet sampling, the degradation is limited if the same sampling rate is used for both training and testing data.

Evidently, supervised learning yields high classification accuracy, due to a priori information about the characteristics of the classes of interest. However, it is infeasible to expect complete a priori information for applications, since often network operators do not even know all the applications that are running in the network. Therefore, Zhang et al. [496] present a traffic classification scheme suitable for a small set of supervised training dataset. The small labeled training set can be trivially hand-labeled based on the limited knowledge of the network operators. They use *discretized* statistical flow features and Bag-of-Flow (BoF)-based traffic classification. A BoF consists of discretized flows (i.e. correlated flows) that share the destination IP address, destination port and transport layer protocol.

Traditional NB classifiers are simple and effective in assigning the test data a posterior conditional probability of belonging to a class of interest. The BoF-based traffic classification leverages NB to generate predictions for each flow, and aggregate the predictions using rules, such as sum, max, median, majority, since flows are correlated. The F-measure, used to evaluate per class performance, of BoF-NB outperforms NB irrespective of the aggregation rule. For example, the F-measure of BoF-NB with the sum rule is over 15 and 10% higher than NB for DNS and POP3, respectively. Evidently, the accuracy increases and error sensitivity decreases as the size of BoFs increase, due to the growth in BoF intra-diversity [496].

Zhang et al. [497] propose a scheme, called Robust Traffic Classification (RTC). They combine supervised and unsupervised ML techniques for the classification of previously unknown zero-day application traffic, and improving the accuracy of known classes in the presence of zero-day application traffic. Their motivation is that unlabeled data contains zero-day traffic. The proposed RTC framework consists of three modules, namely unknown discovery, BoF-based traffic classification, and system update.

The unknown discovery module uses *k*-Means to identify zero-day traffic clusters, and RF to extract zero-day samples. The BoF module guarantees the purity of zero-day samples, which classifies correlated flows together, while the system update module complements knowledge by learning new classes in identified zero-day traffic. RTC is novel in its ability to reflect realistic scenarios using correlated flows and identify zero-day applications. Therefore, even with small labeled training datasets, RTC can achieve a higher flow and byte accuracy of up to 15% and 10%, respectively, in comparison to the second best technique.

The accuracy of traffic classification can be increased to over 99%, by using the discriminative MLP-NN classifier to assign membership probabilities to flows. Auld et al. [26] employ hyperbolic tangent for activation function and a softmax filter to ensure that activation to output generates a positive, normalized distribution over the classes of interest. Their MLP-NN with Bayesian trained weights (BNN) also increases the temporal accuracy of the classifier to 95%. The increase in accuracy is primarily achieved due to the ability to reject predictions. Though the NN with Bayesian weights attain very high performance, it comes at the cost of high compute and storage overhead. Furthermore, some employed features, such as effective bandwidth based on entropy and fourier transform of packet inter-arrival time are computationally intensive, inhibiting its use for online classification. The authors purport that their Bayesian trained weights are robust and efficient, and require only zero or one hidden layer.

On the other hand, Probabilistic Neural Network (PNN) uses Bayes inference theory for classification. Sun et al. [431] leverage PNN that requires no learning processes, no initial weights, no relationship between learning and recalling processes, and the difference between inference and target vectors are not used to modify weights. They employ an activation function that is typical in radial basis function networks and filter out mice flows. Elephant versus mice flows is a prevalent problem in traffic classification, since there is often a lack of representative data for the short-lived mice flows. Often, these flows are discarded for efficient classification. The authors detect mice flows as those flows that contain less than 10 packets and the duration is less than 0.01s. They show that PNN outperforms RBFNN, a feed forward neural network with only two layers and a typical non-linear RBF activation function.

In contrast, Este et al. [140] use single-class SVM followed by multi-class SVM for traffic classification. They consider “semantically valid” bidirectional TCP flows, while ignoring short flows. A grid is maintained to keep track of the percentage of vectors of training sets that are correctly and incorrectly classified as a class. To reduce the overhead in the grid search, they randomly select a small number of flows from the training set to satisfactorily train both single and multi-class classifiers to classify using the first few packets payload sizes. The Multi-class stage is only resorted to, if the single-class stage is unable to clearly identify the application classes. The authors apply their technique to different datasets, with TP of over 90% and low FP for most classes. However, the performance is compromised for encrypted traffic, where ground truth is established using unreliable port-based labeling.

A traffic classification problem with more than two classes, naïvely transforms the SVM into *N one-vs-rest* binary subproblems, resulting in a higher computation cost for a large number of classes. However, Jing et al. [223] propose a SVM based on tournaments for multi-class traffic classification. In the tournament design of SVM, in each round the candidate classes are randomly organized into pairs, where one class of each pair is selected by a binary SVM classifier, reducing the candidate classes by half. This limits the number of support vectors, which is now based only on the two classes in the pair in contrast to using the entire training dataset. This *all-vs-all* approach to multi-class classification in SVM results in a much lower computational cost for classification. The tournament in [223] results in only one candidate class being left as the classified class.

It is important to note that it is possible that the most appropriate class is eliminated, resulting in higher misclassification. To overcome this, a fuzzy policy is used in the tournament. It allows competing classes to proceed to the next round without being eliminated, if neither class has a clear advantage over the other. However, if two classes are continually paired against each other, the fuzzy rule will break the tie. Unfortunately, this special handling results in higher computational cost. The authors compare their proposed basic tournament and fuzzy tournament (FT-SVM) schemes with existing SVM [270] and [140]. The FT-SVM scheme achieves a high overall accuracy of up to 96%, reduces classification error ratio by up to 2.35 times, and reduces average computation cost by up to 7.65 times.

Traditional SVM and multi-class SVM fall short in efficiency for large datasets. Therefore, Wang et al. [464] use multi-class SVM along with an unbalanced binary SVM to perform statistics-based app-level classification for P2P traffic. Unlike the typical approach of decomposing a multi-class problem into multiple binary classification problems and using one-vs-all approach, the authors employ the *all-together* approach. They leverage NetFlow to collect TCP flows on the edge router of their campus network. In the classification process, unknown flows go through the unbalanced binary model first. Only if identified as P2P, they go through a weighted multi-class model. The unbalanced binary SVM model is built using non-P2P and N types of P2P traffic to help decrease FP (i.e. misclassification of non-P2P traffic as P2P). Whereas, the weighted multi-class model is trained using N types of P2P traffic, giving more weight to data traffic than control/signaling traffic. The proposed scheme correctly classifies atleast 75% and atmost 99% of the entire P2P traffic with generally low misclassification.

4.3.2 Unsupervised complete flow feature-based traffic classification

It is not always possible to apply supervised learning on network traffic, since information about all applications running in the network is rarely available. An alternate is unsupervised learning, where the training data is not labeled. Therefore, the classes of interest are unknown. In this case, ML techniques are leveraged to learn about similarities and patterns in data and generate clusters that can be used to identify classes of interest. Therefore, clustering essentially identifies patterns in data and groups them together. In hard clustering, an unknown data point must belong to a single cluster, whereas, in soft clustering, a data point can be mapped into multiple different clusters.

Hard clustering often relies on distance and similarity metrics to select a cluster that most closely resembles a data point. Liu et al. [283] employ hard clustering using k-Means with unsupervised training and achieve an accuracy of up to 90%. They use complete TCP flows and log transformation of features to extract and approximate features to a normal distribution, disposing of any noise and abnormality in data. However, it is unrealistic to apply hard clustering for membership, since flow features from applications, such as HTTP and FTP can exhibit high similarity [303]. Therefore, it is impractical to assume a cluster can accurately represent an application [492]. Hence, a fine-grained view of applications is often necessary by employing soft clustering and assigning an unknown data point to a set of clusters. EM is an iterative and probabilistic soft clustering technique, which guesses the expected cluster(s) and refines the guess using statistical characteristics, such as mean and variance.

McGregor et al. [303] employ EM to group flows with a certain probability and use cross-validation to find the optimal number of clusters. To refine the clusters, they generate classification rules from the clusters and use the rules to prune features that have insignificant impact on classification and repeat the clustering process. Though, promising preliminary results indicate stable clusters and an alternative to disaggregate flows based on traffic types, very limited results are provided.

Similarly, AutoClass is an EM approximation approach employed in Zander et al. [492] that automatically creates clusters from unlabeled training datasets. The boundaries of the classes are improved using an intra-class homogeneity metric, defined as the largest fraction of flows belonging to one application in the class. Their objective is to maximize the mean of intra-class homogeneities and achieve a good separation between different application data. On average, the intra-class homogeneity improves as number of features increase. It eventually plateaus at approximately 0.85-0.89, which implies that it is possible to achieve a good separation between classes without using the full set of features. However, this is still computationally expensive.

Erman et al. [136] uncover that this computational overhead can be reduced by training with half of the clusters, since majority of flows were grouped into these clusters. The trade-off between an intra-class homogeneity metric [492] versus iterative clustering remains to be investigated. The suitability of unsupervised learning in traffic classification reached an milestone when AutoClass was employed to achieve an accuracy of over 91%, higher than the 82.5% accuracy of the supervised NBKE [136]. Thus, it became possible to identify previously unknown applications. Unfortunately, the training time of AutoClass is magnitudes higher than the training time of NBKE [321].

In contrast to EM-based unsupervised clustering, density-based clustering has been found to have a significantly lower training time. Furthermore, density-based spatial clustering of applications with noise (DBSCAN) has the ability to classify noisy data in contrast to k-Means and AutoClass. It differs from conventional clustering, as it exploits the idea of a core object and objects connected to it. An object that does not belong in the neighborhood of a core object and is not a core object itself, is noise. Noisy objects are not assigned to any clusters. The neighborhood around an object is demarcated by epsilon (*eps*). An object is determined to be the core of the cluster when the number of objects in its neighborhood exceeds minimum number of points (*minPts*). In this manner, data points are evaluated to be core points, neighbors of core, or noise, and assigned to clusters or discarded accordingly.

Erman et al. [135] leverage DBSCAN with transport layer statistics to identify application types. Flows are collected from TCP-based applications, identifying flow start and end based on the TCP three way handshake and FIN/RST packets, respectively. They employ logarithms of features due to their heavy-tail distribution and deduce similarity based on Euclidean distance. The authors use WEKA [288] and Cluster 3.0 [194] for evaluating DBSCAN, AutoClass and k-Means clustering and found that AutoClass outperforms DBSCAN. However, training time of DBSCAN is magnitudes lower than AutoClass, 3 min *vs.* 4.5 h. Furthermore, its non-spherical clusters are more precise than the spherical clusters of k-Means. Uniquely, DBSCAN has the ability to classify data into the smallest number of clusters, while the accuracy of k-Means is dependent on the cluster size.

Erman et al. [138] extend their previous work [136] to employ unidirectional TCP flows to classify network traffic using k-Means. The motivation for unidirectional flows is justified, since it may not always be possible to collect bidirectional flows due to routing asymmetries [138, 347]. Therefore, the authors analyze the effectiveness of flows in one direction. To this end, they divide their dataset into three sets, consisting of server-to-client flows, client-to-server flows, and random flows that have a combination of both. The beginning and ending of the TCP flows are identified using SYN/SYNACK packets and FIN/RST packets, respectively. Whereas, a cluster is labeled to the traffic class with the majority of flows. As the number of clusters increase, it is possible to identify applications with a finer granularity.

It is observed that server-to-client flows consistently exhibit the highest average classification accuracy of 95 and 79% for flows and bytes, respectively. However, the random flows attain an average accuracy of 91 and 67% for flows and bytes, respectively. Whereas, the client-to-server flows show the average classification accuracy of 94 and 57% for flows and bytes, respectively. Also, the ML model using the server-to-client dataset has precision and recall values of 97% for Web traffic, while the P2P traffic had a precision of 82% and a recall of 77%. It is apparent that features from traffic in the server-to-client direction act as a good discriminators to classify unidirectional TCP flows. Furthermore, many network application's payload size is much higher in the server-to-client direction.

4.3.3 *Early and sub-flow-based traffic classification*

Relying on the completion of a flow for traffic classification not only surmounts to extensive classifier training time and memory overhead, but also delays time-sensitive classification decisions. Therefore, Bernaille et al. [55] leverage the size and direction of the first few P packets from an application's negotiation phase (i.e. during TCP connection setup) for early traffic classification. They inspect packet payload to establish flow labels and employ unsupervised k-Means clustering to model the application classes. The authors empirically deduce the optimal P and number of clusters (k) that strikes a balance between behavior separation and complexity. They represent a flow in a P-dimensional space, where the p^{th} coordinate is the size of the p^{th} packet in the flow and compute a behavioral similarity between flows using Euclidean distance between their spatial representations.

In the online classification phase, the distance between the spatial representation of a new flow and the centroid of all the clusters is computed. The flow is classified to the cluster with the minimum distance, hence to the dominant application class in the cluster. This approach achieves a classification accuracy exceeding 80% with $P = 5$ and $k = 50$ for most application classes. However, if different application classes exhibit similar behavior during the application's negotiation phase, their flows can be easily misclassified. For example, POP3 is always misclassified as NNTP and SMTP, the dominant application classes in corresponding clusters. However, this issue can be resolved by leveraging port number as a feature during cluster composition [56], which increases the classification accuracy of POP3 to over 90%.

Key advantages of the traffic classification approach in [55] is the ability to classify the same set of application classes from another network, since network packet sizes are similar across different traces. Furthermore, as their approach does not depend on packet payload, it is suitable for classifying encrypted traffic. Though, the packet size may increase due to encryption method used, it can be adjusted for based on a simple heuristic [56]. However, a fundamental requirement is to extract the first few packets during the negotiation phase of a TCP connection. Though simplistic, it is not always possible, especially in networks that use packet sampling. Furthermore, it is impractical for networks that have high load or miss packet statistics.

The classification accuracy of stand-alone classifiers that perform early classification, can be significantly improved by combining the outputs of multiple classifiers using combination algorithms [108, 121]. Donato et al. [121] propose Traffic Identification Engine (TIE), a platform that allows the development and evaluation of ML (and non-ML) classification techniques as plugins. Furthermore, TIE capitalizes on complementarity between classifiers to achieve higher accuracy in online classification. This is realized by using classifier output fusion algorithms, called combiners, including NB, majority voting (MV), weighted majority voting (WMV), Dempster-Shafer (D-S) [484], Behavior-Knowledge Space (BKS) method [199], and Wernecke (WER) method [473]. Note, BKS-based combiners overcome the independent classifier assumption of the other combiners [108]. However, due to the reliance of classifiers on the first few packets, it inherits the limitations of [56].

The authors [108, 121] evaluate the accuracy of the stand-alone classifiers and the combiners. They extract features for ML from flows in proprietary dataset, which is split into 20% classifier training set, 40% classifier and combiner validation set, and 40% classifier and combiner test set. The authors label the dataset using a ground truth classifier e.g. payload-based (PL) classifier. In stand-alone mode, the J48 classifier achieves the highest overall accuracy of 97.2%. Combining the output of J48 with other classifiers (i.e. Random Tree, RIPPER, PL) using BKS method, increases the overall accuracy to 98.4%, when considering first 10 packets per bidirectional flow. Most notably, an average gain in accuracy of 42% is achieved when extracting features from only the first packet, which is significant for online classification. However, higher accuracies are achieved when PL classifier is considered by the combiners in the pool of classifiers, thus increasing computational overhead.

The objective of Nguyen et al. [337] is to design a traffic classifier that can perform well, irrespective of missing statistics, using a small number of most recent packets, called sub-flows [336]. These sub-flows are derived by using a small sliding window over each flow, of N consecutive packets and a step fraction S, to start at packet 0 and slide across the training dataset in steps of N/S. Parameters N and S are critically chosen based on the memory limitations and the trade-off between classification time and accuracy.

To ensure high accuracy of the classifier it is imperative to identify and select sub-flows that best capture the distinctive statistical variations of the complete flows. To this end, the authors manually identify sub-flow starting positions based on protocol knowledge. They leverage Assistance of Clustering Technique (ACT) [338] to automate the selection of sub-flows using unsupervised EM clustering to establish ground truth. To account for directionality, Synthetic Sub-flow Pairs (SSP) are created for every sub-flow recorded with forward and backward features swapped, both labeled as the same application class [335].

Finally, the authors in [337] use NetMate [21] to compute feature values and employ supervised NB and C4.5 DT for traffic classification using WEKA [288]. Both classifiers perform well when evaluated with missing flow start, missing directionality, or 5% independent random packet loss. However, the C4.5 DT classifier performs better with 99.3% median recall and 97% median precision, and achieved 95.7% median recall and 99.2% median precision, for sub-flow size $N = 25$ packets for Enemy Territory and VoIP traffic, respectively. The authors also evaluate a real-world implementation of their approach, called DIFFUSE, for online traffic classification. DIFFUSE achieves a high accuracy of 98-99% for Enemy Territory (online game) and VoIP traffic replayed across the network, while monitoring one or more 1 Gb/s links. Despite the high accuracy, this technique lacks in flexibility, since the classifier can only recognize the application classes that were known a priori.

Erman et al. [137] propose a semi-supervised TCP traffic classification technique that partially overcomes a limitation of [55] and [337]—a priori knowledge of application class or protocol. They have the following objectives: (i) use a small number of labeled flows mixed with a large number of unlabeled flows, (ii) accommodate both known and unknown applications and allow iterative development of classifiers, and (iii) integrate with flow statistics collection and management solutions, such as Bro [350] and NetFlow [100], respectively.

To accomplish this, the authors employ backward greedy feature selection (BGFS) [173] and k-Means clustering to group similar training flows together, while using Euclidean distance as the similarity metric. Here, the objective is to use the patterns hidden in flows to assign them together in clusters, without pre-determined labels. Note, a small set of pre-determined labels are assigned to clusters using maximum likelihood, mapping clusters to application classes. While, other clusters remain unlabeled, accommodating for new or unknown applications. Thus, unknown flows are assigned to an unlabeled cluster. This gives network operators the flexibility to add new unlabeled flows and improve classifier performance by allowing identification of application classes that were previously unknown. The authors establish ground truth using payload-based signature matching with hand classification for validation.

In offline classification, the authors in [137] achieve over 94% flow accuracy with just two randomly labeled flows per cluster, amongst a mix of 64,000 unlabeled flows and $k = 400$. For real-time classification, the authors leverage a layered classification approach, where each layer represents a packet milestone, that is, the number of packets a flow has sent or received within a pre-defined sliding window. Each layer uses an independent model to classify ongoing flows based on statistics available at the given milestone.

Though, the model is trained with flows that have reached each specific packet milestone, previously assigned labels are disregarded upon reclassification. A significant increase in the average distance of new flows to their nearest cluster mean is indicative of the need for retraining, which could be achieved by incremental learning. This approach not only has a small memory footprint, it allows to update the model and potentially improve classification performance [137]. The authors integrate their layered online classification in Bro and achieve byte accuracies in the 70-90% range. Furthermore, the classifier remains fairly robust over time for different traces.

Similar to [137, 337], Li et al. [270] classify TCP traffic into application classes, including unknown application classes using a few packets in a flow. Their approach uniquely trains the ML model for the application class "Attack", that enables early detection and classification of anomalous traffic. They employ C4.5 DT to achieve high accuracy for online classification and reduce complexity in the number of features, by using correlation-based filtering. They perform their evaluations on WEKA [288], and find C4.5 DT to outperform C4.5 DT with AdaBoost and NBKE.

The classification accuracy of C4.5 DT with 0.5% randomly selected training flows, exceed 99% for most classes except Attack, which exhibits moderate-high recall. This is because Attack is a complex application class that shows no temporal stability and its characteristics dynamically change over time. However, it may be possible to overcome this by iterative retraining of the classifier, either by using approach similar to [137] or introducing rules (e.g. based on port numbers or flow metrics) in the DT to increase temporal stability of the classifier. Furthermore, the use of port numbers in conjunction with other features results in a slightly higher accuracy. However, this leaves the classifier vulnerable to issues in port-based classification.

In contrast to the semi-supervised and unsupervised techniques for TCP and UDP traffic classification, Jin et al. [222] employ a supervised approach. They classify network traffic using complete flows, while achieving high accuracy, temporal and spatial stability, and scalability. For accuracy and scalability, their system offers two levels of modularity, partitioning flows and classifying each partition. In the first level, domain knowledge is exploited to partition a flow into m non-overlapping partitions based on flow features, such as protocol or flow size. Second, each partition can be classified in parallel, leveraging a series of k-binary classifiers. Each binary classifier, i, assigns a likelihood score that is reflective of the probability of the flow belonging to the i^{th} traffic class. Eventually, the flow is assigned to the class with the highest score.

They design and leverage weighted threshold sampling and logistic calibration to overcome the imbalance of training and testing data across classes. Though, non-uniform weighted threshold sampling creates smaller balanced training sets, it can distort the distribution of the data. This may violate the independent and identically distributed assumption held by most ML algorithms, invalidating the results of the binary classifiers. Therefore, logistic calibrators are trained for each binary classifier and used at runtime to adjust the prediction of the binary classifiers.

The authors in [222] evaluate their system with respect to spatial and temporal stability, classification accuracy, and training and runtime scalability. With training and testing data collected two months apart from two different locations, result in low error rates of 3 and 0.4% for TCP and UDP traffic, respectively. However, with a larger time difference between training and testing data collection, the error rates increase to 5.5 and 1.2% for TCP and UDP traffic, respectively. By employing collective traffic statistics [221] via colored traffic activity graphs (TAGs) improves the accuracy for all traffic classes, reducing the overall error rate by 15%.

This diminishes the need for frequent retraining of the classifiers. Their system also provides flexible training configuration. That is, given a training time budget it can find the suitable amount of training data and iterations of the ML algorithm. It took the system about two hours to train the classifiers resulting in the reported error rates. Furthermore, their system on a multi-core machine using multi-threads, was able to handle 6.5 million new flows arriving per minute.

4.3.4 *Encrypted traffic classification*

Various applications employ encryption, obfuscation and compression techniques, that make it difficult to detect the corresponding traffic. Bonfiglio et al. [69] perform controlled experiments to reverse engineer the structure of Skype messages between two Skype clients (E2E) and between a Skype client and a traditional PSTN phone (E2O). The proposed framework uses three technique to identify Skype traffic, with a focus on voice calls, regardless of the transport layer protocol, TCP or UDP. The first technique uses a classifier based on the Pearson's χ^2 test that leverages the randomness in message content bits, introduced by the Skype encryption process, as a signature to identify Skype traffic. Whereas, the second technique is based on NB classifier that relies on the stochastic characteristics of traffic, such as message size and average inter-packet gap, to classify Skype traffic over IP. The third technique uses DPI to create a baseline payload-based classifier.

In the evaluation, the NB classifier is effective in identifying all voice traffic, while the χ^2 classifier accurately identifies all Skype traffic over UDP and all encrypted or compressed traffic over TCP. Jointly, NB and χ^2 classifier outperform the classifiers in isolation by detecting Skype voice traffic over UDP and TCP with nearly zero FP. However, higher FNs are noticeable in comparison to the isolated classifiers, as the combination disregards video and data transfers, and correctly identify only those Skype flows that actually carry voice traffic.

The identification of Skype traffic at the flow level is also addressed in Alshammari et al. [17] by employing supervised AdaBoost, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), SVM, NB and C4.5 DT classifiers. Additionally, these classifiers are used to identify Secure Shell (SSH) encrypted traffic. The authors use flow-based statistical features extracted using NetMate [21] and leverage WEKA [288] to train the classifiers using a sampled dataset for SSH, non-SSH, and Skype, non-Skype traffic. The trained models are applied to complete datasets to label flows as SSH, non-SSH, Skype and non-Skype.

In the evaluation, C4.5 DT outperform the other classifiers for the majority of datasets. For SSH traffic, it achieves 95.9% DR and 2.8% FPR on the Dalhousie dataset, 97.2% DR and 0.8% FPR on the AMP dataset, and 82.9% DR and 0.5% FPR on the MAWI dataset. Furthermore, when trained and tested across datasets (i.e. across networks), it achieves 83.7% DR and 1.5% FPR. Hence, it generalizes well from one network to another. The C4.5 DT classifier also performed well for Skype traffic with 98.4% DR and 7.8% FPR in the Dalhousie dataset. However, secure communication in SSH and HTTPS sessions can contain a variety of applications, identification of which may be needed for granularity. Unfortunately, Alshammari et al. [17] do not detect the precise applications within a secure session.

This problem is addressed by Shbair et al. [409] by adopting a hierarchical classification to identify the service provider (e.g. google.com, dropbox.com), followed by the type of service (e.g. maps.google.com, drive.google.com) that are encapsulated in TLS-based HTTPS sessions. They start with the reconstruction of TLS connections from the HTTPS traces and label them using the Server Name Identification (SNI) field, creating the service provider-service hierarchy. These labeled connections are used to build: (i) a classifier to differentiate between service providers, and (ii) a classifier for each service provider to differentiate between their corresponding services. This hierarchical approach reduces the effort required to retrain the classifiers in the event of an addition of a new service. They use statistical features extracted over encrypted payload with CFS and employ C4.5 DT and RF classifiers.

In the evaluation, RF performs better in comparison to C4.5 DT with a precision of 92.6%, recall of 92.8%, and F-measure of 92.6%, to classify service providers with selected features. Furthermore, the accuracy of service classification is between 95-100% for majority of the providers. Thus, asserting the benefit of a hierarchical approach to traffic classification. Also, overall accuracy of the system across both levels is 93.10% with a degradation of less than 20% over a period of 23 weeks without retraining.

4.3.5 NFV and SDN for traffic classification

Recent advances in network paradigms, such as Network Functions Virtualization (NFV) and SDN enable flexible and adaptive techniques for traffic classification. The efforts discussed in this subsection present contrasting approaches for traffic classification using ML in softwarized and virtualized networks.

It is well-known that the performance of classifiers vary significantly based on the type of flow features used. Furthermore, flows inherently exhibit specific characteristics of network applications and protocols. Therefore, finding the ideal set of features is fundamental to achieve efficiency in traffic classification. In a preliminary effort, He et al. [182] propose a NFV-based traffic-driven learning framework for traffic classification, called *ν*TC. *ν*TC consists of a controller, and a set of ML classifiers and feature collectors as virtual network functions (VNFs). Their objective is to dynamically select the most effective ML classifiers and the most cost-efficient flow features, by leveraging a controller and a group of VNFs, for traffic classification. The *ν*TC framework strives to achieve a balance between classification accuracy and speed, and the choice of features have a significant impact on these criteria. Therefore, it is critical to determine the most suitable classifier and dynamically adjust feature collection for a given flow protocol (e.g. TCP, UDP, ICMP).

The cost of extracting different features vary from one another. The same holds true for the execution of classifiers. Therefore, it is important to: (i) identify whether a feature should be collected on the data plane or the control plane, and (ii) have a centralized view of network resources while selecting the appropriate classifier. The controller in *ν*TC is responsible for maintaining the ML models from offline training, and selecting the most suitable classifier and flow features to collect by chaining the corresponding VNFs at runtime. It also monitors the load on the VNFs for scaling resources, if necessary. The adaptive selection of features and classifiers in *v*TC, based on the flow protocol, result in an accuracy of 95.6%. However, the performance overhead of adaptive selection of classifiers and features, and chaining of corresponding VNFs in *v*TC is not discussed. Furthermore, fine-grained classification and corresponding results are missing.

SDN offers built-in mechanisms for data collection via the OpenFlow (OF) protocol. Amaral et al. [19] harness SDN and OF to monitor and classify TCP enterprise network traffic. They leverage ML to extract knowledge from the collected data. In their architecture, a SDN application collects flow statistics from controlled switches and pro-actively installs a flow entry to direct all packets to the controller. For TCP traffic, the controller skips the TCP control packets, and stores the features of sizes, timestamps, MAC and IP addresses, and port numbers for the first five packets, along with their inter-arrival times. Then, the controller installs a flow entry with an idle timeout for local processing at the switch. Upon timeout, flow features of packet count, byte count and duration are collected at the controller.

The collected features are pruned using PCA and adjusted to eliminate high variability and scaling effects. However, the use of port numbers as a feature leaves the classifiers susceptible to issues in port-based classification. Nevertheless, the authors evaluate three ensemble ML classifiers, namely RF, Stochastic Gradient Boosting (SGBoost) and Extreme Gradient Boosting (XGBoost). The results exhibit high accuracy for some application classes (e.g. Web Browsing), while poor performance for others (e.g. LinkedIn). The authors do not provide justifications for the performance of the classifiers. However, this can be attributed to the fairly small training dataset used in their evaluation.

In contrast, Wang et al. [462] propose a framework to classify network traffic to QoS classes rather than applications. They assume that applications with similar QoS requirements exhibit similar statistical properties. This allows for equal treatment of different applications having similar QoS requirements. Their framework consists of two components: (i) traffic identification component that resides in switches at the network edge, to detect QoS-significant (i.e. elephant or long-lived) flows, and (ii) QoS aware traffic classification engine in the SDN controller that leverages DPI (for offline labeling) and semi-supervised ML to map long-lived flows to QoS classes. A significant number of flows remain unlabeled due to limited information on all possible/existing applications, thus calling for semi-supervised learning.

Similar to [137], periodic retraining of classifier is required to cater to new applications. The Laplacian-SVM classifier employed uses flow features from the first twenty packets to classify flows into QoS classes. Furthermore, they employ forward feature selection to reduce the number of features to nine from the initial sixty features. In the evaluation, the accuracy of classifying long-lived flows to QoS classes exceed 90%. However, the performance of the proposed framework is not evaluated, especially in light of the entropy-based features used for traffic classification.

4.4 Summary

Traditionally, Internet traffic has been classified using port numbers, payload and host-based techniques. Port-based techniques are unreliable and antiquated, largely due to the use of dynamic port negotiation, tunneling and misuse of port numbers assigned to well-known applications for obfuscating traffic and avoiding firewalls [54, 109, 176, 286]. In contrast, payload-based techniques are designed to inspect application payload. Though, they are computationally intensive and complicated due to encryption, supervised and unsupervised ML has been successfully applied for traffic classification with high accuracy. Generally, unencrypted handshake payload is used for traffic classification, which is infeasible for high data rate links. On the other hand, long-lived UDP traffic lends itself to supervised payload-based traffic classification, where payload is inspected randomly in an observation window [146]. However, it is not widely applicable and is highly sensitive to the observation window size. Similarly, host-based traffic classification is highly susceptible to routing asymmetries.

In contrast to these myopic approaches, flow feature-based traffic classification techniques inspect the complete communication session, which includes all consecutive, unidirectional packets in the network. This is the most widely studied technique for traffic classification that leverages both supervised and unsupervised ML. In supervised learning, various kernel estimation, NN and SVM-based ML techniques have been employed to achieve high accuracy. Though, traditional kernel estimation techniques are simple and effective, their underlying assumptions are unrealistic and infeasible for traffic classification. In this light, NBKE has been explored for traffic classification, but NN-based traffic classification has shown higher accuracy with probabilistic and, or Bayesian trained weights. Similarly, traditional and multi-class SVM have been applied jointly to increase the accuracy of traffic classification and its applicability to large datasets [464].

Rarely do network operators have complete information about all the applications in their network. Therefore, it is impractical to expect complete a priori knowledge about all applications for traffic classification. Therefore, unsupervised ML techniques have been explored for practical traffic classification using flow features. For traffic classification with unsupervised ML, both hard and soft clustering techniques have been investigated. Since flow features from applications can exhibit high similarity, it is unrealistic to apply hard clustering for fine-grained traffic classification. On the other hand, soft clustering achieves the required granularity with density-based clustering techniques, which also has a lower training time than EM-based soft clustering technique.

Complete flow feature-based traffic classification has been shown to achieve high spatial and temporal stability, high classification accuracy, and training and runtime scalability [222]. However, it requires extensive memory for storage and delays time sensitive classifier decisions. Flow feature-based traffic classification can be achieved using only a small of number of packets in a flow, rather than the complete flow. These sub-flows can be extended synthetically [55] or derived from a small sliding window over each flow [337]. Sub-flow-based traffic classification achieves high accuracy using fast and efficient C4.5 DT classifier with correlation-based filtering. Similar to payload-based traffic classification, encryption can also complicate flow feature-based traffic classification. However, it is possible to circumvent these challenges. For instance, a hierarchical method that identifies service provider followed by type of service, using statistical features from encrypted payload has been highly accurate and temporally stable [409].

Undoubtedly, supervised ML lends itself to accuracy in traffic classification, while unsupervised techniques are more robust. Consequentially, joint application of supervised and unsupervised ML for traffic classification [137, 497] has demonstrated success. Not only are semi-supervised classifiers resilient, they can be easily adapted for zero-day traffic or retrained for increased accuracy against previously unknown applications. Recent advances in networking increase the opportunities in traffic classification with SDN- and NFV-based identification of applications and classes of QoS. Though, some preliminary work in this area has achieved high accuracy, more scrutiny is required with respect to their resilience, temporal and spatial stability, and computational overhead. Most importantly, it is imperative to assess the feasibility of these technologies for time sensitive traffic classification decisions.

5 Traffic routing

Network traffic routing is fundamental in networking and entails selecting a path for packet transmission. Selection criteria are diverse and primarily depend on the operation policies and objectives, such as cost minimization, maximization of link utilization, and QoS provisioning. Traffic routing requires challenging abilities for the ML models, such as the ability to cope and scale with complex and dynamic network topologies, the ability to learn the correlation between the selected path and the perceived QoS, and the ability to predict the consequences of routing decisions. In the existing literature, one family of ML techniques has dominated research in traffic routing, Reinforcement Learning.

Recall, RL employs learning agents to explore, with no supervision, the surrounding environment, usually represented as a MDP with finite states, and learn from trial-and-error the optimal action policy that maximizes a cumulative reward. RL models are as such defined based on a set of states $\mathcal{S}$, a set of actions per state $\mathcal{A}(s_t)$, and the corresponding rewards (or costs) r_t. When $\mathcal{S}$ is associated with the network, a state s_t represents the status at time t of all nodes and links in the network. However, when it is associated with the packet being routed, s_t represents the status of the node holding the packet at time t. In this case, $\mathcal{A}(s_t)$ represents all the possible next-hop neighbor nodes, which may be selected to route the packet to a given destination node. To each link or forwarding action within a route may be associated an immediate static or dynamic reward (respectively cost) r_t according to a single or multiple reward (respectively cost) metrics, such as queuing delay, available bandwidth, congestion level, packet loss rate, energy consumption level, link reliability, retransmission count, etc.

At routing time, the cumulative reward, i.e. the total reward accumulated by the time the packet reaches its destination, is typically unknown. In Q-learning, a simple yet powerful model-free technique in RL, an estimate of the *remaining* cumulative reward, also known as *Q-value*, is associated with each state-action pair. A Q-learning agent learns the best action-selection policy by greedily selecting at each state the action a_t with highest expected Q-value $\max_{a \in \mathcal{A}(s_t)} Q(s_t, a)$. Once the action a_t is executed and the corresponding reward r_t is known, the node updates the Q-value $Q(s_t, a_t)$ accordingly as follows:

$$Q(s_t, a_t) \leftarrow (1-\alpha) Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a) \right)$$

α $(0 < \alpha \leq 1)$ and γ $(0 \leq \gamma \leq 1)$ denote the learning rate and discount factor respectively. The closer α is to 1, the higher is the impact of the most recently learned Q-value. While higher γ values make the learning agent aim for longer-term high rewards. Indeed, the greedy action-selection approach is only optimal if the learning agent knows the current Q-values of all possible actions. The agent can then *exploit* this knowledge to select the most rewarding action. If not, an ϵ-greedy approach may be used such that with probability ϵ the agent chooses to *explore* a random action rather than choosing deterministically the one with highest Q-value.

Though, RL is gaining a lot of attention these days, its application in network traffic routing dates back to the early 1990s. Boyan and Littman's [71, 280] seminal work introduced Q-routing, a straight-forward application of the Q-learning algorithm to packet routing. In Q-routing, a router x learns to map a routing policy, such as routing to destination d via neighbor y, to its Q-value. The Q-value is an estimate of the time it will take for the packet to reach d through y, including any time the packet would have to spend in node x's queue plus the transmission time over the link x, y. Upon reception of the packet, y sends back to x the new estimated remaining routing delay, and x adjusts accordingly its Q-value based on a learning rate. After convergence of the algorithm, optimal routing policies are learned.

Q-routing does not require any prior knowledge of the network topology or traffic patterns. However, experiments on a 36-node network demonstrated the Q-routing outperforms the shortest path first routing algorithm in terms of average packet delivery time. It was also found that, although Q-routing does no exploration or fine-tuning after policies and Q-values are learned, it still outperforms in a dynamically changing network topology, a full-echo Q-routing algorithm where the policy is dynamically adjusted to the current estimated time to destination. In fact, under heavy load, the full-echo Q-routing algorithm constantly changes the routing policy creating bottlenecks in the network. On the contrary, the original Q-routing shows better stability and robustness to topology changes under higher loads.

Since then the application of Q-learning to packet routing has attracted immense attention. A number of research efforts from late 1990s and early 2000s, built on and proposed improvements to Q-learning, resulting in three main research directions: *(a)* improving the performance of Q-routing to increase learning and convergence speed [96, 254], *(b)* leveraging the low complexity of Q-learning and devising Q-learning-inspired algorithms adapted to the specificities of the network (e.g. energy-constrained networks) and/or routing paradigm (e.g. multicast routing [430]), and *(c)* enforcing further collaboration between the routing learning agents to achieve complex global performance requirements [424, 479].

In 1996, a memory-based Q-learning algorithm called predictive Q-routing (PQ-routing) is proposed to keep past experiences to increase learning speed. PQ-routing keeps past best estimated delivery times to destination via each neighboring node y and reuses them in tandem with more current ones. In 1997, Kumar et al. apply dual reinforcement Q-routing (DRQ-Routing) to minimize packet delivery time [254]. DRQ-Routing integrates dual reinforcement learning [167] with Q-routing, so that nodes along the route between the source and the destination receive feedbacks in both directions (i.e. from both the up-stream and down-stream nodes). Both PQ-routing and DRQ-Routing are fully distributed as in Q-routing, and use only local information plus the feedbacks received from neighboring nodes. While PQ-routing shows better performance than Q-routing at lower-loads, DRQ-routing converges faster and the protocol shows better overall performance at the cost of slightly increased communication overhead due to backward rewards.

The problem of ML-based multicast routing was first addressed by Sun et al. [430] in the context of MANETs. Q-MAP, a Q-learning-based algorithm, was proposed to find and build the optimal multicast tree in MANETs. In Q-MAP, Q-values are associated with different upstream nodes and the best Q-values are disseminated directly from the sinks to the nodes thus making exploration of routes unnecessary, while speeding up the convergence of the learning process. Indeed an exploration-free approach eventually leads to maximum routing performance since only actions with maximum Q-values are selected, however it reduces the protocol to a static approach that is insensitive to topology changes.

The traditional single-agent RL model, which is greedy in nature, provides local optimizations regardless of the global performance. Therefore, it is not sufficient to achieve global optimizations such as network lifetime maximization or network-wide QoS provisioning. Multi-Agent Reinforcement Learning (MARL) entails that, in addition to learning information from the environment, each node exchanges local knowledge (i.e. state, Q-value, reward) and decisions (i.e. actions) with other nodes in the network in order to achieve global optimizations. This helps the nodes to consider not only their own performance, but also the one of their neighbors and eventually others, in selecting the routing policy. Generally, this comes at the price of increased complexity as the state is a joint state of all the learning agents, and the transitions are the result of the joint action of all the agents in the system. Q-routing and Q-routing-inspired approaches like PR-routing and DRQ-Routing do use a form of MARL, as Q-values are exchanged between neighboring nodes. This form of MARL is *soft* in that it is easy to implement and has low communication and computational complexity as opposed to the general more complex form of MARL like in [124, 425].

Team-partitioned opaque-transition reinforcement learning (TPOT-RL), proposed by Stone and Veloso for the RoboCup-1998 (Robot Soccer World Cup II) [425], is the first fully collaborative MARL technique to be applied to packet routing [424]. Routing was used by the authors as a proof-of-concept of the applicability of their algorithm to real world problems. However, in practice this algorithm has high computational complexity considering the very large number of states to be explored, and high communication overhead as every routed packet is acknowledged back by the sink along the path from the source for reward computation.

These early works paved the way to a decade of continued and prolific research in the area. While existing research studies preeminently consider routing as a decentralized operation function, and as such distribute the learning function across the routing nodes, works like [276, 461] take a centralized and a partially decentralized approach respectively. In the following we discuss representative works in the area and summarize them in Table 9.

5.1 Routing as a decentralized operation function

RL when applied in a fully distributed fashion, turns each routing node into a learning agent that makes local routing decisions from information learned from the environment. Routing nodes can take their decisions either independently or through collaboration in a multi-agent system fashion.

In [151], Forster et al. use a Q-learning approach in a multicast routing protocol, called FROMS (Feedback Routing for Optimizing Multiple Sinks). The goal of FROMS is to route data efficiently, in terms of hop count, from one source to many mobile sinks in a WSN by finding the optimal shared tree. Like in [430], a FROMS node is a learning agent that runs Q-learning to incrementally learn the real costs of different possible routes. Its state is updated with every data packet that needs to be routed, and the set of actions is defined by the possible next hop neighbors (n_i) and their route to the sinks $\left(hops^{n_i}_{D_p}\right)$. Rewards are received back from the upstream nodes and used to update the Q-values of the corresponding actions. However, unlike [430], next-hop neighbors are selected using a variant of the $\epsilon - greedy$ algorithm, such that the routing algorithm alternates between an exploration phase and a greedy exploitation phase. FROMS shows up to 5 times higher delivery rates than the popular directed diffusion algorithm [205] in the presence of node failure, and 20% less network overhead per packet due to route aggregation.

Arroyo-Valles et al. [24] propose Q-probabilistic routing (Q-PR), a localization-aware routing scheme for WSNs that applies Q-learning to achieve a trade-off between packet delivery rate, expected transmission count (ETX), and network lifetime. A node's decision as to drop a packet or forward it to one of the neighbors is a function of the energy cost at transmission and reception, packet priority, and the ETX to the sink through the neighbor. A node greedily chooses among its next-hop candidate neighbors the one that minimizes the cost of the route to the sink, which is estimated by the Q-value of the nodes. It updates its Q-value every time it relays a packet, and broadcast it so it is received by its neighbors. Experimental evaluations are carried out through simulations with over 50 different topologies of connected networks.

Q-PR is compared to the greedy perimeter stateless routing algorithm (GPSR) and the Expected progress-Face-Expected progress (EFE), both localization-aware routing algorithms. Results show that Q-PR as well as EFE outperform GPSR in terms of successful delivery rate (over 98% against 75.66%). Moreover, Q-PR shows lower number of retransmission retries and acknowledgements (on average over 50% and 40% less than GPSR and EFE respectively). Thus the Q-PR algorithm preserves better the lifetime of the WSN (3× and 4× more than GPSR and EFE respectively). However the algorithm requires that each node maintains locally a number of information regarding each of its neighbors. These include the distance between the nodes, the distance of the neighbor to the sink, the delivery probability between nodes, the estimated residual energy at the neighboor, and the 2-hop neighboors. This hampers the scalability of the approach.

Hu and Fei [197] propose QELAR, a model-based variant of the Q-routing algorithm, to provide faster convergence, route cost reduction, and energy preservation in underwater WSNs. In QELAR, rewards account for both the packet transmission energy (incurred for forwarding the packet to the neighbor node) and the neighbor node's residual energy. Taking into account the residual energy helps achieve a balanced energy distribution among nodes by avoiding highly utilized routes (hotspots). A model representation for each packet is adopted such that the state is defined as per which node holds the packet. Next-hop nodes are selected greedily based on their expected Q-values. The latter are maintained by the node along with corresponding transition probabilities learned at runtime. Each time a node forwards a packet, it appends its Q-value along with its energy level.

QELAR is evaluated and compared against the vector-based forwarding protocol (VBF) through simulations with 250 mobile sensor nodes uniformly deployed in a 3D space. Results show that QELAR is 25% more energy efficient than VBF. The lifetime of the network is 25% $\sim$ 30% higher with QELAR in the presence of failures and network partition compared with VBF with comparable transmission range, which makes QELAR more robust to faults. Whereas, both show comparable routing efficiency and delivery rates. On the other hand, further research could be pursued to study the convergence speed of the model-based learning algorithm of QELAR compared to the model-free Q-learning when appropriate learning rate and discount factor are used.

In [277] Lin and Schaar address the problem of routing delay-sensitive applications in the more general context of multi-hop wireless ad hoc networks. They rely on a n-step temporal difference (TD) [433] learning method, and aim at reducing the frequency of message exchange, and thus the communication overhead without jeopardizing the convergence speed. The routing protocol is evaluated

Table 9 Summary of RL-based decentralized, partially decentralized, and centralized routing models

Ref.	Technique	Application	Dataset	Features[a]	Action set	Evaluation	
	(selection)	*(network)*				Settings[a]	Improvement[b]
AdaR [461]	Partially decentralized LSPI (*ϵ-greedy)*	Unicast routing *(WSN)*	Simulations ·400 sensors ·20 data sources ·1 sink	State: $\mathcal{N}_i$ Reward: function of · node load · residual energy · hop cost to sink · link reliability	Next-hop nodes to destination	· $S = \#nodes$ · $A = \#neighbors$	Compared to Q-learning: · Faster convergence (by 40 episodes) · Less sensitive to initial parameters
FROMS [151]	Q-learning *(variant of ϵ-greedy)*	Multicast routing *(WSN)*	Omnet++ Mobility Framework with 50 random topologies ·50 nodes ·5 sources ·45 sinks	State: $(\mathcal{N}_j^k, D_k)$ Reward: function of hop cost	$\{a_1 \cdots a_m\}$ $a_k = (\mathcal{N}_j^k, D_k)$ N_j^k = next hop along the path to sink D_k	· $S = \#nodes$ · $A = \#neighbors$	Compared to directed diffusion: · up to 5× higher delivery rate · ≈ 20% lower overhead
Q-PR [24]	Variant of Q-learning *(ϵ-greedy)*	Localization-aware routing to achieve a trade-off between packet delivery rate, ETX, and network lifetime *(WSN)*	Simulations ·50 different topologies ·100 nodes	State: $\mathcal{N}_i$ Reward: function of · distance($\mathcal{N}_i$,$\mathcal{N}_j$) · distance($\mathcal{N}_j$,d) · energy at $\mathcal{N}_j$ · ETX · $\mathcal{N}_j$'s neighbors for any neighbor $\mathcal{N}_j$ and destination	Next-hop nodes to destination	· $S = \#nodes$ · $A = \#neighbors$	Delivery rate: · 25% more than GPSR Network lifetime · 3× more than GPSR · 4× more than EFE
Ref.	Technique	Application	Dataset	Features[a]	Action set	Evaluation	
	(selection)	*(network)*				Settings[a]	Improvement[b]
Xia et al. [482]	DRQ-learning *(greedy)*	Spectrum-aware routing *(CRN)*	OMNET++ simulations · stationary multi-hop CRN · 10 nodes · 2 PUs	State: $\mathcal{N}_i$ Reward: # available channels between current node and next-hop node	Next-hop nodes to destination	· $S = \#nodes$ · $A = \#neighbors$	Compared to Q-routing: ·50% faster at lower activity level Compared to Q-routing and SP-routing: · lower converged end-to-end delay
QELAR [197]	Model-based Q-learning *(greedy)*	Distributed energy-efficient routing *(underwater WSN)*	Simulations (ns-2) ·250 sensors in $500^3 m^3$ space ·100m transmission range · fixed source/sink ·1m/s maximum speed for intermediate nodes	State: $\mathcal{N}_i$ Reward: function of the residual energy of the node receiving the packet and the energy distribution among its neighbor nodes.	Next-hop nodes to destination ∪ packet withdrawal	· $S = \#nodes$ · $A = 1 + \#neighbors$	Compared to Q-learning: · Faster convergence (40 episodes less) · Less sensitive to initial parameters

Table 9 Summary of RL-based decentralized, partially decentralized, and centralized routing models *(Continued)*

Lin et al. [277]	$n-$step TD *(greedy)*	Delay-sensitive application routing *(multi-hop wireless ad hoc networks)*	Simulations 2 users transmitting video sequences to the same destination node ·3 ∼ 4-hops wireless network	State: current channel states and queue sizes at the nodes in each hop Reward: goodput at destination	Next-hop nodes to destination	$\cdot\, S = n_q^N \times n_c^H$ $\cdot\ A = (N_h^2)^{H-1} \times N_h$ N = *#nodes* N_h = *#nodes at hop h* H = *#hops* n_q = *#queue states* n_c = *#channel states*	Complexity≈ 2 × 10^8 for the 3−hop network With 95% less information exchanges · ∼ 10% higher PSNR · slightly slower convergence (+1 ∼ 2*sec*)
d-AdaptOR [59]	Q-learning with adaptive learning rate *($\epsilon-$greedy)*	Opportunistic routing *(multi-hop wireless ad hoc networks)*	Simulations on QualNet with 36 randomly placed wireless nodes in a 150*m* × 150*m*	State: $\mathcal{N}_i$ Reward: · fixed negative transmission cost is receiver is not the destination · fixed positive reward if receiver is the destination · 0 if packet is withdrawn	Next-hop nodes to destination ∪ packet withdrawal	$\cdot\, S =$ *#nodes* $\cdot\, A = 1 +$ *#neighbors*	After convergence (≈ 300*sec*) · ETX comparable to a topology-aware routing algorithm · > 30% improvement over greedy-SR, greedy ExOR and SRCR with a single flow · Improvement decreases with # flows
QAR [276]	Centralized SARSA *(ϵ-greedy)*	QoS-aware adaptive routing *(SDN)*	Sprint GIP network trace-driven simulations [418] · 25 switches, 53 links	State: $\mathcal{N}_i$ Reward: function of delay, loss, throughput	Next-hop nodes to destination	$\cdot\, S =$ *#nodes* $\cdot\, A =$ *#neighbors*	Compared to Q-learning with QoS-awareness: · Faster convergence time (20 episodes less)

[a] $\mathcal{N}_i$: node *i*; D_k: sink *k*; *S*: number of state variables; *A*: number of possible actions per state; #: number of

[b] Average values. Results vary according to experimental settings.

in a simulated multi-hop network with 2 sources transmitting videos to a same destination node. Results show that by reducing the frequency of message exchange by 95% (from every $1ms$ to every $20ms$), the goodput and effective data rate are increased by over 40%, and the video quality, calculated in terms of peak signal-to-noise ratio (PSNR), is increased by 10%. The convergence time seems to be only slightly affected ($1 \sim 2sec$). This is an interesting finding considering the bandwidth that can be saved and the interferences that can be avoided by spacing information exchanges.

Bhorkar et al. also address the problem of routing in multi-hop wireless ad hoc networks. They propose d-AdaptOR [59], a distributed adaptive opportunistic routing protocol which minimizes the average packet routing cost. d-AdaptOR is based on Q-learning with adaptive learning rate.

In opportunistic routing, instead of pre-selecting a specific relay node at each packet transmission as in traditional routing, a node broadcasts the data packet so that it is overheard by multiple neighbors. Neighbors who successfully acknowledge the packet form the set of candidate relays. The node will then choose among the candidate relays the one that will be forwarding the packet to destination. This property is an opportunity for the Q-learner to receive from the candidate relays their *up-to-date* Q-values. Traditionally, in Q-learning action selection is based on older, previously received Q-values.

Routing in d-AdaptOR consists of four main steps: (1) the sender transmits the data packet, (2) neighbors acknowledge the packet while sending its Q-value, the estimated cumulative cost-aware packet delivery reward, (3) the sender selects a routing action, either a next-hop relay or the termination of packet transmission, based on the outcome of the previous step using an ϵ-greedy selection rule (4) after the packet is transmitted, the sender updates its own Q-value at a learning rate that is specific to the selected next-hop relay. The learning rate is adjusted using a counter that keeps track of the number of packets received from that neighbor node. The higher the value of the counter, the higher is the convergence rate, though at the expense of Q-values fluctuations. Indeed the value of the counter depends also on the frequency of explorations. Further research could be pursued to investigate the optimal exploration-exploration strategy and the effects of different strategies on the convergence rate.

d-AdaptOR performance was investigated on the QualNet simulator using a random network benchmark consisting of 36 randomly placed wireless nodes. Simulations show that d-AdaptOR consistently outperforms existing adaptive routing algorithms, in terms of number of retransmissions per packet. Further study could be pursued to investigate the added value of node-specific learning rates in Q-value computation, compared to the traditional node-oblivious learning rate that is more efficient in terms of storage and computation.

Xia et al. [482] apply a spectrum-aware DRQ-routing approach in cognitive radio networks. In CRNs, the availability of a channel is dynamic, and is dependent on the activity level of the primary user (PU). The purpose of the routing scheme is to enable a node to select a next-hop neighbor node with higher estimate of total number of available channels up to destination. Indeed, higher number of available channels reduces channel contention, and hence reduces the MAC layer delay. However, relying on the total number of available channels along the path to destination can lead to very poor results in practice. The dual DRQ-routing approach was tested through simulations on a tailored stationary (non-mobile) multi-hop network topology with 10 cognitive radio nodes and 2 PUs operating on different channels. DRQ-routing was also compared against spectrum-aware Q-routing and spectrum-aware shortest path routing (SP-routing) at different activity levels. Simulation results show that after convergence, DRQ-routing minimizes end-to-end delay, is faster to converge than Q-routing (50% faster at lower activity level), and that it significantly reduces end-to-end delay compared to SP-routing at higher activity levels. However, although the nodes are not mobile and the topology is fixed, the convergence time at a 2 packet/s activity level is around 700sec which implies that 1400 periods have elapsed before DQR-routing has converged. As the activity level reaches 2.75packet/s, over 3000 periods are necessary for DRQ-routing to converge. These numbers are quite significant, but that is not surprising considering that a discount factor of 1 was used.

Elwhishi et al. [133] propose a Collaborative Reinforcement Learning (CRL) -based routing scheme for delay tolerant networks. CRL is an extension to RL introduced by Dowling et al. in 2004 for solving system-wide optimization problems in decentralized multi-agent systems with no global state [123], and was first applied to routing in MANETs by the same authors in [124]. Routing schemes for delay tolerant networks are characterized by the lack of end-to-end aspect, and each node explores network connectivity through finding a new link to a next-hop neighbor node when a new packet arrives, which must be kept in the buffer while a link is formed. SAMPLE, the proposed routing mechanism, selects a reliable next-hop neighbor node while taking into account three factors; two factors relevant to the channel availability (node mobility and congestion level), and a factor relevant to the buffer utilization (remaining space in the buffer). These are learned through feedback exchange among agents. Tested with different network topologies and mobility models, SAMPLE shows better performance than the traditional AODV and DSR routing algorithms in terms of packet delivery ratio and throughput.

5.2 Routing as a partially decentralized operation function

In [461] Wang et al. present AdaR, a routing mechanism for WSNs based on a centralized implementation of the model-free Least Squares Policy Iteration (LSPI) RL technique [258]. AdaR uses an offline learning procedure, and is claimed to converge to the fixed point routing policy faster than the traditional Q-learning. The algorithm takes into account the node's load, its residual energy, and hop count to the sink, as well as the reliability of the links. The algorithm runs in learning episodes. The base station is the learning agent, while the routing nodes are passive in terms of learning. However, actions are selected by the routing nodes in a decentralized fashion based on the Q-values assigned by the base station, and the ϵ-greedy selection algorithm. During each episode, the current Q-values are used to select a route to the base station. At each hop, the full hop information is appended to the packet and is used by the base station to calculate immediate rewards. When the base station has received enough information (the required number of packets is undefined), it calculates the new Q-values of the nodes offline, and disseminates them via a network-wide broadcast.

AdaR is tested on a simulated WSN with varying node residual energy and link reliability. Results show that the algorithm converges faster than Q-learning; a routing success rate of $\sim$ 95% with a low deviation was reached even before the 5th learning episode, whereas, it took 40 episodes for Q-learning to reach comparable success rates. This can be explained by Q-learning's initial Q-values and the selected learning rate ($\alpha = 0.5$). Appropriate initial Q-values and higher learning rate would have helped Q-learning converge faster. In fact, the authors show that Q-learning is more sensitive to the initial choice of Q-values than AdaR. Indeed AdaR has some useful properties, like taking into account different routing cost metrics and having faster convergence time. However, this comes at the price of higher computational complexity, and communication overhead due to the growing size of the packets at each hop and the broadcasting of Q-values, which also makes it more sensitive to link failures and node mobility.

5.3 Routing as a centralized control function

More recently, a centralized SARSA with a softmax policy selection algorithm has been applied by Lin et al. [276] to achieve QoS-aware adaptive routing (QAR) in SDN. Although a multi-layer hierarchical SDN control plane is considered by the authors, the proposed SARSA-based routing algorithm is not specific to such an architecture, and is meant to run on any controller that has global visibility of the different paths and links in the network.

For each new flow, the first packet is transmitted by the switch to the controller. The controller implicitly recognizes the QoS requirements of the flow, calculates the optimal route using the SARSA-based QAR algorithm, and accordingly updates the forwarding tables of the switches along the path. The QoS requirements consist in what metric to minimize/maximize (delay, loss, throughput, etc.). They are used to control the weight of each metric in the reward function.

It is suggested that the controller iterates the SARSA algorithm until convergence, which in practice results in delayed routing. The question is, how long is the delay and how suitable is the solution for real-time traffic. Also the impact of routing new flows on the QoS of other flows in the network is overlooked. If the flow is an elephant flow, it may congest the links and severely impact the QoS of flows with tight delay requirements.

5.4 Summary

The low computational and communication requirements of traditional RL algorithms, in particular Q-learning, and their ability to perform well at finding an optimal solution and adapting to changes in the environment, have motivated their—reportedly successful—application to traffic routing in a variety of network settings, as shown in Table 9.

Different approaches have been considered in applying RL to the traffic routing problem. These approaches vary in terms of: (i) level of distribution of the learning capability, and (ii) level of collaboration among multiple learners. Clearly, different approaches lend themselves more naturally to different network topologies and utility functions. For instance, in SDN [276] as well as WSN, the existence of a central node—the controller in SDN and the sink in WSN, respectively—allows for centralized learning. Whereas, routing in wireless ad hoc networks calls for decentralized RL [59, 277] where the learning capability is distributed among the routing nodes.

For the nodes to select the optimal routing policy, they need to evaluate different routing policies (actions) against a given utility function (reward). Rewards can be calculated in a central node, such as a sink or base station like in AdaR [461]. Alternatively, rewards are locally estimated by the nodes, which requires the nodes to exchange information. The nature and the amount of information, as well as the dissemination process, vary according to the utility function, as shown in Table 9. Indeed utility functions such as QoS provisioning, load balancing and network lifetime maximization, as in Q-PR [24], QELAR [197, 277], require more information to be disseminated at the cost of an increased complexity and communication overhead.

It is also important to notice that learners are very loosely coupled in most recently adopted decentralized RL approaches, where routers tend to select routing policies in an asynchronous, independent, very soft MARL fashion. Clearly, MARL aims at coordinating learning agents in order to achieve the optimal network-wide performance. This should further enhance the routing performance. However, several challenges arise from MARL. In fact, the difficulty of defining a good global learning goal, the overhead for an agent to coherently coordinate with other learning agents, and the longer convergence time can be prohibitive when applying MARL to realistic problem sizes. Indeed, there is a need for understanding the trade-off between benefits and overhead when applying MARL, particularly in resource-constrained and dynamic wireless networks where coordination has eventually a lot to offer.

6 Congestion control

Congestion control is fundamental to network operations and is responsible for throttling the number of packets entering the network. It ensures network stability, fairness in resource utilization, and acceptable packet loss ratio. Different network architectures deploy their own set of congestion control mechanisms. The most well-known congestion control mechanisms are those implemented in TCP, since TCP along with IP constitute the basis of the current Internet [13]. TCP congestion control mechanisms operate in the end-systems of the network to limit the packet sending rate when congestion is detected. Another well-known congestion control mechanism is queue management [72] that operates inside the intermediate nodes of the network (e.g. switches and routers) to complement TCP. There have been several improvements in congestion control mechanisms for the Internet and evolutionary network architectures, such as Delay-Tolerant Networks (DTN) and Named Data Networking (NDN). Despite these efforts, there are various shortcomings in areas such as packet loss classification, queue management, Congestion Window (CWND) update, and congestion inference.

This section describes several research works that demonstrate the potential of applying ML to enhance congestion control in different networks. Majority of the techniques have been applied to TCP/IP networks. It is important to note that the first ML-based approaches for congestion control were proposed in the context of asynchronous t ransfer mode (ATM) networks [175, 264, 284, 437]. However, we exclude these works from the survey because, to the best of our knowledge, this type of network has a low impact on present and future networking research interests [177].

6.1 Packet loss classification

In theory, TCP works well regardless of the underlying transmission medium, such as wired, wireless, and optical. In practice the standard TCP congestion control mechanism has been optimized for wired networks. However, the major problem in TCP is that it recognizes and handles all packet losses as network congestion, that is buffer overflow. Hence, performing unjustified congestion control when a loss is due to other reasons, such as packet reordering [150], fading and shadowing in wireless networks [130], and wavelength contention in optical networks [214]. As a consequence, TCP unnecessarily reduces its transmission rate at each detected packet loss, lowering the end-to-end throughput.

Therefore, the TCP throughput for wireless networks can be improved by accurately identifying the cause of packet loss [34, 62, 490] and reducing the TCP transmission rate only when congestion is detected. However, TCP congestion control has no mechanism for identifying the cause of packet loss. We term this problem as packet loss classification and various efforts have been made to propose solutions to this problem. In general, the solutions for packet loss classification fall in two broad categories, depending on where the solution is implemented in the network, that is, at intermediate nodes or in end-systems. The former requires additional implementation at the intermediate nodes that either hide the error losses from the sender [32, 33], or communicate to the sender extra statistics about the network state, such as congestion notification [483] and burst acknowledgment (ACK) [490]. It is important to mention that hiding error losses may violate TCP end-to-end principle as it may require splitting the TCP connection by sending an ACK to the sender before the packet arrives at the receiver [129].

In the latter approach, end-systems are complemented with solutions, such as TCP-Veno [156] and TCP-Westwood [463]. These leverage information available at end-systems, such as inter-arrival time (IAT), round-trip time (RTT), and one-way delay, to distinguish causes of packet loss and aid TCP congestion control mechanism. However, it has been shown that it is difficult to perform a good classification using simple tests, such as the ones implemented by TCP-Veno and TCP-Westwood, on these metrics, since they lack correlation to the cause for packet loss [60].

Therefore, various ML-based solutions have been proposed for packet loss classification in end-systems for different networks, such as hybrid wired-wireless [38, 129, 130, 163, 282], wired [150], and optical networks [214]. Generally, the classifier is trained offline, leveraging diverse supervised and unsupervised ML algorithms for binary classification. The majority of these techniques use the metrics readily available at end-systems, and evaluate their classifier on synthetic data on network simulators, such as ns-2 [203]. We delineate the proposed ML-based solutions for packet loss classification in Table 10 and discuss these techniques in this subsection.

Liu et al. [282] proposed, to the best of our knowledge, the first approach using ML for inferring the cause of packet loss in hybrid wired-wireless networks. Particularly, they distinguish between losses due to congestion and errors in wireless transmission. They employ EM to train a 4-state HMM based on *loss pair* RTT values, that is RTT measured before a packet loss. The Viterbi algorithm [455] is applied on the trained HMM to infer the cause of packet loss. The resultant ML-based packet loss classifier exhibits greater flexibility and superiority over TCP-Vegas [73]. Since, TCP-Vegas has been shown to outperform non-ML-based packet loss classifiers [60], the ML-based solution of [282] was fundamental in creating a niche and instigating the feasibility of ML-based solutions for packet loss classification problems. However, the authors assume that the RTT values never change during measurement. This is an unrealistic assumption since a modification in the return path changes the RTT values without affecting the cause of packet loss. Thus, affecting the correlation between RTT and cause of packet loss.

Barman and Matta [38] use EM on a 2-state HMM and consider discrete delay values to improve the accuracy of the above packet loss classifier, though at the expense of a higher computational cost. This work substitutes the Viterbi algorithm with a Bayesian binary test that provides comparable accuracy, while being computationally efficient. However, this ML-based packet loss classifier, unlike others, requires support from the network to obtain one of its input features, the estimated probability of wireless loss. Furthermore, [38, 282] evaluate their packet loss classifiers on simple linear topologies, which is far from realistic network topologies.

In contrast, El Khayat et al. [129, 130, 163] simulate more than one thousand random hybrid wired-wireless topologies for collecting a dataset of congestion and wireless error losses. The authors compute 40 input features from this dataset by using information that is only available at end-systems, including one-way delay and IAT of packets preceding and succeeding a packet loss. Several supervised ML algorithms are leveraged to build packet loss classifiers using these features. All the classifiers achieve a much higher classification accuracy than non-ML solutions, such as TCP-Veno and TCP-Westwood. In particular, Boosting DT with 25 trees provide the highest accuracy and the second fastest training time. It is important to realize that the training time of DT is the fastest, with a small reduction in accuracy of less than 4% compared to Boosting DT. Therefore, in case of computational constraints, DT achieves the best balance between accuracy and training time.

The authors continue on to improve TCP with the Boosting DT classifier, which exhibit throughput gains over the standard TCP-NewReno [185] and TCP-Veno. The results also show that the improved TCP can maintain a fair link share with legacy protocols (i.e. TCP-friendly). Their ML-based packet loss classifier is flexible and enables the selection between TCP throughput gain and fairness without retraining the classifier.

On the other hand, Fonseca and Crovella [150] focus on detecting the presence of packet loss by differentiating Duplicated ACKs (DUPACK) caused by congestion losses and reordering events. Similar to [282], they employ loss pair RTT as an input feature, however, to infer the network state and not the state of a single TCP connection. Thus, avoiding the poor correlation between RTT and the cause of packet loss. The authors construct a Bayesian packet loss classifier that achieves up to 90% detection probability with a false alarm of 20% on real wired network datasets from the Boston University (BU) and Passive Measure Analysis (PMA) [1]. The performance is superior for the BU dataset due to the poor quality of RTT measurements in the PMA dataset. In addition, the authors adapt an analytic Markov model to evaluate a TCP variant enhanced with the Bayesian packet loss classifier, resulting in a throughput improvement of up to 25% over the standard TCP-Reno.

In the context of optical networks, Jayaraj et al. [214] tackle the classification of congestion losses and contention losses in Optical Burst Switching (OBS) networks. The authors collect data by simulating the National Science Foundation Network (NSFNET) with OBS modules and derive a new feature from the observed losses, called the number of burst between failures (NBBF). They construct two ML-based packet loss classifiers by applying EM for both HMM and clustering. These classifiers integrate two TCP variants that keep a low control overhead for providing better performance (e.g. higher throughput and fewer timeouts) over the standard TCP-NewReno [185] and TCP-SACK [299], and Burst-TCP [490] for OBS networks. The TCP variant using EM for clustering perform slightly better than EM for HMM, as the former produce states (clusters) with a higher degree of similarity, while requiring a similar training time.

6.2 Queue management

Queue management is a mechanism in the intermediate nodes of the network that complements TCP congestion control mechanisms. Specifically, queue management is in charge of dropping packets when appropriate, to control the queue length in the intermediate nodes [72]. The conventional technique for queue management is Drop-tail, which adopts the First-In-First-Out (FIFO) scheme to handle packets that enter a queue. In Drop-tail, each queue establishes a maximum length for accepting

Table 10 Summary of packet loss classification using offline training at end-systems of the network

Ref.	ML Technique	Network	Dataset	Features	Classification	Evaluation	
						Settings	Results
Liu et al. [282]	Unsupervised: · EM for HMM	Hybrid wired and wireless	Synthetic data: · ns-2 simulation · 4-linear topology Data distribution: · Training = 10k	· Loss pair RTT	· Congestion loss · Wireless loss	· 4-state HMM · Gaussian variables · Viterbi inference	HMM accuracy[a]: · 44 − 98%
Barman and Matta [38]	Unsupervised: · EM for HMM	Hybrid wired and wireless	Synthetic data: · ns-2 simulation · Topology: - 4-linear - Dumbbell	· Loss pair delay · Loss probabilities: - Congestion - Wireless *(nw)* *nw: network support*	· Congestion loss · Wireless loss	· 2-state HMM · Gaussian variables · Bayesian inference · Discretized values: - 10 symbols	HMM accuracy[a]: · 92 − 98%
El Khayat et al. [129, 130, 163]	Supervised: · Boosting DT · DT · RF · Bagging DT · Extra-trees · MLP-NN · k-NN	Hybrid wired and wireless	Synthetic data: · Simulation in: - ns-2 - BRITE · > 1k random topologies Data distribution: · Training = 25k · Testing = 10k	40 features applying avg, stdev, min, and max on parameters: · One-way delay · IAT And on packets: · 3 following loss · 1 before loss · 1/2 before RTT [130] *finds that adding the number of losses is insignificant*	· Congestion loss · Wireless loss	Ensemble DT: · 25 trees NN: · 40 input neurons · 2 hidden layers with 30 neurons · 1 output neuron · LMA[b] learning k-NN: · $k = 7$	AUC (%)[c]: · 98.40 · 94.24 · 98.23 · 97.96 · 98.13 · 97.61 · 95.41
Fonseca and Crovella [150]	Supervised: · Bayesian	Wired	Real data: · PMA project · BU Web server	· Loss pair RTT	· Congestion loss · Reordering	· Gaussian variables · 0 to 3 historic samples	In PMA: · TPR = 80% · FPR = 40% In BU: · TPR = 90% · FPR = 20%
Jayaraj et al. [214]	Unsupervised: · EM for HMM · EM-clustering	Optical	Synthetic data: · ns-2 simulation · NSFNET topology Data distribution: · Training = 25k · Testing = 15k	· Number of bursts between failures	· Congestion loss · Contention loss	HMM: · 8 states · Gaussian variables · Viterbi inference · 26 EM iterations Clustering: · 8 clusters · 24 EM iterations	CV[c]: · 0.16 − 0.42 · 0.15 − 0.28 HMM accuracy[a]: · 86 − 96%

[a] Varies according to HMM prior estimates and network simulation settings (e.g. loss rate, error model, delay, traffic)

[b] Levenberg-Marquardt Algorithm (LMA)

[c] Respectively to the list of elements in the column ML technique

incoming packets. When the queue becomes full, the subsequent incoming packets are dropped until the queue becomes available again. However, the combination of Drop-Tail with the TCP congestion avoidance mechanism leads to TCP synchronization that may cause serious problems [68, 72]: (i) inefficient link utilization and excessive packet loss due to a simultaneous decrease in TCP rate, (ii) unacceptable queuing delay due to a continuous full queue state; and (iii) TCP unfairness due to a few connections that monopolize the queue space (i.e. *lock-out* phenomenon).

Active Queue Management (AQM) is a proactive approach that mitigates the limitations of Drop-tail by dropping packets (or marking them for drop) before a queue becomes full [72]. This allows end-systems to respond to congestion before the queue overflows and intermediate nodes to manage packet drops. Random Early Detection (RED) [148] is the earliest and most well known AQM scheme. RED continually adjusts a dropping (marking) probability according to a predicted congestion level. This congestion level is based on a pre-defined threshold and a computed average queue length. However, RED suffers from poor responsiveness, fails to stabilize the queue length to a target value, and its performance (w.r.t. link utilization and packet drop) greatly depends on its parameter tuning, which has not been successfully addressed [269]. Many AQM schemes have been proposed to improve these shortcomings [4]. However, they rely on fixed parameters that are insensitive to the time-varying and nonlinear network conditions.

For this reason, significant research has been conducted to apply ML for building an effective and reliable AQM scheme, which is capable of intelligently managing the queue length and tuning its parameters based on network and traffic conditions. The proposals presented in this survey conduct online training in the intermediate nodes of the network and evaluate their solutions by simulating diverse network topologies, mostly in ns2, using characteristics of wired networks. As highlighted in Table 11, these AQM schemes apply different supervised techniques for TSF [160, 179, 212, 498] and reinforcement-based methods for deducing the increment in the packet drop probability [298, 427, 428, 485, 499]. It is important to note that in this section we use the term *increment* to refer to a small positive or negative change in the value of the packet drop probability. The accuracy results depict the quality of the ML technique, for either correctly predicting future time series values or stabilizing the queue length. In addition, the computational complexity of these AQM schemes depend on the learning algorithm employed and the elements that constitute the ML component. For example, the NN structure and its complementing components. In the following, we discuss these ML-based AQM schemes.

PAQM [160], to the best of our knowledge, is the first approach using ML for improving AQM. Specifically, PAQM used OLS on time series of traffic samples (in bytes) for predicting future traffic volume. Based on such predictions, PAQM dynamically adjusted the packet dropping probability. The proposed OLS method relies on the normalized least mean square (NLMS) algorithm to calculate the linear minimum mean square error (LMMSE). Through simulations, the authors demonstrated that their linear predictor achieves a good accuracy, enabling PAQM to enhance the stability of the queue length when compared to RED-based schemes. Therefore, PAQM is capable of providing high link utilization while incurring low packet loss. Similarly, APACE [212] configure the packet dropping probability by using a similar NLMS-based OLS on time series of queue lengths to predict the current queue length. Simulations show that APACE is comparable to PAQM in terms of prediction accuracy and queue stability, while providing better link utilization with lower packet loss and delay under multiple bottleneck links. However, these NLMS-based predictors have a high computational overhead that is unjustified in comparison to a simpler predictor based on, for instance, a low pass filter.

To address these shortcomings, α_SNFAQM [498] was proposed to predict future traffic volume by applying the BP algorithm to train a neuro-fuzzy hybrid model using NN and fuzzy logic, called α_SNF. This α_SNF predictor uses time series of traffic samples and the predicted traffic volume as features. Then, α_SNFAQM leverage the predicted traffic volume and the instantaneous queue length to classify the network congestion as either severe or light. On this basis, α_SNFAQM decides to either drop all packets, drop packets with probability, or drop none. Simulations demonstrate that the α_SNF predictor slightly exceeds the accuracy of the NMLS-based predictor and incurs lower computational overhead. Furthermore, α_SNFAQM achieves smaller and more stable queue length than PAQM and APACE, while providing comparable goodput. However, α_SNFAQM produce more packet drops in order to notify the congestion earlier.

Similarly, to keep a low computational overhead, NN-RED [179] apply an SLP-NN on time series of queue length to predict a future queue length. The predicted queue length is compared to a threshold to decide if packet dropping is needed for preventing severe congestion. The SLP-NN is trained using the least mean square (LMS) algorithm (*a.k.a.*, delta-rule), which is marginally less complex than NLMS. Basic simulations exhibit that NN-RED outperforms RED and Drop-tail in terms of queuing delay, dropped packets, and queue stability. However, this work lacks comparison of NN-RED with similar approaches, such as PAQM, APACE, and α_SNFAQM, in terms of performance and computational overhead.

Table 11 Summary of AQM schemes with online training in the intermediate nodes of a wired network

Ref.	ML Technique	Multiple	Synthetic data from	Features	Output	Evaluation	
		Bottleneck[a]	ns-2 simulation		*(action-set for RL)*	Settings	Results
PAQM [160]	Supervised: · OLS	✓	Topology: · 6-linear · Arbitrary dumbbell Time = 50*s*	· Traffic volume (bytes)	TSF: · Traffic volume	· NMLS algorithm based on LMMSE	Accuracy: · 90 − 92.3%
APACE [212]	Supervised: · OLS	✓	Topology: · Dumbbell (1-sink) · 6-linear Time = 40*s*	· Queue length	TSF: · Queue length	· NMLS algorithm based on LMMSE	Accuracy: · 92%
α_SNFAQM [498]	Supervised: · MLP-NN	–	Topology: · Dumbbell (1-sink) Time = 300*s*	· Traffic volume · Predicted traffic volume	TSF: · Traffic volume	· 2 input neurons · 2 hidden layers with 3 neurons · 1 output neuron	Accuracy: · 90 − 93%
NN-RED [179]	Supervised: · SLP-NN	–	Topology: · Dumbbell Time = 900*s*	· Queue length	TSF: · Queue length	· 1+*N* input neurons (*N* past values) · 0 hidden layers · 1 output neuron · Delta-rule learning	*N/A*
DEEP BLUE [298]	Reinforcement: · Q-learning - ϵ-greedy	–	Topology: · Dumbbell Time = 50*s* *OPNET simulator instead of ns-2*	States: · Queue length · Packet drop prob. Reward: · Throughput · Queuing delay	Decision making: · Increment of the packet drop probability *(finite: 6 actions)*	· *N/A* states · 6 actions · ϵ-greedy ASS[b]	Optimal packet drop probability: · Outperforms BLUE [144]
Neuron PID [428]	Reinforcement: · PIDNN	✓	Topology: · Dumbbell Time = 100*s*	· Queue length error	Decision making: · Increment of the packet drop probability *(continuous)*	· 3 input neurons · 0 hidden layers · 1 output neuron · Hebbian learning · 1 PID component	QL_{Acc} error[c]: · 7.15 QL_{Jit}: · 20.18
AN-AQM [427]	Reinforcement: · PIDNN	✓	Topology: · Dumbbell · 6-linear Time = 100*s*	· Queue length error · Sending rate error	Decision making: · Increment of the packet drop probability *(continuous)*	· 6 input neurons · 0 hidden layers · 1 output neuron · Hebbian learning · 2 PID components	QL_{Acc} error[c]: · 6.44 QL_{Jit}: · 22.61
FAPIDNN [485]	Reinforcement: · PIDNN	✓	Topology: · Dumbbell Time = 60*s*	· Queue length error	Decision making: · Increment of the packet drop probability *(continuous)*	· 3 input neurons · 0 hidden layers · 1 output neuron · 1 PID component · 1 fuzzy component	QL_{Acc} error[c]: · 3.73 QL_{Jit}: · 31.8
NRL [499]	Reinforcement: · SLP-NN	✓	Topology: · Dumbbell Time = 100*s*	· Queue length error · Sending rate error	Decision making: · Increment of the packet drop probability *(continuous)*	· 2 input neurons · 0 hidden layers · 1 output neuron · RL learning	QL_{Acc} error[c]: · 38.73 QL_{Jit}: · 128.84

[a] Specifies if the approach was evaluated for multiple bottleneck links (✓) or simply for a single bottleneck link (–)
[b] Action Selection Strategy (ASS)
[c] Value computed using RMSE on the results presented in [269] for different network conditions

On the other hand, DEEP BLUE [298] focus on addressing the limitations of BLUE [144], an AQM scheme proposed for improving RED. BLUE suffers from inaccurate parameter setting and is highly dependent on its parameters. DEEP BLUE addresses these problems by introducing a fuzzy Q-learning (FQL) approach that learns to select the appropriate increment (*actions*) for achieving the optimal packet drop probability. The features for inferring the FQL *states* are the current queue length and packet drop probability. Whereas, the *reward* signal adopts a linear combination of the throughput and queuing delay. The authors use the OPNET simulator to show that DEEP BLUE improves BLUE in terms of queue stabilization and dropping policy. In addition, the authors mention that DEEP BLUE only generates a slight surplus of storage and computational overhead over BLUE, though no evaluation results are reported.

Other NN-based AQM schemes adopt an RL approach for deciding the proper increment of the packet drop probability. Neuron PID [428] uses a Proportional-Integral-Derivative (PID) controller that incorporates an SLP-NN to tune the controller parameters. Specifically, the SLP-NN receives three terms from the PID component and updates their weights by applying the associative Hebbian learning. The three terms of this PID-based SLP-NN (PIDNN) are computed from the queue length error, which is the difference between the target and current queue lengths. The latter represents the reward signal of the PID control loop. It is important to note that the PID component includes a transfer function that increases the computational overhead of the PIDNN, when compared to a simple SLP-NN.

AN-AQM [427] extends Neuron PID by including another PID component. Therefore, the SLP-NN of AN-AQM receives three terms more from the second PID component for updating their weights. The three terms of the second PID component are generated from the sending rate error, which is the mismatch between the bottleneck link capacity and the queue input rate. The latter serves as the reward signal for the PID control loop. This modification improves the performance of the PIDNN in more realistic scenarios. However, it incurs a higher computational overhead, due to an additional PID transfer function and the increase in the number of input neurons. Similarly, FAPIDNN [485] adopts a fuzzy controller to dynamically tune the learning rate of a PIDNN. As in Neuron PID, FAPIDNN includes only one PID component to calculate the three terms from the queue length error. However, the fuzzy controller of FAPIDNN also adds computational complexity. Alternatively, NRL [499] directly uses an SLP-NN—without a PID or fuzzy compo nent—that relies on a reward function to update the learning parameters. This reward function is computed from the queue length error and the sending rate error.

Li et al. [269] carry out extensive simulations on ns-2 to perform a comparative evaluation of the above NN-based AQM schemes (i.e. Neuron PID, AN-AQM, FAPIDNN, and NRL) and AQM schemes based on RED and Proportional-Integral (PI) controllers. For a single bottleneck link, the results demonstrate that the NN-based schemes outperform the RED/PI schemes in terms of queue length accuracy (QL_{Acc}) and queue length jitter (QL_{Jit}), under different network settings. Where, QL_{Acc} is the difference between the average queue length and a target value, while QL_{Jit} is the standard deviation of the average queue length. However, the NN-based schemes result in a higher packet drop than PI schemes. For multiple bottleneck links, one of the PI schemes (i.e. IAPI [429]) present better QL_{Acc} and packet drop, yet producing higher QL_{Jit}. When comparing only NN-based schemes, FAPIDNN provides the best QL_{Acc}, while Neuron PID has the least QL_{Jit} and packet drop. Nevertheless, AN-AQM is superior in these performance metrics for realistic scenarios involving UDP traffic noise.

6.3 Congestion window update

CWND is one of the TCP per-connection state variables that limits the amount of data a sender can transmit before receiving an ACK. The other state variable is the Receiver Window (RWND), which is a limit advertised by a receiver to a sender for communicating the amount of data it can receive. The TCP congestion control mechanisms use the minimum between these state variables to manage the amount of data injected into the network [13]. However, TCP was designed based on specific network conditions and assumes all losses as congestion (cf., Section 6.1). Therefore, TCP in wireless lossy links unnecessarily lowers its rate by reducing CWND at each packet loss, negatively affecting the end-to-end performance. Furthermore, the CWND update mechanism of TCP is not suitable for the diverse characteristics of different network technologies [30, 122]. For example, networks with a high Bandwidth-Delay Product (BDP), such as satellite networks, require a more aggressive CWND increase. Whereas, networks with a low BDP, such as Wireless Ad hoc Networks (WANET), call for a more conservative CWND increase.

The challenge of properly updating CWND in resource-constrained wireless networks, like WANET and IoT, is difficult. This is due to their limited bandwidth, processing, and battery power, and their dynamic network conditions [271, 380]. In fact, the deterministic nature of TCP is more prone to cause higher contention losses and CWND synchronization problems in WANET, due to node mobility that continuously modifies the wireless multi-hop paths [29, 379]. Several TCP variations, such as TCP-Vegas and TCP-Westwood, have been proposed to overcome these shortcomings. However, the fixed rule strategies used by such TCP variations are inadequate for adapting CWND to the rapidly changing wireless environment. For example, TCP-Vegas fails to fully utilize the available bandwidth in WANETs, as its RTT-based rate estimate is incorrect under unstable network conditions

[219]. Furthermore, methods for improving TCP-Vegas (e.g. Vegas-W [119]) are still insufficient to account for such variability, as their operation relies on the past network conditions rather than present or future.

As summarized in Table 12, this survey reviews several approaches based on RL that have been proposed to cope with the problems of properly updating CWND (or sending rate) according to the network conditions. Some of these approaches are particularly designed for resource-constrained networks, including WANETs [29, 219, 379, 380] and IoT [271], while others address a wider range of network architectures [30, 122, 477], such as satellite, cellular, and data center networks. Unless otherwise stated, the RL component conducts online training in the end-systems of the network to decide the increment for updating CWND. Although some approaches may apply the same RL technique, they differ in either the defined action-set (i.e. finite or continuous) or the utilization of a function approximation.

The evaluation of these RL-based approaches rely on synthetic data generated from multiple network topologies simulated in tools, such as GloMoSim, ns-2, and ns-3. A couple of these approaches [29, 122] also include experimental evaluation. The performance results show the improvement ratio of each RL-based approach against the best TCP implementation baseline. For example, if an approach is compared to TCP-Reno and TCP-NewReno, we present the improvement over the latter, as it is an enhancement over the former. It is important to note that an optimal CWND update reduces the number of packets lost and delay, and increases the throughput and fairness. Therefore, the selected improvement metrics allow to measure the quality of the RL component for deciding the best set of actions to update CWND.

To the best of our knowledge, TCP-FALA [380] is the first RL-based TCP variant that focuses on CWND adaptation in wireless networks, particularly in WANETs. TCP-FALA introduces a CWND update mechanism that applies FALA to learn the congestion state of the network. On receipt of a packet, it computes five states using IATs of ACKs, and distinguishes DUPACKs to compute the states in a different way. Each state corresponds to a single action that defines the increment for updating CWND. The probabilities for each possible action are continually updated, which are used by TCP-FALA for stochastically selecting the action to be executed. Such stochastic decision facilitates in adapting to changing network conditions and prevents CWND synchronization problem. Simulations in GloMoSim demonstrate that TCP-FALA experiences lower packet loss and higher throughput than standard TCP-Reno in different network conditions. However, the limited size of the action-set makes difficult mapping the range of responses provided by the network to the appropriate actions. In addition, WANETs require a much finer update of the CWND due to their constrained bandwidth.

To overcome this limitation, Learning-TCP [29, 379] extends TCP-FALA by employing CALA for enabling a finer and more flexible CWND update. Instead of separately calculating probabilities for each action, Learning-TCP continually updates an action probability distribution, which follows a normal distribution and requires less time to compute. Similar to TCP-FALA, Learning-TCP uses IATs of ACKs for computing the states, though without distinguishing DUPACKs and reducing the number of states to two. Several simulations in ns-2 and GloMoSim show that both Learning-TCP and TCP-FALA outperform standard TCP-NewReno in terms of packet loss, goodput, and fairness. Furthermore, the simulations demonstrate that Learning-TCP is superior to TCP-FALA and TCP-FeW [329] (a non-ML TCP variant enhanced for WANETs) with respect to these performance metrics. Whereas, TCP-FALA only achieves better fairness than TCP-FeW. The authors also provide experimental results that are consistent with the simulations.

TCP-GVegas [219] also focuses on updating CWND in WANETs. It improves TCP-Vegas by combining a grey model and a Q-learning model. The grey model predicts the real throughput of the next stage, while the Q-learning model adapts CWND to network changes. This Q-learning model uses the three stages of CWND changes (defined by TCP-Vegas) as the states and the throughput as the reward. The state is determined from CWND, RTT, and actual and predicted throughput. The action-set is continuous and limited by a range computed from RTT, throughput, and a pre-defined span factor. TCP-GVegas adopts an ϵ-greedy strategy for selecting the optimal action that maximizes the quality of the state-action pair. Simulations in ns-2 reveal that TCP-GVegas outperforms TCP-NewReno and TCP-Vegas in terms of throughput and delay for different wireless topologies and varying network conditions. However, TCP-GVegas has higher computational and storage overhead compared to standard TCP and TCP-Vegas. In fact, TCP-GVegas even has a higher computational and storage overhead than TCP-FALA and Learning-TCP, though a more thorough performance evaluation is required to determine the trade-off between these RL-based TCP variants.

In the similar context of resource constrained networks, TCPLearning [271] apply Q-learning for updating CWND in IoT networks. This Q-learning model computes the states by using a 10-interval discretization of each of the following four features: IAT of ACKs, IAT of packets sent, RTT, and Slow Start Threshold (SSThresh). TCPLearning defines a finite action-set that provides five increments for updating CWND and a selection strategy based on ϵ-greedy. The reward for each state-action pair is calculated from the throughput and RTT. To cope with the

Table 12 Summary of decision making of the increment for updating CWND by using online training at end-systems of the network

Ref.	RL	Network	Synthetic Dataset	Features	Action-set	Evaluation	
	Technique				*(action selection)*	Settings	Results[a]
TCP-FALA [380]	FALA	WANET	GloMoSim simulation: · Topology: - Random - Dumbbell	States and reward: · IAT of ACKs *(distinguish ACKS and DUPACKs)*	Finite: · 5 actions *(stochastic)*	· 1 input feature · 5 states · 5 actions	To TCP-NewReno[b]: · Packet loss = 66% · Goodput = 29% · Fairness = 20% To TCP-FeW[‡]: · Packet loss = −5% · Goodput = −10% · Fairness = 12%
Learning-TCP [29, 379]	CALA	WANET	Simulation: · ns2 and GloMoSim · Topology: - Chain - Random node - Grid Experimental: · Linux-based · Chain topology	States and reward: · IAT of ACKs	Continuous: · Normal action probability distribution *(stochastic)*	· 1 input feature · 2 states · ∞ actions	To TCP-FeW: · Packet loss = 37% · Goodput = 13% · Fairness = 23% To TCP-FALA: · Packet loss = 28% · Goodput = 36% · Fairness = 14%
TCP-GVegas [219]	Q-learning	WANET	ns-2 simulation: · Topology: - Chain - Random	States: · CWND · RTTz · Throughput Reward: · Throughput	Continuous: · Range based on RTT, throughput, and a span factor *(ϵ-greedy)*	· 3 input features · 3 states · *N/A* actions	To TCP-Vegas: · Throughput = 60% · Delay = 54%
FK-TCPLearning [271]	FKQL	IoT	ns-3 simulation: · Dumbbell topology: - Single source/sink - Double source/sink	States: · IAT of ACKs · IAT of packets sent · RTT · SSThresh Reward: · Throughput · RTT	Finite: · 5 actions *(ϵ-greedy)*	· 5 input features · 10k states · 5 actions · FK approx: - 100 prototypes	To TCP-NewReno: · Throughput = 34% · Delay = 12% To TCPLearning based on pure Q-learning: · Throughput= −1.5% · Delay = −10%
UL-TCP [30]	CALA	Wireless: · Single-hop: - Satellite - Cellular - WLAN · Multi-hop: - WANET	ns-2 simulation: · Single-hop dumbbell · Multi-hop topology: - Chain - Random - Grid	States and reward: · RTT · Throughput · RTO CWND	Continuous: · Normal action probability distribution *(stochastic)*	· 3 input features · 2 states · ∞ actions	For single-hop, to ATL: · Packet loss = 51% · Goodput: = −14% · Fairness = 53% For multi-hop, similar to Learning-TCP
Remy [477]	Own *(offline training)*	· Wired · Cellular	ns-2 simulation: · Wired topology: - Dumbbell - Datacenter · Cellular topology	States: · IAT of ACKs · IAT of packets sent · RTT Reward: · Throughput · Delay	Continuous with 3-dimensions: · CWND multiple · CWND increment · Time between successive sends *(ϵ-greedy)*	· 4 input features · $(16k)^3$ states · 100^3 actions · 16 network configurations	To TCP-Cubic: · Throughput = 21% · Delay = 60% To TCP-Cubic/SFQ-CD: · Throughput = 10% · Delay = 38%
PCC [122]	Own	· Wired · Satellite	Experimental: · GENI · Emulab · PlanetLab	States: · Sending rate Reward: · Throughput · Delay · Loss rate	Finite: · 2 actions of the increment for updating sending rate (not CWND) *(gradient ascent)*	· 3 input features · 4 states · 2 actions	To TCP-Cubic: · Throughput = 21% · Delay = 60%

[a] Average value of improvement ratio. Results vary according to the configured network parameters (e.g. topology, mobility, traffic)

[b] Based on the results from the simulated and experimental evaluations in [29]

memory restrictions of IoT devices, the authors use two function approximation methods: tile coding [435] and Fuzzy Kanerva (FK) [481]. The latter significantly reduces the memory requirements, hence, is incorporated in a modification of TCPLearning, called FK-TCPLearning. Specifically, FK-TCPLearning with a set of 100 prototypes, needs only 1.2% (2.4KB) of the memory used by TCPLearning based on pure Q-learning (200KB), for storing 50,000 state-action pairs. Furthermore, basic simulations in ns-3 reveal that FK-TCPLearning improves the throughput and delay of TCP-NewReno, while being marginally inferior to TCPLearning.

The approaches above are specifically designed for resource constrained networks, hence, restricting their applicability. For example, TCP-FALA and Learning-TCP estimate the congestion state from IATs of ACKs, which are prone to fluctuations in single-hop-wireless networks with high and moderate BDP, such as satellite networks, cellular networks, and WLAN. UL-TCP [30] address this gap by modifying Learning-TCP to compute the two congestion states from three different network features: RTT, throughput, and CWND at retransmission timeout (RTO). Simulations of single-hop-wireless networks in ns-2 show that UL-TCP achieves significantly better packet loss and fairness than TCP-NewReno and TCP-ATL [10] (a non-ML TCP variant designed for single-hop-wireless networks). However, UL-TCP is slightly inferior in terms of goodput than TCP-ATL. It is important to note that, unlike UL-TCP, TCP-ATL requires additional implementation in the intermediate nodes of the network. For multi-hop-wireless networks (i.e. WANETs), UL-TCP is compared to TCP-NewReno and TCP-FALA, and exhibit similar results to Learning-TCP. However, UL-TCP is slightly more complex than Learning-TCP due to the storage and usage of more parameters for computing the states.

Remy [477] and PCC [122] went further by introducing congestion control mechanisms that learn to operate in multiple network architectures. Remy designed an RL-based algorithm that is trained offline under many simulated network samples. It aims to find the best rule map (i.e. RemyCC) between the network state and the CWND updating actions to optimize a specific objective function. The simulated samples are constructed based on network assumptions (e.g. number of senders, link speeds, traffic model) given at design time—along with the objective function—as prior knowledge to Remy. The generated RemyCC is deployed in the target network without further learning to update CWND according to the current network state and the rule map. Several tests on simulated network topologies, such as cellular and data center networks, reveal that most of the generated RemyCCs provide a better balance between throughput and delay, in comparison to the standard TCP and its many enhanced variants, including TCP-Vegas, TCP-Cubic [174], and TCP-Cubic over Stochastic Fair Queuing [304] with Controlled-Delay AQM [340] (SFQ-CD). However, if the target network violates the prior assumptions or if the simulated samples incompletely consider the parameters of the target network, the performance of the trained RemyCC may degrade.

To tackle this uncertainty, PCC [122] avoids network assumptions and proposes an online RL-based algorithm that continually selects the increment for updating the sending rate, instead of CWND, based on a utility function. This utility function aggregates performance results (i.e. throughput, delay, and loss rate) observed for new sending rates during short periods of time. The authors emulate various network topologies, such as satellite and data center networks, on experimental testbeds for evaluating their proposal. The results demonstrate that PCC outperforms standard TCP and other variants specially designed for particular networks, such as TCP-Hybla [83] for satellite networks and TCP-SABUL [172] for inter-data center networks.

6.4 Congestion inference

Network protocols adapt their operation based on estimated network parameters that allow to infer the congestion state. For example, some multicast and multipath protocols rely on predictions of TCP throughput to adjust their behavior [238, 316], and the TCP protocol computes the retransmission timeout based on RTT estimations [22]. However, the conventional mechanisms for estimating these network parameters remain inaccurate, primarily because the relationships between the various parameters are not clearly understood. This is the case of analytic and history-based models for predicting the TCP throughput and the Exponential Weighted Moving Average (EWMA) algorithm used by TCP for estimating RTT.

For the aforementioned reasons, several ML-based approaches have addressed the limitations of inferring the congestion in various network architectures by estimating different network parameters: throughput [238, 316, 371], RTT [22, 128], and mobility [309] in TCP-based networks, table entries rate in NDNs [230], and congestion level in DTNs [412]. As depicted in Table 13, the majority of these proposals apply diverse supervised learning techniques, mostly for prediction. While, the one focused on DTN uses Q-learning for building a congestion control mechanism. The location of the final solution and the training type (i.e. online or offline) differ throughout the proposals, as well as the dataset and tools used for evaluating them. Similarly, the accuracy column shows a variety of metrics mainly due to the lack of consistency from the authors for evaluating the quality of their ML-based components for correctly predicting a specific parameter.

El Khayat et al. [238] apply multiple supervised learning techniques for predicting the TCP throughput in a wired network. From the different features used in the learning phase for building the ML models, the authors find that the Timeout Loss Rate (TLR) adds significant improvement in prediction accuracy. This is because TLR helps to discriminate two types of losses: triple duplicates and timeout. The ML models are trained and tested using synthetic data collected from ns-2 simulations. MLP-NN achieve the lowest accuracy error, followed by MART with

Table 13 Summary of congestion inference from the estimation of different network parameters

Ref.	ML Technique	Network	Dataset	Features	Output	Evaluation	
		(location)				Settings	Results[ab]
El Khayat et al. [238]	Supervised: · MLP-NN · MART · Bagging DT · Extra-trees *(offline)*	Wired *(end-system)*	Synthetic data: · ns-2 simulation · > 1k random topologies Data distribution: · Training = 18k · Testing = 7.6k	· Packet size · RTT: avg, min, max, stdev · Sesion loss rate · Initial timeout · Packets ACK at once · Session duration · TLR	Prediction: · Throughput	Ensemble DT: · 25 trees NN: *N/A*	MSE (10^{-3})[c]: · 0.245 · 0.423 · 0.501 · 0.525
Mirza et al. [316]	Supervised: · SVR *(offline)*	Multi-path wired *(end-system)*	Synthetic data: · Laboratory testbed - Dumbbell multi-path topology · RON testbed	· Queuing delay · Packet loss · Throughput	Prediction: · Throughput	· 2 input features · RBF kernel	Rate of predictions with RPE $\leq$ 10%: · Lab: 51% · RON: 87%
Quer et al. [371]	Supervised: · BN *(offline)*	WLAN *(access point)*	Synthetic data: · ns-3 simulation · Star topology Data distribution: · Training = 40k · Testing = 10k	· MAC-TX · MAC-RTX · MAC contention window · CWND · CWND status · RTT · Trhoughput	Prediction: · Throughput	DAG: · 7 vertices · 6 edges	Using MAC-TX: · NRMSE = 0.37 Using all features: · NRMSE = 0.27
Mezzavilla et al. [309]	Supervised: · BN *(offline)*	WANET *(end-system)*	Synthetic data: · ns-3 simulation · Topology: - *(not mentioned)*	· MAC-TX · MAC-RTX · Slots before TX · Queue TX packets · Missing entries in IP table	Classification: · Static · Mobile	DAG: · 6 vertices · 5 edges	Using MAC-TX and MAC-RTX: · Precision = 0.88 · Recall = 0.91
Fixed-Share Experts [22]	Supervised: · WMA *(online)*	· WANET · Wired · Hybrid wired and wireless *(end-system)*	Synthetic data: · QualNet simulation · Topology: - Random WANET - Dumbbell wired Real data: · File transfer · Wired and WLAN	· RTT	Prediction: · RTT	· 1 input feature · 100 experts · Simple experts	MAE (ticks): · Synthetic data (ticks of 500ms): = 0.53 · Real data (ticks of 4ms): = 2.95
SENSE [128]	Supervised: · WMA *(online)*	Hybrid wired and wireless *(end-system)*	Real data: · Dataset from [22]	· RTT	Prediction: · RTT	· 1 input feature · 100 experts · EWMA experts	MAE (ticks of 4ms): = 1.55
ACCPndn [230]	Supervised: · TLFN - PSO - GA *(online)*	NDN *(controller node)*	Synthetic data: · ns-2 simulation · Topology: - DFN - SWITCH Data distribution: · Training = 70% · Validation = 15% · Testing = 15%	· PIT entries rate	Prediction: · PIT entries rate	· R input neurons · 2 hidden layers with R neurons · R output neurons R: *number of contributing routers*	MSE: · PSO-GA = 2.23 · GA-PSO = 3.25 · PSO = 4.05 · GA = 5.65 · BP = 7.27
Smart-DTN-CC [412]	Reinforcement: · Q-learning - Boltzmann - WoLF *(online)*	DTN *(node)*	Synthetic data: · ONE simulation: · Random topology	States: · Input rate · Output rate · Buffer space Reward: · State transition	Decision-making: · Action to control the congestion *(finite action-set:* *12 actions)*	· 3 input features · 4 states · 12 actions	Improvement to CCC: · Delivery ratio = 53% · Delay = 95%

[a] Average values. Results vary according to the configured network parameters (e.g. topology, mobility, traffic)

[b] Error metrics: MAE, MSE, NRMSE, and Relative Prediction Error (RPE)

[c] Respectively to the list of elements in the column ML technique

25 trees. Both methods are recommended by the authors when learning from the full feature set. In addition, the authors demonstrate that all their ML-based predictors are more TCP-friendly than conventional throughput analytic models, like SQRT [300] and PFTK [343], which are generally used by multicast and real-time protocols.

Mirza et al. [316] also focus on throughput prediction, however, for multi-path wired networks, such as multi-homed and wide-area overlay networks. The authors train and test a supervised time series regression model using SVR on synthetic data collected from two distinct testbeds: authors laboratory deployment and the Resilient Overlay Networks (RON) project [332]. They also include a confidence interval estimator that triggers retraining if the predicted throughput falls outside the interval. The results reveal that the SVR model yields more predictions with a relative prediction error (RPE) of at least 10% than a simple history-based predictor. Moreover, the evaluation show that using active measurement tools for computing the model features, provide predictions as accurate as relying on ideal passive measurements. This is important because it is difficult to correctly collect passive measurements in real wide-area paths.

Another approach for predicting TCP throughput is proposed by Quer et al. [371]. Their ML solution resides in the access point of a WLAN, instead of the end-systems as in the above approaches. The authors apply BN for constructing a DAG that contains the probabilistic structure between the multiple features that allow predicting the throughput. A simplified probabilistic model is derived from the constructed DAG by using a subset of the features for inference. The training and testing of the BN model rely on synthetic data collected from ns-3 simulations. The results demonstrate that for a good amount of training samples (≥ 1000), this model provides a low prediction error. Furthermore, the authors exhibit that the prediction based only on the number of MAC Transmissions (MAC-TX) achieves a comparable error—and sometimes even lower—than using the full set of features.

A similar BN-based approach is proposed by Mezzavilla et al. [309], for classifying the mobility of the nodes in WANETs as either static or mobile. The DAG is built from a fewer number of features, therefore, reducing its number of vertices and edges. As in [371], the authors derive a simplified probabilistic model from the DAG by using two features for inference: MAC-TX and MAC Retransmissions (MAC-RTX). The results reveal that the simplified BN model achieves a good accuracy for classifying mobility in WANETs, when varying the radio propagation stability. This mobility classifier was used to implement a TCP variant that outperforms TCP-NewReno in terms of throughput and outage probability.

Fixed-Share Experts [22] and SENSE [128] concentrate on a different challenge, i.e. predicting RTT for estimating the congestion state at the end-systems of the network. Both are based on the WMA ensemble method and conduct online training for TFS. It is important to note that WMA uses the term *experts* to refer to algorithms or hypotheses that form the ensemble model. Particularly, SENSE extends Fixed-Share Experts by adding: (i) EWMA equations with different weights as experts, (ii) a meta-learning step for modifying experts penalty regarding recent past history, and (iii) a level-shift for adapting to sudden changes by restarting parameter learning. The two RTT predictors are trained and tested on real data collected from file transfers in a hybrid wired-wireless network. Only Fixed-Share Experts is evaluated on synthetic data collected from QualNet simulations. The results on real data show that SENSE achieves a lower prediction error—measured in ticks of $4ms$—than Fixed-Share Experts for predicting RTT. For synthetic data, Fixed-Share Experts provide a lower prediction error in comparison to real data even with a higher tick value of $500ms$. In terms of complexity, it is important to mention that SENSE requires more computational resources than Fixed-Share Experts, due to the adoption of EWMA as experts and the meta-learning step.

Finally, other works apply ML techniques to build novel congestion control mechanisms for non-TCP-based networks. ACCPndn [230] propose to include a TLFN into a controller node for predicting the rate of entries arriving to the Pending Interest Table (PIT) of NDN routers. The controller node gathers historical PIT entries rate from contributing routers and sends the prediction back to the corresponding router. The defined TLFN consists of two hidden layers between the input and output layers. The number of neurons for each layer corresponds to the number of contributing routers. To improve the parameter tuning of the TLFN trained using BP, the authors introduce a hybrid training algorithm that combines two optimization methods: PSO and GA. Various tests on synthetic data collected from ns-2 simulations demonstrate that the TLFN trained with PSO-GA provides a lower prediction error than the TLFN with other training algorithms, such as GA-PSO, GA or PSO only, and BP. Additionally, ACCPndn incorporate fuzzy decision-making in each router that uses the predicted PIT entries rate to proactively respond to network congestion. This congestion control mechanism considerably outperforms other NDN congestion control protocols, such as NACK [487] and HoBHIS [392], in terms of packet drop and link utilization.

Smart-DTN-CC [412] is another congestion control mechanism based on ML for DTN nodes. In particular, Smart-DTN-CC applies Q-learning for adjusting the congestion control behavior to the operating dynamics of the environment. Four congestion states are computed from information locally available at each node. The actions are selected from a finite set of 12 actions based on Boltzmann or Win-or-Learn Fast (WoLF) strategies. The reward of the state-action pairs depend on the transition between the states caused by a specific action. Simulations in the Opportunistic Network Environment (ONE) tool show that Smart-DTN-CC achieves higher delivery ratio and significantly lower delay than existing DTN congestion control mechanisms, such as CCC [265] and SR [406].

6.5 Summary

In the current Internet, TCP implements the most prevalent congestion control mechanism. TCP degrades the throughput of the network when packet losses are due to reasons other than congestion. Therefore, identifying the cause of packet loss can improve TCP throughput. Table 10 summarizes various solutions that have leveraged ML for classifying packet losses at the end-systems of different network technologies. In hybrid wired-wireless networks, the unsupervised EM for HMM and various supervised techniques were used to differentiate wireless losses (e.g. fading and shadowing) from congestion. In wired networks, a supervised Bayesian classifier was proposed to distinguish DUPACKs caused by reordering from the ones due to congestion. In optical networks, the unsupervised algorithm EM was employed on HMM training and clustering for classifying contention and congestion losses.

TCP variants built upon these ML-based classifiers outperform standard and diverse non-ML TCP versions (e.g. TCP-Veno and Burst-TCP). The majority of the ML-based classifiers were tested using synthetic data collected from simulations. EM-based classifiers simulate simpler topologies. Only the Bayesian classifier was evaluated on real data, though the small number of losses in the data negatively affects the results. In addition, all the classifiers perform binary classification of packet losses. Therefore, it would be interesting to explore an ML-based classifier that distinguishes between multiple causes of packet loss.

The other well-known congestion control mechanism is queue management. Several variations of AQM schemes (e.g. RED) have been proposed to overcome the TCP synchronization problem. However, these schemes suffer from poor responsiveness to time-varying and nonlinear network conditions. Therefore, different AQM schemes have integrated ML for better queue length stabilization and parameter tuning in changing network traffic conditions. As depicted in Table 11, half of the ML-based AQM schemes apply supervised OLS and NN for predicting future time series values of either traffic volume or queue length. The predicted values are used to dynamically adjust the packet drop probability. The other half of ML-based schemes employ reinforcement-based methods for deducing the increment in the packet drop probability.

All these ML-based AQM schemes improve and speed up the queue stabilization over non-ML AQM schemes for varying network conditions. However, the evaluation was based only on simulations of wired networks, though including single and multiple bottleneck topologies. Additionally, none of the ML-based AQM schemes have considered providing a fair link share among senders and have not been tested under coexisting legacy schemes in other bottleneck links in the network.

Another shortcoming in TCP is that its CWND update mechanism does not fit the distinct characteristics of different networks. For example, while satellite networks demand an aggressive CWND increase, WANETs perform better under a conservative approach. Table 12 outlines several solutions that have used RL techniques to appropriately update CWND according to the network conditions. Half of these ML-based approaches apply FALA, CALA, or Q-learning (including the FK function approximation) on resource-constrained networks (i.e. WANET and IoT). Whereas, the other half either use CALA or an own RL design on a wider range of network architectures, including satellite, cellular, and data center networks.

TCP variants built upon these ML-based CWND updating mechanisms perform better in terms of throughput and delay than standard and non-ML TCP versions particularly enhanced for specific network conditions (e.g. TCP-FeW and TCP-Cubic). Some of the ML-based TCP also show improvements in packet loss and fairness. The evaluation has been only based on synthetic data collected from simulations and experimental testbeds. In this case, it would be interesting to explore other ML techniques rather than RL for properly updating CWND.

TCP, as well as some multicast and multipath protocols, infer the congestion state from estimated network parameters (e.g. RTT and throughput) to adapt their behavior. However, such estimation remains imprecise mainly because of the difficulty in modeling the relationships between the various parameters. As summarized in Table 13, several solutions have leveraged ML for inferring congestion in different network architectures by estimating diverse network parameters. In the context of TCP-based networks, various supervised techniques were employed to predict the throughput in wired and WLAN networks. A supervised BN was also built to classify the node mobility in WANETs, while the ensemble WMA was used for predicting RTT in WANETs and hybrid wired-wireless networks. In the context of evolutionary network architectures, a supervised TLFN predicted the rate of entries arriving to the PIT of NDN routers, whereas Q-learning was employed in DTN nodes to select the proper action for the corresponding congestion state.

All these ML-based estimators outperform the accuracy of conventional estimation mechanisms (e.g. EWMA and history-based). The evaluation was mostly performed using synthetic data collected from simulations and laboratory testbeds. Only the WMA-based estimators collected real data to test their approaches.

As final remarks, note that the majority of the ML-based solutions rely on synthetic data to conduct the evaluation. However, synthetic data rely on simulations that may differ from real conditions. Therefore, there is a need to collect data from real networks to successfully apply and evaluate ML-based solutions. In some cases, such as in queue management, real collected data might not be enough to accomplish a realistic evaluation because the solutions impact the immediate network conditions. Therefore, recent networking technologies, like SDN and NFV, might support the evaluation in real networks. In addition, despite some ML-based solutions work on the same problem and report similar evaluation metrics, there is still the need of establishing a common set of metrics, data, and conditions that facilitate their comparison in terms of performance and complexity.

7 Resource management

Resource management in networking entails controlling the vital resources of the network, including CPU, memory, disk, switches, routers, bandwidth, AP, radio channels and its frequencies. These are leveraged collectively or independently to offer services. Naïvely, network service providers can provision a fixed amount of resources that satisfies an expected demand for a service. However, it is non-trivial to predict demand, while over and under estimation can lead to both poor utilization and loss in revenue. Therefore, a fundamental challenge in resource management is predicting demand and dynamically provisioning and reprovisioning resources, such that the network is resilient to variations in service demand. Despite the widespread application of ML for load prediction and resource management in cloud data centers [367], various challenges still prevail for different networks, including cellular networks, wireless networks and ad hoc networks. Though, there are various challenges in resource management, in this survey, we consider two broad categories, admission control and resource allocation.

Admission control is an indirect approach to resource management that does not need demand prediction. The objective in admission control is to optimize the utilization of resources by monitoring and managing the resources in the network. For example, new requests for compute and network resources are initiated for a VoIP call or connection setup. In this case, admission control dictates whether the new incoming request should be granted or rejected based on available network resources, QoS requirements of the new request and its consequence on the existing services utilizing the resources in the network. Evidently, accepting a new request generates revenue for the network service provider. However, it may degrade the QoS of existing services due to scarcity of resources and consequentially violate SLA, incurring penalties and loss in revenue. Therefore, there is an imminent trade-off between accepting new requests and maintaining or meeting QoS. Admission control addresses this challenge and aims to maximize the number of requests accepted and ser ved by the network without violating SLA.

In contrast, resource allocation is a decision problem that actively manages resources to maximize a long-term objective, such as revenue or resource utilization. The underlying challenge in resource allocation is to adapt resources for long-term benefits in the face of unpredictability. General model driven approaches for resource allocation have fallen short in keeping up with the velocity and volume of the resource requests in the network. However, resource allocation is exemplar for highlighting the advantages of ML, which can learn and manage resource provisioning in various ways.

7.1 Admission control

As shown in Table 14, Admission control has leveraged ML extensively in a variety of networks, including ATM networks [95, 189, 190], wireless networks [8, 36, 359], cellular networks [66, 67, 281, 372, 458], ad hoc networks [452], and next generation networks [311]. To the best of our knowledge, Hiramatsu [189] was the first to propose NN based solutions controlling the admission of a service requesting resources for a basic call setup in ATM networks. He demonstrated the feasibility of NN based approaches for accepting or rejecting requests, for resilience to dynamic changes in network traffic characteristics. However, there were unrealistic underlying assumptions. First, all calls had similar traffic characteristics, that is, single bit or multi bitrate. Second, cell loss rate was the sole QoS parameter.

Later, Hiramatsu [190] overcome these limitations, by integrating admission control for calls and link capacity control in ATM networks using distributed NNs. The NNs could now handle a number of bit-rate classes with unknown characteristics and adapt to the changes in traffic characteristics of each class. Cheng and Chang [95] use a congestion-status parameter, a cell-loss probability, and three traffic parameters, including peak bitrate, average bitrate, and mean peak-rate duration, to achieve a 20% improvement over Hiramatsu. To reduce the dimensionality of the feature space, they transform peak bitrate, average bitrate, and mean peak-rate duration of a call into a unified metric.

Piamrat et al. [359] propose an admission control mechanism for wireless networks based on subjective QoE perceived by end-users. This is in contrast to leveraging quantitative parameters, such as bandwidth, loss and

Table 14 Summary of ML-based Admission Control

Ref.	ML Technique	Network	Dataset	Features	Output	Evaluation	
						Settings	Results
Hiramatsu [189, 190]	Supervised: NN	ATM	Simulation	· Link capacity ·Observed call generation rate	· Call loss rate	2-10-1[a]	Improved call loss rate
Cheng and Chang [95]	Supervised: MLP-NN	ATM	Simulation	· Congestion-status · Cell-loss probability · Peak bitrate · Average bitrate · Mean peak-rate duration	Acceptance or rejection	30-30-1[a]	20% system utilization improvement over [189]
Piamrat et al. [359]	Supervised: · RandNN	Wireless	Videos (distorted) generated by streaming application	Codec, bandwidth, loss, delay, and jitter	· MOS	N/A	N/A
Baldo et al. [36]	Supervised: · MLP-NN	Wireless LAN	ns-3 simulator and testbed	Link load and frame loss	Service quality	9-10-1[a]	98.5% (offline) 92% (online)
Liu et al. [281]	Supervised: · MLP-NN	Cellular (CDMA)	Simulation of cellular networks	· Network environment · User behavior · Call class · Action	GoS	5-10-1[a]	Performs better than the static algorithms
Bojovic et al. [66]	Supervised: · MLP-NN	Cellular (LTE)	ns-3 network simulator	· Application throughput · Average packet error rate · Average size of packet data unit	QoS fulfillment ratio	N/A	Accuracy: 86%
Vassis et al. [452]	Supervised: · MLP · Probabilistic RBFNN · LVQ-NN · HNN ·SVM network	Ad hoc networks	Pamvotis WLAN simulator	· Network throughput · Packet generation rate	Average packet delays	N/A	Correctness: ·77% - 88% (Probabilistic RBFNN) Others do not converge
Ahn et al. [8]	Un-Supervised: · HNN	Wireless network	Simulation	· Usable QoS levels	QoS assignment matrix for each connection	N × M, where N and M are the number of connections and the number of QoS levels	Minimized connection blocking and dropping probabilities
Blenk et al. [63]	Supervised: · RNN	VN	Simulation	· Different graph features	Acceptance or rejection of VN	18 different Recurrent NNs	89% - 98%
Bojovic et al. [67]	Supervised: · NN ·BN	Cellular (LTE) network	ns-3 simulator	· Channel quality indicator	R-factor	Two layers with Number of nodes in the hidden layer: 10 and 20	Accuracy: 98% (BN)
Quer et al. [372]	Supervised: · BN	Wireless LAN	ns-3 simulator	· Link Layer conditions	Voice call quality	Nodes: 9, Links: 14	Accuracy: 95%
Mignanti et al. [311]	RL: · Q-learning	NGN	OMNET simulator	States · Environment state based on number of active connections of each traffic class	Action · Accept or reject *(ϵ-greedy)*	Not provided	10%-30% better than a greedy approach
Wang et al. [458]	RL: · Q-learning	LTE femtocell networks	Simulation	States · Queue length of handoff and new calls	Action · Maintain, degrade, or upgrade proportion levels	$RRl \times 3$, where l is QoS proportion levels	Reduction in blocking probability
Tong et al. [446]	RL: · Q-learning	Multimedia networks	Simulation	States · The number ongoing calls of each class · Call arrival or termination event · QoS and capacity constraints	Action · Accept or reject or no action	$K \times 2$, where K is number of constraints	Improvement in rejection rates
Marbach et al. [295]	RL: · TD(0)	Integrated service networks	Simulation	States · The number active calls of each class · Routing path of each active call	Action · Accept with a route or reject	States 1.4×10^{256}	2.2% improvement in rewards

latency. To do so, they first choose configuration parameters, such as codec, bandwidth, loss, delay, and jitter, along with their value ranges. Then, the authors synthetically distort a number of video samples by varying the chosen parameters. These distorted video samples are evaluated by human observers who provide a mean opinion score (MOS) for each sample. The configurations and corresponding MOSs are used to construct the training and testing datasets for a Random Neural Network (RandNN), which predicts MOSs in real-time without human interaction. Though, they evaluate their admission control mechanism for user satisfaction and throughput based metrics, no accuracy or error analysis is reported for the RandNN.

Baldo et al. [36] propose a ML-based solution using MLP-NN to address the problem of user driven admission control for VoIP communications in a WLAN. In their solution, a mobile device gathers measurements on the link congestion and the service quality of past voice calls. These measurements are used to train the MLP-NN to learn the relationship between the VoIP call quality and the underlying link layer, thus inferring whether an access point can satisfactorily sustain the new VoIP call. The authors report 98.5% and 92% accuracy for offline and online learning, respectively.

On the other hand, Liu et al. [281] propose a self-learning call admission control mechanism for Code Division Multiple Access (CDMA) cellular networks that have both voice and data services. Their admission control mechanism is built atop a novel learning control architecture (e.g., adaptive critic design) that has only one controller module, namely, a critic network. The critic network is trained with an 3:6:1 MLP-NN that uses inputs such as network environment (e.g. total interference received at the base station), user behavior (e.g. call type—new or hand off call), call class (e.g. voice, data), and the action to accept or reject calls. The output is the Grade of Service (GoS) measure. The MLP-NN is retrained to adapt to changes in the admission control requirements, user behaviors and usage patterns, and the underlying network itself. Through simulation of cellular networks with two classes of services, the authors demonstrate that their admission control mechanism outperforms non-adaptive admission control mechanisms, with respect to GoS, in CDMA cellular networks.

In contrast, Bojovic et al. [66] design an ML-based radio admission control mechanism to guarantee QoS for various services, such as voice, data, video and FTP, while maximizing radio resource utilization in long term evolution (LTE) networks. In their mechanism, the MLP-NN is trained using features, such as application throughput, average packet error rate, and average size of payload. The MLP-NN is then used to predict how the admission of a new session would affect the QoS of all sessions to come. Using a LTE simulator, it is shown that MLP-NN can achieve up to 86% accurate decisions provided it has been trained over a relatively long period of time. Despite its high accuracy, a critical disadvantage of MLP-NN is over-fitting, thus it fails to generalize in the face of partial new inputs.

Vassis et al. [452] propose an adaptive and distributed admission control mechanism for variable bitrate video sessions, over ad hoc networks with heterogeneous video and HTTP traffic. Unlike previous admission control approaches that only consider the new request, this mechanism takes into account the QoS constraints of all the services in the network. The authors evaluate five different NNs, namely MLP, probabilistic RBFNN, learning vector quantization network (LVQ) –a precursor to SOM–, HNN, and SVM network. Using network throughput and packet generation rates of all nodes prior to starting each session and the average packet delays of those sessions as the training and validation data, respectively, they found probabilistic RBFNN to always converge with a success rate between 77 and 88%.

Similarly, Ahn et al. [8] propose a dynamic admission control algorithm for multimedia wireless networks based on unsupervised HNN. In this mechanism, new or hand-off connections requesting admission are only granted admission if the bandwidth of the corresponding cell is sufficient to meet the bandwidth required for their best QoS level. Otherwise, the QoS levels of the existing connections are degraded to free up some bandwidth for the new or hand-off connection requesting admission. The compromised QoS levels of existing connections and the QoS levels of the new or hand-off connections are then computed using a hardware-based HNN that permits real-time admission control. Most importantly, the HNN does not require any training, and can easily adapt to dynamic network conditions. This admission control mechanism achieves significant gains in ATM networks, in terms of minimizing the blocking and dropping probabilities and maximizing fairness in resource allocation.

Recently, Blenk et al. [63] employ RNN for admission control for the online virtual network embedding (VNE) problem. Before running a VNE algorithm to embed a virtual network request (VNR), the RNN predicts the probability whether VNR will be accepted by the VNE algorithm based on the current state of the substrate and the request. This allows the VNE algorithm to process only those requests that are accepted by the RNN, thus reducing the overall runtime and improving the system performance. The RNN is trained with new representations of substrate networks and VNRs that are based on topological and network resource features. To obtain a compact representation, the authors apply PCA on a set of feature vectors and select features that are sensitive to high load, including number of nodes, spectral radius, maximum effective eccentricity, average neighbor degree, number of eigenvalues, average path length, and number of edges. A total of 18 different RNNs are trained offline using a supervised learning algorithm and a dataset generated through simulation of two VNE algorithms, namely Shortest Distance Path and Load Balanced. These RNNs achieve accuracies between 89% and 98%, demonstrating that this admission control mechanism can learn from the historical performances of VNE algorithms.

However, a potential disadvantage of NN based systems is that the confidence of the predicted output is unknown. As a remedy, a BN can predict the probability distribution of certain network variables for better performance in admission control [67, 372]. Specifically, Bojovic et al. [67] compare NN and BN models by applying them for admission control of calls in LTE networks. Both models are trained to learn the network behavior from the observation of the selected features. Upon arrival of an incoming VoIP call and assuming that the call is accepted, these two models are used to estimate the R-factor [206] QoS metric. A major difference between NN and BN is that NN can directly predict the value of the R-factor, while BN provides a distribution over its possible values. In NN, if the estimated R-factor is greater or smaller than a QoS threshold, the call is accepted or rejected, respectively. In contrast, the BN model accepts a call if the probability of the R-factor exceeding a threshold is greater than a probability threshold, or drops it otherwise. This gives the admission control mechanism additional flexibility to choose the probability threshold that allows to meet different system requirements by opportunistically tuning these thresholds. Through a simulation of macro cell LTE admission control scenario in ns-3, the BN model shows less FPs and FNs compared to NN.

Similarly, Quer et al. [372] develop an admission control mechanism for VoIP calls in a WLAN. They employ BN to predict the voice call quality as a function of link layer conditions in the network, including the fraction of channel time occupied by voice and background best effort traffic, estimated frame error probabilities of voice and background traffic, and R-factor representing the posteriori performance. The BN model is built upon four phases, (i) a structure learning phase to find qualitative relationships among the variables, (ii) a parameter learning phase to find quantitative relationships, (iii) the design of an inference engine to estimate the most probable value of the variable of interest, and (iv) an accuracy verification to obtain the desired level of accuracy in the estimation of the parameter of interest. The authors evaluate the BN model via ns-3 based simulation of a WLAN, having both VoIP and TCP traffic, and show an accuracy of 95%.

Besides NN and BN, the admission control problem has also been formulated as an MDP [311, 458]. Traditionally, dynamic programming (DP) is used to solve a MDP. However, DP suffers from two limitations in the context of admission control. First, it expects the number of states in the MDP to be in polynomial order, which is seldom the case in real networks. Second, DP requires explicit state transition probabilities, which are non-trivial to determine a priori. Therefore, RL, that can handle MDP problems with very large state spaces and unknown state transition probabilities, has been successfully applied to solve MDP-based admission control problems in networking.

Mignanti et al. [311] employ Q-learning to address admission control for connections in next generation networks. In their approach, when a connection request arrives, the Q-values of accepting and rejecting the request are computed. The request is accepted or rejected depending on whether the Q-value for acceptance or rejection is higher. Similarly, Q-learning has been used to allocate guard channels as part of the admission control mechanism for new calls in the LTE femtocell networks [458]. It is important to realize that allocating a guard channel for a new or hand-off call can raise the blocking probability. Therefore, Q-learning has to find the optimal policy that minimizes the cumulative blocking probability.

RL has also been leveraged for more complex problems that pertain to admission control with routing [295, 446]. In such problems, when a request is admitted, a route has to be established such that each link in the route meets the QoS requirements of the request. Therefore, RL-based solutions discussed earlier for admission control, with only two possible actions, are infeasible for admission control with routing. Here, the action space consists of selecting a route from a predefined set of routes in the network. Tong et al. [446] formulate this problem as a semi-MDP, and leverage Q-learning to define policies for route selection, such that the revenue is maximized and QoS requirements of the requests are met. In the formulation, they consider two important classes of QoS constraints, (i) state dependent constraint (e.g. capacity constraint) that is a function of only the current state, and (ii) past dependent constraint (e.g. fairness constraint) that depends on statistics over the past history. Since a detailed specification of a network state is computationally intractable [446], they exploit statistical independence of the links in the network for developing a decentralized RL training and decision making algorithm. In this approach, each link in the network performs Q-learning locally using only the link state information, instead of network state information. The authors evaluate their approach to admission control with routing via simulation of a network with 4 nodes and 12 links. The results show significant improvement over heuristic based algorithms.

Similarly, Marbach et al. [295] use RL to construct a dynamic admission control with routing policy for new calls in integrated service networks. As the traditional

DP-based models for admission control with routing are computationally intractable, Marbach et al. [295] propose an approximation architecture consisting of an MLP with internal tunable weights that can be adjusted using TD(0). However, TD(0) has a slow rate of convergence, hence the authors integrate it with decomposition approach to represent the network as a set of decoupled link processes. This allows to adopt a decentralized training and decision making, which not only significantly reduce training time, but also achieve sophisticated admission control with routing policies that are otherwise difficult to obtain via heuristics approaches.

7.2 Resource allocation

Recall that the challenge in resource allocation lies in predicting demand variability and future resource utilization. ML-based techniques can be leveraged to learn the indicators that can aid in resource allocation as summarized in Table 15. The most suitable ML-based approach for the resource allocation decision problem is RL. The primary advantage of RL is that it can be deployed without any initial policies, and it can learn to adapt to the dynamic demands for a reactive resource allocation. For instance, Tesauro [442] use decompositional RL to allocate and reallocate data center server resources to two different workloads, a web-based time-varying transactional workload and a non-web-based batch workload. Since the impact of a resource allocation decision is Markovian, the RA problem benefits largely from an MDP-based formulation. However, the state and action space of an MDP grows exponentially and leads to the dimensionality problem.

To address this problem, the authors in [442] propose a decompositional formula of RL for composite MDPs. The decompositional RL uses a localized version of SARSA(0) algorithm to learn a local value function based on local state and local resource allocation of a request instead of global knowledge. Vengerov [454] go further in applying RL to the allocation of multiple resource types (e.g. CPU, memory, bandwidth), using fuzzy rules where some or all the fuzzy categories can overlap. Whereas, Mao et al. [294] use DNN to approximate functions in large scale RL task in order to develop a multi-resource cluster scheduler. Most recently, Pietrabissa et al. [361] propose a scalable RL based solution to the MDP problem for resource allocation using policy reduction mechanism proposed in [360] and state aggregation that combines lightly loaded states into one single state.

More specifically, Baldo et al. [35] and Bojovic et al. [65] optimize network resource allocation. Baldo et al. [35] use a supervised MLP-NN for real-time characterization of the communication performance in wireless networks and optimize resource allocation. On the other hand, Bojovic et al. [65] use MLP-NN to select the AP that will provide the best performance to a mobile user in IEEE 802.11 WLAN. In their proposals, each user collects measurements from each AP, such as signal to noise ratio (SNR), probability of failure, business ratio, average beacon delay, and number of detected stations. These metrics are used to describe different APs and train a two layer MLP-NN. The output of the MLP-NN is the downlink throughput, which is a standard performance metric used by mobile clients. The MLP-NN is trained rigorously with different configuration parameters to result in the lowest normalized RMSE (NRMSE). Finally, the MLP-NN is deployed to select the AP that will yield the optimal throughput in different scenarios and evaluated on EXTREME testbed [364]. Undoubtedly, the ML-based AP selection for network resource allocation, outperforms AP selection mechanisms based on the signal to noise ratio (SNR), the load based scheme and the beacon delay scheme, especially in dynamic environments.

Similarly, Adeel et al. [6] leverage RNN to build an intelligent LTE-Uplink system that can optimize radio resource allocation based on user requirements, surrounding environments, and equipment's ability. In particular, their system can allocate the optimal radio parameters to serving users and suggest the acceptable transmit power to users served by adjacent cells for inter-cell-interference coordination. To analyze the performance of RNN, three learning algorithms are analyzed, namely GD, adaptive inertia weight particle swarm optimization (AIWPSO), and differential evolution (DE). One RNN is trained and validated using each of the above learning algorithms with a dataset of 6000 samples. The dataset is synthetically generated by executing multiple simulations of the LTE environment using a SEAMCAT simulator. Evaluation results show that AIWPSO outperforms the other learning algorithms, with respect to accuracy (based on MSE). However, AIWPSO's better accuracy is achieved at the expense of longer convergence time due to extra computational complexity. Unfortunately, [6] does not evaluate the effectiveness of resource allocation for the proposed LTE system. However, the analysis of the learning algorithms can provide valuable insights in applying ML to similar networking problems.

Though, admission control and resource allocation have been studied separately, Testolin et al. [443] leverage ML to address them jointly for QoE-based video requests in wireless networks. They combine unsupervised learning, using stochastic RNN, also known as RBM, with supervised classification using a linear classifier, to estimate video quality in terms of the average Structural SIMilarity (SSIM) index. The corresponding module uses video frame size that is readily available at the network layer to control admission and resource provisioning. However, the relationship between video frame size and SSIM in non-linear and RBM extracts an abstract representation of the features that describe the video. The linear classifier

Table 15 Summary of ML-based Resource Allocation

Ref.	ML Technique	Network	Dataset	Features	Output	Evaluation	
						Settings	Results
Baldo et al. [35]	Supervised: · MLP-NN	Wireless networks	Simulation data generated using ns-Miracle simulator	· Signal to noise ratio · Received frames · Erroneous frames · Idle time	· Throughput · Delay · Reliability	2 layers with 6 neurons in the hidden layer	Very good accuracy
Bojovic [65]	Supervised: · MLP-NN	Wireless LAN	Synthetic data generated using testbed	· Signal to noise ratio · Probability of failure · Business ratio · Average beacon delay · Number of detected stations	· Throughput of an access point	2 layers with varying number of nodes in the hidden layer, maximum number of epochs, and learning rate	NRMSE = 8%
Adeel et al. [6]	RNN with GD, AIWPSO, and DE	Cellular network	Synthetically generated using a SEAMCAT LTE simulator	· Signal to interference noise ratio · Inter-cell-interference · Modulation/coding schemes · Transmit power	Throughput	5-8-1[a]	Mean square error · AIWPSO: 8.5×10^{-4} · GD: 1.03×10^{-3} · DE: 9.3×10^{-4}
Testolin et al. [443]	Supervised: · Linear classifier Unsupervised: · RNN	Wireless networks	38 video clips taken from CIF	· Video frame size	· Quality level of each video in terms of the average SSIM index	32 visible units with a varying number of hidden units	RMSE < 3%
Mijumbi et al. [312]	RL · Q-learning (ϵ-greedy and softmax)	VNs	Simulation on ns-3 and real Internet traffic traces	States · Percentages of allocated and unused resources in substrate nodes and links	Actions · Increase or decrease the percentages of allocated resource	2^9 states, 9 actions	Improved the acceptance ratio
Mijumbi et al. [313]	Supervised: · FNN	VNF chains	VoIP traffic traces	· Dependency of resource requirements of each VNFC on its neighbor VNFCs · Historical local VNFC resource utilization	· Resource requirements of each VNFC	2 NNs for each VNFC	Accuracy ∼90%
Shi et al. [410]	Supervised: · MDP · BN	VNF chains	Simulation data generated using WorkflowSim	· Historical resource usage	· Future resource reliability	Running time for MDP: $O(t^{v+1})$, where t and v stand for the number of NFV component tasks and the number of VMs, respectively	Better than other greedy methods in terms of cost

[a] Number of neurons at the input layer, hidden, and output layers, respectively

maps the abstractions to the SSIM coefficients, which are leveraged to accept or reject new video requests, and to adapt resource provisioning to meet the network resource requirements. The authors report a RMSE of below 3% using videos from a pool of 38 video clips with different data rates and durations.

Virtualization of network resources through NFV and virtual networks brings forward a new dimension to the resource allocation problem, that is, provisioning virtual resources sitting on top of physical resources. To leverage the benefits of virtualization, Mijumbi et al. [312] propose a dynamic resource management approach for virtual networks (VNs) using distributed RL that dynamically and opportunistically allocates resources to virtual nodes and links. The substrate network is modeled as a decentralized system, where multiple agents use Q-learning on each substrate node and link. These agents learn the optimal policy to dynamically allocate substrate network resources to virtual nodes and links. The percentage of allocated and unused resources (e.g. queue size, bandwidth) in substrate nodes or links represent the states of Q-learning, with two explicit actions to increase or decrease the percentage of allocated resource. A biased learning policy is exploited with an initialization phase to improve the convergence rate of Q-learning. This Q-learning based action selection approach for resource allocation outperforms ϵ-greedy and softmax in ns-3 simulation with real Internet traffic traces. Furthermore, in comparison to static allocation, the proposed method improve the ratio of accepting VNs without affecting their QoSs.

In addition, Mijumbi et al. [312] use FNN to predict future resource requirements for each VNF component (VNFC) in a service function chain [313]. Each VNFC is modeled using a pair of supervised FNNs that learn the trend of resource requirements for the VNFC by combining historical local VNFC resource utilization information with the information collected from its neighbors. The first FNN learns the dependence of the resource requirements for each of the VNFCs, which is used by the second FNN to forecast the resource requirements for each VNFC. The predictions are leveraged to spin-up and configure new VNFCs or deallocate resources to turn off VNFCs. Evaluation based on real-time VoIP traffic traces on a virtualized IP Multimedia Subsystem (IMS) reveals a prediction accuracy of approximately 90%.

In contrast, Shi et al. [410] use BN to predict future resource reliability, the ability of a resource to ensure constant system operation without disruption, of NFV components based on historical resource usage of VNFC. The learning algorithm is triggered when an NFV component is initially allocated to resources. As time evolves, the BN is continuously trained with resource reliability responses and transition probabilities of the BN are updated, resulting in improved prediction accuracy. The predictions are leveraged in an MDP to dynamically allocate resources for VNFCs. Using WorkflowSim simulator, the authors demonstrate that the proposed method outperforms greedy methods in terms of overall cost.

7.3 Summary

As evident from Tables 14 and 15, the ML-based resource management schemes studied in this paper can be broadly classified into two groups—supervised learning-based and RL-based. Application of unsupervised techniques in resource management is rather unexplored, with the exception of a few works. In addition, MLP-NN, though applied with a variety of parameter settings, is the most popular supervised technique, while Q-learning dominates the choice of RL-based approaches. Furthermore, other works have leveraged BN techniques, to introduce the flexibility of having a probability distribution rather than individual values produced by NN-based approaches. However, MLP-NNs offer better scalability than BN and RL, since the number of neurons in different layers of an MLP-NN can be tuned based on the problem dimension. Whereas, the number of states in RL can explode very quickly in a moderate size network. In the past, several techniques, such as decomposition, decentralization, and approximation have been used to deal with the dimensionality issue of applying RL. Recently, RL combined with deep-learning has been shown as a promising alternative [294] that can be leveraged to tackle various resource management problems in practical settings. Nonetheless, NN-based supervised approaches exhibit steady performance in terms of both accuracy and convergence.

Although the ML-based resource management schemes studied in this paper differ in terms of the feature sets, they either predict one or more QoS metrics of interest or generate an acceptance/rejection decision for an incoming request, based on a QoS estimation. The ML-based resource management approaches also exhibit a similarity regarding the network and dataset. The focus of the majority of approaches is on wireless networks, where resource contention is more profound than wired networks. Due to the lack of real-life traces, these approaches adopt different methods to simulate the network of interest and produce training and testing data. Therefore, more research is needed that can evaluate the performance of the proposed ML techniques in real networks and with real data.

8 Fault management

Fault management involves detection, isolation, and correction of an abnormal condition of a network. It requires network operators and administrators to have a thorough knowledge of the entire network, its devices and all the applications running in the network. This is an unrealistic expectation. Furthermore, recent advances in technology, such as virtualization and softwarization makes today's network monumental in size, complexity and highly dynamic. Therefore, fault management is becoming increasingly challenging in today's networks.

Naïve fault management is reactive and can be perceived as a cyclic process of detection, localization and mitigation of faults. First, fault detection jointly correlates vairous different network symptoms to determine whether one or more network failures or faults have occurred. For example, faults can occur due to reduced switch capacity, increased rate of packet generation for a certain application, disabled switch, and disabled links [37]. Therefore, the next step in fault management is localization of the root cause of the fault(s), which requires pinpointing the physical location of the faulty network hardware or software element, and determining the reason for the fault. And lastly, fault mitigation aims to repair or correct the network behaviour. In contrast, fault prediction is proactive and aims to prevent faults or failures in the future by predicting them and initiating mitigation procedures to minimize performance degradation. ML-based techniques have been proposed to address these challenges and promote cognitive fault management in the areas of fault prediction, detection, localization of root cause, and mitigation of the faults. In the following subsections, we describe the role ML has played in these prominent challenges for fault management.

8.1 Predicting fault

One of the fundamental challenges in fault management is fault prediction to circumvent upcoming network failures and performance degradation. One of the first ML-based approaches for detecting anomalous events in communication networks is [301]. This approach performs fault prediction by continuously learning to distinguish between normal and abnormal network behaviors and triggering diagnostic measures upon the detection of an anomaly. The continuous learning enables adaptation of the fault prediction and diagnostic measure to the network dynamics without explicit control. Although the work in [301] leverages ML in fault prediction, it does not mention any specific technique. On the other hand, BNs have been widely used in communication and cellular networks to predict faults [193, 247, 248].

In a BN, the normal behavior of a network and deviations from the normal are combined in the probabilistic framework to predict future faults in communication and cellular networks. However, one shortcoming of the system in [193] is that it cannot predict impact on network service deterioration. Nevertheless, a common drawback of BN is that they are not sensitive to temporal factors, and fail to model IP networks that dynamically evolve over time. For such networks, Ding et al. [118] apply dynamic BN to model both static and dynamic changes in managed entities and their dependencies. The dynamic BN model is robust in fault prediction of a network element, localization of fault and its cause and effect on network performance.

Snow et al. [414] use a NN to estimate the dependability of a 2G wireless network that is used to characterize availability, reliability, maintainability, and survivability of the network. Though the NN is trained vigorously with analytical and empirical datasets, it is limited, due to the wireless network topology having a fixed topology. This is far from reality. Furthermore, network fault predictions are tightly coupled with wireless link quality. Therefore, in Wang et al. [466], the estimation of the link quality in WSNs is postulated as a classification problem, and solved by leveraging supervised DT, rule learner, SVM, BN, and ensemble methods. The results reveal that DTs and rule learners achieve the highest accuracy and result in significant improvement in data delivery rates.

A daunting and fundamental prerequisite for fault prediction is feature selection. It is non-trivial to extract appropriate features from an enormous volume of event logs of a large scale or distributed network system [285]. Therefore, feature selection and dimensionality reduction are imperative for accurate fault prediction. Wang et al. [466] propose to employ local over the global features, as local features can be collected without costly communications in a wireless network. In contrast, Lu et al. [285] use a manifold learning technique called Supervised Hessian Locally Linear Embedding (SHLLE), to automatically extract the failure features and generate failure prediction. Based on an empirical experiment, the authors show that SHLLE outperforms the feature extraction algorithm, such as PCA, and classification methods, including k-NN and SVM.

Pellegrini et al. [355] propose an ML-based framework to predict the remaining time to failure (RTTF) of applications. Their framework is application-agnostic, that is, it is applicable to scenarios where a sufficient number of observations of the monitored phenomena can be collected in advance. The framework uses different ML techniques for building prediction models, namely linear regression, M5P, REPTree, LASSO, SVM, and Least-Square SVM, allowing network operators to select the most suitable technique based on their needs. In addition, other ML techniques can be easily integrated in the framework. The ML techniques in the framework are compared for a multi-tier e-commerce web application running on a virtualized testbed, and show that the REPTree and M5P outperform the other ML techniques for predicting RTTF. It is essential to note that the model has a high prediction error when the network system is temporally far from the occurrence of the failure. However, as the network system approaches the time of the occurrence of the failure, the number of accumulated anomalies increase and the model is able to predict the RTTF with a high accuracy.

Wang et al. [469] present a mechanism for predicting equipment failure in optical networks using ML-based techniques and TSF. The operational states of an equipment are built by leveraging physical indicators, such as input optical power, laser bias current, laser temperature offset, output optical power, environmental temperature, and unusable time. A double-exponential smoothing time series algorithm uses the historical data from time $t - n$ to time $t - 1$ to predict the values of the physical indicators at a future time instance $t + T$. This is accomplished by using a kernel function and penalty factor in an SVM to model non-linear relationships and reduce misclassification, respectively. The enhanced SVM accomplish an accuracy of 95% in predicting equipment failure based on real data from an optical network operator.

Most recently, Kumar et al. [255] explore the applicability of a wide range of regression and analytical models to predict inter-arrival time of faults in a cellular network. They analyze time-stamped faults over a period of one month from multiple base stations of a national mobile operator in USA. The authors observe that current networks barely reside in a healthy state and patterns of fault occurrence is non-linear. In a comparison of the different ML-based techniques for fault prediction, they show that DNN with autoencoders outperform other ML techniques, including autoregressive NN, linear and non-linear SVM, and exponential and linear regression. An autoencoder is a variant of NN that consists of an encoder and a decoder and used for dimensionality reduction. The autoencoder is pre-trained on the testing data and then converted into a traditional NN for computing prediction error. The pre-training of each layer in an unsupervised manner allows for better initial weights, and results in higher prediction accuracy.

8.2 Detecting fault

Unlike fault prediction, fault detection is reactive and identifies and, or classifies a failure *after* it has occurred, using network symptoms, performance degradation, and other parameters. Rao [382] propose fault detection for cellular networks that can detect faults at different levels, base station, sector, carrier, and channel. They employ a statistical hypothesis testing framework which combines parametric, semi-parametric, and non-parametric test statistics to model expected behavior. In parametric and semi-parametric statistical tests, a fault is detected when significant deviations from the expected activity is observed. In the case of non-parametric statistical tests, where the expected distribution is not known a-priori, the authors use a combination of empirical data and statistical correlations to conduct the hypothesis test. The test is dependent on a threshold value that is initially set through statistical analysis of traffic patterns. However, improper threshold settings may lead to high FPs and FNs. Hence, the threshold should be adapted to changing traffic patterns due to spatial, temporal, and seasonal effects. In the background, an open loop routine continuously learns and updates the threshold, in an adjustable period of time. However, the time for learning may be large for certain applications that may impact fault detection time.

Baras et al. [37] implement a reactive system to detect and localize the root cause of faults for X.25 protocol, by combining an NN with an expert system. Performance data, such as blocking of packets, queue sizes, packet throughput from all applications, utilization of links connecting subnetworks, and packet end-to-end delays, are used to train a RBFNN for various faults. The output of the NN is a fault code that represents one of the various fault scenarios. A classifier leverages the aggregated output of the NN to determine the current status of the network as normal or faulty. The detection phase is repeated until a confidence of K out of M is achieved, which activates the expert system to collect and deduce the location and cause of the fault.

Recently, Adda et al. [5] build a real-time fault detection and classification model using k-Means, Fuzzy C Means (FCM), and EM. They leverage SNMP to collect information from the routers, switches, hubs, printers and servers in an IP network of a college campus. The authors select 12 features that exhibit sensitivity to the behavior of network traffic [370], and use the traffic patterns to form clusters that represent normal traffic, link failure, server crash, broadcast storm and protocol error. Their evaluation results reveal that though k-Means and EM are relatively faster than FCM, FCM is more accurate.

Moustapha and Selmic [324] detect faulty nodes in a WSN using RNN. The nodes in the RNN hidden layers model sensor nodes in WSN, while the weights on the edges are based on confidence factors of the received signal strength indicators (RSSI). Whereas, the output of the RNN is an approximation of the operation of the WSN. Fault detection is achieved by identifying discrepancies between approximated and real WSN values. The RNN successfully detect faults, without early false alarms, for a small scale WSN with 15 sensors and synthetically introduced faults.

Recall, supervised fault detection requires models to be trained with normal and failure-prone datasets. However, Hajji [178] propose an unsupervised fault detection mechanism for fast detection of anomalies in LAN through traffic analysis. They design a parametric model of network traffic, and a method for baselining normal network operations using successive parameter identification, instead of EM. The fault detection problem is formulated as a change point problem that observes the baseline random variable and raises an alarm as soon as the variable exceeds an expected value. Experimental evaluation validate the fault detection mechanism in real-time on a real network with high detection accuracy.

Recently, Hashmi et al. [181] use different unsupervised algorithms, such as *k*-Means, FCM, Kohonens SOM, Local Outlier Factor, and Local Outlier Probabilities, to detect faults in a broadband service provider network that serves about 1.3 million customers. For this purpose, they analyze a real network failure log (NFL) dataset that contains status of customer complaints, along with network generated alarms affecting a particular region during a certain time. The selected data spans a duration of 12 months and contains about 1 million NFL data points from 5 service regions of the provider. The collected NFL dataset has 9 attributes, out of which 5 are selected for the analysis: (i) fault occurrence date, (ii) time of the day, (iii) geographical region, (iv) fault cause, and (v) resolution time. At first, *k*-Means, FCM and Kohonens SOM clustering techniques are applied to cluster the NFL dataset that is completely unlabeled. Afterwards, density-based outlier determination algorithms, such as Local Outlier Factor, and Local Outlier Probabilities, are used on the clustered data to determine the degree of anomalous behavior for every SOM node. The evaluation results show that SOM outperforms *k*-Means and FCM in terms of error metric. Furthermore, Local Outlier Probabilities algorithm applied on SOM is more reliable in identifying the spatio-temporal patterns linked with high fault resolution times.

8.3 Localizing the root cause of fault

The next step in fault management is to identify the root cause and physically locate the fault to initiate mitigation. This minimizes the mean time to repair in a network that does not deploy a proactive fault prediction mechanism. Chen et al. [91, 92] use DTs and clustering to diagnose faults in large network systems. The DTs are trained using a new learning algorithm, MinEntropy [91], on datasets of failure prone network traces. To minimize convergence time and computational overhead, MinEntropy uses an early stopping criteria and follows the most suspicious path in the DT. Chen et al. [91] complement the DT with heuristics, to correlate features with the number of detected failures to aid in feature selection and fault localization. MinEntropy is validated against actual failures observed for several months on eBay [127]. For single fault cases, the algorithm identifies more than 90% of the faults with low FPRs. In contrast, Chen et al. [92] employ clustering to group the successes and failures of requests. A faulty component is detected and located by analyzing the components that are only used in the failed requests. In addition to the single fault cases, the clustering approach can also locate faults occurring due to interactions amongst multiple components, with a high accuracy and relatively low number of false positives.

Ruiz et al. [393] use a BN to localize and identify the most probable cause of two types of failures, the tight filtering and inter-channel interference, in optical networks. They discretize the continuous real-valued features of Quality of Transmission (QoT), such as received power and pre-forward error correction bit error rate (pre-FEC BER) for categories. The authors use these categories and type of failures to train the BN, which can identify the root cause of the failure at the optical layer when a service experiences excessive errors. The BN achieves high accuracy of 99.2% on synthetically generated datasets.

Similarly, Khanafer et al. [237] develop an automated diagnosis model for Universal Mobile Telecommunications System (UMTS) networks using BN. The core elements of the diagnosis model are the causes and symptoms of faults. The authors consider two types of symptoms, i.e., alarms and Key Performance Indicators (KPI). To automatically specify KPI thresholds, they investigated two different discretization methods, an unsupervised method called Percentile-based Discretization (PBD) and a supervised method called Entropy Minimization Discretization (EMD). The performances of the two discretization methods are evaluated on a semi-dynamic UMTS simulator that allows the generation of a large amount of causes and symptoms data required to construct the diagnosis model. As EMD technique outperforms PBD by a large margin in the simulation study, the authors analyze the diagnosis model consisting of BN and EMD in a real UMTS network, utilizing alarms and KPIs extracted from an operations and maintenance center. Using a 3-fold cross-validation test, the correct faults are diagnosed in 88.1% of the cases. In the remaining cases, the diagnosis is incorrect for the first cause but correct for the second, and the diagnosis model converges from around 100 data points.

Kiciman and Fox [241] propose PinPoint for fault detection and localization that requires no a priori knowledge of the faults. The models capture the runtime path of each request served by the network and delineates it as the causal path in the network. It exploits the paths to extract two low-level behaviors of the network, the path shape and the interaction of the components. Using the set of previous path shapes modeled as a Probabilistic Context-Free Grammar (PCFG), it builds a dynamic and self-adapting reference model of the network. Therefore, fault prediction is a search for anomalies against the reference model. Pinpoint uses DT with ID3 to correlate the anomaly to its probable cause in the network. The DT is converted to an equivalent set of rules by generating a rule for each path from the root of the tree to a leaf. PinPoint ranks the rules, based on the number of paths classified as anomalous, to identify the hardware and, or software components that are correlated with the failures.

Johnsson et al. [225] use discrete state-space particle filtering to determine the locations of performance degradations in packet switched networks. Their approach is based on active network measurements, probabilistic inference, and change detection in the network. They define a PMF to define the location of faulty components in the network. It is a lightweight fault detection and isolation mechanism, which is capable of automatically detecting and identifying the location of the fault in simulation of different sized tree topologies. It is imperative to realize that time to fault localization is dependent on precise position of the fault in the topology. This is because the links closer to the root are measured more often in comparison to links close to the leaf nodes. Hence, the filter is able to learn the positions close to the root. In addition, the algorithm minimizes false positives or false negatives for the chosen parameter values.

Barreto et al. [40] develop an unsupervised approach to monitor the condition of cellular networks using competitive neural algorithms, including Winner-Take-All (WTA), Frequency-Sensitive Competitive Learning (FSCL), SOM, and Neural-Gas algorithm (NGA). The model is trained on state vectors that represent the normal functioning of a CDMA2000 wireless network. Global and local normality profiles (NPs) are built from the distribution of quantization errors of the training state vectors and their components, respectively. The overall state of the cellular network is evaluated using the global NP and the local NPs are used to identify the causes of faults. Evidently, the joint use of global and local NPs is more accurate and robust than applying these methods in isolation.

8.4 Automated mitigation

Automated mitigation improves fault management by minimizing and, or eliminating human intervention, and reducing downtime. For proactive fault prediction, automated mitigation involves gathering information from the suspected network elements to help find the origin of the predicted fault. For building this information base, a fault manager may either actively poll selected network elements, or rely on passive submission of alarms from them. In both cases, actions should be selected carefully since frequent polling wastes network resources, while too many false alarms diminish the effectiveness of automated mitigation. On the other hand, in the case of reactive fault detection, automated mitigation selects a workflow for troubleshooting the fault. Therefore, the fundamental challenge in automated mitigation is to select the optimal set of actions or workflow in a stochastic environment.

He et al. [183] address this fundamental challenge for proactive fault management using a POMDP, to formulate the trade-off between monitoring, diagnosis, and mitigation. They assume partial observability, to account for the fact that some monitored observations might be missing or delayed in a communication network. They propose an RL algorithm to obtain approximate solutions to the POMDP with large number of states representing real-life networks. The authors devise a preliminary policy where the states are completely observable. Then, they fine-tune this policy by updating the belief space and transition probabilities in the real world, where the states are incompletely observed.

In contrast, for reactive fault detection, Watanabe et al. [470] propose a method for automatically extracting a workflow from unstructured trouble tickets to troubleshoot a network fault. A trouble ticket contains free-format texts that provide a complete history of troubleshooting a failure. The authors use supervised NB classifier to automatically classify the correct labels for each sentence of a trouble ticket and remove unrelated sentences. They propose an efficient algorithm to align the same actions described with different sentences by using multiple sequence alignment. Furthermore, clustering is used to find the actions that have different mitigation steps depending on the situation. This aid the operators in selecting the appropriate next action. Using real trouble tickets, obtained from an enterprise service, the authors report a precision of over 83%.

8.5 Summary

As summarized in Tables 16, 17 and 18, most of the ML-based fault management approaches use different supervised learning techniques that depend on training data to predict/detect/locate faults in the network. However, a common challenge faced by these techniques is the scarcity of fault data generated in a production network. While both normal and fault data are easily available for a test or simulated network, only normal data with infrequent faults are routinely available for a production network. Although injecting faults can help produce the required data [285], it is unrealistic to inject faults in the production network just for the sake of generating training data. On the other hand, synthetic data generated in a test or simulated network may not perfectly mimic the behavior of a production network. Such limitations increase the probability of ML techniques being ill-trained in an unfamiliar network setting. As a remedy, some approaches leverage unsupervised techniques that rely on detecting changes in network states instead of using labeled fault data. However, unsupervised techniques can take longer time to converge than supervised approaches, potentially missing any fault occurring before the convergence. Therefore, a potential research direction can be to explore the applicability of semi-supervised and RL-based techniques for fault management.

Table 16 Summary of ML-based fault prediction

Ref.	ML Technique	Network	Dataset	Features	Output	Evaluation	
		(*location*)			(*training*)	Settings	Results
Hood et al. [193]	Supervised: · BN	Campus network	Data collected from router	Management information base (MIB) variables for following network functions · Interface group · IP group · UDP group	Predict network health	500 samples for each of 14 MIB variables of the 3 network functions	Predict approximately 8 min before fault occurrence
Kogeda et al. [248]	Supervised: · BN	Cellular network	Simulation with fault injection	·Power ·Multiplexer ·Cell ·Transmission	Faulty or not	4 nodes each with 3 states	Confidence level of 99.8%
Snow et al. [414]	Supervised: · NN (MLP)	Wireless network	Generated using discrete time event simulation	·Mean time to failure ·Mean time to restore ·Time Profile ·Run Time	Dependability of a network ·Survivability ·Availability ·Failed components ·Reportable outages	14 inputs, 10 and 5 nodes in the first and second hidden layer, respectively	Closely approximates reportable outages
Wang et al. [466]	Supervised: · DT (J4.8) · Rule learners (JRip) · SVM · BN · Ensemble	Wireless sensor network	Generated using sensor network testbed	·Received signal strength indication ·Send and forward buffer sizes ·Channel load assessment ·Forward and backward	Link quality estimation	10-fold cross validation was used with 5000 samples	Accuracy · 82% for J4.8 ·80% for JRip
Lu et al. [285]	Manifold learning: ·SHLLE	Distributed systems	Generated from a testbed of a distributed environment with a file transfer application	System performance ·interface group ·IP group ·TCP group ·UDP group	Prediction of network, CPU, and memory failures	Not provided	· Precision: 0.452 ·Recall: 0.456 · False positive rate: 0.152
Pellegrini et al. [355]	Different ML methods: ·Linear Regression · M5P · REP-Tree · LASSO · SVM · Least-Square SVM	Multi-tier e-commerce web application	Generated from a testbed of a virtual architecture	Different system performance	Remaining Time to Failure (RTTF)	Not provided	Soft mean absolute error · Linear regression: 137.600 · M5P: 79.182 · REP-Tree: 69.832 · LASSO as a Predictor: 405.187 · SVM: 132.668 · Least-Square SVM: 132.675
Wang et al. [469]	Supervised: · Double-exponential smoothing (DES) and SVM	Optical network	Real data collected from an optical network of a telecommunications operator	Indicators In Board Data: ·Input Optical Power · Laser Bias Current · Laser Temperature Offset · Output Optical Power · Environmental Temperature ·Unusable Time	Predicting equipment failure	10-fold cross-validation was used to test model accuracy	DES with SVM · Prediction accuracy: 95%
Kumar et al. [255]	Unsupervised: · DNN with Autoencoders	Cellular Network	Fault data from one of the national mobile operators of USA for a month	Historical data of fault occurrence and their inter-arrival times	Prediction of inter-arrival time of faults	10 neurons in the hidden layer	DNN with autoencoders · NRMSE: 0.122092 ·RMSE: 0.504425

Table 17 Summary of ML-based Fault Detection

Ref.	ML Technique	Network	Dataset	Features	Output	Evaluation	
		(*location*)			(*training*)	Settings	Results
Rao [382]	Statistical learning	Cellular network	Data collected from real cellular networks	Mobile user call load profile	Detect faults at ·Base station level ·Sector level ·Carrier level ·Channel level	Not provided	Bounded probability of false alarm
Baras et al. [37]	A combination of NN (radial basis functions)	Cellular network (X.25 protocol)	Simulation with OPNET	For each fault scenario ·Blocking of packets ·Queue sizes ·Packet throughput ·Utilization on links connecting subnetworks ·Packet end-to-end delays	Detect one of the fault scenarios ·Reduced switch capacity ·Increased packet generation rate of a certain application ·Disabled switch ·Disabled links	Varying number of hidden nodes between 175 and 230	Different rates of errors
Adda et al. [5]	Supervised: · *k*-Means ·FCM ·EM	IP network of a school campus	Obtained from a network with heavy and light traffic scenarios	12 variables of interface (IF) category collected through SNMP	Fault classes: · Normal traffic ·Link failure traffic ·Server crash ·Broadcast storm ·Protocol error	Not provided	Precision for heavy scenario in router dataset · *k*-Means = 40 · FCM = 85 · EM = 40
Moustapha and Selmic [324]	Supervised: · RNN	Wireless sensor network	Collected from a simulated sensor network	·Previous outputs of sensor nodes· Current and previous output samples of neighboring sensor nodes	Approximation of the output of the sensor node	8-10-1[a]	Constant error smaller than state-of-the-art
Hajji [178]	Unsupervised change detection method	Local area networks	Collected from a real network using remote monitoring agents	Baseline random variable	An alarm as soon as an anomaly occurs	Time to detect : · 50 s to 17 min	Accuracy: 100% · Low alarm rate: 0.12 alarms per hour
Hashmi et al. [181]	Supervised: · *k*-Means · FCM · SOM	Broadband service provider network	1 million NFL data points from 5 service regions	·Fault occurrence date ·Time of the day ·Geographical region ·Fault cause ·Resolution time	Identify the spatio-temporal patterns linked with high fault resolution times	SOM on a 15x15 network grid for 154 epochs	Sum of squared errors: ·*k*-Means = 2156788 · FCM = 2822823 · SOM = 1136

[a] Number of neurons at the input layer, hidden, and output layers, respectively

Table 18 Summary of ML-based fault localization

Ref.	ML Technique	Network	Dataset	Features	Output	Evaluation	
		(*location*)			(*training*)	Settings	Results
Chen et al. [91]	DT (C4.5)	Network systems	Snapshots of logs from eBay	A complete request trace ·Request type ·Request name ·Pool ·Host ·Version ·Status of each request	Different faulty elements	10 one hour snapshots with 14 faults in total	·Precision: 92% ·Recall: 93%
Ruiz et al. [393]	BN	Optical network	Synthetically generated time series	Quality of Transmission (QoT) parameters ·Received power ·Pre-forward error correction bit error rate (pre-FEC BER)	Detect one of the two fault scenarios ·Tight filtering ·Inter-channel interference	5,000 and 500 time series for training and testing, respectively	Accuracy: 99.2%
Khanafer et al. [237]	BN and EMD	Cellular network	Synthetically generated from a simulated and a real UMTS network	·Causes of faults ·Symptoms, i.e., alarms and KPIs	Identify the cuase of the fault	77 and 42 faulty cells for training and testing, respectively	Accuracy: 88.1%
Kiciman and Fox [241]	Supervised: · DT (ID3)	Three-tier enterprise applications	Generated using small testbed platform	Paths classified as normal or anomalous	Hardware and software components that are correlated with the failures	Three different DTs were evaluated	Correctly detect 89% to 96% of major failures
Johnsson et al. [225]	Unsupervised: discrete state-space particle filtering	IP network	Discrete event simulator	·Active network measurements ·Probabilistic inference ·Change detection	Probability mass function indicating the location of the faulty components	Operations per filter: $O(\|G\|)$, where $\|G\|$ is the number of edges in a graph G	Found the location of faults and performance degradations in real time
Barreto et al. [40]	Unsupervised: · Winner-Take-All (WTA) · Frequency-Sensitive Competitive Learning (FSCL) · SOM ·Neural-Gas algorithm (NGA)	Cellular network	Simulation study	State vectors representing the normal functioning of a network	State vector causing the abnormally	400 vectors were used for training and 100 vectors were used for testing	False alarm: · WTA: 12.43 · FSCL: 10.20 · SOM: 8.75 ·NGA: 9.50

The ML-based fault management approaches surveyed in this paper focus on a variety of networks. Consequently, the fault scenarios studied in these approaches vary greatly as they depend both on the layer (e.g. physical, link, or IP layer) and the type (e.g. cellular, wireless, local area network) of the network. The same holds for feature set and output of these schemes, as both features and outputs depend on the fault scenario of a particular network. In addition, the evaluation settings adopted by these approaches lack uniformity. Therefore, a pairwise comparison between the evaluation results of two approaches in any of the Tables 16, 17 and 18 may be misleading. Nonetheless, it is clear that ML techniques can aid the cumbersome and human centric fault management process, by either predicting faults in advance, or narrowing down the cause or location of the fault that could not be avoided in the first place.

9 QoS and QoE management

The knowledge about the impact of network performance on user experience is crucial, as it determines the success, degradation or failure of a service. User experience assessment has attracted a lot of attention. In early works, there was no differentiation between user experience and network QoS. User experience was then measured in terms of network parameters (e.g. bandwidth, packet loss rate, delay, jitter), and application parameters, such as bitrate for multimedia services. While monitoring and controlling QoS parameters is essential for delivering high service quality, it is more crucial, especially for service providers, to evaluate service quality from the user's perspective.

User QoE assessment is complex as individual experience depends on individual expectation and perception. Both are subjective in nature, and hard to quantify and measure. QoE assessment methods went through different stages this last decade, from subjective testing to engagement measurement through objective quality modeling. Subjective testing, where users are asked to rate or assign opinions scores averaged into a mean opinion score (MOS), has been and is still widely used. Subjective testing is simple and easy to implement, and the MOS metric is easy to compute. However because one cannot force users to rate a service and rate it objectively, MOS scores can be unfair and biased, and are subjected to outliers. Objective quality models, such as the video quality metric (VQM) [362], the perceptual evaluation of speech quality (PESQ) metric [386] and the E-model [51] for voice and video services, were proposed to objectively assess service quality by human beings and infer more "fair" and unbiased MOS. Full-reference (FR) quality models, like PESQ and VQM, compute quality distortion by comparing the original signal against the received one. They are as such accurate, but at the expense of a high computational effort. On the contrary, no-reference (NR) models like E-model try to assess the quality of a distorted signal without any reference to the original signal. They are more efficient to compute, however they may be less accurate. More recently, measurable user *engagement* metrics, such as service time and probability of return, have emerged from data-driven QoE analysis. Such metrics are found to draw more directly the impact of user quality perception to content providers; business objectives.

Statistical and ML techniques have been found useful in linking QoE to network- and application-level QoS, and understanding the impact of the latter on the former. Linear and non-linear regression (e.g. exponential, logarithmic, power regression) was used to quantify the individual and collective impact of network- and application- level QoS parameters (e.g. packet loss ratio, delay, throughput, round-trip time, video bitrate, frame rate, etc.) on the user's QoE. In the literature, simple-regression models with a single feature are most dominant [145, 240, 383, 408], although the collective impact of different QoS parameters was also considered [23, 132].

Simple regression: In [408], Shaikh et al. study existing correlation between different network-level QoS parameters and MOS in the context of a web surfing. They show that a correlation does exist and that among 3 forms of regression (linear, exponential, and logarithmic), linear regression renders the best correlation coefficient between QoE and packet loss rate, exponential regression captures the correlation between QoE and file download time with highest accuracy, whereas logarithmic regression is the best fit for linking QoE to throughput.

Reichl et al. [383], in alignment with the Weber-Fechner law from the field of psychophysics, use logarithmic regression to quantify the correlation between available bandwidth and mobile broadband service users' MOS.

In [145], Fiedler et al. test the IQX hypothesis according to which QoE and QoS parameters are connected through an exponential relationship. Their experiment validates the IQX hypothesis for VoIP services, where PESQ-generated MOS is expressed as a function of packet loss, and reordering ratio caused by jitter. For web surfing, exponential mappings are shown to outperform a previously published logarithmic function.

Steven's power law from the field of psychophysics, according to which there is a power correlation between the magnitude of a physical stimulus and the intensity or strength that people feel, was applied by Khorsandroo et al. [239, 240] to find a power mapping function between MOS and packet loss ratio. A comparative study shows that the proposed power correlation is outperformed by the logarithmic correlation from [383].

Multi-parameter regression: In order to gasp the impact of the global network condition on the QoE, Elkotob et al. [132] propose to map MOS to a set of QoS parameters (e.g. packet loss rate, frame rate, bandwidth, round trip time and jitter) as opposed to a single one. This idea was further promoted by Aroussi et al. [23] who propose a generic exponential correlation model between QoE and several QoS parameters based on the IQX hypothesis.

More complex regression and classification models based on supervised and unsupervised ML techniques (including deep learning) were also proposed and tested against real-life and trace-driven datasets. We report below on the characteristics of surveyed models and their performance in terms of accuracy, generally measured in terms of MSRE, and linearity, generally measured in terms of Pearson correlation coefficient (PCC), all summarized in Tables 19 and 20.

9.1 QoE/QoS correlation with supervised ML

In [235, 236], Khan et al. propose an Adaptive Neural Fuzzy Inference System (ANFIS)-based model to predict streamed video quality in terms of MOS. They also investigate the impact of QoS on end-to-end video quality for H.264 encoded video, and in particular the impact of radio link loss models in UMTS networks. A combination of physical and application layer parameters is used to train both models. Simulation results show that both models give good prediction accuracy. However, the authors conclude that the choice of parameters is crucial in achieving good performance. The proposed models in this paper need to be validated by more subjective testing. Other works like [501] have also used the ANFIS approach to identify the causal relationship between the QoS parameters that affect the QoE and the overall perceived QoE.

MLP-NNs are also reported to efficiently estimate the QoE by Machado et al. [287], who adopt a methodology that is similar to Khan et al. [235]. In this work, QoE is estimated by applying an MLP over network-related features (delay, jitter, packet loss, etc.) as well as video-related features (type of video, e.g. news, football, etc.). Different MLP models are generated for different program-generated QoE metrics, including Peak-Signal-to-Noise-Ratio (PSNR), MOS, Structural SIMilarity (SSIM) [468], and VQM. A synthetic video streaming dataset of 565 data points is created with EvalVid integrated to NS-2, and the models are trained over 70% of the database for parameter fine-tuning. It is observed that different QoE metrics can lead to very different model parameters. For instance, for the estimated MOS metric, best results are achieved by a single hidden-layer MLP with 10 neurons trained over 2700 epochs. Whereas for SSIM, 2 hidden layers with, respectively, 12 and 24 neurons trained over 1800 epochs are needed to achieve similar results. With a MSE of ≈ 0.01, the MOS-MLP model outperforms the other models. Nevertheless, with appropriate configuration all the models are able to predict the QoE with very high accuracy.

In [328], Mushtaq et al. apply six ML classifiers to model QoE/QoS correlation, namely NB, SVM, k-NN, DT, RF and NN. A dataset is generated from a controlled network environment where streamed video traffic flows through a network emulator and different delay, jitter, and packet loss ratio are applied. Opinion scores are collected from a panel of viewers and MOS are calculated. ML models are fed with nine features related to the viewers, network condition and the video itself, namely, viewer gender, frequency of viewing, interest, delay, jitter, loss, conditional loss, motion complexity and resolution. A 4-fold cross-validation is performed to estimate the performance of the models. Results show that DT and RF perform slightly better than the other models with a mean absolute error of 0.126 and 0.136 respectively, and a TPR of 74% and 74.8% respectively. The parameters of the models are not disclosed, and neither is the significance of the selected features in particular the viewer-related ones, whose usefulness and practicality in real-life deployment are questionable.

In [89] Charonyktakis et al. develop a modular user-centric algorithm MLQoE based on supervised learning to correlate the QoE and network QoS metrics for VoIP services. The algorithm is modular in that it trains several supervised learning models based on SVR, single hidden layer MLP-NN, DT, and GNB, and after cross-validation, it selects the most accurate model. The algorithm is user-centric in that a model is generated for each user, which makes it computationally costly and time consuming. 3 datasets are generated synthetically with calls established in 3 different testbeds under different network conditions: during handover (dataset 1), in a network with heavy UDP traffic (dataset 2), in a network with heavy TCP traffic (dataset 3). OMNET++ and a VoIP tool are used in this matter. The QoE of the received calls are assessed through both subjective testing (user-generated MOS) and objective measurement (PESQ and E-model). The no-reference ML models are trained with 10 network metrics (including average delay, packet loss, average jitter, and more) to output predicted MOS. The accuracy of the MLQoE model in predicting MOS and the accuracies of pure SVR, NN, DT and GNB models are further compared against the full-reference PSEQ's, the no-reference E-model's, as well as the predictive accuracies of the single-feature simple-regression WFL [383] and IQX [145] models. Experiments show that, in terms of mean absolute error MAE, the supervised learning models generally outperform E-model and even the full-reference PESQ, only one exception is observed with dataset 2. It also shows that there is no single ML model that outperforms

Table 19 Summary of supervised ML-based QoS/QoE correlation models

Ref.	ML	Application	Dataset	Features	Output	Evaluation	
	Technique	*(approach)*	*(availability)*			Settings	Results[ab]
Khan et al. [235, 236]	ANFIS	Impact of network and application-level QoS on MPEG4 video streaming over wireless mobile networks *(NR regression)*	Simulations with Evalvid and *ns* − 2 · MPEG4 video source · 3 video types · variable network conditions · mobile video streaming client · PSNR-generated MOS	Video type Application-related: frame rate, send bitrate network-level: link bandwidth, packet error rate	MOS	Number of features= 5 5-layer ANFIS-NN: fuzzy layer, product layer, normalized layer, defuzzy layer, total output layer	For slight/gentle/rapid motion video type: · RMSE= 0.15/0.18/0.56 · R^2 = 0.7/0.8/0.75 *(on normalized data)* Outperformed by a simple regression model [235]
Machado et al. [287]	MLP-NN	Impact of QoS and video features over QoE *(FR/NR regression)*	Simulations on Evalvid integrated to *NS* − 2 ·3 video types (slight, gentle, rapid motion) ·565 data points · MOS, PSNR, SSIM and VQM generated by Evalvid and the VQMT tool	Delay, jitter, total/I/P/B frame loss · not clear if type of video is considered	A model is created for each output · MOS · PSNR · SSIM · VQM	Number of features= 6 ∼ 7 (-,10,1) MOS-MLP (-,10,1) PSN-MLP (-,12,24,1) SSIM-MLP (-,10,1) VQM-MLP	MOS-MLP · MSE≈ 0.01 PSNR-MLP · MSE≈ 0.14 SSIM-MLP · MSE≈ 0.01 VQM-MLPMSE · MSE≈ 0.3 *(on normalized data)*
Mushtaq et al. [328]	DT, RF, NB, SVM, *k*-NN, and NN	Impact of QoS, video features and viewer features over QoE *(NR classification)*	Collected from streaming videos over QoS-controlled emulated network, and MOS collected from a panel of viewers	network-level: · delay, jitter, packet loss, etc. application-related: · resolution type of video: · motion complexity viewer-related: · gender, interest, etc.	MOS	Number of features= 9 *k*-NN ($k = 4$) Other settings *(N/A)*	RF · MAE= 0.136 · TP= 74.8% DT · MAE= 0.126 · TP= 74% NB · MAE≈ 0.23 · TP≈ 57% SVM · MAE= 0.26 · TP≈ 61% 4-NN · MAE≈ 0.2 · TP= 49% NN · MAE≈ 0.18 · TP≈ 65% *(on normalized data)*

Table 19 Summary of supervised ML-based QoS/QoE correlation models *(Continued)*

Ref.	ML	Application	Dataset	Features	Output	Evaluation	
	Technique	*(approach)*	*(availability)*			Settings	Results[ab]
MLQoE [89]	SVR, MLP-NN, DT, and GNB	modular user-centric correlation of QoE and network QoS metrics for VoIP services *(NR regression)*	3 datasets of VoIP sessions under different network conditions generated with OMNET++: during handover (dataset 1), in a network with heavy UDP traffic (dataset 2), in a network with heavy TCP traffic (dataset 3) QoE assessed with user-generated MOS and program-generated PESQ and E-model QoE	network-related: · delay, jitter, packet loss, etc.	MOS	Number of features= 10 MLP-NN $(10, 2 \sim 5, 1)$ Gaussian, linear, and polynomial kernel SVR	SVR · $\text{MAE}_1 = 0.66$ · $\text{MAE}_2 = 0.65$ · $\text{MAE}_3 = 0.47$ MLP-NN · $\text{MAE}_1 = 0.75$ · $\text{MAE}_2 = 0.68$ · $\text{MAE}_3 = 0.53$ DT · $\text{MAE}_1 = 0.73$ · $\text{MAE}_2 = 0.55$ · $\text{MAE}_3 = 0.5$ GNB · $\text{MAE}_1 = 0.69$ · $\text{MAE}_2 = 0.68$ · $\text{MAE}_3 = 0.53$ *(on normalized data)*
Dermibilek et al. [114]	RF, BG, and DNN	Correlation of QoE and network and application QoS metrics for video streaming services *(NR regression)*	INRS dataset, including user-generated MOS on audiovisual sequences encoded and transmitted with varying video and network parameters, and other pub *(public [112])*	network-related: delay, jitter, packet loss, etc. application-related: video frame rates, quantization parameters, filters, etc.	MOS	Number of features: ·RF_1, $\text{BG}_1 = 34$ ·RF_2, $\text{BG}_2 = 5$ ·DNN_{21}, $\text{DNN}_{22} = 5$ RF, BG tree size= 200 Number of hidden layers: ·$DNN_{21} = 1$ · DNN_{22} hidden= 20	RF_1 · RMSE= 0.340 · PCC= 0.930 RF_2 · RMSE= 0.340 · PCC= 0.930 BG_1 · RMSE= 0.345 · PCC= 0.928 BG_2 · RMSE= 0.355 · PCC= 0.925 DNN_{21} · RMSE= 0.403 · PCC= 0.909 DNN_{22} · RMSE= 0.437 · PCC= 0.894 *(on normalized data)*

[a] Average values. Results vary according to experimental settings

[b] Accuracy metrics: mean square error (MSE), mean absolute error (MAE), root MSE (RMSE), correlation coefficient (R), Pearson correlation coefficient (PCC)

Table 20 Summary of ML-based QoS/QoE prediction models for HAS and DASH

Ref.	ML Technique	Application	Dataset	Features	Output	Evaluation	
	(training)	*(approach)*	*(availability)*			Settings	Results[ab]
CS2P [432]	Supervised: HMM *(offline)*	Throughput prediction for midstream bitrate adaptation in HAS clients to improve the QoE for video streaming *(regression)*	iQIYI dataset consisting of 20 million sessions covering ·3 million unique clients IPs and ·18 server IPs ·87 ISPs	Throughput samples	Throughput 1 ∼ 10 periods ahead	HMM model per cluster of similar sessions: · Number of states= 6 · Number of samples= 100 SVM, GBR single model for all sessions: · Number of features=6	MAE= 7% *(on normalized data)* · up to 50% more accurate than SVR, GBR and HMM with no clustering · 3.2% improvement on overall QoE · 10.9% improved bitrate over MPC
Claeys et al. [102]	Reinforcement learning: Q-Learning *(online)*	Video quality adaptation in a HAS client to maximize QoE under varying network conditions *(rule extraction)*	ns-3 simulation based on TCP streaming sessions in Norway's Telenor 3G/HSDPA mobile wireless network dataset. *(public [384])*	State: · client buffer filling level · client throughput level Reward: QoE as function of · targeted quality level · span between current and targeted video quality level · rebuffering level	Finite action set of $N = 7$ possible video quality levels (sotmax selection)	Improvement compared to Microsoft MSS: ·9.12% higher estimated MOS ·16.65% lower standard deviation	$\cdot S = (N+1)\frac{B_{max}}{T_{seg}+1}$ $\cdot A = N$

[a] Average values. Results vary according to experimental settings

[b] Evaluation metrics: mean absolute error (MAE); *S*: number of state variables; *A*: number of possible actions per state

all others; while the SVR model has the lowest MAE with dataset 1 (0.66), DT achieves the best result with dataset 2 (0.55) and GBN with dataset 3 (0.43). MLQoE further outperforms the WFL-model and the IQX-model with a MAE improvement of 18 $\sim$ 42%. Indeed this motivates the need for a modular ML-based QoE prediction algorithm. However, further research could be pursued to study the correlation between the performance of the different ML-models and the way the QoS parameters evolve in each of the 3 datasets.

Another subset of ML techniques are considered by Demirbilek et al. [114] and used to develop no-reference models to predict QoE for audiovisual services. These techniques include: decision tree ensemble methods (RF and BG), and deep learning (DNN). Genetic programming (GP) is also considered and compared against the ML techniques. All models are trained and validated through 4 $\sim$ 10-fold cross-validation on the INRS dataset [113]. The dataset includes user-generated MOS on audiovisual sequences encoded and transmitted with varying video frame rates, quantization parameters, filters and network packet loss rates. 34 no-reference application- and network-level features are considered. Experiments with different feature sets show that, apart from the DNN model, all models perform better with the complete set of features, and hence do not require feature processing. On the contrary the DNN model performs better when trained only with 5 independent features, namely: video frame rate, quantization, noise reduction, video packet loss rate, and audio packet loss rate. Also, the one-hidden layer DNN model outperforms the model with 20 hidden layers in terms of RMSE (0.403 vs. 0.437) and PCC (0.909 vs. 0.894). The conducted experiments also show that all models perform quite well and that the RF model with complete set of features performs the best (lowest RMSE 0.340 and highest PCC 0.930). The video packet loss rate seems to be the most influential feature on the RF model. The model is further trained on other publicly available audiovisual datasets and still performs well. However it is not compared to the other models, which would be useful to confirm or infirm the supremacy of RF.

9.2 QoE prediction under QoS impairments

In [453], Vega et al. propose an unsupervised deep learning model based on Restricted Boltzmann Machines (RBMs) for real-time quality assessment of video streaming services. More precisely, the model is intended to infer the no-reference features of the received video from only a subset of those features that the client extracts in real-time fashion. 10 video-related features are extracted: one related to the bit stream, five to the frame, two to the inter-frame and the last two to the content. Network QoS parameters are not considered in the feature set, however the impact of the network conditions is studied in the paper based on two synthetic network-impaired video datasets, namely ReTRiEVED (for general condition networks) and LIMP (for extremely lossy networks). It is observed that the PCC between the VQM of the received video and the *bit rate* feature is the highest amongst the ten features, under network delay, jitter and throughput impairments. However, it is the *blur ratio* that correlates the most with VQM under severe packet loss condition. A discrepancy between video types was also recorded. This eventually motivated the need for one RBM model (with different feature set) per video type and network impairment, which raised the number of devised models to 32. Video-type and network-condition specific RBMs (with 100 hidden neurons) eventually shows a better performance than the single video-type and network-condition oblivious model on the ReTRiEVED dataset, according to the authors, which contradicts the results shown on the tables. Assuming that there is improvement, the practicality and overhead of the multi-RBMs solution are yet to be evaluated. In fact, delay, jitter, and throughput impairments are treated as if they were independent conditions and a condition-specific model is created. In practice, however impairments are correlated and happen together. Therefore, if the client has to assess the quality of the streamed video, it will also have to find out what impairment there is prior to selecting the appropriate predictor.

9.3 QoS/QoE prediction for HAS and DASH

Recently, the widespread adoption of HTTP adaptive streaming (HAS) drove increasing interest in developing QoE/QoS-aware HAS clients. Data-driven approaches, in particular ML, have been employed mainly in two different ways: *(1)* to predict changes in network QoS, namely throughput, and trigger adaptation mechanism to reduce rebuffering time [432], and *(2)* to select appropriate adaptation action [102].

It has been shown in recent work [432] that accurate throughput prediction can significantly improve the QoE for adaptive video streaming. ML has been widely used in throughput prediction in general as shown in Section 3. In the particular context of adaptive video streaming, Sun et al. propose in [432] the Cross Session Stateful Predictor (CS2P), a throughput prediction system to help with bitrate selection and adaptation in HAS clients. CS2P uses HMM for modeling the state-transition evolution of throughput, one model per session *cluster*, where sessions are clustered according to common features (e.g. ISP, region). The system is testing with a video provider (iQIYI) dataset consisting of 20 million sessions covering 3 million unique client IPs, 18 server IPs, and 87 ISPs. The HMM model is trained offline via the expectation maximization algorithm, and 4-fold cross-validation is used for tuning the number of states (6 states in total). Online prediction provides an estimate of the throughput

$1 \sim 10$ epochs ahead using maximum likelihood estimation. Throughput is continuously monitored and the model is updated online accordingly. Midstream throughput prediction experiments show that the model achieves 7% median absolute normalized prediction error ($\sim$ 20% 75th-percentile error) reducing the median prediction error by up to 50% compared to history-based predictors (last sample, harmonic mean, AR) as well as other ML-based predictors (SVR, GBR, and HMM trained on all sessions as opposed to the session cluster). It is also shown that CS2P achieves 3.2% improvement on overall QoE and 10.9% higher average bitrate over state-of-art Model Predictive Control (MPC) approach, which uses harmonic mean for throughput prediction. The authors claim that SVR and GBR perform poorly when trained on the session cluster. This might be due to the smaller size of the session cluster dataset, but requires further investigation.

In [102] (that extends [103] - [357]), a Q-learning-based HAS client is proposed to dynamically adjust to the current network conditions, while optimizing the QoE. Adaptation is assumed at the segment level; the quality level (e.g. bitrate) of the next video segment may go higher or lower depending on network conditions. States are defined as a combination of the client buffer filling level and throughput level. $B_{max}/T_{seg} + 1$ different buffer filling levels are considered where B_{max} denotes the maximum client buffer size in seconds, and T_{seg} the segment duration in seconds. Whereas $N + 1$ throughput levels are considered, ranging between 0 and the client link capacity, where N is the number of quality levels. The reward function to be maximized is a measure of the QoE, calculated on the basis of the targeted segment quality level, the span between the current and targeted quality level, and the rebuffering level (which may result in video freezes).

The model is trained and tested on $NS - 3$ with 10-min different video sequences (6 in total), split into 2sec segments each encoded at $N = 7$ different bitrates. The algorithm is trained for 400 episodes of streaming each of the video sequences over a publicly available $3G$ network bandwidth trace [384, 385]. The authors claim that the Q-learning client achieves in average 9.12% higher estimated MOS (program-generated), with 16.65% lower standard deviation, than the traditional Microsoft ISS Smooth Streaming (MSS) client. Similar performance is recorded when alternating between 2 video sequences every 100 streaming episodes. However, shifting to a random new video sequence after convergence time was not investigated.

9.4 Summary

Research in QoS/QoE provisioning has been leveraging ML for both prediction and adaptation, as shown in Tables 19 and 20. Clearly, research has been dominated by works on predicting QoE based on video-level and network-level features. As such, a number of different QoS/QoE correlation models have been proposed in the literature for different media types (e.g. voice, video and image) ranging from simple regression models to NNs, including SVM, DT, RF, etc. For each media type, different QoE assessment methods and metrics have been used (e.g. MOS, VQM), each with its own set of computational and operational requirements. The lack of a common, standard QoE measure makes it difficult to compare the efficiency of different QoS/QoE prediction and correlation models. In addition, there is a lack of a clear quantitative description of the impact of network QoS on QoE. This impact is poorly understood and varies from one scenario to another. While some find it sufficient to correlate the QoE to a single network QoS parameter [145, 240, 383, 408], e.g. delay or throughput, others argue that multiple QoS parameters impact the QoE and need to be considered in tandem as features in a QoE/QoS correlation model [89, 102, 114, 235, 287, 328, 432]. Still othe rs consider QoS as a confounding parameter and build different QoE assessment models for different network QoS conditions [453].

This motivates the need for an efficient methodology for QoE/QoS correlation, based on a combination of quantifiable subjective and objective QoS measures and outcomes of service usage. This calls for the identification of the influential factors of QoE for a given type of service and understanding their impact on user's expectation and satisfaction. QoE measures, such as MOS, and user engagement metrics are very sensitive to contextual factors. Though, contextual information undoubtedly influences QoE and is necessary to develop relevant QoE optimization strategies, it can raise privacy concerns.

Results depicted in Table 19 show that supervised learning techniques, such as NNs, SVR, DT and RF have consistent low MOS prediction errors. According to [89], RF is a better classifier than NN when it comes to predicting MOS. Table 20 also shows that using ML in HAS and DASH for prediction and adaptation, using supervised learning and RL, can improve QoE. However, this still needs to be validated in a real-world testbed.

10 Network security

Network security consists of protecting the network against cyber-threats that may compromise the network's availability, or yield unauthorized access or misuse of network-accessible resources. Undoubtedly, businesses are constantly under security threats [231], which not only costs billions of dollars in damage and recovery [227], but could also have a detrimental impact to their reputation. Therefore, network security is a fundamental cornerstone in network operations and management.

It is undeniable that we are now facing a cyber arms race, where attackers are constantly finding clever ways to attack networks, while security experts are developing new measures to shield the network from known attacks, and most importantly zero-day attacks. Examples of such security measures include:

– *Encryption of network traffic*, especially the payload, to protect the integrity and confidentiality of the data in the packets traversing the network.
– *Authorization using credentials*, to restrict access to authorized personnel only.
– *Access control*, for instance, using security policies to grant different access rights and privileges to different users based on their roles and authorities.
– *Anti-viruses*, to protect end-systems against malwares, e.g. Trojan horse, ransom-wares, etc.
– *Firewalls*, hardware or software-based, to allow or block network traffic based on pre-defined set of rules.

However, encryption keys and login credentials can be breached, exposing the network to all kinds of threats. Furthermore, the prevention capabilities of firewalls and anti-viruses are limited by the prescribed set of rules and patches. Hence, it is imperative to include a second line of defense that can detect early symptoms of cyber-threats and react quickly enough before any damage is done. Such systems are commonly referred to as Intrusion Detection/Prevention Systems (IDS/IPS). IDSs monitor the network for signs of malicious activities and can be broadly classified into two categories—Misuse- and Anomaly-based systems. While the former rely on signatures of known attacks, the latter is based on the notion that intrusions exhibit a behavior that is quite distinctive from normal network behavior. Hence, the general objective of anomaly-based IDSs is to define the "normal behavior" in order to detect deviations from this norm.

When it comes to the application of ML for network security, through our literature survey we have found that the majority of works have focused on the application of ML for intrusion detection. Here, intrusion detection refers to detecting any form of attacks that may compromise the network e.g. probing, phishing, DoS, DDoS, etc. This can be seen as a classification problem. While there is a body of work on host-based intrusion detection (e.g. malware and botnet detection), we do not delve into this topic, as most of these works utilize traces collected from the end-host (sometimes in correlation with network traces). Concretely, in our discussion, we focus on network-based intrusion detection and we classify the works into three categories, namely misuse, anomaly, and hybrid network IDSs.

Previous surveys [82, 161, 447] looked at the application of ML for cyber-security. However, [161, 447] cover the literature between 2000-2008, leaving out a decade of work. More recently, [82] looked at the application of Data Mining and ML for cyber-security intrusion detection. The proposed taxonomy consists of the different ML techniques with a sample of efforts that apply the corresponding technique. Our discussion is different, as we focus on ML-based approaches with a quantitative analysis of existing works (Tables 21, 22 23, 24 and 25). Furthermore, we survey efforts related to SDN and reinforcement learning, which have been recently published.

10.1 Misuse-based intrusion detection

Misuse-based IDSs consist of monitoring the network and matching the network activities against the expected behavior of an attack. The key component of such a system is the comprehensiveness of the attack signatures. Typically, the signatures fed to a misuse-IDS rely on expert knowledge [84]. The source of this knowledge can either be human experts, or it can be extracted from data. However, the huge volume of generated network traces renders manual inspection practically impossible. Furthermore, attack signatures extracted by sequentially scanning network traces will fail to capture advanced persistent threats or complex attacks with intermittent symptoms. Intruders can easily evade detection if the signatures rely on a stream of suspicious activities by simply inserting noise in the data.

In light of the above, ML became the tool of choice for misuse-based IDSs. Its ability to find patterns in big datasets, fits the need to learn signatures of attacks from collected network traces. Hence, it comes as no surprise to see a fair amount of literature [20, 84, 90, 252, 322, 344, 354, 402, 421] that rely on ML for misuse-detection. These efforts are summarized in Table 21. Naturally, all existing works employ supervised learning, and the majority perform the detection offline. Note, we classify all work that use normal and attack data in their training set as misuse-detection.

The earliest work that employed ML for misuse detection is [84]. It was among the first to highlight the limitations of rule-based expert systems, namely that they (i) fail to detect variants of known attacks, (ii) require constant updating, and (iii) fail to correlate between multiple individual instances of suspicious activities if they occur in isolation. Following the success of NN in the detection of computer viruses, the application of NN for misuse detection as an alternative to rule-based systems is proposed. The advantages of NN are its ability to analyze network traces in a less structured-manner (as opposed to rule-based systems), and to provide prediction in the form of a probability. The latter can enable the detection of variants of known attacks. For evaluation, training and testing dataset are generated using *RealSecure*^{TM}—a tool that monitors network data and compares it against signatures of known attacks. For attack dataset, *InternetScanner*^{TM} [368] and Satan Scanner [143] tools are used to generate port scans and syn-flood attacks on the monitored

Table 21 Summary of ML-based Misuse Detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Cannady [84]	Supervised NN (*offline*)	Normal: RealSecure Attack: [143, 368]	TCP, IP, and ICMP header fields and payload	-1 Layer MLP: 9, [a], 2 -Sigmoid function -Number of nodes in hidden layers determined by trial & error	DR: 89%-91% Training + Testing runtime: 26.13 hrs
Pfahringer [358]	Supervised Ensemble of C5 DTs (*offline*)	KDD Cup [257]	all 41 features	-Two-processor (2x300Mhz) -512M memory, 9 GB disc Solaris OS 5.6 -10-folds cross-validation	DR Normal: 99.5% DR Probe: 83.3% DR DoS: 97.1% DR U2R: 13.2% DR R2L: 8.4% Training: 24 h
Pan et al. [344]	Supervised NN and C4.5 DT (*offline*)	KDD Cup [257]	all 41 features	-29,313 training data records -111,858 testing data records -1 Layer MLP: 70-14-6 -NN trained until MSE = 0.001 or # Epochs = 1500 -Selected attacks for U2L and R2L -After-the-event analysis	DR Normal : 99.5% DR DoS: 97.3% DR Probe (Satan): 95.3% DR Probe (Portsweep): 94.9% DR U2R: 72.7% DR R2L: 100% ADR: 93.28% FP: 0.2%
Moradi et al. [322]	Supervised NN (*offline*)	KDD Cup [257]	35 features	-12,159 training data records -900 validation data records -6,996 testing data records -Attacks: SYN Flood and Satan -2 Layers MLP: 35 35 35 3 -1 Layer MLP: 35 45 35 -ESVM Method	2 Layers MLP DR: 80% 2 Layers MLP Training time > 25 hrs 2 Layers MLP w/ ESVM DR: 90% 2 Layers MLP w/ ESVM Training time < 5 hrs 1 Layers MLP w/ ESVM DR: 87%
Chebrolu et al. [90]	Supervised BN and CART (*offline*)	KDD Cup [257]	Feature Selection using Markov Blanket and Gini rule	-5,092 training data records -6,890 testing data records - AMD Athlon 1.67 GHz processor with 992 MB of RAM	DR Normal: 100% DR Probe: 100% DR DoS: 100% DR U2R: 84% DR R2L: 99.47% Training BN time: 11.03 ∼ 25.19 sec Testing BN time: 5.01 ∼ 12.13 sec Training CART time : 0.59 ∼ 1.15 sec Testing CART time: 0.02∼ 0.13 sec
Amor et al. [20]	Supervised NB (*offline*)	KDD Cup [257]	all 41 features	-494,019 training data records -311,029 testing data records -Pentium III 700 Mhz processor	DR Normal: 97.68% PCC DoS: 96.65% PCC R2L: 8.66% PCC U2R: 11.84% PCC Probing: 88.33%
Stein et al. [421]	Supervised C4.5 DT (*offline*)	KDD Cup [257]	GA-based feature selection	-489,843 training data records -311,029 testing data records -10-fold cross validation -GA ran for 100 generations	Error rate DoS: 2.22% Error rate Probe: 1.67% Error rate R2L: 19.9% Error rate U2R: 0.1%
Paddabachigari et al. [354]	Supervised Ensemble of SVM, DT, and SVM-DT *Offline*	KDD Cup [257]	all 41 features	5,092 training data records 6,890 testing data records AMD Athlon, 1.67 GHz processor with 992 MB of RAM -Polynomial kernel	DR Normal: 99.7% DR Probe:100% DR DoS: 99.92% DR U2R: 68% DR R2L: 97.16% Training time: 1∼ 19 sec Testing time: 0.03∼ 2.11 sec
Sangkatsanee et al. [402]	Supervised C4.5 DT (*online*)	Normal: Reliability Lab Data 2009 (RLD09) Attack: [341, 444, 475]	TCP, UPD, and ICMP header fields	-55,000 training data records -102,959 testing data records -12 features -2.83 GHz Intel Pentium Core2 Quad 9550 processor with 4 GB RAM and 100 Mbps LAN -Platform used: Weka V.3.6.0	DR Normal: 99.43% DR DoS: 99.17% DR Probe: 98.73% Detection speed: 2∼ 3 sec

Table 21 Summary of ML-based Misuse Detection *(Continued)*

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Miller et al. [314]	Supervised Ensemble MPML (*Offline*)	NSL-KDD [438]	all 41 features	-125,973 training records -22,544 testing records -3 NBs trained w/ 12, 9, 9 features -Platform used Weka [288]	TP: 84.137% FP: 15.863%
Li et al. [272]	Supervised TCM K-NN (*Offline*)	KDD Cup [257]	all 41 features 8 features selected using Chi-square	-Intel Pentium 4, 1.73 GHz, 1 GB RAM, Windows XP Professional - Platform Weka [288] -49,402 training records -12,350 testing records -$K = 50$	41 features: TP 99.7% 41 features: FP 0% 8 features: TP 99.6% 8 features: FP 0.1%

[a] Determined empirically, Mean Square Error (MSE), Percentage Correct Classification (PCC), Average Detection Rate (ADR), Early Stop Validation Method (ESVM)

host. Results show that the NN is able to correctly identify normal and attack records 89-91% of the time.

In 1999, the KDD cup was launched in conjunction with the KDD'99 conference. The objective of the contest was the design of a classifier that is capable of distinguishing between normal and attack connections in a network. A dataset was publicly provided for this contest [257], and since then became the primary dataset used in ML-based intrusion detection literature. It consists of 5 categories of attacks, including DoS, probing, user-to-root (U2R) and root-to-local (R2L), in addition to normal connections. The top three contestants employed DT-based solutions [421]. The winner of the contest [358] used an ensemble of 50 times 10 C5 DTs with a mixture of bagging and boosting [377]. The results of the proposed method are presented in Table 21. Clearly, the proposed approach performs poorly for U2R and R2L attack categories. The authors do mention that many of the decisions were pragmatic and encouraged more scientific endeavors. Subsequently, an extensive body of literature emerged for ML-based intrusion detection using the KDD'99 dataset, in efforts to improve on these results, where some use the winners' results as a benchmark.

For instance, Moradi et al. [322] investigate the application of NN for multi-class classification using the KDD'99 dataset. Specifically, the authors focused on DoS and probing attacks. As opposed to the work of [84], two NNs were trained: one with a single hidden layer and the second with two hidden layers, to increase the precision of attack classification. They leverage the *Early Stopping Validation Method* [366] to reduce training and validation time of the NN to less than 5 hours. As expected, the NN with 2 hidden layers achieves a higher accuracy of 91%, compared to the 87% accuracy of the NN with a single hidden layer.

Amor et al. [20] compare NB and DT also using KDD'99 dataset, and promote NB's linear training and classification times as a competitive alternative to DT. NB is found to be 7 times faster in learning and classification than DT. For whole attacks, DT shows a slightly higher accuracy over NB. However, NB achieves better accuracy for DoS, R2L, and probing attacks. Both NB and DT perform poorly for R2L and U2R attacks. In fact, Sabhnani and Serpen [398] expose that no classifiers can be trained successfully on the KDD dataset to perform misuse detection for U2R or R2L attack categories. This is due to the deficiencies and limitations of the KDD dataset rather than the inadequacies of the proposed algorithms.

The authors found via multiple analysis techniques that the training and testing datasets represent dissimilar hypothesis for the U2R and R2L attack categories; so if one would employ any algorithm that attempts to learn the signature of these attacks using the training dataset is bound to perform poorly on the testing dataset. Yet, the work in [344] reports surprisingly impressive detection accuracy for U2R and R2L. Here, a hybrid of BP NN with C4.5 is proposed, where BP NN is used to detect DoS and probing attacks, and C4.5 for U2R and R2L. For U2R and R2L only a subcategory of attacks is considered (yielding a total of 11 U2R connections out of more than 200 in the original dataset and $\sim$2000 out of more than 15000 for R2L connections). *After-the-event* analysis is also performed to feed C4.5 with new rules in the event of misclassification.

Other seminal works consider hybrid and ensemble methods for misuse detection [90, 354, 421]. The goal of ensemble methods is to integrate different ML techniques to leverage their benefits and overcome their individual limitations. When applied to misuse detection, and more specifically for the KDD'99 dataset, these work focused on looking at which ML technique works best for a class of connections. For instance, Peddabachigari et al. [354] propose an IDS that leverages an ensemble of DT, SVM with polynomial kernel based function, and hybrid DT-SVM to detect various different cases of misuse. Through

Table 22 Summary of ML for flow feature-based anomaly detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Kayacik et al. [232]	Unsupervised Hierarchical SOM (*Offline*)	KDD Cup [257]	6 TCP features	-494,021 training records -311,029 records in test set 1 -4,898,431 records in test set 2 -Platforms: SOM-Toolbox [12] & SOM PAK [250] -3-level SOM w/ # Epochs: 4000	DR Test-set 1: 89% FP Test-set 1: 4.6% DR Test-set 2: 99.7% FP Test-set 2: 1.7%
Kim et al. [242]	Supervised SVM (*Offline*)	KDD Cup [257]	selected using GA	Training set: kddcup.data.gz [257] Testing set: corrected.gz [257] -Detect only DoS attacks -10-fold cross validation -GA ran for 20 generations	DR w/ Neural Kernel: 99% DR w/ Radial Kernel:87% DR w/ Inverse Multi-Quadratic Kernel: 77%
Jiang et al. [220]	Unsupervised Improved NN (*Offline*)	KDD Cup [256, 257]	all 41 features	-40,459 training records -429,742 testing records -Cluster Radius Thresh *r*=[0.2-0.27]	DR DoS: 99.10%%99.15 DR Probe: 64.72%80.27% DR U2R: 25.49%60.78% DR R2L 6.34%8.67% DR new attacks: 32.44%42.12% FP: 0.05%1.30%
Zhang et al. [495]	Unsupervised Random Forests (*Offline*)	KDD Cup [257]	40 features labeled by service type	-4 datasets used with % of attack connections: 1%, 2%, 5%, 10% -Platform used: Weka [288]	1% attacks: FP: 1% DR: 95% 10% attacks: FP: 1% DR: 80%
Ahmed et al. [7]	Supervised Kernel Function (*Online*)	From Abilene backbone network	number of packets, number of individual IP flows	-2 timeseries binned at 5 min intervals -Timeseries dimensions = FxT -F = 121 flows, T = 2016 timesteps	T#1 DR: 21/34-30/34 FP:0-19 T#2 DR:28/44-39/44 FP:5-16
Shon et al. [411]	Unsupervised Soft-margin SVM and OCSVM (*Offline*)	KDD Cup [257] Data collected from Dalhousie U.	selected using GA	-SVM Toolkits [88, 396] -100,000 packets for training -1,000-1,500 packet for testing -GA run for 100 generations 3-cross fold validation	KDD w/ 9 attack types DR: 74.4% Dalhousie Dataset DR: 99.99% KDD w/ 9 attack types FN:31.3% Dalhousi Dataset FP:0.01%
Giacinto et al. [165]	Unsupervised Multiple Classifiers (*Offline*)	KDD Cup [257]	29 features for HTTP 34 features for FTP 16 features for ICMP 31 features for Mail 37 features for Misc 29 features for Private&Other	-494,020 training records -311,029 testing records -1.5% of data records is attacks	v-SVC DR: 67.31%-94.25% v-SVC FP: 0.91%-9.62%
Hu et al. [198]	Supervised Decision stumps with AdaBoost (*Offline*)	KDD Cup [257]	all 41 features	-494,021 training records -311,029 testing records -Pentium IV with 2.6-GHz CPU and 256-MB RAM -Platform used Matlab 7	DR: 90.04%-90.88% FP: 0.31%-1.79% Mean Training time: 73 sec
Muniyandi et al. [327]	Unsupervised K-Means, C4.5 DT (*Offline*)	KDD Cup [257]	all 41 features	-15,000 training records -2,500 testing records -Intel Pentium Core 2 Duo CPU 2.20GHz, 2.19GHz, 0.99GB of RAM w/ Microsoft Windows XP (SP2) -Platform: Weka 3.5 [288]	DR: 99.6% FP: 0.1% Precision: 95.6% Accuracy: 95.8% F-measure: 94.0%
Panda et al. [345]	Unsupervised RF, ND, END (*Offline*)	NSL-KDD [438]	all 41 features	-25,192 training instances -IBM PC of 2.66GHz CPU with 40GB HDD and 512 MB RAM -10-fold cross validation	TP: 99.5 FP: 0.1% F-measure: 99.7% Precision: 99.9% Recall 99.9% Time to build model: 18.13 sec
Boero et al. [64]	Supervised RBF-SVM (*Offline*)	Normal: from U. of Genoa Malwares: [126, 292, 348, 351]	7 SDN OpenFlow features	-RBF Complexity par: 20 -RBF kernel par: 2	Normal-TP: 86% Normal-FP: 1.6% Malware-TP: 98.4% Malware-FP: 13.8%

empirical evaluation, the resultant IDS consist of using DT for U2R, SVM for DoS, and and DT-SVM to detect normal traffic. The ensemble of the 3 methods together (with a voting mechanism) is used to detect probing and R2L attacks. The resultant accuracy for each class is presented in Table 21.

Table 23 Summary of ML for payload-based anomaly detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Zanero et al. [493]	Unsupervised A two-tier SOM-based architecture (*Offline*)	Normal: KDD Cup [257] Attack: Scans from Nessus [44]	Packet headers and payload	-2,000 training packets -2,000 testing packets -10x10 SOM trained for 10,000 epochs -Platform used: SOM toolbox [12]	Improves DR by 75% over 1-tiered S.O.M
Wang et al. [459]	Unsupervised Centroid model (*Offline*)	KDD Cup [257] & CUCS	Payload of TCP traffic	-2 weeks training data -3 weeks testing data -Inside network TCP data only -Incremental learning	DR w/ payload of a packet: 58.8% DR w/ first 100 bytes of a packet: 56.7% DR w/ last 100 bytes of a packet: 47.4% DR w/ all payloads of a con: 56.7% DR w/ first 1000 bytes of a Con: 52.6% Training time: 4.6-26.2 sec Testing time: 1.6-16.1 sec
Perdisci et al. [356]	Supervised Ensemble of single-class SVM (*Offline*)	Normal: KDD Cup [257] Normal: GATECH Attack: CLET [117] Attack: PBA [149] Generic [204]	Payload	-50% of dataset for training -50% of dataset for testing -11 OCSVM trained with 2_v-grams; v=1...10 -5-fold cross validation on KDD cup -7-fold cross validation on GATECH -2 GHz Dual Core AMD Opteron Processor and 8GB RAM	Generic DR w/ FP $10^{-}5$: 60% shell-code DR w/ FP $10^{-}5$: 90% CLET DR w/ FP $10^{-}5$: 90% Detection time KDD Cup: 10.92 ms Detection time GATECH: 17.11 ms
Gornitz et al. [171]	Supervised SVDD (*Online*)	Normal: from Fraunhofer Inst. Attack: Metasploit	payload	-2,500 training network events -1,250 testing network events -Active Learning -Fraction of Labeled data: 1.5%	DR: 96% FP: 0.0015%

Stein et al. [421] employ DT with GA. The goal of GA is to pick the best feature set out of the 41 features provided in KDD'99 dataset. DT with GA is performed for every category of attacks, rendering a total of 4 DTs. The average error rate achieved by each DT at the end of 20 runs is reported in Table 21. Another interesting ensemble learning approach is the one proposed in [90], where the ensemble is composed of pairs of feature set and classification technique. More specifically, BN and CART classification techniques are evaluated on the KDD'99 dataset with different feature sets. Markov blanket [353] and Gini [76] are adopted as feature selection techniques for BN and CART, respectively. Markov blanket identifies the only knowledge needed to predict the behavior of a particular node; a node here refers to the different categories of attacks. Gini coefficient measures how well the splitting rules in CART separates between the different categories of attacks. This is achieved by pruning away branches with high classification error. For BN, 17 features out of 41 are chosen during the data reduction phase. For CART, 12 variables are selected. CART and BN are trained on the 12 and 17 features set, as well as 19 features set from [326]. They describe the final ensemble method using pairs (#features, classification), which delineates the reduced feature set and the classification technique that exhibits the highest accuracy for the different categories of attacks and normal traffic. The ensemble model achieves 100% accuracy for normal (12 features set, CART), probe (17 features set, CART), and DoS (17 features set, Ensemble), and 84% accuracy for U2R (19 features set, CART), and 99.47% accuracy for R2L (12 features set, Ensemble).

Miller et al. [314] also devise an ensemble method but based on NB classifiers, denoted as Multi-perspective Machine Learning (MPML). The key idea behind MPML is that an attack can be detected by looking at different network characteristics or "perspective". These characteristics in turn are represented by a subset of network features. Hence, they group the features of a perspective together, and train a classifier using each feature set. The intuition behind this approach is to consider a diverse and rich set of network characteristics (each represented by a classifier), to enhance the overall prediction accuracy. The predictions made by each classifier are then fed to another NB model to reach a consensus.

A limitation of the aforementioned approaches is that they are all employed offline, which inhibits their application in real life. A few related works focused on the training and detection times of their IDS. Most classifiers (e.g., image, text recognition systems) require re-training from time to time. However, for IDSs this retraining may

Table 24 Summary of deep and reinforcement learning for intrusion detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Cannady et al. [85]	RL CMAC-NN (*Online*)	Prototype Application	Patterns of Ping Flood and UDP Packet Storm attacks	-3 Layers NN -Prototype developed w/ C & Matlab	Learning Error: 2.199-1.94^{-07}% New Attack Error:2.199-8.53^{-14}% Recollection Error: 0.038-3.28^{-05}% Error after Refinement: 1.24%
Servin et al. [407]	RL Q-Learning (*Online*)	Generated using NS-2	Congestion, Delay, and Flow-based	-Number of Agents: 7 -DDoS attacks only -Boltzmann's rules for E2	FP: 0-10% Accuracy:∼ 70%-∼ 99% Recall: ∼ 30%-∼ 99%
Li et al. [273]	DL DBN w/ Auto-Encoder (*Offline*)	KDD Cup [257]	all 41 features	-494,021 training records -311,029 testing records -Intel Core Duo CPU 2.10 GHz and 2GB RAM -Platform used: Matlab v.7.11 -3 Layers Encoder: 41,300,150,75,*	TPR: 92.20%-96.79% FPR: 1.58%-15.79% Accuracy: 88.95%-92.10% Training time: [1.147-2.625] sec
Alom et al. [14]	DL DBN (*Offline*)	NSL-KDD [438]	39 features	-25,000 training & testing records	DR w/ 40% data for training: 97.45% Training time w/ 40% data for training: 0.32 sec
Tang et al. [436]	DL DNN (*Offline*)	NSL-KDD [438]	6 basic features	-125,975 training records -22,554 testing records -3-Layers DNN: 6,12,6,3,2 -Batch Size: 10 # Epochs: 100 -Best Learning Rate: 0.001	Accuracy: 72.0%5-75.75% Precision: 79%-83% Recall: 72%-76% F-measure: 72%-75%
Kim et al. [245]	DL LSTM-RNN (*Offline*)	KDD Cup [257]	all 41 features	-1,930 training data records -10 test datasets of 5000 records -Intel Core I7 3.60 GHZ, RAM 8GB, OS Ubuntu 14.04 -# Nodes in Input Layer: 41 -# Nodes in Output Layer: 5 -Batch Size:50 #Epoch:500 -Best Learning Rate:0.01	DR: 98.88% FP: 10.04% Accuracy: 96.93%
Javaid et al. [213]	DL Self-taught Learning (*Offline*)	NSL-KDD [438]	all 41 features	-125,973 training records -22,544 testing records -10-fold cross validation	2-class TP: 88.39% 2-class Precision: 85.44% 2-class Recall: 95.95% 2-class F-measure: 90.4%

Table 25 Summary of ML for Hybrid Intrusion Detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Mukkamala et al. [325]	Supervised RBF-SVM (*Online*)	KDD cup [257]	all 41 features	7,312 training records -6,980 testing records -Platform used: SVMLight [224]	Accuracy: 99.5% Training time: 17.77 sec Testing Time: 1.63 sec
Zhang et al. [494]	Hybrid Hierarchical-RBF (*Online*)	KDD Cup	all 41 features	-32,000 training records -32,000 testing records	SHIDS Normal DR:=99.5% SHIDS Normal FP: 1.2% SHIDS Attack DR: [98.2%-99.3%] SHIDS Attack FP: [0%-5.4%] PHIDS level 1 DR: 99.8% PHIDS level 1 DR:1.2% PHIDS level 2 DR:[98.8%-99.7%] PHIDS level 2 FP:[0%-4%] PHIDS level 3 DR: 86.9% PHIDS level 3 FP: 0% Training time: 5 min
Depren et al. [116]	Hybrid SOM w./ J.48 (*Offline*)	KDD Cup	6 basic features for SOM all 41 features for J.48	-10-fold cross validation -Two-phases SOM Training -Phase 1 learning rate:0.6 -Phase 2 learning rate: 0.05 -Confidence Val. for J.48 pruning: 25%	DR: 99.9% Missed Rate: 0.1% FP: 1.25%

be performed daily (or even hourly) due to the fast and ever changing nature of cyber-threats [180]. Hence, fast training times are critical for an adaptable and robust IDS. [198] tackled the challenge of devising an IDS with fast training time using an Adaboost algorithm. The proposed algorithm consists of an ensemble of weak classifiers (decision stumps), where their decisions are then fed to a strong classifier to make the final decision. The fast training time achieved (of 73 s) is attributed to the use of weak classifiers. Another advantage of decision stumps is the ability to combine weak classifiers for categorical features with weak classifiers for continuous features, without any forced conversation as is typically done in most works. During the evaluation, a subset of attack types are omitted from the training set in order to evaluate the algorithm's ability to detect unknown attacks. While the reported accuracy is not significantly high (90%), the training time is promising for real-time deployment. Clearly, there is still a need for a model that can achieve fast training time, without sacrificing the detection accuracy.

Sangkatsanee et al. [402] propose a real-time misuse-based IDS. Information gain is applied to reduce the number of features used (for faster detection), resulting in 12 features. Different ML techniques were assessed, among which DT provided the best empirical results. They developed a tool that runs on traces collected in 2 s time intervals, and shows a detection accuracy of 98%. A post-processing technique is also proposed to reduce FP, which consists of flagging an attack only if 3-out-of 5 consecutive records belonging to the same connection were classified as an attack. While this work is indeed promising, given it is performed in real-time, it suffers from a few limitations: (i) it can only detect two types of attacks (DoS and probe), (ii) it is not compared against other real-time signature-based IDS (e.g. Snort [87]), (iii) it only looks at attacks in windows of 2 s, and (iv) its post-processing approach correlates records between 2 IPs, making it vulnerable to persistent threats and distributed attacks.

A final effort that merits a discussion here is [272]. This work employs Transductive Confidence Machine for *k*-NN (TCM-KNN), a supervised classification algorithm with a strangeness measure. A high strangeness measure indicates that the given instance is an outlier in a particular class (for which the measurement is being conducted). The strangeness measure is calculated for every instance against each possible classification class. This is achieved by measuring the ratio of the sum of the k-nearest distances from a given class to the sum of the *k*-nearest distances from all other classes. The strangeness measure is also employed for active learning. Since getting labeled data for attacks is a cumbersome task, active learning can relieve part of this tedious process by indicating the subset of data points that should be labeled to improve the confidence of the classifier. TCM-KNN is evaluated over the KDD'99 dataset and the results are reported in Table 21. The benefits of active learning is also evaluated. Starting with a training set of just 12 instances, TCM-KNN requires the labeling of an additional 40 actively selected instances to reach a TP of 99.7%. Whereas, random sampling requires the labeling of 2000 instances to attain the same accuracy.

10.2 Anomaly-based intrusion detection

Though misuse-based IDSs are very successful at detecting known attacks, they fail to identify new ones. Network cyber-threats are constantly changing and evolving, making it crucial to identify "zero-day" attacks. This is where anomaly-based intrusion detection comes in. Anomaly-based IDS models normal network behavior, and identify anomalies as a deviation from the expected behavior. A big issue with anomaly-based IDSs is false alarms, since it is difficult to obtain a complete representation of normality. ML for anomaly detection has received significant attention, due to the autonomy and robustness it offers in learning and adapting profiles of normality as they change over time. With ML, the system can learn patterns of normal behavior across environments, applications, group of users, and time. In addition, it offers the ability to find complex correlations in the data that cannot be deduced from mere observation. Though anomaly detection can be broadly divided into flow feature or payload-based detection, recently, deep learning and reinforcement learning are being aptly exploited. Primarily, this is due to their intrinsic ability to extrapolate data from limited knowledge. We delineate and summarize the seminal and state-of-the-art ML-based techniques for anomaly detection in Tables 22, 23 and 24.

10.2.1 Flow feature-based anomaly detection

Flow-based anomaly detection techniques rely on learning the expected (benign) network activities from flow features. The immediate observation in contrast to misuse detection is the application of unsupervised learning and hybrid supervised/unsupervised learning. Some works employed supervised learning for anomaly detection as well. The main difference is instead of teaching the model the expected behavior, in unsupervised learning the model is fed with an unlabeled training set to find a structure, or a hidden pattern, in the data. In anomaly detection, the notion is that benign network behavior is more common and will naturally group together, whereas, anomalous behavior is more sparse and will appear as outliers in the dataset. Hence, the larger and more dense clusters will indicate normal connections, while the smaller more distant data points (or clusters of data points) will indicate malicious behavior. A quick glance at Tables 22, 23, and 24 will reveal that the KDD'99 dataset is the dataset of choice in most anomaly-based intrusion detection literature, where some have also employed the improved version of the dataset, NSL-KDD [438] released in 2009. In the sequel, we elucidate the most influential work in the application of flow feature-based ML for anomaly detection.

We start-off our discussion by looking at supervised learning techniques. KOAD [7] is an online kernel function-based anomaly detection IDS. The key feature of KOAD is its ability to model normal behavior in face of variable traffic characteristics. It leverages a real-time anomaly detection algorithm that incrementally constructs and maintains a dictionary of input vectors defining the region of normal behavior. This dictionary is built using time series of the number of packets and IP flows. In the evaluation, the authors use a dataset collected by monitoring 11 core routers in the Abilene backbone network for a week. It comprises of two multi-variate time series, the number of packets and the number of individual IP flows. KOAD is evaluated against PCA and One-Class Neighbor Machine (OCNM). In packet time series, OCNM flags 26 out of 34 anomalies but generates 14 FPs, while KOAD gives different TP and FP under different parameters. For instance, it can detect 30 anomaly records with 17 FPs, and 26 anomaly records with 1 FP. However, PCA can detect 25 anomalies with 0 FP. On the other hand, for the flow-count time series, KOAD outperforms PCA and OCNM in terms of detection rate but at the cost of a higher FP.

More recently, Boero et al. [64] leverage a SVM with radial basis function kernel (RBF-SVM) to devise an IDS for SDN-based malware detection. A reduced feature set is evaluated based on features that are collectible via OF and commercial SDN switches. This limits the number of features to 7 consisting of the number of packets, number of bytes, flow duration, byte rate, packet rate, length of the first packet, and average packet length. The dataset used for evaluation consists of normal traffic traces from a university campus and malware traffic traces from [126, 292, 348, 351]. For a dataset with known attacks, both RBF-SVM with limited and all features return a TP above 98% for the malware traces, while TP of RBF-SVM is 86.2% for normal traces.

However, detecting new attacks using the RBF-SVM with limited and full features achieve comparable TP with a high FP of approximately 18% for normal traces. This shows that restricting the features set to those that can be collected via SDN switches slightly impacts the TP rate; however it comes at a cost of a higher FP. Hence there is a need to enlarge the features set that SDN switches monitor and collect. As we will see in the following, the battle between FP and TP will constantly resurface throughout our discussion. This is expected since guaranteeing the ground truth is difficult and requires manual labeling. Furthermore, obtaining a complete representation of normal behavior is extremely challenging. Thus, any future legitimate behavior that was not part of the trained set might be flagged as an anomaly.

The main application of unsupervised learning for anomaly detection is clustering on the basis that normal data connections will create larger more dense clusters. Jiang et al. [220] challenge this notion by showcasing that the size of the cluster is not sufficient to detect anomalies and has to be coupled with the distance of the cluster from other clusters, to increase accuracy of detection. To this end, the authors propose an Improved Nearest Neighbor (IMM) technique for calculating cluster radius threshold. The KDD dataset is used for evaluation and shows that IMM outperforms three related works [131, 139, 363] in terms of detection rate and FP. A snippet of their reported results is presented in Table 22.

Kayacik et al. [232] leverage unsupervised NN with SOM and investigate their detection capabilities when trained with only 6 of the most basic TCP features, including protocol type, service type, status flag, connection duration, and total bytes sent to destination/source host. They evaluate their work on the KDD dataset, and observe that SOM-based anomaly detection achieves an average DR (ADR) of 89% with FP in the range of [1.7%-4.6%]. Other interesting applications of unsupervised learning for anomaly detection is RF [495] and an ensemble of single-class classifiers [165]. Giacinto et al. [165] train a single-class classifier, based on ν-SVC [405], for each individual protocol and network service; e.g. ICMP, HTTP, FTP, and Mail. This ensures that each classifier is specialized in detecting normal and abnormal characteristics for one these protocols and services. The application of one-class classifier is particularly interesting for cases where there is a skewness in the data. This is in-line with the fact that normal traffic traces are more common than malicious network activities. Thus, the one-class classifier learns the behavior of the dominant class, and dissimilar traffic patterns are then flagged as an anomaly. Results of the evaluation can be found in Table 22.

The majority of works in anomaly-based IDS employed a hybrid of supervised/unsupervised learning techniques. Panda et al. [345] evaluate several hybrid approaches to identify the best combination of supervised and unsupervised data filtering and base classifiers for detecting anomalies. The authors evaluate DT, PCA, stochastic primal estimated sub-gradient solver for SVM (SPegasos), ensembles of balanced nested dichotomies (END), Grading, and RF. They show that RF with nested dichotomies (ND) and END achieve the best results, with a detection rate of 99.5% and a FP of 0.1%. It is also the fastest in terms of performance, requiring 18.13 s to build and provides F-measure, precision , and recall of 99.7%, 99.9, and 99.9%, respectively. Enhanced SVM [411] combines a supervised version of SVM: soft-margin SVM with an unsupervised version: one-class SVM. The intuition here is that this combination will allow to get the best of both worlds: low FP with ability to detect zero-day attacks. Enhanced SVM consists of 4 phases:

- Create a profile of normal packets using Self-organized Feature Map.
- Packet filtering scheme, using *p0f* [491], based on passive TCP/IP fingerprinting to reject incorrectly formed TPC/IP packets.
- GA to perform feature selection
- Temporal correlation of packets during packet processing

Enhanced-SVM is only trained with normal traffic. The normal to abnormal ration in the data set consists of 98.5-99 to 1-1.5%. Compared to two commercial IDSs, Bro and Snort, the Enhanced-SVM slightly improves in anomaly detection accuracy on a real dataset with *unknown* traffic traces. However, for known attacks, Snort and Bro significantly outperform Enhanced-SVM.Wagner et al. [456] also leverage a hybrid supervised and unsupervised single-class SVM to detect anomalies in IP NetFlow records. A new kernel function is proposed to measure the similarity between two windows of IP flow records of n seconds. The hybrid SVM is evaluated on a normal dataset obtained from an ISP, with synthetically generated attacks using Flame [74], and with $n = 5$ s. Results show that the hybrid SVM can achieve an ADR of 92%, FP in the range [0-0.033], and TN in the range [0.967-1]. Finally, Muniyandi et al. [327] propose a hybrid anomaly detection mechanism that combines k-Means with C4.5 DT. They build k clusters using k-Means and employ DT for each cluster. DT overcomes the forced assignment problem in k-Means, where k is too small and a class dominates due to skewed dataset. The authors evaluate the hybrid detection on the KDD dataset and show that it outperforms k-Means, ID3, NB, k-NN, SVM, and TCM-KNN, over 6 different metrics, including TP, FP, precision, accuracy, F-measure, and ROC. However, TCM-KNN achieves better results in terms of TPR and FPR.

10.2.2 Payload-based anomaly detection

Payload-based anomaly detection systems learn patterns of normality from packet payload. This provides the ability to detect attacks injected inside the payload that can easily evade flow feature-based IDSs. In this subsection, we discuss ML techniques that have been employed to detect anomalies using packet payload alone or in conjunction with flow features.

PAYL [459] use the *1-gram* method to model packet payloads. n-gram is widely used for text analysis. It consists of a sliding window of size n that scans the payload while counting the occurrence/frequency of each n-gram. In addition to counting the frequency of each byte in the payload, the mean and the standard deviation is computed. As the payload exhibits different characteristics for different services, PAYL generates a payload model for each service, port, direction of payload, and payload length range. Once the models are generated, Mahalanobis distance is used to measure the deviation between incoming packets and the payload models. The larger the distance, the higher the likelihood that the newly arrived packet is abnormal. The authors leverage incremental learning to keep the model up to date, by updating the Mahalanobis distance to include new information gathered from new packets. PAYL's ability to detect attacks on TCP connections is evaluated using the KDD dataset and data traces collected from Columbia University Computer Science (CUCS) web server. PAYL is able to detect 60% of the attacks on ports 21 and 80 with a FP of 1%. However, it performs poorly when the attacks target applications running on ports 23 and 25. This is due to the fact that attacks on ports 21 and 80 exhibit distinctive patterns in the format of the payload, making them easier to detect than attacks on ports 23 and 25. PAYL can be used as an unsupervised learning technique under the assumption that malicious payloads are a minority, and will have a large distance to the profile than the average normal samples. Hence, by running the learned model on the training set, malicious packets in the set can be detected, omitted, and then the models are retrained on the new training set.

Perdisci et al. [356] design Multiple-Classifier Payload-based Anomaly Detector (McPAD) to infer shell and polymorphic shell code attacks. Shell code attacks inject malicious executable code in the packet payload. As opposed to 1-gram analysis performed by PAYL, McPAD runs a 2_{ν}-gram analysis technique to model the payload ($\nu = [0 - 10]$). It measures the occurrence of a pair of bytes that are ν positions apart. By varying ν and applying feature reduction, different compact representations of the payload are obtained. Each of these representations is then fed to a 1-class classifier model and majority vote is used to make the final prediction. For evaluation, normal traffic is extracted from two datasets: the 1st week of KDD dataset and 7 weeks of HTTP traffic collected from College of Computing School at the Georgia Tech (GATECH). Attack traffic is collected from a generic dataset in [204], in addition to synthetically generated polymorphic attacks [117] and Polymorphic Blending Attacks (PBAs). In comparison to PAYL, McPAD achieves a DR of 60, 80 and 90% for generic, polymorphic CLET, and shell-code attacks, respectively, with an FP of 10^{-5} for all attacks. While, PAYL reports very low DRs for the same FP. However, the computational overhead of McPAD is much higher than that of PAYL with an average processing time of 10.92 ms over KDD and 17.11 ms over GATECH whereas PAYL runs in 0.039 ms and 0.032 ms, respectively.

Zanero et al. [493] propose a two-tier architecture for anomaly detection. The first tier consists of an unsupervised outliers detection algorithm that classifies each packet. This tier provides a form of feature reduction as the result of the classification "compresses" each packet into a single byte of information. The results from the first tier are fed into the second tier anomaly detection algorithm. In the first tier, both packet header and payload are used for outliers detection. The authors compare three different techniques, including SOM, Principal Direction Divisive Partitioning (PDDP) algorithm and k-Means, with SOM outperforming PDDP and k-Means in terms of classification accuracy with a reasonable computational cost. A preliminary prototype that combines a first tier SOM with a second tier SOM is evaluated over the Nessus [44] vulnerabilities scans. The results show a 75% improvement in DR over an IDS that does not include the first tier.

Gornitz et al. [171] leverage semi-supervised Support Vector Data Description (SVDD) and active learning to build the active SVDD (ActiveSVDD) model for payload-based anomaly detection. It is first trained with unlabeled examples, and subsequently refined by incorporating labeled data that has been queried by active learning rules. The empirical evaluation consists of comparing an unsupervised SVDD with random sampling against ActiveSVDD. The dataset used for the evaluation is HTTP traffic recorded within 10 days at Fraunhofer Institute. Attack data is generated using Metasploit [307] framework. In addition, mimicry attacks are added in the form of cloaked data to evaluate the ability to detect adversarial attacks. The model achieve high accuracy, with random sampling for online applications with cloaked data, 96% DR with a very low FP and 64% DR for ActiveSVDD and SVDD, respectively.

10.3 Deep and reinforcement learning for intrusion detection

As we contemplate the applications of ML for misuse and anomaly detection, we observe that all applications of NN were restricted to networks with at most 2 hidden-layers. DNNs are attractive for the ability to train large NNs with several hidden-layers. As we survey the literature on DL for intrusion detection, we will observe much larger and deeper NNs in terms of number of nodes in each layer, and the number of hidden layers. Conceptually, the results of DNNs get better with more data and larger models.

10.3.1 Deep learning for anomaly detection

Over the past decade, anomaly detection has particularly benefited from self-taught learning (STL) [213], DBN [14, 273], and RNN [245]. Once more, all these works have been evaluated using KDD dataset, and its enhanced version NSL-KDD [438] dataset. Their results are summarized in Table 24.

In 2007, STL [378] emerged as an improvement over semi-supervised learning. STL uses unlabeled data from other, but relevant, object class to enhance a supervised classification task e.g. using random unlabeled images from the Internet to enhance the accuracy of a supervised classification task for cat images. This is achieved by learning a good feature representation from the unlabeled data and then applying this representation to the supervised classifier. The potential benefit of STL for anomaly detection is clear: intrusion detection suffers from the lack of sufficient amount of labeled data, more specifically for attacks. To this extent, the work in [213] explore the application of STL for anomaly detection. Their proposed model consists of two stages, an Unsupervised Feature Learning (UFL) stage using sparse auto encoder, followed by a classification stage that uses the learned features with soft-max regression (SMR). They evaluate their solution using the NSL-KDD Cup dataset for 2-class and 5-class classifications, and compare against a SMR technique that is not preceded by a UFL stage. The 2-class classification achieves a higher accuracy of 88.39% compared to 78.06% of SMR, and outperforms SMR with respect to recall and F-measure. However, SMR outperforms STL in precision.

Li et al. [273] and Alom et al. [14] explore the use of DBN for anomaly detection. DBN is an interesting class of NN, when trained using unlabeled data it works as a features selector, and when trained with labeled data it acts as a classifier. In [273] DBN is used to perform both of these two tasks. More specifically, an auto-encoder is first used for dimensionality reduction. The proposed DBN is composed of multi-layers of RBM and a layer of BP NN. Unsupervised training is performed on every layer of RBM and the final output is fed to the BP NN for classification. Pre-training and pre-tuning the DBN with auto-encoder over 10 iterations, result in an accuracy of 92.10%, FP of 1.58% and TP of 92.20%. DBN without auto-encoder achieves an accuracy, FP, and TP of 91.4, 9.02, and 95.34%, respectively.

In [14], the authors perform a compare analysis to evaluate the performance of DBN (composed of two-layers RBM) against against SVM, and a hybrid DBN-SVM. This comparative analysis was performed using the NSL-KDD dataset. The results show that DBN runs in 0.32 s, and achieves an accuracy of 97.5% when trained with only 40% of the NSL-KDD dataset, outperforming SVM and DBN-SVM. This exceeds the performance, with respect to training time of similar existing work [400]. In contrast, Tang et al. [436] use DNN for a flow-based anomaly detection in SDNs. They extract six features from the SDN switches and evaluate the accuracy of anomaly detection using the NSL-KDD Cup dataset. As the learning rate is varied, the DNN achieves an accuracy, precision, recall, and F-measure in the range [72.05-75.75%], [79-83%], [72-76%], and [72-75%] , respectively. It is important to note that for the highest learning rate, DNN achieves the highest accuracy on the training dataset. However, its accuracy, recall and F-measure on the test datasets drops. The authors note that such an accuracy drop occurs with a high learning rate since the model becomes trained “too accurately”, i.e. over-fitting. Nevertheless, the accuracy of the DNN is lower than the winner of the KDD Cup, RF, which has an accuracy of 81.59%.

10.3.2 Reinforcement learning for intrusion detection (RL)

MARL [407] is a Multi-Agent Reinforcement Learning system for the detection of DoS and DDoS attacks. MARL is based on Q-learning, and the system consists of a set of heterogeneous sensor agents (SA) and a hierarchy of decision agents (DA). In the proposed setup three SAs and 1 DA are used. Each SA is responsible of collecting either congestion, delay, or flow-based network metrics. These collected metrics represent the local state of every SA. Every SA runs a local RL mechanism to match its local state to a particular communication action-signal. These signals are received by the DA, which given a global view of the state of the network triggers a final action signal that is forwarded to a human in the loop. If the DA (or the higher-layer agent in case of a hierarchy of DAs) makes the appropriate call, all the agents in the system are rewarded. Otherwise, they will all be penalized. MARL is evaluated on NS-2 with 7 nodes, where two nodes generate normal FTP and UDP traffic and one generates the UDP attacks. The remaining four nodes constitute the SA agents and a single DA agent. There is a single baseline run and seven tests are conducted, where each test differs in the normal traffic, attack patterns, or both. The corresponding accuracy, recall, and FPR for each test is presented in Table 24. MARL is also tested on a dataset that contains mimicry attacks, it achieved a recall and accuracy of $\sim 30\%$ and $\sim 70\%$. When little change is inflicted in the traffic pattern, MARL can achieve high 99% accuracy and recall with 0 FP.

A less conventional application of RL is [85], which consists of an online IDS based on adaptive NN with modified RL. Here RL consists of a feedback mechanism. The focus is to detect DoS attacks using Cerebellar Model Articulation Controller (CMAC) NN. The learning algorithm incorporates feedback from the protected system in the form of system state (i.e. response rate, heartbeat). The objective is to leverage the system state to assist in detecting the attacks earlier since the responsiveness of the system reduces under attack. The authors evaluate CMAC NN using a prototype application that simulates ping flooding and UDP packet storm attacks. First, they assess the system’s ability to autonomously learn attacks. They find that when the system is trained with gradual ping flood attack vectors, the error rate is 2.199%, which reduces to $1.94^{-7}\%$ as the training progresses. The authors also evaluate the system’s ability to learn new attacks and recognize learned attacks. The error rate results are presented in Table 24. Finally, the benefit of the system’s feedback mechanism illustrates that as attacks progress, the system state’s responsiveness approaches 0 and the error rate reaches $8.53^{-14}\%$.

10.4 Hybrid intrusion detection

We conclude our survey of ML for intrusion detection by looking at hybrid IDSs that apply both misuse and anomaly-based intrusion detection. Such a hybrid system can make the best of both worlds i.e. high accuracy in detecting patterns of known attacks, along with the ability to detect new attacks. Every time a new attack is detected, it can then be fed to the misuse-detection system to enhance the comprehensiveness of its database. We start off our discussion by looking at the work of Depren et al. [116] that leverages J.48 DT and SOM for misuse and anomaly detection, respectively. Three SOM modules are trained, one for each of the TCP, UDP and ICMP traffic. The output of the misuse and anomaly detection modules are combined using a simple decision support system, that raises an alarm if either one of the modules detect an attack. The authors evaluate their work over the KDD Cup dataset and find that their hybrid IDS achieves a DR of 99.9% with a missed rate of 0.1% and a FP of 1.25%.

Similarly, Mukkamala et al. [325] compare a SVM-based with an NN-based hybrid misuse and anomaly detection models. Their models are trained with normal and attack data and evaluated using the KDD Cup dataset. The SVM-based hybrid model achieves 99.5% accuracy with training and testing times of 17.77 s and 1.63 s, respectively. While, three different NNs are trained and tested, each with a different structure of hidden layers. The three NNs achieve an accuracy of 99.05, 99.25, and 99%, respectively, with a training time of 18 min. Therefore, SVM outperforms NN, slightly in accuracy and significantly in runtime.

Zhang et al. [494] develop a hierarchical IDS framework based on RBF to detect both misuse and anomaly attacks, in real-time. Their hierarchical approach is modular and decreases the complexity of the system. It enables different modules to be retrained separately, instead of retraining the entire system. This is particularly useful in the event of a change that only affects a subset of the modules. Serial hierarchical IDS (SHIDS) is compared against a parallel hierarchical IDS (PHIDS). SHIDS begins by training a classifier with only normal data and as the classifier detects abnormal packets, it logs them in a database. *c*-Means clustering [58] groups the data based on their statistical distributions, and as the number of attack records in the largest group exceeds a pre-defined threshold, a new classifier is trained with that specific attack data and appended to the end of the SHIDS. PHIDS on the other hand consists of three layers. The anomaly and misuse classifiers are in the first two layers, while the third layer is dedicated to different attack categories. Over time, the data in each attack category is updated as new attacks are identified. The performance of RBF is evaluated using the KDD dataset against a Back-Propagation learning algorithm (BPL). Though, BPL achieves a higher DR for misuse detection, RBF has a smaller training time of 5 min compared to 2 h for BPL. Training time is critical for online IDSs. Further, when training the model with just normal data for anomaly detection, RBF outperforms BPL for each attack category, with respect to DR and FP. Overall, RBF achieves a DR of 99.2% and FP of 1.2%, compared to BPL with a DR of 93.7% and FP of 7.2%. The evaluation of SHIDS and PHIDS are in Table 25.

10.5 Summary

Our survey on the application of ML for network security focused on network-based intrusion detection. We grouped the work into misuse, anomaly, and hybrid network IDSs. In each category, we expose the different ML techniques that were applied, including recent applications of DL and RL. One clear take-away message is the significant benefit that ML has brought to misuse-based intrusion detection. It has really improved on the rule-based systems, and allowed the extraction of more complex patterns of attacks from audit data. It even allowed the ability to detect variants of known attacks. In the field of misuse-detection, a preference is given to white-box models (e.g. DT) as their decision rules can be extracted, as opposed to black-box models (e.g. NN). Ensemble-based methods were also heavily employed by training ML models on different subsets of the dataset or with different feature sets. Ensemble-based methods have been particularly useful in achieving very fast training time.

While the benefits of ML for IDS is clear, there is a lot of speculation on the application of ML for anomaly detection. Despite the extensive literature on ML-based anomaly detection, it has not received the same traction in real deployments [415]. Indeed, the most widely deployed IDS (Snort [45]) is in fact misuse-based [101]. The main culprit for this aversion is not only the susceptibility of anomaly detection to high FPs, but also the high-cost of misclassification in the event of FNs. Compared to the cost of misclassification in an ads recommender system, a missed malicious activity can bring down the system or cause a massive data breach. Another main weakness that we observe in the literature is that most works consist of raising an alarm if an anomaly is detected without giving any hints or leads on the observed malicious behavior (e.g. the attack target). Providing such semantics can be extremely valuable to network analysts [415], and even in reducing FP.

The dataset of choice in the majority of the surveyed literature has been based on KDD'99, an out-dated dataset. On one hand, this has provided the community with the ability to compare and contrast different methods and techniques. On the other hand, it does not reflect the recent more relevant types of attacks. Moreover, even the normal connection traces represent basic applications (e.g. email and file-transfer) without any inclusion to more recent day-to-day applications that swarms the network (e.g. social media and video streaming). This is further aggravated by the several limitations and flaws reported about this dataset [438]. Indeed, there is a dire need for a new dataset for intrusion detection.

To conclude, most works on the application of ML for intrusion detection are offline, and amongst the few real-time IDSs, there is no consideration for early detection (i.e. detecting a threat from the first few packets of a flow). Moreover, there is a gap in the ML for intrusion detection literature with regards to intrusion detection for persistent threats, or correlating among isolated anomaly instances over time. Finally, only a handful of works have actually evaluated the robustness of their algorithm in the event of mimicry attacks, an aspect of critical importance as attackers are constantly looking for ways to evade detection.

11 Lessons learned, insights and research opportunities

We have discussed the existing efforts in employing ML techniques to address various challenges and problems in networking. The success of ML primarily lies in the availability of data, compounded with improved and resilient ML algorithms to solve complex problems. Future networks are envisaged to support an explosive growth in traffic volume and connected devices with unprecedented access to information. In addition, these capabilities will have to be achieved without significantly increasing CAPEX, OPEX or customer tariffs.

In order to be sustainable in a competitive environment, network operators must adopt efficient and affordable deployment, operations and management. Enabling technologies for future networks include SDN, network slicing, NFV, and multi-tenancy, which reduce CAPEX, increase resource utilization and sharing. Similarly, autonomic network management frameworks coupled with SDN is envisioned to reduce OPEX. The aforementioned technologies will allow future networks to host a wide variety of applications and services, and a richer set of use cases, including massive broadband, ultra low latency and highly reliable services, machine to machine communications, tactile Internet, industrial applications, autonomous vehicles, real-time monitoring and control.

In this subsection, we describe and delineate prominent challenges and open research opportunities pertaining to the application of ML in current and future networks, from the network, system and knowledge acquisition perspectives.

11.1 Network perspective

11.1.1 Cost of predictions

The accuracy of network monitoring data comes at the cost of increased monitoring overhead (e.g. consumed network bandwidth and switch memory). This raises the need for network monitoring schemes that are both accurate and cost-effective. Monitoring applications in traditional networks rely on a predefined set of monitoring probes built into the hardware/firmware, which limits their flexibility. With SDN customizable software-based monitoring probes can be deployed on-demand to collect more diverse monitoring data. However, in many instances, e.g. monitoring traffic volume over a given switch interface, these probes need to operate at line rate, which is very expensive over very high speed links and difficult to achieve in software. This makes TSF-based approaches for traffic prediction prohibitive.

Recently, two solutions have been investigated in order to overcome this issue, (i) traffic sampling and interpolation [274], and (ii) leveraging features other than traffic volume for traffic prediction [365]. Indeed, various flow sampling techniques (stochastic/deterministic, spacial/temporal, etc.) to reduce monitoring overhead have been proposed in the literature. Unfortunately, the current ML-based solution proposed in [274], is not conclusive and shows contradicting prediction accuracy results. Instead, Poupart et al. [365] use classifiers to identify elephant flows. Indeed, classifiers operate at a coarser granularity. Therefore, their accuracy can not be compared to the accuracy of regression model operating on the same set of features. Using features other than traffic volumes for accurate traffic prediction remains an open research direction.

11.1.2 Cost of errors and detailed reports

ML for anomaly detection has received significant interest in the literature, without gaining traction in the industry. This is primarily due to the high FPR [27, 415], making them inapplicable in an operational setting. FPRs waste expensive analyst time to investigate the false alarms, and reduce the trust and confidence in the IDS. Another major concern with anomaly detection techniques is the lack of detailed reports on detected anomalies [415]. Typically, a flag is raised and an alarm is triggered whenever there is a deviation from the norm. An efficient IDS is not only responsible for detecting attacks and intrusions in the network, it must provide a detailed log of anomalies for historical data collection and model retraining.

11.1.3 Complexity matters

When performing traffic prediction, classification, routing and congestion control on intermediate nodes in the network, it is crucial that they consume less time and computing resources to avoid degradation in network performance. This requirement is non-trivial, especially, in resource-constrained networks, such as WANETs and IoT. Though, performance metrics for ML evaluation are well-defined, it is difficult to evaluate the complexity of ML-based approaches a priori. Unlike traditional algorithms, the complexity of ML algorithms also rely on the size and quality of data, and the performance objectives. The issue is further exacerbated, if the model is adaptive and relearning is intermittently triggered due to varying network conditions over time. The traditional complexity metrics fail to cover these aspects. Therefore, it is important to identify well-rounded evaluation metrics that will help in assessing the complexity of given ML techniques, to strike a trade-off between performance improvement and computational cost.

11.1.4 ML in the face of the new Web

In an effort to improve security and QoE for end-users, new application protocols (e.g. HTTP/2 [48], SPDY [47], QUIC [211]) have emerged that overcome various limitations of HTTP/1.1. For instance, HTTP/2 offers payload encryption, multiplexing and concurrency, resource prioritization, and server push. Though, the WEB applications over HTTP/2 enjoy the benefits of these enhancements, it further complicates traffic classification by introducing unpredictability in the data used for ML. For example, if we employ flow feature-based traffic classification, the feature statistics can be skewed, as several requests can be initiated over the same TCP connection and responses can be received out of order. Therefore, the challenge lies in exploring the behavior and performance of ML techniques when confronted with such unpredictability in a single TCP connection and even parallel TCP connections [293] in HTTP/2. Similarly, prioritization requested by different WEB clients diminish the applicability of a generic ML-based classification technique for identifying WEB applications.

11.1.5 Rethinking evaluation baseline

Often, proposed ML-based networking solutions are assessed and evaluated against existing non-ML frameworks. These latter act as baseline and are used to demonstrate the benefits, if any, of using ML. Unfortunately, these baseline solutions are often deprecated and outdated. For instance, ML-based congestion control mechanisms are often compared against default TCP implementations, e.g. CTCP, CUBIC, or BIC with typical loss recovery mechanisms, such as Reno, NewReno, or SACK. However, Yang et al. [486] applied supervised learning techniques to identify the precise TCP protocol used in Web traffic and uncovered that though majority of the servers employ the default, there is a small amount of web traffic that employs non-default TCP implementation for congestion control and loss recovery. Therefore, it is critical to consider TCP variants as comparison baselines that have taken the lead, and are prominently employed for congestion control and loss recovery.

ML-based congestion control mechanisms should be designed and evaluated under the consideration that the standard TCP is no longer the de facto protocol, and current networks implement heterogeneous TCP protocols that are TCP-friendly. Furthermore, it is a good practice to consider TCP variants, particularly enhanced for specific network technologies, such as TCP-FeW for WANETs and Hybla for satellite networks. ML-based approaches, such as Learning-TCP [29] and PCC [122], have already taken these considerations into account and provide an enhanced evaluation of their proposed solutions. Therefore, it is imperative to design a standardized set of performance metrics for enabling a fair comparison between various ML-based approaches to different problems in networking.

11.1.6 RL in face of network (in)stability and QoS

There are various challenges in finding the right balance between exploration of and exploitation in RL. When in comes to traffic routing, various routes must be explored before the system can converge to the optimal routing policy. However, exploring new routes can lead to performance instability and fluctuation in network delay, throughput and other parameters that impact QoS. On the other hand, exploiting the same "optimal" route to forward all the traffic may lead to congestion and performance degradation, which would also impact the QoS. Different avenues can be explored to overcome these challenges. For example, increasing the learning rate can help detect early signs of performance degradation. While, load balancing can be achieved with selective routing, which can be implemented by assigning different reward functions to different types of flows (elephant vs. mice, ToS, etc.). Furthermore, instability-awareness at exploration time can be implemented by limiting the scope of the routes to explore those with highest rewards. Indeed, this requires an in-depth study to gauge the impact of such solutions on network performance and their convergence time to optimal routing.

Another direction worth pursuing is to correlate the reward function of an RL-based routing to a desired level of QoS. This involves finding ways to answer questions, such as, which reward function can guarantee that the delay in the network does not exceed a given threshold? or, given a reward function, what would be the expected delay in the network?

11.1.7 Practicality and applicability of ML

Benchmarks used in the literature for the training and validation of proposed ML-based networking solutions are often far from being realistic. For instance, ML-based admission control mechanisms, are based on simulations that consider traffic from only a small set of applications or services. Furthermore, they disregard diversity of QoS parameters when performing admission control. However, in practice, networks carry traffic from heterogeneous applications and services, each having its own QoS requirements, with respect to throughput, loss rate, latency, jitter, reliability, availability, and so on. Hence, the optimal decision in the context of a simulated admission control mechanism may not be the optimal for a practical network. Furthermore, often synthetic network datasets are used in training and validation. Although, ML models perform well in such settings, their applicability in practical settings remains questionable. Therefore, more research is needed to develop practical ML-based network solutions.

11.1.8 SDN meets ML

Though, there has been a growing interest in leveraging ML to realize autonomic networks, there is little evidence of its application to date. Prohibiting factors include the distributed control and vendor-specific nature of legacy network devices. Several technological advances have been made in the last decade to overcome these limitations. The advent of network softwarization and programmability through SDN and NFV offers centralized control and alleviates vendor lock-in.

SDN can facilitate adaptive and intelligent network probing. Probes are test transactions that are used to monitor network behavior and obtain measurements from network elements. Finding the optimal probe rate will be prohibitively expensive in future networks, due to the large number of devices, the variety of parameters to measure, and the small time intervals to log data. Aggressive probing can exponentially increase the amount of traffic overhead resulting in network performance degradation. In contrast, conservative probing may have the risk of missing some significant anomalies or critical network events. Hence, it is imperative to adapt probing rates that keep traffic overhead within a target value, while minimizing performance degradation. SDN can leverage ML techniques to offer the perfect platform to realize adaptive probing. For example, upon predicting a fault or detecting an anomaly, the SDN controller can probe suspected devices at a faster rate. Similarly, during network overload, the controller may reduce the probing rate and rely on regression to predict the value of the measured parameters.

11.1.9 Virtualization meets ML

Due to the anticipated rise in the number of devices and expansion in network coverage, future networks will be exposed to a higher number of network faults and security threats. If not promptly addressed, such failures and, or attacks can be detrimental, as a single instance may affect many users and violate the QoS requirements of a number of applications and services. Thus, there is a dire need for an intelligent and responsive fault and security management framework. This framework will have to deal with new faults and attacks across different administrative and technological domains within a single network, introduced by concepts of network slicing, NFV, and multi-tenancy. For instance, any failure in the underlying physical resource can propagate to the hosted virtual resources, though the reverse is not always true. Hence, it will be nearly impossible for traditional approaches to locate the root cause or compromised elements of the fault or an attack, in such a complex network setting.

On the other hand, ML-based approaches on fault and security management focus mostly on single tenant in single layer networks. To develop the fault and security management framework for future networks, existing ML-based approaches need to be extended or re-engineered to take into account the notion of multi-tenancy in multi-layer networks. Due to the versatility of the problem, DNN can be explored to model complex multi-dimensional state spaces.

11.1.10 ML for smart network policies

The unprecedented scale and degree of uncertainty in future networks will amplify the complexity of traffic engineering tasks, such as congestion control, traffic prediction, classification, and routing. Although ML-based solutions have shown promising results to address many traffic engineering challenges, their time complexity needs to be evaluated with the envisioned dynamics, volume of data, number of devices and stringent applications requirements in future networks. To address this, *smart* policy-based traffic engineering approaches can be adopted where operators can efficiently and quickly apply adaptive traffic engineering policies. Policy-based traffic classification using SDN has shown promising results in the treatment of QoS requirements based on operator-engineered policies [334]. Incorporating ML to assist in developing and extracting adaptive policies for policy-based traffic engineering solutions, remains rather unexplored. One possible avenue is to apply RL for generating policies for traffic engineering in future networks.

11.1.11 ML in support of autonomy

Networks are experiencing a massive growth in traffic, and will continue to grow even faster with the advent of IoT devices, tactile Internet, virtual/augmented reality, high definition media delivery, etc. Furthermore, Cisco reports that there is a substantial difference between busy hour and average Internet traffic, such that in 2016, the busy hour Internet traffic increased by 51% in comparison to the 32% growth in average Internet traffic [99]. Such difference is expected to grow further in the next half a decade, where Cisco predicts that the growth rate of busy hour traffic will be almost 1.5 times that of average Internet traffic.

To accommodate such dynamic traffic, network operators can no longer afford the CAPEX for static resource provisioning as per the peak traffic requirements. Therefore, network operators must employ dynamic resource allocation that can scale based on the varying traffic demand. ML is an integral part of dynamic resource allocation that enables demand prediction, facilitates proactive provisioning and release of network resources. In addition, contextual information can be leveraged by ML to anticipate exceptional resource demand and reserve emergency resource in highly volatile environments.

Networks are also experiencing an exponential growth in terms of the number and diversity of supported applications and services. These have stringent and heterogeneous QoS requirements, in terms of latency, jitter, reliability, availability and mobility. It is likely that network operators may not only be unaware of all the devices in their network but also unconscious of all the applications and their QoS requirements. Therefore, it is challenging to devise efficient admission control and resource management mechanisms with limited knowledge. Existing works have demonstrated that both admission control and resource management can be formulated as learning problems, where ML can help improve performance and increase efficiency. A further step would be to explore if admission control and resource management strategies can be learned directly from network operation experience. Considering the intricate relationship between network experience and management strategies, DL can be leveraged to characterize the inherent relationship between inputs and outputs of a network.

11.2 System perspective

11.2.1 Support for adaptive, incremental learning in dynamic network environments

Networks are dynamic in nature. Traffic volume, network topology, and security attack signatures, are some of the many aspects that may change, often in an unexpected and previously unobserved way. Thus, it is fundamental to constantly retrain the ML model to account for these changes. Most ML models are trained offline. Retraining a model from scratch can be computationally intensive, time consuming, and prohibitive. The ability to retrain the model as new data is generated is fundamental to achieve fast incremental learning, which remains an open research direction. Indeed incremental learning comes with special system needs. In the particular case of RL applied to routing in SDN, a number of simulations are required before the model can converge to the optimal observation-to-action mapping policy. Every time a new flow is injected in the network, the SDN controller is required to find the optimal routing policy for that flow, and a number of simulations are performed as changes are observed in the link status. This calls for a system that fully exploits data and model parallelism to provide millisecond-level training convergence time.

11.2.2 Support for secure learning

ML is prone to adversarial attacks [39], also known as mimicry attacks, that aim to confuse learning. For instance, when employing ML for intrusion detection, an adversarial attack can trick the model into misclassifying malicious events as benign by poisoning the training data. Hence, it is fundamental to train robust ML models that are capable of detecting mimicry attacks. An interesting initiative worth mentioning is Cleverhans [346], a useful library that allows to craft adversarial examples. It provides training datasets that can be used to build robust ML models, capable of distinguishing legitimate datasets from poisoned ones, in the particular area of image recognition. There is indeed an urgent need for a system capable of generating adversarial use cases to be used in training robust models. Secure learning also demands a system that protects the training data from leakage and tampering, enforces privacy, data confidentiality and integrity, and support the secure sharing of data across domains.

11.2.3 Architectures for ML-driven networking

Modern networks generate massive volumes of different types of data (e.g. logs, traffic flow records, network performance metrics, etc.). At 100's of Gbps, even with high sampling rates, a single large network infrastructure element can easily generate hundreds of millions of flow records per day. Recently, the availability of massive data drove rapid advancement in computer hardware and software systems, for storage, processing and analytics. This is evidenced by the emergence of massive-scale datacenters, with tens of thousands of servers and EB storage capacity, the widespread deployment of large-scale software systems like Hadoop MapReduce and Apache Spark, and the increasing number of ML and in particular deep learning libraries built on top of these systems, such as Tensor-Flow, Torch, Caffe, Chainer, Nvidia's CUDA and MXNet. Mostly open-source, these libraries are capable of scaling out their workloads on CPU clusters enabled by specialized hardware, such as GPUs and TPUs.

GPUs are anticipated to be a key enabler for the next generation SDN [166, 465]. GPU-accelerated SDN routers are reported to have a much improved packet processing capability. Furthermore, the GPUs on SDN controllers may be particularly useful for executing ML and DL algorithms for learning various networking scenarios, and acting according to the acquired knowledge. On the other hand, smaller, resource constrained, smart networked devices, are more likely to benefit from a cloud-edge ML system. A cloud-edge ML system would leverage the large processing and memory resources, robust networks, and massive storage capabilities of the cloud for training computationally intensive models and sharing these with edge devices. Data collection and analytics that require immediate or near-immediate response time would be handled by edge devices. Light-weight ML software systems, such as Caffe2Go and TensorFlowLite, would eventually enable edge devices to by-pass the cloud and build leaner models locally.

11.3 Knowledge perspective

11.3.1 Lack of real-world data

As we surveyed the literature, we observed that numerous works relied on synthetic data, particularly in resource and fault management, network security, and QoE/QoS correlation. Synthetic datasets are usually simplistic and do not truly reflect the complexity of real-world settings. This is not surprising, since obtaining real-world data traces is difficult due to the critical and private nature of network traffic, especially the payload. Furthermore, establishing the ground truth is particularly challenging, given the voluminous amount of traffic making any manual inspection intractable. Although injecting faults and, or attacks in the network can help produce the required data as adopted by [285], it is unrealistic to jeopardize a production network for the sake of generating training data. Such limitations increase the probability of ML techniques being ill-trained and inapplicable in real-world network settings. Thus, it remains unclear how the numerous works in the literature would perform over real data traces. Therefore, a combined effort from both academia and industry is needed, to create public repositories of data traces annotated with ground truth from various real networks.

11.3.2 The need for standard evaluation metrics

As we survey existing works, it became apparent that comparing them within each networking domain is not possible. This is due to the adoption of non-standardized performance metrics, evaluation environments, or datasets [109]. Furthermore, even when the same dataset is adopted, different portions of the data are used for training and testing, thereby inhibiting any possibility for comparative analysis. Standardization of metrics, data, and environment for evaluating similar approaches is fundamental to provide the ability to contrast and compare the different techniques, and evaluate their suitability for different networking tasks. To fulfill this need, standard bodies such as the Internet Engineering Task Force (IETF), can play a pivotal role by promoting standardization of evaluation procedures, performance metrics, and data formats through Requests for Comments (RFCs).

11.3.3 Theory and ML techniques for networking

As the compute and data storage barriers that thwarted the application of ML in networking are no longer an issue, *what is now preventing an ML-for-networking success story as in games, vision and speech recognition?* Lack of a theoretical model is one obstacle that ML faces in networking. This concern was raised by David Meyer during his talk at IETF97 on machine intelligence and networking [308]. Without a unified theory, each network has to be learned separately. This could truly hinder the speed of adoption of ML in networking. Furthermore, the currently employed ML techniques in networking have been designed with other applications in mind. An open research direction in this realm is to design ML algorithms tailored for networks [306]. Another key issue is the lack of expertise, that is, ML and networking are two different fields, and there is currently a scarcity in the number of people that are experts in both domains. This mandates more cross-domain collaborations involving experts from both networking and ML communities.

12 Conclusion

Over the past two decades, ML has been successfully applied in various areas of networking. This survey provides a comprehensive body of knowledge on the applicability of ML techniques in support of network operation and management, with a focus on traffic engineering, performance optimization and network security. We review representative literature works, explore and discuss the feasibility and practicality of the proposed ML solutions in addressing challenges pertaining to the autonomic operation and management of future networks.

Clearly, future networks will have to support an explosive growth in traffic volume and connected devices, to provide exceptional capabilities for accessing and sharing information. The unprecedented scale and degree of uncertainty will amplify the complexity of traffic engineering tasks, such as congestion control, traffic prediction, classification, and routing, as well as the exposure to faults and security attacks. Although ML-based solutions have shown promising results to address many traffic engineering challenges, their scalability needs to be evaluated with the envisioned volume of data, number of devices and applications. On the other hand, existing ML-based approaches for fault and security management focus mostly on single-tenant and single-layer networks. To develop the fault and security management framework for future networks, existing ML approaches should be extended or re-architected to take into account the notion of multi tenancy in multi layer networks.

In this survey, we discuss the above issues along with several other challenges and opportunities. Our findings motivate the need for more research to advance the state-of-the-art, and finally realize the long-time vision of autonomic networking.

Acknowledgments
We thank the anonymous reviewers for their insightful comments and suggestions that helped us improve the quality of the paper. This work is supported in part by the Royal Bank of Canada, NSERC Discovery Grants Program, the Quebec FRQNT postdoctoral research fellowship, the ELAP scholarship, and the COLCIENCIAS Scholarship Program No. 647-2014, Colombia.

Authors' contributions
All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details
[1]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. [2]Department of Telematics, University of Cauca, Popayan, Colombia.

Received: 3 January 2018 Accepted: 4 May 2018
Published online: 21 June 2018

References

1. Aben E. NLANR PMA data. 2010. https://labs.ripe.net/datarepository/data-sets/nlanr-pma-data. Accessed 27 Dec 2017.
2. Ackley DH, Hinton GE, Sejnowski TJ. A learning algorithm for boltzmann machines. Cogn Sci. 1985;9(1):147–69.
3. ACM Special Interest Group on KDD. KDD cup archives. 2016. http://www.kdd.org/kdd-cup. Accessed 22 Nov 2017.
4. Adams R. Active queue management: A survey. IEEE Commun Surv Tutor. 2013;15(3):1425–76.
5. Adda M, Qader K, Al-Kasassbeh M. Comparative analysis of clustering techniques in network traffic faults classification. Int J Innov Res Comput Commun Eng. 2017;5(4):6551–63.
6. Adeel A, Larijani H, Javed A, Ahmadinia A. Critical analysis of learning algorithms in random neural network based cognitive engine for lte systems. In: Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st. IEEE; 2015. p. 1–5.
7. Ahmed T, Coates M, Lakhina A. Multivariate online anomaly detection using kernel recursive least squares. IEEE; 2007. pp. 625–33.
8. Ahn CW, Ramakrishna RS. Qos provisioning dynamic connection-admission control for multimedia wireless networks using a hopfield neural network. IEEE Trans Veh Technol. 2004;53(1):106–117.
9. Aizenberg IN, Aizenberg NN, Vandewalle JP. Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications. Norwell: Kluwer Academic Publishers; 2000.
10. Akan OB, Akyildiz IF. ATL: an adaptive transport layer suite for next-generation wireless internet. IEEE J Sel Areas Commun. 2004;22(5):802–17.

11. Albus JS. A new approach to manipulator control: the cerebellar model articulation controller (cmac). J Dyn Syst Meas. Control. 1975;97:220–7.
12. Alhoniemi E, Himberg J, Parhankangas J, Vesanto J. S.o.m toolbox; 2000. http://www.cis.hut.fi/projects/somtoolbox/. Accessed 28 Dec 2017.
13. Allman M, Paxson V, Blanton E. TCP Congestion Control. RFC 5681, Internet Engineering Task Force. 2009. https://tools.ietf.org/html/rfc5681.
14. Alom MZ, Bontupalli V, Taha TM. Intrusion detection using deep belief networks. In: Aerospace and Electronics Conference (NAECON), 2015 National. IEEE; 2015. p. 339–44.
15. Alpaydin E. Introduction to Machine Learning, 2nd ed. Cambridge: MIT Press; 2010.
16. Alpaydin E. Introduction to Machine Learning, 3rd ed. Cambridge: MIT Press; 2014.
17. Alshammari R, Zincir-Heywood AN. Machine learning based encrypted traffic classification: Identifying ssh and skype. In: Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on. IEEE; 2009. p. 1–8.
18. Alsheikh MA, Lin S, Niyato D, Tan HP. Machine learning in wireless sensor networks: Algorithms, strategies, and applications. IEEE Commun Surv Tutor. 2014;16(4):1996–2018.
19. Amaral P, Dinis J, Pinto P, Bernardo L, Tavares J, Mamede HS. Machine learning in software defined networks: Data collection and traffic classification. In: Network Protocols (ICNP), 2016 IEEE 24th International Conference on. IEEE; 2016. p. 1–5.
20. Amor NB, Benferhat S, Elouedi Z. Naive bayes vs decision trees in intrusion detection systems. In: Proceedings of the 2004 ACM symposium on Applied computing. ACM; 2004. p. 420–4.
21. Arndt D. HOW TO: Calculating Flow Statistics Using NetMate. 2016. https://dan.arndt.ca/nims/calculating-flow-statistics-using-netmate/. Accessed 01 Aug 2017.
22. Arouche Nunes BA, Veenstra K, Ballenthin W, Lukin S, Obraczka K. A machine learning framework for tcp round-trip time estimation. EURASIP J Wirel Commun Netw. 2014;2014(1):47.
23. Aroussi S, Bouabana-Tebibel T, Mellouk A. Empirical QoE/QoS correlation model based on multiple parameters for VoD flows. In: Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE; 2012. p. 1963–8.
24. Arroyo-Valles R, Alaiz-Rodriguez R, Guerrero-Curieses A, Cid-Sueiro J. Q-probabilistic routing in wireless sensor networks. In: Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on. IEEE; 2007. p. 1–6.
25. Astrom K. Optimal control of markov processes with incomplete state information. J Mathl Anal Appl. 1965;10(1):174–205.
26. Auld T, Moore AW, Gull SF. Bayesian neural networks for internet traffic classification. IEEE Trans neural networks. 2007;18(1):223–39.
27. Axelsson S. The base-rate fallacy and the difficulty of intrusion detection. ACM Trans Inf Syst Secur (TISSEC). 2000;3(3):186–205.
28. Ayoubi S, Limam N, Salahuddin MA, Shahriar N, Boutaba R, Estrada-Solano F, Caicedo OM. Machine learning for cognitive network management. IEEE Commun Mag. 2018;1(1):1.
29. Badarla V, Murthy CSR. Learning-tcp: A stochastic approach for efficient update in tcp congestion window in ad hoc wireless networks. J Parallel Distrib Comput. 2011;71(6):863–78.
30. Badarla V, Siva Ram Murthy C. A novel learning based solution for efficient data transport in heterogeneous wireless networks. Wirel Netw. 2010;16(6):1777–98.
31. Bakhshi T, Ghita B. On internet traffic classification: A two-phased machine learning approach. J Comput Netw Commun. 2016;2016:2016.
32. Bakre A, Badrinath BR. I-tcp: Indirect tcp for mobile hosts. In: Proceedings of the 15th International Conference on Distributed Computing Systems. ICDCS '95. Washington: IEEE Computer Society; 1995. p. 136.
33. Balakrishnan H, Seshan S, Amir E, Katz RH. Improving tcp/ip performance over wireless networks. In: Proceedings of the 1st Annual International Conference on Mobile Computing and Networking. MobiCom '95. New York: ACM; 1995. p. 2–11.
34. Balakrishnan H, Padmanabhan VN, Seshan S, Katz RH. A comparison of mechanisms for improving tcp performance over wireless links. IEEE/ACM Trans Netw. 1997;5(6):756–69.
35. Baldo N, Zorzi M. Learning and adaptation in cognitive radios using neural networks. In: Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE. IEEE; 2008. p. 998–1003.
36. Baldo N, Dini P, Nin-Guerrero J. User-driven call admission control for voip over wlan with a neural network based cognitive engine. In: Cognitive Information Processing (CIP), 2010 2nd International Workshop on. IEEE; 2010. p. 52–6.
37. Baras JS, Ball M, Gupta S, Viswanathan P, Shah P. Automated network fault management. In: MILCOM 97 Proceedings. IEEE; 1997. p. 1244–50.
38. Barman D, Matta I. Model-based loss inference by tcp over heterogeneous networks. In: Proceedings of WiOpt 2004 Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks. Cambridge; 2004. p. 364–73.
39. Barreno M, Nelson B, Sears R, Joseph AD, Tygar JD. Can machine learning be secure? In: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ACM, ASIACCS '06. New York: ACM; 2006. p. 16–25.
40. Barreto GA, Mota JCM, Souza LGM, Frota RA, Aguayo L. Condition monitoring of 3g cellular networks through competitive neural models. IEEE Trans Neural Netw. 2005;16(5):1064–75.
41. Baum LE, Petrie T. Statistical inference for probabilistic functions of finite state markov chains. Ann Math Statist. 1966;37(6):1554–63.
42. Bay SD, Kibler D, Pazzani MJ, Smyth P. The uci kdd archive of large data sets for data mining research and experimentation. SIGKDD Explor Newsl. 2000;2(2):81–5.
43. Bayes M, Price M. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s. Philos Trans. 1763;53(1683-1775): 370–418.
44. Beale J, Deraison R, Meer H, Temmingh R, Walt CVD. Nessus network auditing. Burlington: Syngress Publishing; 2004.
45. Beale J, Baker AR, Esler J. Snort: IDS and IPS toolkit. Burlington: Syngress Publishing; 2007.
46. Bellman R. Dynamic Programming, 1st ed. Princeton: Princeton University Press; 1957.
47. Belshe M, Peon R. SPDY Protocol. Tech. rep., Network Working Group. 2012. https://tools.ietf.org/pdf/draft-mbelshe-httpbis-spdy-00.pdf.
48. Belshe M, Peon R, Thomson M. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, IETF. 2015. http://www.rfc-editor.org/info/rfc7540.txt.
49. Bengio Y, Lamblin P, Popovici D, Larochelle H. Greedy layer-wise training of deep networks. In: Proceedings of the 19th, International Conference on Neural Information Processing Systems. NIPS'06. Cambridge: MIT Press; 2006. p. 153–60.
50. Benson T. Data Set for IMC 2010 Data Center Measurement. 2010. http://pages.cs.wisc.edu/tbenson/IMC10_Data.html. Accessed 28 Dec 2017.
51. Bergstra JA, Middelburg C. Itu-t recommendation g. 107: The e-model, a computational model for use in transmission planning: ITU; 2003.
52. Bermolen P, Rossi D. Support vector regression for link load prediction. Comput Netw. 2009;53(2):191–201.
53. Bermolen P, Mellia M, Meo M, Rossi D, Valenti S. Abacus: Accurate behavioral classification of P2P-tv traffic. Comput Netw. 2011;55(6): 1394–411.
54. Bernaille L, Teixeira R. Implementation issues of early application identification. Lect Notes Compu Sci. 2007;4866:156.
55. Bernaille L, Teixeira R, Akodkenou I, Soule A, Salamatian K. Traffic classification on the fly. ACM SIGCOMM Comput Commun Rev. 2006a;36(2):23–6.
56. Bernaille L, Teixeira R, Salamatian K. Early application identification. In: Proceedings of the 2006 ACM CoNEXT Conference. ACM; 2006b. p. 61–6:12.
57. Bernstein L, Yuhas CM. How technology shapes network management. IEEE Netw. 1989;3(4):16–9.
58. Bezdek JC, Ehrlich R, Full W. Fcm: The fuzzy c-means clustering algorithm. Comput Geosci. 1984;10(2-3):191–203.
59. Bhorkar AA, Naghshvar M, Javidi T, Rao BD. Adaptive opportunistic routing for wireless ad hoc networks. IEEE/ACM Trans Netw. 2012;20(1): 243–56.
60. Biaz S, Vaidya NH. Distinguishing congestion losses from wireless transmission losses: a negative result. In: Proceedings 7th International Conference on Computer Communications and Networks (Cat. No.98EX226). Piscataway: IEEE; 1998. p. 722–31.
61. Bkassiny M, Li Y, Jayaweera SK. A survey on machine-learning techniques in cognitive radios. IEEE Commun Surv Tutor. 2013;15(3): 1136–59.

62. Blanton E, Allman M. On making tcp more robust to packet reordering. SIGCOMM Comput Commun Rev. 2002;32(1):20–30.
63. Blenk A, Kalmbach P, van der Smagt P, Kellerer W. Boost online virtual network embedding: Using neural networks for admission control. In: Network and Service Management (CNSM), 2016 12th International Conference on. Piscataway: IEEE; 2016. p. 10–8.
64. Boero L, Marchese M, Zappatore S. Support vector machine meets software defined networking in ids domain. In: Proceedings of the 29th International Teletraffic Congress (ITC), vol. 3. New York: IEEE; 2017. p. 25–30.
65. Bojovic B, Baldo N, Nin-Guerrero J, Dini P. A supervised learning approach to cognitive access point selection. In: GLOBECOM Workshops (GC Wkshps), 2011 IEEE. Piscataway: IEEE; 2011. p. 1100–5.
66. Bojovic B, Baldo N, Dini P. A cognitive scheme for radio admission control in lte systems. In: Cognitive Information Processing (CIP), 2012 3rd International Workshop on. Piscataway: IEEE; 2012. p. 1–3.
67. Bojovic B, Quer G, Baldo N, Rao RR. Bayesian and neural network schemes for call admission control in lte systems. In: Global Communications Conference (GLOBECOM), 2013 IEEE. Piscataway: IEEE; 2013. p. 1246–52.
68. Bonald T, May M, Bolot JC. Analytic evaluation of red performance. In: Proceedings IEEE INFOCOM 2000. Conference on, Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064), vol 3. 2000. p. 1415–24.
69. Bonfiglio D, Mellia M, Meo M, Rossi D, Tofanelli P. Revealing skype traffic: when randomness plays with you. In: ACM SIGCOMM Computer Communication Review. ACM; 2007. p. 37–48.
70. Boser BE, Guyon IM, Vapnik VN. A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, ACM, New York, NY, USA, COLT '92. New York: ACM; 1992. p. 144–52.
71. Boyan JA, Littman ML. Packet routing in dynamically changing networks: A reinforcement learning approach. In: Advances in neural information processing systems; 1994. p. 671–8.
72. Braden B, Clark D, Crowcroft J, Davie B, Deering S, Estrin D, Floyd S, Jacobson V, Minshall G, Partridge C, Peterson L, Ramakrishnan K, Shenker S, Wroclawski J, Zhang L. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, Internet Engineering Task Force. 1998. https://tools.ietf.org/html/rfc2309.
73. Brakmo LS, O'Malley SW, Peterson LL. Tcp vegas: New techniques for congestion detection and avoidance. In: Proceedings of the Conference on Communications Architectures, Protocols and Applications, ACM, New York, NY, USA, SIGCOMM '94. New York: ACM; 1994. p. 24–35.
74. Brauckhoff D, Wagner A, May M. FLAME: A flow-level anomaly modeling engine. In: Proceedings of the conference on Cyber security experimentation and test (CSET). Berkley: USENIX Association; 2008. p. 1.
75. Breiman L. Bagging predictors. Mach Learn. 1996;24(2):123–40.
76. Breiman L, Friedman J, Stone C, Olshen R. Classification and Regression Trees. The Wadsworth and Brooks-Cole statistics-probability series. New York: Taylor & Francis; 1984.
77. Brill E, Lin JJ, Banko M, Dumais ST, Ng AY, et al. Data-intensive question answering. In: TREC, vol. 56. 2001. p. 90.
78. Broomhead DS, Lowe D. Radial basis functions, multi-variable functional interpolation and adaptive networks. Memorandum No. 4148. Malvern: Royal Signals and Radar Establishment; 1988.
79. Brownlee J. Practical Machine Learning Problems. 2013. https://machinelearningmastery.com/practical-machine-learning-problems/. Accessed 28 Dec 2017.
80. Bryson A, Ho Y. Applied optimal control: optimization, estimation and control. Blaisdell book in the pure and applied sciences. Waltham: Blaisdell Pub. Co; 1969.
81. Bryson AE. A gradient method for optimizing multi-stage allocation processes. In: Harvard University Symposium on Digital Computers and their Applications. Boston; 1961. p. 72.
82. Buczak AL, Guven E. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Commun Surv Tutor. 2016;18(2):1153–76.
83. Caini C, Firrincieli R. Tcp hybla: a tcp enhancement for heterogeneous networks. Int J Satell Commun Netw. 2004;22(5):547–66.
84. Cannady J. Artificial neural networks for misuse detection. In: Proceedings of the 21st National information systems security conference, vol. 26. Virginia; 1998. p. 368–81.
85. Cannady J. Next generation intrusion detection: Autonomous reinforcement learning of network attacks. In: Proceedings of the 23rd national information systems security conference. Baltimore; 2000. p. 1–12.
86. Chabaa S, Zeroual A, Antari J. Identification and prediction of internet traffic using artificial neural networks. J Int Learn Syst Appl. 2010;2(03):147.
87. Chakrabarti S, Chakraborty M, Mukhopadhyay I. Study of snort-based ids. In: Proceedings of the International Conference and Workshop on Emerging Trends in Technology. New York: ACM; 2010. p. 43–7.
88. Chang CC, Lin CJ. Libsvm: a library for support vector machines. ACM trans int syst technols (TIST). 2011;2(3):27.
89. Charonyktakis P, Plakia M, Tsamardinos I, Papadopouli M. On user-centric modular qoe prediction for voip based on machine-learning algorithms. IEEE Trans Mob Comput. 2016;15(6):1443–56.
90. Chebrolu S, Abraham A, Thomas JP. Feature deduction and ensemble design of intrusion detection systems. Comput secur. 2005;24(4):295–307.
91. Chen M, Zheng AX, Lloyd J, Jordan MI, Brewer E. Failure diagnosis using decision trees. In: Autonomic Computing, 2004. Proceedings. International Conference on. Piscataway: IEEE; 2004. p. 36–43.
92. Chen MY, Kiciman E, Fratkin E, Fox A, Brewer E. Pinpoint: Problem determination in large, dynamic internet services. In: Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on. Piscataway: IEEE; 2002. p. 595–604.
93. Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM; 2016. p. 785–94.
94. Chen Z, Wen J, Geng Y. Predicting future traffic using hidden markov models. In: Proceddings of 24th IEEE International Conference on Network Protocols (ICNP). IEEE; 2016. p. 1–6.
95. Cheng RG, Chang CJ. Neural-network connection-admission control for atm networks. IEE Proc-Commun. 1997;144(2):93–8.
96. Choi SP, Yeung DY. Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In: Advances in Neural Information Processing Systems. 1996. p. 945–51.
97. Chow CK. An optimum character recognition system using decision functions. IRE Trans Electron Comput EC. 1957;6(4):247–54.
98. Chun-Feng W, Kui L, Pei-Ping S. Hybrid artificial bee colony algorithm and particle swarm search for global optimization. Math Probl Eng. 2014;2014.
99. Cisco. The Zettabyte Era: Trends and Analysis. 2017. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html. Accessed 28 Dec 2017.
100. Cisco Systems. Cisco IOS Netflow. 2012. http://www.cisco.com/go/netflow. Accessed 01 Aug 2017.
101. Cisco Systems. Snort: The worlds most widely deployed ips technology. 2014. https://www.cisco.com/c/en/us/products/collateral/security/brief_c17-733286.html. Accessed 25 Apr 2018.
102. Claeys M, Latre S, Famaey J, De Turck F. Design and evaluation of a self-learning http adaptive video streaming client. IEEE commun lett. 2014a;18(4):716–9.
103. Claeys M, Latré S, Famaey J, Wu T, Van Leekwijck W, De Turck F. Design and optimisation of a (fa) q-learning-based http adaptive streaming client. Connect Sci. 2014b;26(1):25–43.
104. Cortez P, Rio M, Rocha M, Sousa P. Internet traffic forecasting using neural networks. In: Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN). IEEE; 2006. p. 2635–42.
105. Cox DR. The regression analysis of binary sequences. J R Stat Soc Ser B (Methodological). 1958;20(2):215–42.
106. Cybenko G. Approximation by superpositions of a sigmoidal function. Mathematics of Control. Signals Syst (MCSS). 1989;2(4):303–14.
107. Dagum P, Galper A, Horvitz EJ. Temporal Probabilistic Reasoning: Dynamic Network Models for Forecasting. Stanford: Knowledge Systems Laboratory, Medical Computer Science, Stanford University; 1991.
108. Dainotti A, Pescapé A, Sansone C. Early classification of network traffic through multi-classification. In: International Workshop on Traffic Monitoring and Analysis. Springer; 2011. p. 122–35.
109. Dainotti A, Pescape A, Claffy KC. Issues and future directions in traffic classification. IEEE Netw. 2012;26(1):35–40.

110. Dean T, Kanazawa K. A model for reasoning about persistence and causation. Comput Intell. 1989;5(2):142–50.
111. Dechter R. Learning while searching in constraint-satisfaction-problems. In: Proceedings of the Fifth AAAI National Conference on Artificial Intelligence, AAAI Press, AAAI'86. Palo Alto: AAAI Press; 1986. p. 178–83.
112. Demirbilek E. The INRS Audiovisual Quality Dataset. 2016. https://github.com/edipdemirbilek/TheINRSAudiovisualQualityDataset. Accessed 28 Dec 2017.
113. Demirbilek E, Grégoire JC. INRS audiovisual quality dataset. ACM; 2016, pp. 167–71.
114. Demirbilek E, Grégoire JC. Machine learning–based parametric audiovisual quality prediction models for real-time communications. ACM Transactions on Multimedia Computing. Commun Appl (TOMM). 2017;13(2):16.
115. Dempster AP, Laird NM, Rubin DB. Maximum likelihood from incomplete data via the em algorithm. J R Stat Soc Ser B (Method). 1977;39(1):1–38.
116. Depren O, Topallar M, Anarim E, Ciliz MK. An intelligent intrusion detection system (ids) for anomaly and misuse detection in computer networks. Expert syst Appl. 2005;29(4):713–22.
117. Detristan T, Ulenspiegel T, Malcom Y, Underduk M. Polymorphic shellcode engine using spectrum analysis. 2003. http://www.phrack.org/show.php?p=61&a=9. Accessed 25 May 2018.
118. Ding J, Kramer B, Xu S, Chen H, Bai Y. Predictive fault management in the dynamic environment of ip networks. In: IP Operations and Management, 2004. Proceedings IEEE Workshop on. Piscataway: IEEE; 2004. p. 233–9.
119. Ding L, Wang X, Xu Y, Zhang W. Improve throughput of tcp-vegas in multihop ad hoc networks. Comput Commun. 2008;31(10):2581–8.
120. Domingos P. The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. Basic Books; 2015.
121. Donato W, Pescapé A, Dainotti A. Traffic identification engine: an open platform for traffic classification. IEEE Netw. 2014;28(2):56–64.
122. Dong M, Li Q, Zarchy D, Godfrey PB, Schapira M. Pcc: Re-architecting congestion control for consistent high performance. In: Proceedings of the 12th, USENIX Conference on Networked Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, NSDI'15. Berkley: USENIX Association; 2015. p. 395–408.
123. Dowling J, Cunningham R, Curran E, Cahill V. Collaborative reinforcement learning of autonomic behaviour. In: Proceedings. 15th International Workshop on Database and Expert Systems Applications, 2004. 2004. p. 700–4. https://doi.org/10.1109/DEXA.2004.1333556.
124. Dowling J, Curran E, Cunningham R, Cahill V. Using feedback in collaborative reinforcement learning to adaptively optimize manet routing. IEEE Transactions on Systems. Man and Cybern-Part A: Syst Hum. 2005;35(3):360–72.
125. Dreger H, Feldmann A, Mai M, Paxson V, Sommer R. Dynamic application-layer protocol analysis for network intrusion detection. In: USENIX Security Symposium. Berkeley: USENIX Security Symposium; 2006. p. 257–72.
126. Dump CM. Dde command execution malware samples. 2017. http://contagiodump.blogspot.it. Accessed 1 Mar 2017.
127. eBay Inc. eBay. 2017. https://www.ebay.com/. Accessed 01 Aug 2017.
128. Edalat Y, Ahn JS, Obraczka K. Smart experts for network state estimation. IEEE Trans Netw Serv Manag. 2016;13(3):622–35.
129. El Khayat I, Geurts P, Leduc G. Improving TCP in Wireless Networks with an Adaptive Machine-Learnt Classifier of Packet Loss Causes. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005, pp. 549–60.
130. El Khayat I, Geurts P, Leduc G. Enhancement of tcp over wired/wireless networks with packet loss classifiers inferred by supervised learning. Wirel Netw. 2010;16(2):273–90.
131. Elkan C. Results of the kdd'99 classifier learning. ACM SIGKDD Explor Newsl. 2000;1(2):63–4.
132. Elkotob M, Grandlund D, Andersson K, Ahlund C. Multimedia qoe optimized management using prediction and statistical learning. In: Local Computer Networks (LCN) ,2010 IEEE 35th Conference on. IEEE; 2010. p. 324–7.
133. Elwhishi A, Ho PH, Naik K, Shihada B. Arbr: Adaptive reinforcement-based routing for dtn. In: Wireless and Mobile Computing, Networking and Communications (WiMob), 2010 IEEE 6th International Conference on. IEEE; 2010. p. 376–85.
134. Erickson BJ, Korfiatis P, Akkus Z, Kline TL. Machine learning for medical imaging. RadioGraphics. 2017;37(2):505–15.
135. Erman J, Arlitt M, Mahanti A. Traffic classification using clustering algorithms. In: Proceedings of the 2006 SIGCOMM workshop on Mining network data. ACM; 2006a. p. 281–6.
136. Erman J, Mahanti A, Arlitt M. Internet traffic identification using machine learning. In: Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE. IEEE; 2006b. p. 1–6.
137. Erman J, Mahanti A, Arlitt M, Cohen I, Williamson C. Offline/realtime traffic classification using semi-supervised learning. Perform Eval. 2007a;64(9):1194–213.
138. Erman J, Mahanti A, Arlitt M, Williamson C. Identifying and discriminating between web and peer-to-peer traffic in the network core. In: Proceedings of the 16th international conference on World Wide Web. ACM; 2007b. p. 883–92.
139. Eskin E, Arnold A, Prerau M, Portnoy L, Stolfo S. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. Appl data min comput secur. 2002;6:77–102.
140. Este A, Gringoli F, Salgarelli L. Support vector machines for tcp traffic classification. Comput Netw. 2009;53(14):2476–90.
141. Eswaradass A, Sun XH, Wu M. Network bandwidth predictor (nbp): A system for online network performance forecasting. In: Proceedings of 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID). IEEE; 2006. p. 4–pp.
142. Fadlullah Z, Tang F, Mao B, Kato N, Akashi O, Inoue T, Mizutani K. State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. IEEE Commun Surv Tutor. 2017;PP(99):1.
143. Farmer D, Venema W. Satan: Security administrator tool for analyzing networks. 1993. http://www.porcupine.org/satan/. Accessed 28 Dec 2017.
144. Feng W-C, Shin KG, Kandlur DD, Saha D. The blue active queue management algorithms. IEEE/ACM Trans Netw. 2002;10(4):513–28.
145. Fiedler M, Hossfeld T, Tran-Gia P. A generic quantitative relationship between quality of experience and quality of service. IEEE Netw. 2010;24(2).
146. Finamore A, Mellia M, Meo M, Rossi D. Kiss: Stochastic packet inspection classifier for udp traffic. IEEE/ACM Trans Netw. 2010;18(5):1505–15.
147. Fix E, Hodges JL. Discriminatory analysis-nonparametric discrimination: consistency properties. Report No. 4, Project 21-49-004, USAF School of Aviation Medicine. 1951.
148. Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. IEEE/ACM Trans Netw. 1993;1(4):397–413.
149. Fogla P, Sharif MI, Perdisci R, Kolesnikov OM, Lee W. Polymorphic blending attacks. In: USENIX Security Symposium. Berkley: USENIX Association; 2006. p. 241–56.
150. Fonseca N, Crovella M. Bayesian packet loss detection for tcp. In: Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies., vol 3. 2005. p. 1826–37.
151. Forster A, Murphy AL. Froms: Feedback routing for optimizing multiple sinks in wsn with reinforcement learning. In: Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International, Conference on. IEEE; 2007. p. 371–6.
152. Fraleigh C, Diot C, Lyles B, Moon S, Owezarski P, Papagiannaki D, Tobagi F. Design and deployment of a passive monitoring infrastructure. In: Thyrrhenian Internatinal Workshop on Digital Communications. Springer; 2001. p. 556–75.
153. Freund Y, Schapire RE. Experiments with a new boosting algorithm. In: Proceedings of the Thirteenth International Conference on International Conference on Machine Learning, ICML'96. San Francisco: Morgan Kaufmann Publishers Inc.; 1996. p. 148–56.
154. Friedman JH. Greedy function approximation: A gradient boosting machine. Ann Statist. 2001;29(5):1189–232.
155. Friedman JH. Stochastic gradient boosting. Comput Stat Data Anal. 2002;38(4):367–78.
156. Fu CP, Liew SC. Tcp veno: Tcp enhancement for transmission over wireless access networks. IEEE J Sel Areas Commun. 2003;21(2): 216–28.
157. Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybern. 1980;36(4):193–202.

158. Funahashi KI. On the approximate realization of continuous mappings by neural networks. Neural netw. 1989;2(3):183–92.
159. Gagniuc P. Markov Chains: From Theory to Implementation and Experimentation. Hoboken: Wiley; 2017.
160. Gao Y, He G, Hou JC. On exploiting traffic predictability in active queue management. In: Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3. Piscataway: IEEE; 2002. p. 1630–9.
161. Garcia-Teodoro P, Diaz-Verdejo J, Maciá-Fernández G, Vázquez E. Anomaly-based network intrusion detection: Techniques, systems and challenges. Comput Secur. 2009;28(1-2):18–28.
162. Gartner Inc. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. 2017. https://www.gartner.com/newsroom/id/3598917. Accessed 01 Aug 2017.
163. Geurts P, Khayat IE, Leduc G. A machine learning approach to improve congestion control over wireless computer networks. In: Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on; 2004. p. 383–6.
164. Geurts P, Ernst D, Wehenkel L. Extremely randomized trees. Mach Learn. 2006;63(1):3–42.
165. Giacinto G, Perdisci R, Del Rio M, Roli F. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. Inf Fusion. 2008;9(1):69–82.
166. Go Y, Jamshed MA, Moon Y, Hwang C, Park K. Apunet: Revitalizing gpu as packet processing accelerator. In: NSDI. 2017. p. 83–96.
167. Goetz P, Kumar S, Miikkulainen R. On-line adaptation of a signal predistorter through dual reinforcement learning. In: ICML. 1996. p. 175–81.
168. Goldberger AS. Econometric computing by hand. J Econ Soc Meas. 2004;29(1-3):115–7.
169. Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016. http://www.deeplearningbook.org.
170. Google. Cloud TPUs - ML accelerators for TensorFlow, Google Cloud Platform. 2017. https://cloud.google.com/tpu/. Accessed 01 Aug 2017.
171. Görnitz N, Kloft M, Rieck K, Brefeld U. Active learning for network intrusion detection. In: Proceedings of the 2nd ACM workshop on Security and artificial intelligence. New York: ACM; 2009. p. 47–54.
172. Gu Y, Grossman R. Sabul: A transport protocol for grid computing. J Grid Comput. 2003;1(4):377–86.
173. Guyon I, Elisseeff A. An introduction to variable and feature selection. J Mach Learn Res. 2003;1157–82.
174. Ha S, Rhee I, Xu L. Cubic: A new tcp-friendly high-speed tcp variant. SIGOPS Oper Syst Rev. 2008;42(5):64–74.
175. Habib I, Tarraf A, Saadawi T. A neural network controller for congestion control in atm multiplexers. Comput Netw ISDN Syst. 1997;29(3):325–34.
176. Haffner P, Sen S, Spatscheck O, Wang D. Acas: automated construction of application signatures. In: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data. New York: ACM; 2005. p. 197–202.
177. Haining W. An economical broadband network with high added value. In: Evolving the Access Network, International Engineering Consortium. Chicago: International Engineering Consortium; 2006. p. 67–70.
178. Hajji H. Statistical analysis of network traffic for adaptive faults detection. IEEE Trans Neural Netw. 2005;16(5):1053–63.
179. Hariri B, Sadati N. Nn-red: an aqm mechanism based on neural networks. Electron Lett. 2007;43(19):1053–5.
180. Harrison V, Pagliery J. Nearly 1 million new malware threats released every day. 2015. http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/index.html. Accessed 28 Dec 2017.
181. Hashmi US, Darbandi A, Imran A. Enabling proactive self-healing by data mining network failure logs. In: Computing, Networking and Communications (ICNC), 2017 International Conference on. Piscataway: IEEE; 2017. p. 511–7.
182. He L, Xu C, Luo Y. vtc: Machine learning based traffic classification as a virtual network function. In: Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization. ACM; 2016. p. 53–56.
183. He Q, Shayman MA. Using reinforcement learning for proactive network fault management. In: Proceedings of the International Conference on Communication Technologies. 1999.
184. Hebb D. The Organization of Behavior: A Neuropsychological Theory. New York: Wiley; 1949.
185. Henderson T, Floyd S, Gurtov S, Nishida Y. sThe newreno modification to tcp's fast recovery algorithm. RFC 6582, Internet Engineering Task Force. 2012. https://tools.ietf.org/html/rfc6582.
186. Hinton GE. Training products of experts by minimizing contrastive divergence. Training. 2006;14(8).
187. Hinton GE, McClelland JL, Rumelhart DE. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In: Rumelhart DE, McClelland JL, PDP Research Group C, editors. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge, MA, USA, chap Distributed Representations: MIT Press; 1986. p. 77–109.
188. Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. Neural Comput. 2006;18(7):1527–54.
189. Hiramatsu A. Atm communications network control by neural networks. IEEE Trans Neural Netw. 1990;1(1):122–30.
190. Hiramatsu A. Integration of atm call admission control and link capacity control by distributed neural networks. IEEE J Sel Areas Commun. 1991;9(7):1131–8.
191. Ho TK. Random decision forests. In: Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1, IEEE Computer Society, Washington, DC, USA, ICDAR '95. Piscataway: IEEE; 1995. p. 278.
192. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.
193. Hood CS, Ji C. Proactive network-fault detection. IEEE Trans Reliab. 1997;46(3):333–41.
194. de Hoon M, Imoto S, Nolan J, Miyano S. Open source clustering software. Bioinformatics. 2004;20(9):1453–4.
195. Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci. 1982;79(8):2554–8. http://www.pnas.org/content/79/8/2554.full.pdf.
196. Hornik K. Approximation capabilities of multilayer feedforward networks. Neural Netw. 1991;4(2):251–7.
197. Hu T, Fei Y. Qelar: a machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks. IEEE Trans Mob Comput. 2010;9(6):796–809.
198. Hu W, Hu W, Maybank S. Adaboost-based algorithm for network intrusion detection. IEEE Transactions on Systems, Man, and Cybernetics. Part B (Cybernetics). 2008;38(2):577–83.
199. Huang YS, Suen CY. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. IEEE Trans Pattern Anal Mach Intell. 1995;17(1):90–4.
200. Hubel DH, Wiesel TN. Receptive fields of single neurones in the cat's striate cortex. J Physiol. 1959;148(3):574–91.
201. Hubel DH, Wiesel TN. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J Physiol. 1962;160(1):106–54.
202. IMPACT Cyber Trust. Information Marketplace for Policy and Analysis of Cyber-risk and Trust. 2017. https://www.impactcybertrust.org. Accessed 01 Aug 2017.
203. Information Sciences Institute. The network simulator ns-2. 2014. http://www.isi.edu/nsnam/ns/. Accessed 10 Oct 2017.
204. Ingham KL, Inoue H. Comparing anomaly detection techniques for http. In: International Workshop on Recent Advances in Intrusion Detection (RAID). Berlin: Springer; 2007. p. 42–62.
205. Intanagonwiwat C, Govindan R, Estrin D, Heidemann J, Silva F. Directed diffusion for wireless sensor networking. IEEE/ACM Trans Netw (ToN). 2003;11(1):2–16.
206. International Telecommunications Union. G.107 : The E-model: a computational model for use in transmission planning. 2008. https://www.itu.int/rec/T-REC-G.107. Accessed 28 Dec 2017.
207. Internet Assigned Numbers Authority. IANA. 2017. https://www.iana.org/. Accessed 01 Aug 2017.
208. Internet Engineering TaskForce. n Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. 2002. https://tools.ietf.org/html/rfc3411. Accessed 01 Aug 2017.
209. Internet Engineering Task Force. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. 2008. https://tools.ietf.org/html/rfc5101. Accessed 01 Aug 2017.
210. Ivakhnenko A, Lapa V, ENGINEERING PULISOE. Cybernetic Predicting Devices. Purdue University School of Electrical Engineering; 1965.

211. Iyengar J, Swett I. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Tech. rep. Network Working Group; 2015. https://tools.ietf.org/pdf/draft-tsvwg-quic-protocol-00.pdf.
212. Jain A, Karandikar A, Verma R. An adaptive prediction based approach for congestion estimation in active queue management (apace). In: Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE, vol. 7. Piscataway: IEEE; 2003. p. 4153–7.
213. Javaid A, Niyaz Q, Sun W, Alam M. A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), ICST, (Institute for Computer Sciences Social-Informatics and Telecommunications Engineering). Brussels; 2016. p. 21–6.
214. Jayaraj A, Venkatesh T, Murthy CSR. Loss classification in optical burst switching networks using machine learning techniques: improving the performance of tcp. IEEE J Sel Areas Commun. 2008;26(6):45–54.
215. Jaynes ET. Information theory and statistical mechanics. Phys Rev. 1957a;106:620–30.
216. Jaynes ET. Information theory and statistical mechanics. ii. Phys Rev. 1957b;108:171–90.
217. Jennings A. A learning system for communications network configuration. Eng Appl Artif Intell. 1988;1(3):151–60.
218. Jiang H, Moore AW, Ge Z, Jin S, Wang J. Lightweight application classification for network management. In: Proceedings of the 2007 SIGCOMM workshop on Internet network management. ACM; 2007. p. 299–304.
219. Jiang H, Luo Y, Zhang Q, Yin M, Wu C. Tcp-gvegas with prediction and adaptation in multi-hop ad hoc networks. Wirel Netw. 2017;23(5):1535–48.
220. Jiang S, Song X, Wang H, Han JJ, Li QH. A clustering-based method for unsupervised intrusion detections. Pattern Recog Lett. 2006;27(7):802–10.
221. Jin Y, Duffield N, Haffner P, Sen S, Zhang ZL. Inferring applications at the network layer using collective traffic statistics. In: Teletraffic Congress (ITC), 2010 22nd International. IEEE; 2010. p. 1–8.
222. Jin Y, Duffield N, Erman J, Haffner P, Sen S, Zhang ZL. A modular machine learning system for flow-level traffic classification in large networks. ACM Trans Knowl Discov Data (TKDD). 2012;6(1):4.
223. Jing N, Yang M, Cheng S, Dong Q, Xiong H. An efficient svm-based method for multi-class network traffic classification. In: Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International. IEEE; 2011. p. 1–8.
224. Joachims T. SVMlight. 1999. http://svmlight.joachims.org/. Accessed 28 Dec 2017.
225. Johnsson A, Meirosu C. Towards automatic network fault localization in real time using probabilistic inference. In: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on. Piscataway: IEEE; 2013. p. 1393–8.
226. Jordan MI. Serial order: A parallel distributed processing approach. Tech. Rep. ICS Report 8604. San Diego: University of California; 1986.
227. Juniper Research. Cybercrime will cost businesses over $2 trillion by 2019. 2015. https://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillion. Accessed 10 Nov 2017.
228. Karagiannis T, Broido A, Brownlee N, Claffy KC, Faloutsos M. Is p2p dying or just hiding?[p2p traffic measurement]. In: IEEE Global Telecommunications Conference (GLOBECOM), vol. 3. 2004. p. 1532–8.
229. Karagiannis T, Papagiannaki K, Faloutsos M. BLINC: multilevel traffic classification in the dark. ACM SIGCOMM Comput Commun Rev. 2005;35(4):229–40.
230. Karami A. Accpndn: Adaptive congestion control protocol in named data networking by learning capacities using optimized time-lagged feedforward neural network. J Netw Comput Appl. 2015;56(Supplement C):1–18.
231. Lab Kaspersky. Damage control: The cost of security breaches. IT security risks special report series Report, Kaspersky. 2015. https://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf. Accessed 10 Nov 2017.
232. Kayacik HG, Zincir-Heywood AN, Heywood MI. On the capability of an SOM-based intrusion detection system. In: Proceedings of the International Joint Conference on Neural Networks. New York: IEEE; 2003. p. 1808–13.
233. Kelley HJ. Gradient theory of optimal flight paths. ARS J. 1960;30(10): 947–54.
234. Kennedy J, Eberhart R. Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on, vol 4. 1995. p. 1942–8.
235. Khan A, Sun L, Ifeachor E. Content-based video quality prediction for mpeg4 video streaming over wireless networks. J Multimedia. 2009a;4(4).
236. Khan A, Sun L, Ifeachor E. Content clustering based video quality prediction model for mpeg4 video streaming over wireless networks. IEEE; 2009b, pp. 1–5.
237. Khanafer RM, Solana B, Triola J, Barco R, Moltsen L, Altman Z, Lazaro P. Automated diagnosis for umts networks using bayesian network approach. IEEE Trans veh technol. 2008;57(4):2451–61.
238. Khayat IE, Geurts P, Leduc G. Machine-learnt versus analytical models of tcp throughput. Comput Netw. 2007;51(10):2631–44.
239. Khorsandroo S, Noor RM, Khorsandroo S. The role of psychophysics laws in quality of experience assessment: a video streaming case study. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics. ACM; 2012. p. 446–52.
240. Khorsandroo S, Md Noor R, Khorsandroo S. A generic quantitative relationship to assess interdependency of qoe and qos. KSII Trans Internet Inf Syst. 2013;7(2).
241. Kiciman E, Fox A. Detecting application-level failures in component-based internet services. IEEE Trans Neural Netw. 2005;16(5): 1027–41.
242. Kim DS, Nguyen HN, Park JS. Genetic algorithm to improve svm based network intrusion detection system. New York: IEEE; 2005, pp. 155–8.
243. Kim H, Fomenkov M, claffy kc, Brownlee N, Barman D, Faloutsos M. Comparison of internet traffic classification tools. In: IMRG Workshop on Application Classification and Identification; 2007. p. 1–2.
244. Kim H, Claffy KC, Fomenkov M, Barman D, Faloutsos M, Lee K. Internet traffic classification demystified: myths, caveats, and the best practices. ACM; 2008, p. 11.
245. Kim J, Kim J, Thu HLT, Kim H. Long short term memory recurrent neural network classifier for intrusion detection. International Conference on. IEEE; 2016, pp. 1–5.
246. Klaine PV, Imran MA, Onireti O, Souza RD. A survey of machine learning techniques applied to self organizing cellular networks. IEEE Commun Surv Tutor. 2017;PP(99):1.
247. Kogeda OP, Agbinya JI, Omlin CW. A probabilistic approach to faults prediction in cellular networks. IEEE; 2006, p. 130.
248. Kogeda P, Agbinya J. Prediction of faults in cellular networks using bayesian network model. UTS ePress; 2006.
249. Kohonen T. Self-organized formation of topologically correct feature maps. Biol Cybern. 1982;43(1):59–69.
250. Kohonen T, Hynninen J, Kangas J, Laaksonen J. Som pak: The self-organizing map program package. Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science. 1996.
251. Kotz D, Henderson T, Abyzov I, Yeo J. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). 2009. https://crawdad.org/dartmouth/campus/20090909. Accessed 28 Dec 2017.
252. Kruegel C, Toth T. Using decision trees to improve signature-based intrusion detection. In: Recent Advances in Intrusion Detection. Springer; 2003. p. 173–91.
253. Kumano Y, Ata S, Nakamura N, Nakahira Y, Oka I. Towards real-time processing for application identification of encrypted traffic. In: International Conference on Computing, Networking and Communications (ICNC); 2014. p. 136–40.
254. Kumar S, Miikkulainen R. Dual reinforcement q-routing: An on-line adaptive routing algorithm. In: Proceedings of the artificial neural networks in engineering Conference. 1997. p. 231–8.
255. Kumar Y, Farooq H, Imran A. Fault prediction and reliability analysis in a real cellular network. In: Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International. IEEE; 2017. p. 1090–1095.
256. Labs ML. Kdd cup 1998 data. 1998. https://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html. Accessed 28 Dec 2017.
257. Labs ML. Kdd cup 1999 data. 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed 28 Dec 2017.
258. Lagoudakis MG, Parr R. Model-free least-squares policy iteration. In: Advances in neural information processing systems. 2002. p. 1547–54.
259. Lal TN, Chapelle O, Weston J, Elisseeff A. Embedded methods. In: Feature extraction. Springer; 2006. p. 137–65.
260. Lapedes AS, Farber RM. How neural nets work. 1987.

261. Laplace PS. Théorie analytique des probabilités. Paris: Courcier; 1812.
262. Lawrence Berkeley NationalLaboratoryandICSI. LBNL/ICSI Enterprise Tracing Project. 2005. http://www.icir.org/enterprise-tracing/. Accessed 01 Aug 2017.
263. Le Cun Y. Learning Process in an Asymmetric Threshold Network. Berlin, Heidelberg: Springer Berlin Heidelberg; 1986, pp. 233–40.
264. Lee SJ, Hou CL. A neural-fuzzy system for congestion control in atm networks. IEEE Transactions on Systems, Man, and Cybernetics. Part B (Cybernetics). 2000;30(1):2–9.
265. Leela-Amornsin L, Esaki H. Heuristic congestion control for message deletion in delay tolerant network. In: Proceedings of the Third Conference on Smart Spaces and Next Generation Wired, and 10th International Conference on Wireless Networking, Springer-Verlag, Berlin, Heidelberg, ruSMART/NEW2AN'10; 2010. p. 287–98.
266. Legendre A. Nouvelles méthodes pour la détermination des orbites des comètes. Nineteenth Century Collections Online (NCCO): Science, Technology, and Medicine: 1780-1925, F. Didot. 1805.
267. Lemaître G, Nogueira F, Aridas CK. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. J Mach Learn Res. 2017;18(17):1–5.
268. Leonardi E, Mellia M, Horvath A, Muscariello L, Niccolini S, Rossi D, Young K. Building a cooperative p2p-tv application over a wise network: the approach of the european fp-7 strep napa-wine. IEEE Commun Mag. 2008;46(4):20–2.
269. Li F, Sun J, Zukerman M, Liu Z, Xu Q, Chan S, Chen G, Ko KT. A comparative simulation study of tcp/aqm systems for evaluating the potential of neuron-based aqm schemes. J Netw Comput Appl. 2014;41(Supplement C):274–99.
270. Li W, Moore AW. A machine learning approach for efficient traffic classification. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International, Symposium on. IEEE; 2007. p. 310–7.
271. Li W, Zhou F, Meleis W, Chowdhury K. Learning-based and data-driven tcp design for memory-constrained iot. In: 2016 International Conference on Distributed Computing in Sensor Systems (DCOSS). 2016a. p. 199–205.
272. Li Y, Guo L. An active learning based tcm-knn algorithm for supervised network intrusion detection. Comput Secur. 2007;26(7):459–67.
273. Li Y, Ma R, Jiao R. A hybrid malicious code detection method based on deep learning. Methods. 2015;9(5).
274. Li Y, Liu H, Yang W, Hu D, Xu W. Inter-data-center network traffic prediction with elephant flows: IEEE; 2016b, pp. 206–13.
275. Lin LJ. Reinforcement learning for robots using neural networks. PhD thesis. Pittsburgh, PA, USA: Carnegie Mellon University; 1992. uMI Order No. GAX93-22750.
276. Lin SC, Akyildiz IF, Wang P, Luo M. Qos-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach. In: Services Computing (SCC) 2016, IEEE International Conference on. IEEE; 2016. p. 25–33.
277. Lin Z, van der Schaar M. Autonomic and distributed joint routing and power control for delay-sensitive applications in multi-hop wireless networks. IEEE Trans Wirel Commun. 2011;10(1):102–13.
278. Lincoln Laboratory MIT. DARPA Intrusion Detection Evaluation. 1999. https://www.ll.mit.edu/ideval/data/1999data.html. Accessed 01 Aug 2017.
279. Littlestone N, Warmuth MK. The weighted majority algorithm. In: 30th Annual Symposium on Foundations of Computer Science. 1989. p. 256–61.
280. Littman M, Boyan J. A distributed reinforcement learning scheme for network routing. In: Proceedings of the international workshop on applications of neural networks to telecommunications. Psychology Press; 1993. p. 45–51.
281. Liu D, Zhang Y, Zhang H. A self-learning call admission control scheme for cdma cellular networks. IEEE trans neural netw. 2005;16(5):1219–28.
282. Liu J, Matta I, Crovella M. End-to-end inference of loss nature in a hybrid wired/wireless environment. 2003.
283. Liu Y, Li W, Li YC. Network traffic classification using k-means clustering: IEEE; 2007. pp. 360–5.
284. Liu YC, Douligeris C. Static vs. adaptive feedback congestion controller for atm networks. In: Global Telecommunications Conference, 1995. GLOBECOM '95., vol 1. IEEE; 1995. p. 291–5.
285. Lu X, Wang H, Zhou R, Ge B. Using hessian locally linear embedding for autonomic failure prediction. In: Nature & Biologically Inspired, Computing, 2009. NaBIC 2009. World Congress on. IEEE; 2009. p. 772–6.
286. Ma J, Levchenko K, Kreibich C, Savage S, Voelker GM. Unexpected means of protocol inference. In: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. 2006. p. 313–26.
287. Machado VA, Silva CN, Oliveira RS, Melo AM, Silva M, Francês CR, Costa JC, Vijaykumar NL, Hirata CM. A new proposal to provide estimation of qos and qoe over wimax networks: An approach based on computational intelligence and discrete-event simulation. IEEE; 2011, pp. 1–6.
288. Machine Learning Group, University of Waikato. WEKA. 2017. http://www.cs.waikato.ac.nz/ml/weka/. Accessed 01 Aug 2017.
289. Macleish KJ. Mapping the integration of artificial intelligence into telecommunications. IEEE J Sel Areas Commun. 1988;6(5):892–8.
290. MacQueen J. Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, University of California Press, Berkeley, 1; 1967. p. 281–97.
291. Mahmoud Q. Cognitive Networks: Towards Self-Aware Networks. Wiley-Interscience; 2007.
292. Malware-Traffic-Analysisnet. A source for pcap files and malware samples. 2017. http://www.malware-traffic-analysis.net. Accessed 15 Dec 2017.
293. Manzoor J, Drago I, Sadre R. The curious case of parallel connections in http/2. In: 12th International Conference on Network and Service Management (CNSM). 2016. p. 174–80.
294. Mao H, Alizadeh M, Menache I, Kandula S. Resource management with deep reinforcement learning. In: HotNets. 2016. p. 50–6.
295. Marbach P, Mihatsch O, Tsitsiklis JN. Call admission control and routing in integrated services networks using neuro-dynamic programming. IEEE J Sel areas commun. 2000;18(2):197–208.
296. Markov AA. An example of statistical investigation in the text of eugene onegin illustrating coupling of tests in chains. In: Proceedings of the Royal Academy of Sciences of St. Petersburg, St. Petersburg, Rusia, vol 1. 1913. p. 153.
297. Maron ME. Automatic indexing: An experimental inquiry. J ACM. 1961;8(3):404–17.
298. Masoumzadeh SS, Taghizadeh G, Meshgi K, Shiry S. Deep blue: A fuzzy q-learning enhanced active queue management scheme. In: 2009 International Conference on Adaptive and Intelligent Systems; 2009. p. 43–8.
299. Mathis M, Mahdavi J, Floyd S, Romanow A. Tcp selective acknowledgment options. RFC 2018, Internet Engineering Task Force. 1996. https://tools.ietf.org/html/rfc2018.
300. Mathis M, Semke J, Mahdavi J, Ott T. The macroscopic behavior of the tcp congestion avoidance algorithm. SIGCOMM Comput Commun Rev. 1997;27(3):67–82.
301. Maxion RA. Anomaly detection for diagnosis. In: Fault-Tolerant Computing, 1990. FTCS-20. Digest of, Papers., 20th International Symposium. IEEE; 1990. p. 20–7.
302. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys. 1943;5(4):115–33.
303. McGregor A, Hall M, Lorier P, Brunskill J. Flow clustering using machine learning techniques. Passive and Active Netw Meas. 2004;205–14.
304. McKenney PE. Stochastic fairness queueing. In: INFOCOM '90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration. Proceedings. IEEE; 1990. p. 733–40. vol.2.
305. Messenger R, Mandell L. A modal search technique for predictibe nominal scale multivariate analys. J Am Stat Assoc. 1972;67(340): 768–72.
306. Mestres A, Rodriguez-Natal A, Carner J, Barlet-Ros P, Alarcón E, Solé M, Muntés-Mulero V, Meyer D, Barkai S, Hibbett MJ, et al. Knowledge-defined networking. ACM SIGCOMM Comput Commun Rev. 2017;47(3):2–10.
307. Metasploit L. The metasploit framework. 2007. http://www.metasploit.com. Accessed 28 Dec 2017.
308. Meyer D. Machine Intelligence and Networks. 2016. https://www.youtube.com/watch?v=XORRw6Sqi9Y. Accessed 28 Dec 2017.

309. Mezzavilla M, Quer G, Zorzi M. On the effects of cognitive mobility prediction in wireless multi-hop ad hoc networks. In: 2014 IEEE International Conference on Communications (ICC); 2014. p. 1638–44.
310. Microsoft Corporation. Skype. 2017. https://www.skype.com/. Accessed 01 Aug 2017.
311. Mignanti S, Di Giorgio A, Suraci V. A model based rl admission control algorithm for next generation networks. In: Networks, 2009. ICN'09. Eighth International Conference on. IEEE; 2009. p. 191–6.
312. Mijumbi R, Gorricho JL, Serrat J, Claeys M, De Turck F, Latré S. Design and evaluation of learning algorithms for dynamic resource management in virtual networks. IEEE; 2014, pp. 1–9.
313. Mijumbi R, Hasija S, Davy S, Davy A, Jennings B, Boutaba R. A connectionist approach to dynamic resource management for virtualised network functions. In: Network and Service Management (CNSM) 2016 12th International Conference on. IEEE; 2016. p. 1–9.
314. Miller ST, Busby-Earle C. Multi-perspective machine learning a classifier ensemble method for intrusion detection. In: Proceedings of the 2017 International Conference on Machine Learning and Soft Computing. ACM; 2017. p. 7–12.
315. Minsky M, Papert S. Perceptrons: An Introduction to Computational Geometry. Cambridge: MIT Press; 1972.
316. Mirza M, Sommers J, Barford P, Zhu X. A machine learning approach to tcp throughput prediction. IEEE/ACM Trans Netw. 2010;18(4):1026–39.
317. Mitchell TM. Machine Learning, 1st ed. New York: McGraw-Hill, Inc.; 1997.
318. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–33.
319. Montana DJ, Davis L. Training feedforward neural networks using genetic algorithms. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'89. 1989. p. 762–7.
320. Moore AW, Papagiannaki K. Toward the accurate identification of network applications. In: PAM. Springer; 2005. p. 41–54.
321. Moore AW, Zuev D. Internet traffic classification using bayesian analysis techniques. In: ACM SIGMETRICS Performance Evaluation Review, ACM, vol 33. 2005. p. 50–60.
322. Moradi M, Zulkernine M. A neural network based system for intrusion detection and classification of attacks. In: Proceedings of the IEEE International Conference on Advances in Intelligent Systems-Theory and Applications. 2004. p. 15–8.
323. Morgan JN, Sonquist JA. Problems in the analysis of survey data, and a proposal. J Am Stat Assoc. 1963;58(302):415–34.
324. Moustapha AI, Selmic RR. Wireless sensor network modeling using modified recurrent neural networks: Application to fault detection. IEEE Trans Instrum Meas. 2008;57(5):981–8.
325. Mukkamala S, Janoski G, Sung A. Intrusion detection using neural networks and support vector machines. 2002, pp. 1702–7.
326. Mukkamala S, Sung AH, Abraham A. Intrusion detection using ensemble of soft computing paradigms. In: Intelligent systems design and applications. Springer; 2003. p. 239–48.
327. Muniyandi AP, Rajeswari R, Rajaram R. Network anomaly detection by cascading k-means clustering and c4. 5 decision tree algorithm. Procedia Eng. 2012;30:174–82.
328. Mushtaq MS, Augustin B, Mellouk A. Empirical study based on machine learning approach to assess the qos/qoe correlation. In: Networks and Optical Communications (NOC), 2012 17th European Conference on. IEEE; 2012. p. 1–7.
329. Nahm K, Helmy A, Jay Kuo CC. Tcp over multihop 802.11 networks: Issues and performance enhancement. In: Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, ACM, New York, NY, USA, MobiHoc '05; 2005. p. 277–87.
330. Narendra KS, Thathachar MAL. Learning automata - a survey. IEEE Transactions on Systems. Man Cybern SMC-. 1974;4(4):323–34.
331. Netflix Inc. Netflix. 2017. https://www.netflix.com/. Accessed 01 Aug 2017.
332. Networks and Mobile Systems Group. Resilient overlay networks RON. 2017. http://nms.csail.mit.edu/ron/. Accessed 27 Dec 2017.
333. Ng AY, Jordan MI. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, MIT Press, Cambridge, MA, USA, NIPS'01. 2001. p. 841–8. http://dl.acm.org/citation.cfm?id=2.980539.2980648.
334. Ng B, Hayes M, Seah WKG. Developing a traffic classification platform for enterprise networks with sdn: Experiences amp; lessons learned. In: IFIP Networking Conference. 2015. p. 1–9.
335. Nguyen T, Armitage G. Synthetic sub-flow pairs for timely and stable ip traffic identification. In: Proc. Australian Telecommunication Networks and Application Conference. 2006a.
336. Nguyen TT, Armitage G. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In: Local Computer Networks, Proceedings 2006 31st IEEE Conference on. IEEE; 2006b. p. 369–76.
337. Nguyen TT, Armitage G, Branch P, Zander S. Timely and continuous machine-learning-based classification for interactive ip traffic. IEEE/ACM Trans Netw (TON). 2012;20(6):1880–94.
338. Nguyen TTT, Armitage G. Clustering to assist supervised machine learning for real-time ip traffic classification. In: IEEE International Conference on Communications. 2008a. p. 5857–62.
339. Nguyen TTT, Armitage G. A survey of techniques for internet traffic classification using machine learning. IEEE Commun Surv Tutor. 2008b;10(4):56–76.
340. Nichols K, Jacobson V. Controlling queue delay. Queue. 2012;10(5): 20:20–20:34.
341. NMapWin. Nmap security scanner. 2016. http://nmapwin.sourceforge.net/. Accessed 1 Mar 2017.
342. NVIDIA. Graphics Processing Unit (GPU). 2017. http://www.nvidia.com/object/gpu.html. Accessed 01 Aug 2017.
343. Padhye J, Firoiu V, Towsley D, Kurose J. Modeling tcp throughput: A simple model and its empirical validation. In: Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM, New York, SIGCOMM '98. 1998. p. 303–14.
344. Pan ZS, Chen SC, Hu GB, Zhang DQ. Hybrid neural network and c4.5 for misuse detection. 2003, pp. 2463–7.
345. Panda M, Abraham A, Patra MR. A hybrid intelligent approach for network intrusion detection. Procedia Eng. 2012;30:1–9.
346. Papernot N, Goodfellow I, Sheatsley R, Feinman R, McDaniel P. cleverhans v1. 0.0: an adversarial machine learning library. 2016. arXiv preprint arXiv:161000768.
347. Park J, Tyan HR, Kuo CCJ. Internet traffic classification for scalable qos provision. In: Multimedia and Expo, vol 2006 IEEE International Conference on. IEEE; 2006. p. 1221–4.
348. Parkour M. Pcap traffic patterns. 2013. http://www.mediafire.com/?a49l965nlayad. Accessed 1 Mar 2017.
349. Parzen E. On estimation of a probability density function and mode. Ann Math Statist. 1962;33(3):1065–76.
350. Paxson V. Bro: A system for detecting network intruders in real-time. Comput Netw. 1999;31(23-24):2435–63.
351. Pcap-Analysis. Malware. 2017. http://www.pcapanalysis.com. Accessed 1 Mar 2017.
352. Pearl J. Bayesian networks: A model of self-activated memory for evidential reasoning. Irvine: University of California; 1985, pp. 329–34.
353. Pearl J. Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann; 2014.
354. Peddabachigari S, Abraham A, Grosan C, Thomas J. Modeling intrusion detection system using hybrid intelligent systems. J netw comput appl. 2007;30(1):114–32.
355. Pellegrini A, Di Sanzo P, Avresky DR. A machine learning-based framework for building application failure prediction models. In: Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International. IEEE; 2015. p. 1072–81.
356. Perdisci R, Ariu D, Fogla P, Giacinto G, Lee W. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. Comput netw. 2009;53(6):864–81.
357. Petrangeli S, Claeys M, Latré S, Famaey J, De Turck F. A multi-agent q-learning-based framework for achieving fairness in http adaptive streaming. IEEE; 2014, pp. 1–9.
358. Pfahringer B. Winning the kdd99 classification cup: bagged boosting. ACM SIGKDD Explor Newsl. 2000;1(2):65–6.
359. Piamrat K, Ksentini A, Viho C, Bonnin JM. Qoe-aware admission control for multimedia applications in ieee 802.11 wireless networks. In:

Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th. IEEE; 2008. p. 1–5.

360. Pietrabissa A. Admission control in umts networks based on approximate dynamic programming. Eur J control. 2008;14(1):62–75.
361. Pietrabissa A, Priscoli FD, Di Giorgio A, Giuseppi A, Panfili M, Suraci V. An approximate dynamic programming approach to resource management in multi-cloud scenarios. Int J Control. 2017;90(3): 492–503.
362. Pinson MH, Wolf S. A new standardized method for objectively measuring video quality. IEEE Trans broadcast. 2004;50(3):312–22.
363. Portnoy L, Eskin E, Stolfo S. Intrusion detection with unlabeled data using clustering. In: Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001). Citeseer; 2001.
364. Portoles-Comeras M, Requena-Esteso M, Mangues-Bafalluy J, Cardenete-Suriol M. Extreme: Combining the ease of management of multi-user experimental facilities and the flexibility of proof of concept testbeds. In: Testbeds and Research Infrastructures for the Development of Networks and Communities, s2006. TRIDENTCOM 2006 2nd International, Conference on. IEEE; 2006. p. 10.
365. Poupart P, Chen Z, Jaini P, Fung F, Susanto H, Geng Y, Chen L, Chen K, Jin H. Online flow size prediction for improved network routing. IEEE; 2016, pp. 1–6.
366. Prechelt L. Early stopping-but when? Neural Netw: Tricks of the trade. 1998;553.
367. Prevost JJ, Nagothu K, Kelley B, Jamshidi M. Prediction of cloud data center networks loads using stochastic and neural models. IEEE; 2011. p. 276–281.
368. Proactcouk. ISS Internet Scanner. 2017. http://www.tech.proact.co.uk/iss/iss_system_scanner.htm. Accessed 28 Dec 2017.
369. Puget JF. What is machine learning? (IT best kept secret is optimization). 2016. https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Is_Machine_Learning. Accessed 01 Aug 2017.
370. Qader K, Adda M. Fault classification system for computer networks using fuzzy probabilistic neural network classifier (fpnnc) International Conference on Engineering Applications of Neural Networks. Springer; 2014. p. 217–26.
371. Quer G, Meenakshisundaram H, Tamma B, Manoj BS, Rao R, Zorzi M. Cognitive network inference through bayesian network analysis. 2010, pp. 1–6.
372. Quer G, Baldo N, Zorzi M. Cognitive call admission control for voip over ieee 802.11 using bayesian networks. In: Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE. IEEE; 2011. p. 1–6.
373. Quinlan J. Discovering rules form large collections of examples: A case study. Edingburgh: Expert Systems in the Micro Electronic Age Edinburgh Press; 1979.
374. Quinlan JR. Simplifying decision trees. Int J Man-Mach Stud. 1987;27(3): 221–34.
375. Quinlan JR. Learning with continuous classes. In: Proceedings of Australian Joint Conference on Artificial Intelligence, World Scientific. 1992. p. 343–8.
376. Quinlan JR. C4.5: Programs for Machine Learning. San Francisco: Morgan Kaufmann Publishers Inc.; 1993.
377. Quinlan JR, et al, Vol. 1. Bagging, boosting, and c4. 5; 1996. 725–30.
378. Raina R, Battle A, Lee H, Packer B, Ng AY. Self-taught learning: transfer learning from unlabeled data. In: Proceedings of the 24th international conference on Machine learning. ACM; 2007. p. 759–66.
379. Ramana BV, Murthy CSR. Learning-tcp: A novel learning automata based congestion window updating mechanism for ad hoc wireless networks. In: Proceedings of the 12th International Conference on High Performance Computing, Springer-Verlag, Berlin, Heidelberg, HiPC'05. 2005. p. 454–464.
380. Ramana BV, Manoj BS, Murthy CSR. Learning-tcp: a novel learning automata based reliable transport protocol for ad hoc wireless networks. In: 2nd International Conference on Broadband Networks 2005. 2005. p. 484–493. Vol. 1.
381. Ranzato M, Poultney C, Chopra S, LeCun Y. Efficient learning of sparse representations with an energy-based model. In: Proceedings of the 19th International Conference on Neural Information Processing Systems, MIT Press, Cambridge, MA, USA, NIPS'06; 2006. p. 1137–44.
382. Rao S. Operational fault detection in cellular wireless base-stations. IEEE Trans Netw Serv Manag. 2006;3(2):1–11.
383. Reichl P, Egger S, Schatz R, D'Alconzo A. The logarithmic nature of qoe and the role of the weber-fechner law in qoe assessment. IEEE; 2010, pp. 1–5.
384. Riiser H, Endestad T, Vigmostad P, Griwodz C, Halvorsen P. DATASET: HSDPA-bandwidth logs for mobile HTTP streaming scenarios. 2011. http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/. Accessed 28 Dec 2017.
385. Riiser H, Endestad T, Vigmostad P, Griwodz C, Halvorsen P. Video streaming using a location-based bandwidth-lookup service for bitrate planning. ACM Trans Multimedia Comput Commun Appl. 2012;8(3): 24:1–24:19. https://doi.org/10.1145/2.240136.2240137, http://doi.acm.org/10.1145/2.240136.2240137.
386. Rix AW, Beerends JG, Hollier MP, Hekstra AP. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In: Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on, IEEE, vol 2. 2001. p. 749–52.
387. Rosenblatt F. The perceptron, a perceiving and recognizing automaton. Report No. 85-460-1 Project PARA: Cornell Aeronautical Laboratory; 1957.
388. Rosenblatt M. Remarks on some nonparametric estimates of a density function. Ann Math Statist. 1956;27(3):832–837.
389. Ross DA, Lim J, Lin RS, Yang MH. Incremental learning for robust visual tracking. Int J Comput Vision. 2008;77(1-3):125–141.
390. Roughan M, Sen S, Spatscheck O, Duffield N. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement. ACM; 2004a. p. 135–148.
391. Roughan M, Zhang Y, Ge Z, Greenberg A. Abilene network; 2004b. http://www.maths.adelaide.edu.au/matthew.roughan/data/Abilene.tar.gz. Accessed 28 Dec 2017.
392. Rozhnova N, Fdida S. An effective hop-by-hop interest shaping mechanism for ccn communications. In: 2012 Proceedings IEEE INFOCOM Workshops; 2012. p. 322–327.
393. Ruiz M, Fresi F, Vela AP, Meloni G, Sambo N, Cugini F, Poti L, Velasco L, Castoldi P. Service-triggered failure identification/localization through monitoring of multiple parameters. In: ECOC 2016; 42nd European Conference on Optical Communication: Proceedings of. VDE; 2016. p. 1–3.
394. Rumelhart DE, Hinton GE, Williams RJ. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In: Rumelhart DE, McClelland JL, PDP Research Group C, editors. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol 1. Cambridge, MA, USA, chap Learning Internal Representations by Error Propagation: MIT Press; 1986. p. 318–62.
395. Rummery GA, Niranjan M. On-line Q-learning using connectionist systems. CUED/F-INFENG/TR 166, Cambridge University Engineering Department. 1994.
396. Rüping S. mySVM. 2004. http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/. Accessed 28 Dec 2017.
397. Russell S, Norvig P. Artificial Intelligence: A Modern Approach, 3rd ed. Upper Saddle River: Prentice Hall Press; 2009.
398. Sabhnani M, Serpen G. Why machine learning algorithms fail in misuse detection on kdd intrusion detection data set. Intell data anal. 2004;8(4): 403–15.
399. Salakhutdinov R, Hinton G. Deep boltzmann machines. In: Artificial Intelligence and Statistics; 2009. p. 448–55.
400. Salama M, Eid H, Ramadan R, Darwish A, Hassanien A. Hybrid intelligent intrusion detection scheme. Soft comput ind appl. 2011;293–303.
401. Samuel AL. Some studies in machine learning using the game of checkers. IBM J Res Dev. 1959;3(3):210–29.
402. Sangkatsanee P, Wattanapongsakorn N, Charnsripinyo C. Practical real-time intrusion detection using machine learning approaches. Comput Commun. 2011;34(18):2227–35.
403. Schapire RE. The strength of weak learnability. Mach Learn. 1990;5(2): 197–227.
404. Schatzmann D, Mühlbauer W, Spyropoulos T, Dimitropoulos X. Digging into https: Flow-based classification of webmail traffic. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement; 2010. p. 322–27.
405. Schölkopf B, Platt JC, Shawe-Taylor J, Smola AJ, Williamson RC. Estimating the support of a high-dimensional distribution. Neural comput. 2001;13(7):1443–71.

406. Seligman M, Fall K, Mundur P. Alternative custodians for congestion control in delay tolerant networks. In: Proceedings of the 2006 SIGCOMM Workshop on Challenged Networks, CHANTS '06. New York: ACM; 2006. p. 229–36.
407. Servin A, Kudenko D. Multi-agent reinforcement learning for intrusion detection: A case study and evaluation. In: German Conference on Multiagent System Technologies. Springer; 2008. p. 159–70.
408. Shaikh J, Fiedler M, Collange D. Quality of experience from user and network perspectives. annals of telecommun-annales des telecommun. 2010;65(1-2):47–57.
409. Shbair WM, Cholez T, Francois J, Chrisment I. A multi-level framework to identify https services. In: IEEE/IFIP Network Operations and Management Symposium (NOMS) 2016. p. 240–8.
410. Shi R, Zhang J, Chu W, Bao Q, Jin X, Gong C, Zhu Q, Yu C, Rosenberg S. Mdp and machine learning-based cost-optimization of dynamic resource allocation for network function virtualization. In: Serv Comput (SCC) 2015 IEEE International Conference on. IEEE; 2015. p. 65–73.
411. Shon T, Moon J. A hybrid machine learning approach to network anomaly detection. Inf Sci. 2007;177(18):3799–821.
412. Silva AP, Obraczka K, Burleigh S, Hirata CM. Smart congestion control for delay- and disruption tolerant networks. In: 2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON). 2016. p. 1–9.
413. Smolensky P. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In: Rumelhart DE, McClelland JL, PDP Research Group C, editors. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1" MIT Press, Cambridge, MA, USA, chap Information Processing in Dynamical Systems: Foundations of Harmony Theory. 1986. p. 194–281.
414. Snow A, Rastogi P, Weckman G. Assessing dependability of wireless networks using neural networks. In: Military Communications Conference, 2005. MILCOM 2005. IEEE. IEEE; 2005. p. 2809–15.
415. Sommer R, Paxson V. Outside the closed world: On using machine learning for network intrusion detection. In: Security and Privacy (SP), 2010 IEEE Symposium on, IEEE; 2010. p. 305–16.
416. Sondik EJ. The optimal control of partially observable markov decision processes. PhD thesis. California: Stanford University; 1971.
417. Soysal M, Schmidt EG. Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. Perform Eval. 2010;67(6):451–67.
418. Sprint. IP network performance. 2017. https://www.sprint.net/performance/. Accessed 28 Dec 2017.
419. Srihari SN, Kuebert EJ. Integration of hand-written address interpretation technology into the united states postal service remote computer reader system. In: Proceedings of the 4th International Conference on Document Analysis and Recognition, ICDAR '97. Washington: IEEE Computer Society; 1997. p. 892–6.
420. Stanfill C, Waltz D. Toward memory-based reasoning. Commun ACM. 1986;29(12):1213–28.
421. Stein G, Chen B, Wu AS, Hua KA. Decision tree classifier for network intrusion detection with ga-based feature selection. In: Proceedings of the 43rd annual Southeast regional conference-Volume 2. ACM; 2005. p. 136–41.
422. Steinhaus H. Sur la division des corp materiels en parties. Bull Acad Polon Sci. 1956;1:801–4.
423. Stigler SM. Gauss and the invention of least squares. Ann Statist. 1981;9(3):465–74.
424. Stone P. Tpot-rl applied to network routing. In: ICML. 2000. p. 935–42.
425. Stone P, Veloso M. Team-partitioned, opaque-transition reinforcement learning. In: Proceedings of the third annual conference on Autonomous Agents. ACM; 1999. p. 206–12.
426. Stratonovich RL. Conditional markov processes. Theory Probab Appl. 1960;5(2):156–78.
427. Sun J, Zukerman M. An adaptive neuron aqm for a stable internet. In: Proceedings of the 6th International IFIP-TC6 Conference on Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet, Springer-Verlag, Berlin, Heidelberg, NETWORKING'07. 2007. p. 844–54.
428. Sun J, Chan S, Ko Kt, Chen G, Zukerman M. Neuron pid: A robust aqm scheme. In: Proceedings of the Australian Telecommunication Networks and Applications Conference (ATNAC) 2006. 2006. p. 259–62.
429. Sun J, Chan S, Zukerman M. Iapi: An intelligent adaptive pi active queue management scheme. Comput Commun. 2012;35(18):2281–93.
430. Sun R, Tatsumi S, Zhao G. Q-map: A novel multicast routing method in wireless ad hoc networks with multiagent reinforcement learning. In: TENCON'02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering, vol 1. IEEE; 2002. p. 667–670.
431. Sun R, Yang B, Peng L, Chen Z, Zhang L, Jing S. Traffic classification using probabilistic neural networks. In: Natural computation (ICNC), 2010 sixth international conference on, vol. 4. IEEE; 2010. p. 1914–9.
432. Sun Y, Yin X, Jiang J, Sekar V, Lin F, Wang N, Liu T, Sinopoli B. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In: Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference. ACM; 2016. p. 272–85.
433. Sutton RS. Learning to predict by the methods of temporal differences. Mach learn. 1988;3(1):9–44.
434. Sutton RS, Barto AG. A temporal-difference model of classical conditioning. In: Proceedings of the ninth annual conference of the cognitive science society. Seattle, WA; 1987. p. 355–78.
435. Sutton RS, Barto AG. Reinforcement Learning: An Introduction, 2nd ed. MIT Press; 2016.
436. Tang TA, Mhamdi L, McLernon D, Zaidi SAR, Ghogho M. Deep learning approach for network intrusion detection in software defined networking. In: Wireless Networks and Mobile Communications (WINCOM), 2016 International Conference on. IEEE; 2016. p. 258–263.
437. Tarraf AA, Habib IW, Saadawi TN. Congestion control mechanism for atm networks using neural networks. In: Communications 1995. ICC '95 Seattle, 'Gateway to Globalization' 1995 IEEE International Conference on, vol 1. 1995. p. 206–10.
438. Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the kdd cup 99 data set. IEEE; 2009, pp. 1–6.
439. Telecommunication Networks Group - Politecnico di Torino. Skype testbed traces, TSTAT - TCP Statistic and Analysis Tool. 2008. http://tstat.tlc.polito.it/traces-skype.shtml. Accessed 01 Aug 2017.
440. Tesauro G. Practical issues in temporal difference learning. Mach Learn. 1992;8(3):257–77.
441. Tesauro G. Reinforcement learning in autonomic computing: A manifesto and case studies. IEEE Internet Comput. 2007;11(1):22–30.
442. Tesauro G, et al. Online resource allocation using decompositional reinforcement learning. 2005. pp. 886–91.
443. Testolin A, Zanforlin M, De Grazia MDF, Munaretto D, Zanella A, Zorzi M, Zorzi M. A machine learning approach to qoe-based video admission control and resource allocation in wireless systems. In: Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual, Mediterranean. IEEE; 2014. p. 31–38.
444. TFreak. Smurf tool. 2003. www.phreak.org/archives/exploits/denial/smurf.c.
445. Tibshirani R. Regression shrinkage and selection via the lasso. J R Stat Soc Ser B (Methodol). 1996;58(1):267–288.
446. Tong H, Brown TX. Adaptive call admission control under quality of service constraints: a reinforcement learning solution. IEEE J sel Areas Commun. 2000;18(2):209–21.
447. Tsai CF, Hsu YF, Lin CY, Lin WY. Intrusion detection by machine learning: A review. Expert Systems with Applications. 2009;36(10):11,994–12,000.
448. Tsetlin M. Automaton Theory and Modeling of Biological Systems. Automaton Theory and Modeling of Biological Systems. Academic Press; 1973.
449. Turing AM. Computing machinery and intelligence. Mind. 1950;59(236):433–60.
450. UCI KDD Archive. 2005. https://kdd.ics.uci.edu/. Accessed 01 Aug 2017.
451. University of California San Diego Supercomputer Center. CAIDA: Center for Applied Internet Data Analysis. 2017. http://www.caida.org. Accessed 01 Aug 2017.
452. Vassis D, Kampouraki A, Belsis P, Skourlas C. Admission control of video sessions over ad hoc networks using neural classifiers. IEEE; 2014, pp. 1015–20.
453. Vega MT, Mocanu DC, Liotta A. Unsupervised deep learning for real-time assessment of video streaming services. Multimedia Tools Appl. 2017;1–25.
454. Vengerov D. A reinforcement learning approach to dynamic resource allocation. Eng Appl Artif Intell. 2007;20(3):383–90.

455. Viterbi A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans Inf Theory. 1967;13(2): 260–9.
456. Wagner C, François J, Engel T, et al. Machine learning approach for ip-flow record anomaly detection. In: International Conference on Research in Networking. Springer; 2011. p. 28–39.
457. WAND Network Research Group. WITS: Waikato Internet Traffic Storage. 2017. https://wand.net.nz/wits. Accessed 01 Aug 2017.
458. Wang J, Qiu Y. A new call admission control strategy for lte femtocell networks. In: 2nd international conference on advances in computer science and engineering. 2013.
459. Wang K, Stolfo SJ. Anomalous payload-based network intrusion detection. In: RAID, vol 4. Springer; 2004. p. 203–22.
460. Wang M, Cui Y, Wang X, Xiao S, Jiang J. Machine learning for networking: Workflow, advances and opportunities. IEEE Netw. 2018a;32(2):92–9.
461. Wang P, Wang T. Adaptive routing for sensor networks using reinforcement learning. In: Computer and Information Technology, 2006. CIT'06. The Sixth IEEE International Conference on. IEEE; 2006. p. 219.
462. Wang P, Lin SC, Luo M. A framework for qos-aware traffic classification using semi-supervised machine learning in sdns. In: Services Computing (SCC), 2016 IEEE International Conference on. IEEE; 2016. p. 760–5.
463. Wang R, Valla M, Sanadidi MY, Ng BKF, Gerla M. Efficiency/friendliness tradeoffs in TCP westwood. In: Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications. 2002. p. 304–11.
464. Wang R, Liu Y, Yang Y, Zhou X. Solving the app-level classification problem of p2p traffic via optimized support vector machines. In: Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on, IEEE, vol 2; 2006. p. 534–9.
465. Wang X, Zhang Q, Ren J, Xu S, Wang S, Yu S. Toward efficient parallel routing optimization for large-scale sdn networks using gpgpu. J Netw Comput Appl. 2018b.
466. Wang Y, Martonosi M, Peh LS. Predicting link quality using supervised learning in wireless sensor networks. ACM SIGMOBILE Mob Comput Commun Rev. 2007;11(3):71–83.
467. Wang Y, Xiang Y, Yu S. Internet traffic classification using machine learning: a token-based approach. IEEE; 2011. p. 285–9.
468. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. IEEE trans image process. 2004;13(4):600–12.
469. Wang Z, Zhang M, Wang D, Song C, Liu M, Li J, Lou L, Liu Z. Failure prediction using machine learning and time series in optical network. Optics Express. 2017;25(16):18,553–18,565.
470. Watanabe A, Ishibashi K, Toyono T, Kimura T, Watanabe K, Matsuo Y, Shiomoto K. Workflow extraction for service operation using multiple unstructured trouble tickets. IEEE; 2016, pp. 652–8.
471. Watkins CJ. Models of delayed reinforcement learning. PhD thesis: Psychology Department, Cambridge University; 1989.
472. Werbos P. Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis: Harvard University; 1975.
473. Wernecke KD. A coupling procedure for the discrimination of mixed data. Biometrics. 1992;48(2):497–506.
474. WIDE Project. MAWI Working Group Traffic Archive. 2017. http://mawi.wide.ad.jp/mawi. Accessed 01 Aug 2017.
475. Williams M. Net tools 5. 2011. https://www.techworld.com/download/networking-tools/net-tools-5-3248881/. Accessed 1 Mar 2017.
476. Williams N, Zander S, Armitage G. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. ACM SIGCOMM Comput Commun Rev. 2006;36(5): 5–16.
477. Winstein K, Balakrishnan H. Tcp ex machina: Computer-generated congestion control. In: Proceedings of the ACM SIGCOMM 201 Conference on SIGCOMM, SIGCOMM '13. New York: ACM; 2013. p. 123–34.
478. Witten IH. An adaptive optimal controller for discrete-time markov environments. Inf Control. 1977;34(4):286–95.
479. Wolpert DH, Tumer K, Frank J. Using collective intelligence to route internet traffic. Adv neural inf process syst. 1999;952–60.
480. Wolski R. Dynamically forecasting network performance using the network weather service. Cluster Comput. 1998;1(1):119–32.
481. Wu C, Meleis W. Fuzzy kanerva-based function approximation for reinforcement learning. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). 2009. p. 1257–8.
482. Xia B, Wahab MH, Yang Y, Fan Z, Sooriyabandara M. Reinforcement learning based spectrum-aware routing in multi-hop cognitive radio networks. In: Cognitive Radio Oriented Wireless Networks and Communications, 2009. CROWNCOM'09. 4th International, Conference on. IEEE; 2009. p. 1–5.
483. Xu K, Tian Y, Ansari N. Tcp-jersey for wireless ip communications. IEEE J Sel Areas Commun. 2004;22(4):747–56.
484. Xu L, Krzyzak A, Suen CY. Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE Transactions on Systems. Man Cybern. 1992;22(3):418–35.
485. Yan Q, Lei Q. A new active queue management algorithm based on self-adaptive fuzzy neural-network pid controller. In: 2011 International Conference on, Internet Technology and Applications; 2011. p. 1–4.
486. Yang P, Luo W, Xu L, Deogun J, Lu Y. Tcp congestion avoidance algorithm identification. In: 2011 31st International Conference on Distributed Computing Systems. 2011. p. 310–21.
487. Yi C, Afanasyev A, Moiseenko I, Wang L, Zhang B, Zhang L. A case for stateful forwarding plane. Comput Commun. 2013;36(7):779–791.
488. YouTube LLC. YouTube. 2017. https://www.youtube.com/. Accessed 01 Aug 2017.
489. Yu E, Chen CR. Traffic prediction using neural networks. In: Proceedings of IEEE GLOBECOM. IEEE; 1993. p. 991–5.
490. Yu X, Qiao C, Liu Y. Tcp implementations and false time out detection in obs networks. In: IEEE INFOCOM 2004, vol 2. 2004. p. 774–84.
491. Zalewski M, Stearns W. p0f. 2014. http://lcamtuf.coredump.cx/p0f. Accessed 1 Mar 2017.
492. Zander S, Nguyen T, Armitage G. Automated traffic classification and application identification using machine learning. IEEE; 2005, pp. 250–7.
493. Zanero S, Savaresi SM. Unsupervised learning techniques for an intrusion detection system: ACM; 2004, pp. 412–9.
494. Zhang C, Jiang J, Kamel M. Intrusion detection using hierarchical neural networks. Pattern Recogn Lett. 2005;26(6):779–91.
495. Zhang J, Zulkernine M. Anomaly based network intrusion detection with unsupervised outlier detection. In: Communications, 2006. ICC'06. IEEE International Conference on. IEEE; 2006. p. 2388–93.
496. Zhang J, Chen C, Xiang Y, Zhou W, Xiang Y. Internet traffic classification by aggregating correlated naive bayes predictions. IEEE Trans Inf Forensic Secur. 2013;8(1):5–15.
497. Zhang J, Chen X, Xiang Y, Zhou W, Wu J. Robust network traffic classification. IEEE/ACM Trans Netw (TON). 2015;23(4):1257–70.
498. Zhani MF, Elbiaze H, Kamoun F. α_snfaqm: an active queue management mechanism using neurofuzzy prediction. In: 2007 12th IEEE Symposium on Computers and Communications. 2007. p. 381–6.
499. Zhou C, Di D, Chen Q, Guo J. An adaptive aqm algorithm based on neuron reinforcement learning. In: 2009 IEEE International Conference on Control and Automation; 2009. p. 1342–6.
500. Zhu Y, Zhang G, Qiu J. Network traffic prediction based on particle swarm bp neural network. JNW. 2013;8(11):2685–91.
501. Zineb AB, Ayadi M, Tabbane S. Cognitive radio networks management using an anfis approach with qos/qoe mapping scheme. IEEE; 2015, pp. 1–6.