Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets

Rahul Potharaju* Purdue University
Navendu Jain Microsoft Research
Cristina Nita-Rotaru Purdue University

Abstract

This paper presents NetSieve, a system that aims to do automated problem inference from network trouble tickets. Network trouble tickets are diaries comprising fixed fields and free-form text written by operators to document the steps while troubleshooting a problem. Unfortunately, while tickets carry valuable information for network management, analyzing them to do problem inference is extremely difficult—fixed fields are often inaccurate or incomplete, and the free-form text is mostly written in natural language.

This paper takes a *practical* step towards automatically analyzing natural language text in network tickets to infer the problem symptoms, troubleshooting activities and resolution actions. Our system, NetSieve, combines statistical natural language processing (NLP), knowledge representation, and ontology modeling to achieve these goals. To cope with ambiguity in free-form text, NetSieve leverages learning from human guidance to improve its inference accuracy. We evaluate NetSieve on 10K+ tickets from a large cloud provider, and compare its accuracy using (a) an expert review, (b) a study with operators, and (c) vendor data that tracks device replacement and repairs. Our results show that NetSieve achieves 89%-100% accuracy and its inference output is useful to learn global problem trends. We have used NetSieve in several key network operations: analyzing device failure trends, understanding why network redundancy fails, and identifying device problem symptoms.

1 Introduction

Network failures are a significant contributor to system downtime and service unavailability [12, 13, 47]. To track network troubleshooting and maintenance, operators typically deploy a trouble ticket system which logs all the steps from opening a ticket (e.g., customer complaint, SNMP alarm) till its resolution [21]. Trouble tickets comprise two types of fields: (a) structured data often generated automatically by alarm systems such as ticket id, time of alert, and syslog error, and (b) free-form text written by operators to record the diagnosis steps and communication (e.g., via IM, email) with the customer or other technicians while mitigating the problem. Even though the free-form field is less regular and precise compared to the fixed text, it usually provides a *detailed* view of the problem: what happened? what troubleshooting was done? and what was the resolution? Figure 1 shows a ticket describing continuous reboots of a load balancer even after reseating its power supply units; bad memory as the root cause; and memory replacement as the fix; which would be hard to infer from coarse-grained fixed data.

STRUCTURED

Ticket Title:	Ticket #xxxxxx NetDevice; LoadBalancer Down 100% Summary: Indicates that the root cause is a failed system		
Problem Type	**Problem SubType**	**Priority**	**Created**
Severity - 2	2: Medium		

UNSTRUCTURED (Diary)

Operator 1: Both power supplies have been reseated
Operator 1: The device has been powered back up and it does not appear that it has come back online. Please advise.
Operator 2: Ok. Let me see what I can do.
--- Original Message ---
From: Vendor Support
Subject: Regarding Case Number #yyyyyy
Title: **Device** v9.4.5 **continuously rebooting**
As discussed, the **device** has **bad memory chips** as such we **replace** it. Please completely fill the **RMA** form below and **return** it.

Figure 1: An example network trouble ticket.

Unfortunately, while tickets contain valuable information to infer problem trends and improve network management, mining them automatically is extremely hard. On one hand, the fixed fields are often inaccurate or incomplete [36]. Our analysis (§2.1) on a large ticket dataset shows that the designated problem type and subtype fields had incorrect or inconclusive information in 69% and 75% of the tickets, respectively. On the other hand, since the free-form text is written in natural language, it is often ambiguous and contains typos, grammatical errors, and words (e.g., "cable", "line card", "power supply") having domain-specific meanings different from the dictionary.

Given these fundamental challenges, it becomes difficult to automatically extract *meaning* from raw ticket text even with advanced NLP techniques, which are designed

*Work done during an internship at Microsoft Research, Redmond.

Table 1: Examples of network trouble tickets and their inference output from NetSieve.

		Inference output from NetSieve		
	Ticket Title	**Problems**	**Activities**	**Actions**
1	SNMPTrap LogAlert 100%: Internal link 4.8 is unavailable.	link down, failover, bad sectors	swap cable, upgrade fiber, run fsck, verify HDD	replace cable, HDD
2	HSRPEndpoint SwitchOver 100%: The status of HSRP endpoint has changed since last polling.	firmware error, interface failure	verify and break-fix supervisor engine	replace supervisor engine, reboot switch
3	StandbyFail: Failover condition, this standby will not be able to go active.	unexpected reboot, performance degraded	verify load balancer, run config script	rma power supply unit
4	The machine can no longer reach internet resources. Gateway is set to load balancer float IP.	verify static route	reboot server, invoke failover, packet capture	rehome server, reboot top-of-rack switch
5	Device console is generating a lot of log messages and not authenticating users to login.	sync error, no redundancy	power down device, verify maintenance	replace load balancer
6	Kernel panic 100%: CPU context corrupt.	load balancer reboot, firmware bug	check performance, break-fix upgrade	upgrade BIOS, reboot load balancer
7	Content Delivery Network: Load balancer is in bad state, failing majority of keep-alive requests.	standby dead, misconfigured route	upgrade devices	replace standby and active, deploy hot-fix
8	OSPFNeighborRelationship Down 100%: This OSPF link between neighboring endpoints is down.	connectivity failure, packet errors	verify for known maintenance	replace network card
9	HighErrorRate: Summary: http://domain/characteristics.cgi?<device>.	packet errors	verify interface	cable and xenpak module replaced
10	AllComponentsDown: Summary: Indicates that all components in the redundancy group are down;	down alerts	verify for decommissioned devices	decommission load balancer

to process well-written text (e.g., news articles) [33]. Most prior work on mining trouble tickets use either keyword search and manual processing of free-form content [20, 27, 42], predefined rule set from ticket history [37], or document clustering based on manual keyword selection [36]. While these approaches are simple to implement and can help narrow down the types of problems to examine, they risk (1) inaccuracy as they consider only the presence of a keyword regardless of where it appears (e.g., "do not replace the cable" specifies a negation) and its relationship to other words (e.g., "checking for maintenance" does not clarify whether the ticket was *actually* due to maintenance), (2) a significant human effort to build the keyword list and repeating the process for new tickets, and (3) inflexibility due to predefined rule sets as they do not cover unexpected incidents or become outdated as the network evolves.

Our Contributions. This paper presents NetSieve, a problem inference system that aims to automatically analyze ticket text written in natural language to infer the problem symptoms, troubleshooting activities, and resolution actions. Since it is nearly impractical to understand any arbitrary text, NetSieve adopts a domain-specific approach to first build a knowledge base using existing tickets, automatically to the extent possible, and then use it to do problem inference. While a ticket may contain multiple pieces of useful information, NetSieve focuses on inferring three key features for summarization as shown in Table 1:

1. **Problems** denote the network entity (e.g., router, link, power supply unit) and its associated state, condition or symptoms (e.g., crash, defective, reboot) as identified by an operator e.g., bad memory, line card failure, crash of a load balancer.
2. **Activities** indicate the steps performed on the network entity during troubleshooting e.g., clean and swap cables, verify hard disk drive, run configuration script.
3. **Actions** represent the resolution action(s) performed on the network entity to mitigate the problem e.g., upgrade BIOS, rehome servers, reseat power supply.

To achieve this functionality, NetSieve combines techniques from several areas in a novel way to perform problem inference over three phases. First, it constructs a domain-specific knowledge base and an ontology model to interpret the free-form text using pattern mining and statistical NLP. In particular, it finds important domain-specific words and phrases (e.g., "supervisor engine", "kernel", "configuration") and then maps them onto the ontology model to specify relationships between them. Second, it applies this knowledge base to infer problems, activities and actions from tickets and exports the inference output for summarization and trend analysis. Third, to improve the inference accuracy, NetSieve performs incremental learning to incorporate human feedback.

Our evaluation on 10K+ network tickets from a large cloud provider shows that NetSieve performs automated problem inference with 89%-100% accuracy, and several network teams in that cloud provider have used its inference output to learn global problem trends: (1) compare device reliability across platforms and vendors, (2) analyze cases when network redundancy failover is ineffective, and (3) prioritize checking for the top-k problems

and failing components during network troubleshooting.

This paper makes the following contributions:

- A large-scale measurement study (§2) to highlight the challenges in analyzing structured data and free-form text in network trouble tickets.
- Design and implementation (§3) of NetSieve, an automated inference system that analyzes free-form text in tickets to extract the problem symptoms, troubleshooting activities and resolution actions.
- Evaluation (§4) of NetSieve using expert review, study with network operators and vendor data, and showing its applicability (§5) to improve network management.

Scope and Limitations: NetSieve is based on analyzing free-form text written by operators. Thus, its accuracy is dependent on (a) fidelity of the operators' input and (b) tickets containing sufficient information for inference. NetSieve leverages NLP techniques, and hence is subject to their well-known limitations such as ambiguities caused by *anaphoras* (*e.g.*, referring to a router as *this*), complex negations (*e.g.*, "device gets replaced" but later in the ticket, the action is negated by the use of an anaphora) and truth conditions (*e.g.*, "please replace the unit once you get more in stock" does not clarify whether the unit has been replaced). NetSieve inference rules may be specific to our ticket data and may not apply to other networks. While we cannot establish representativeness, this concern is alleviated to some extent by the size and diversity of our dataset. Finally, our ontology model represents one way of building a knowledge base, based on discussions with operators. Given that the ticket system is subjective and domain-specific, alternative approaches may work better for other systems.

2 Measurement and Challenges

In this section, we present a measurement study to highlight the key challenges in automated problem inference from network tickets. The dataset comprises 10K+ (absolute counts omitted due to confidentiality reasons) network tickets logged during April 2010-2012 from a large cloud provider. Next, we describe the challenges in analyzing fixed fields and free-form text in trouble tickets.

2.1 Challenges: Analyzing Fixed Fields

C1: Coarse granularity. The fixed fields in tickets contain attributes such as 'ProblemType' and 'ProblemSubType', which are either pre-populated by alarm systems or filled in by operators. Figure 2 shows the top-10 problem types and sub-types along-with the fraction of

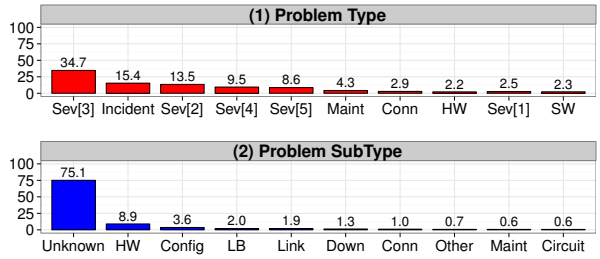

Figure 2: Problem types and subtypes listed in the tickets.

tickets. *Sev* denotes problem severity assigned based on SLAs with the customer. We observe that while problem types such as *Software*, *Hardware*, *Maintenance*, and *Incident* provide coarse granularity information about the problem type, other types e.g., *Sev[1-5]* are highly subjective reflecting operator's judgement and they account for 68.8% of the tickets. As a result, these fields are not useful to precisely infer the observed problems.

C2: Inaccuracy or Incompleteness. Figure 3 shows the problem categorization for a randomly selected subset of tickets labeled by a domain expert (top) and the field values from the tickets (bottom) for three different types of devices: (1) Access Routers (AR), (2) Firewalls, and (3) Load balancers (LB); the number of tickets is 300, 42, and 299, respectively.

We make two key observations. First, the Problem SubType field (in the bottom row) is *Unknown* in about 79%-87% of the tickets. As a result, we may incorrectly infer that devices failed due to unknown problems, whereas the problems were precisely reported in the expert labeled set based on the same ticket data. Second, the categories annotated by the expert and ticket fields for each device type have little overlap, and even when there is a common category, there is a significant difference in the fraction of tickets attributed to that category e.g., 'Cable' accounts for 0.6% of the LB tickets whereas the ground truth shows their contribution to be 9.7%.

The reason that these fields are inaccurate or incomplete is that operators work under a tight time schedule, and they usually have a narrow focus of mitigating a problem rather than analyzing failure trends. Thus, they may not have the time, may not be motivated, or simply forget to input precise data for these fields after closing the tickets. Further, some fixed fields have a drop-down menu of pre-defined labels and every problem may not be easily described using them.

2.2 Challenges: Analyzing Free-form Text

In comparison to structured data, the free-form text in network tickets is descriptive and ambiguous: it has

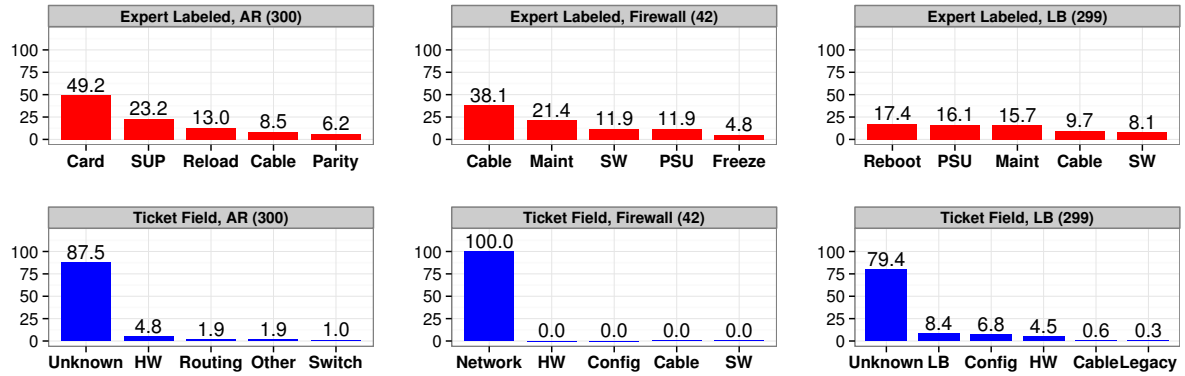

Figure 3: Categorization of the 'Problem SubType' field in tickets for (a) Access Routers (AR), (b) Firewalls, and (c) Load balancers (LB). The top and bottom rows show the major problem subtypes as labeled by an expert and the ticket field, respectively.

domain-specific words and synonyms mixed with regular dictionary words, spelling and grammar errors, and writings from different operators.

Specifically, we highlight the following challenges in mining free-form text in trouble tickets:

C1: Diversity of content. A ticket may contain a variety of semantic elements such as emails, IMs, device debug logs, devices names, and operator notes.

C2: Domain-specific words. Without a prior list of domain-specific keywords, training spell checkers can be hard e.g., DMZ and DNS are both valid technical keywords, but they cannot be found in the dictionary.

C3: Redundant text. Tickets often contain text fragments that appear with high frequency. We observe three types of frequently occurring fragments (see Figure 1): templates, emails and device logs. Templates are text fragments added by operators to meet triage guidelines, but they often do not contain any problem-specific information. Many emails are asynchronous replies to a previous message and thus, it may be hard to reconstruct the message order for inference. Log messages are usually appended to a ticket in progress. Therefore, text mining using metrics such as term frequency may incorrectly give more weightage to terms that appear in these logs.

Overall, these challenges highlight the difficulty in automatically inferring problems from tickets. While we studied only our ticket dataset, our conversation with operators (having a broader industry view and some having worked at other networks), suggests that these challenges are similar to those of many other systems.

3 Design and Implementation

In this section, we first give an overview of NetSieve and then describe its design and implementation.

3.1 Design Goals

To automatically analyze free-form text, NetSieve should meet the following design goals:

1. *Accuracy*: The inference system needs to be accurate as incorrect inference can lead to bad operator decisions, and wasted time and effort in validating inference output for each ticket, thus limiting practicality.
2. *Automation*: Although we cannot completely eliminate humans from the loop, the system should be able to operate as autonomously as possible.
3. *Adaptation*: As the network evolves, the system should be able to analyze new types of problems and leverage human feedback to acquire new knowledge for continuously improving the inference accuracy.
4. *Scalability*: The system should be scalable to process a large number of tickets where each ticket may comprise up to a million characters, in a reasonable time.
5. *Usability*: The output from the inference system should provide a user-friendly interface (e.g., visualization, REST, plaintext) to allow the operator to browse, filter and process the inference output.

3.2 Overview

NetSieve infers three key features from network trouble tickets: (1) Problem symptoms indicating what problem occurred, (2) Troubleshooting activities describing the diagnostic steps, and (3) Resolution actions denoting the fix applied to mitigate the problem.

Figure 4 shows an overview of the NetSieve architecture. NetSieve operates in three phases. First, the knowledge building phase constructs a domain-specific knowledge base and an ontology model using existing tickets and input from a domain-expert. Second, the operational

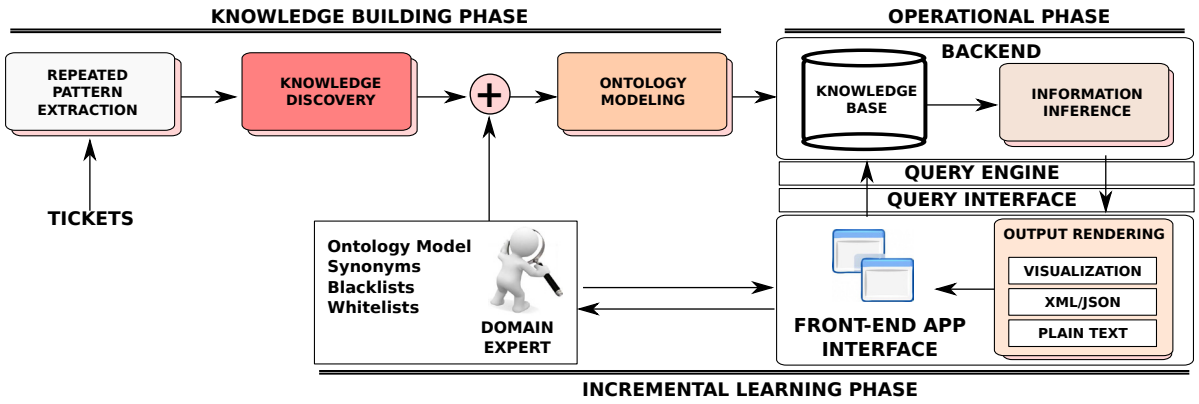

Figure 4: NetSieve Architecture: The first phase builds a domain-specific knowledge base using existing tickets. The second phase uses the knowledge base to make problem inference. The third phase leverages human guidance to improve the inference accuracy.

phase uses the knowledge base to make problem inference from tickets. Third, the incremental learning phase improves the accuracy of knowledge base using human guidance. We next give a brief description of each of these phases.

Knowledge Building Phase: The goal of this phase is to analyze free-form text to extract important domain-specific phrases such as "power supply unit" and "load balancer" using repeated pattern mining (§3.3.1) and statistical NLP (§3.3.2). These domain-specific phrases are then mapped onto an ontology model (§3.3.3) that formally represents the relationships between network entities and stores them in a knowledge base. This phase is executed either when NetSieve is bootstrapped or to retrain the system using expert feedback.

Operational Phase: The goal of this phase is to perform problem inference (§3.4) from a ticket using the knowledge base. To export the inference output, NetSieve supports SQL (through the *Query Engine*) and HTTP GET requests (through a *Query Interface* such as REST [11]) and outputs results in a variety of data formats such as XML/JSON, and through data visualization for ticket summarization and trend analysis.

Incremental Learning Phase: To improve inference accuracy, it is important to continuously update the knowledge base to incorporate any new domain-specific terminologies. NetSieve provides an interface to allow a domain-expert to give feedback for improving the ontology model, synonyms, blacklists and whitelists. After each learning session, NetSieve performs problem inference using the updated knowledge base.

3.3 Knowledge Building Phase

Building a domain-specific knowledge phase requires addressing three key questions. First, what type of information should be extracted from the free-form text to enable problem inference? Second, how do we extract this information in a scalable manner from a large ticket corpus? Third, how do we model the relationships in the extracted information to infer *meaning* from the ticket content. Next we describe solutions to these questions.

3.3.1 Repeated Pattern Extraction

Intuitively, the phrases that would be most useful to build a knowledge base should capture domain-specific information and be related to *hot* (common) and important problem types. As mining arbitrary ticket text is extremely hard (§2), we first extract hot phrases and later apply filters (§3.3.2) to select the important ones.

DESIGN: To find the hot phrases from ticket text, we initially applied conventional text mining techniques for *n*-gram extraction. *N*-grams are arbitrary and recurrent word combinations [4] that are repeated in a given context [45]. Since network tickets have no inherent linguistic model, we extracted n-grams of arbitrary length for comprehensive analysis without limiting to bi-grams or tri-grams. We implemented several advanced techniques [9, 39, 45] from computational linguistics and NLP, and observed the following challenges:

1. Extracting all possible n-grams can be computationally expensive for a large n and is heavily dependent on the size of the corpus. We investigated using a popular technique by Nagao et al. [39] based on extracting word co-locations, implemented in C [56]. On our dataset, this algorithm did not terminate on a 100K word document after 36 hours of CPU time on a Xeon 2.67 GHz eight-core server with 48 GB RAM, as also observed by others [52].
2. Determining and fine-tuning the numerous thresholds and parameters used by statistical techniques [39, 45,

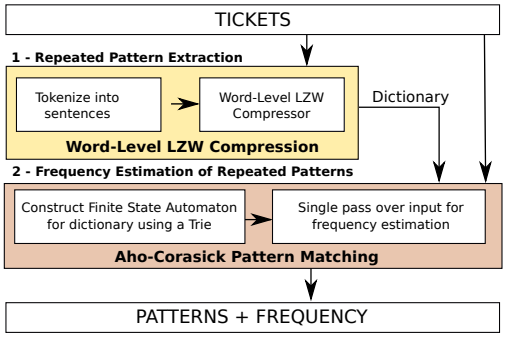

Figure 5: Two Phase Pattern Extraction. First, NetSieve tokenizes input into sentences and applies WLZW to build a dictionary of repeated patterns. Second, it uses the Aho-Corasick pattern matching algorithm to calculate their frequency.

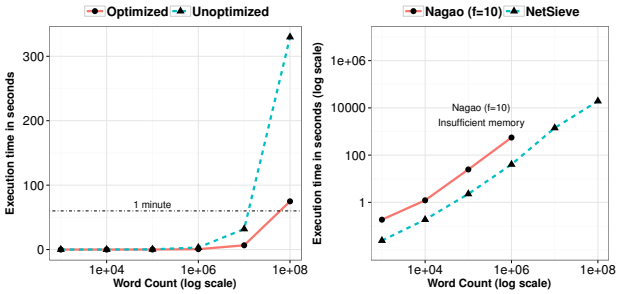

Figure 6: Performance of WLZW (a): Optimized implementation using Cython gives a performance boost of up to 5x-20x over a Python based solution as expected. Comparing NetSieve with N-gram extraction of Nagao et al. [39] (b): NetSieve is able to scale well beyond a million words in comparison to Nagao(f=10), where f is the phrase frequency.

55] is difficult when the corpus size is large.

3. Not all n-grams are *useful* due to their semantic context. For instance, n-grams such as "showing green" (LED status) and "unracking the" (unmounting the server) occurred frequently together but they do not contribute to the domain knowledge.

To address these challenges, we trade completeness in n-gram extraction for scalability and speedup. Our idea to extract hot patterns is to use a data compression algorithm, typically used to compress files by finding recurring patterns in the data and encoding them. A dictionary is maintained to map the repeated patterns to their output codes. Clearly, these dictionaries do not include all possible n-grams, but they contain hot patterns that are frequent enough to bootstrap the knowledge base.

Data compression algorithms typically operate at a byte or character level, and they do not output the frequency of patterns in their dictionary. To address these issues, NetSieve performs pattern extraction in two phases (Figure 5). First, it tokenizes input into sentences and leverages LZW [49] to develop a word-level LZW encoder (WLZW) that builds a dictionary of repeated patterns at the word-level. In the second phase, NetSieve applies the Aho-Corasick algorithm [2] to output frequency of the repeated phrases. Aho-Corasick is a string matching algorithm that runs in a single-pass and has a complexity linear in the pattern length, input size and the number of output matches.

Table 2: Examples of phrases extracted using the Two Phase Pattern Extraction algorithm.

Phrase Type	Phrase Pattern
Frequent messages	team this is to inform you that there has been a device down alarm reported on
Debug messages	errors 0 collisions 0 interface resets 0 babbles 0 late collision 0 deferred <device> sup 1a
Email snippets	if you need assistance outside of these hours please call into the htts toll free number 1 800

IMPLEMENTATION: We implemented the two phase pattern extraction algorithm in Cython [3], that allows translating Python into optimized C/C++ code. To optimize performance, we implemented the Aho-Corasick algorithm using suffix-trees [48] as opposed to the conventional suffix-arrays [30]. As expected, we achieved a 5x-20x performance improvement using Cython compared to a Python-based solution (Figure 6(a)).

Figure 6(b) shows the performance comparison of WLZW to *Nagao (f=10)* [39] which extracts all n-grams that occur at least 10 times. The latter terminated due to insufficient memory for a million word document. In comparison, NetSieve is able to process documents containing 100 million words in under 2.7 hours.

Note that WLZW is one way to extract hot patterns; we will explore other methods [16, 18, 52] in the future.

3.3.2 Knowledge Discovery

The goal of the knowledge discovery phase is to filter important domain-specific patterns from the extracted set in the previous phase; Table 2 shows examples of the extracted phrases. We define a pattern as *important* when it contributes to understanding of the "central topic" in a ticket. Consider the following excerpt from a ticket:

> We found that the **device** <**name**> **Power LED** is **amber** and it is in **hung state**. This device has **silver power supply**. We need to change the **silver power supply** to **black**. We will let you know once the **power supply** is **changed**.

The central topic of the above excerpt is "device failure that requires a power supply unit to be changed" and the

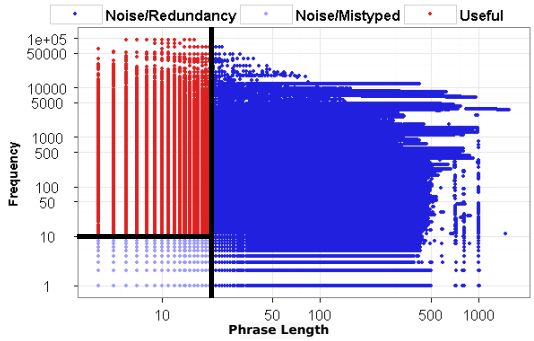

Figure 7: Filtering phrases using phrase length and frequency.

phrases in bold are relevant. While the pattern extractor outputs these phrases as repeated patterns, the key challenge is how to distinguish them from noisy patterns.

DESIGN: An intuitive method is to select the most frequently occurring patterns as important. However, we observed that many of them were warning messages which did not contribute to the central topic. Therefore, we apply a pipeline of three filters to identify the important domain-specific patterns.

Phrase Length/Frequency Filter: The idea behind applying this filter is that both the length and frequency of a phrase can act as good indicators of its importance. Intuitively, we are interested in phrases of short-length, but having a high-frequency. The rationale is that the other length-frequency combinations are either noise, occur due to typos, or can be constructed using short phrases.

We did not use a spell checker because an untrained or an undertrained one may incorrectly modify domain-specific words such as DNS and DMZ, and the probability of an important domain-specific phrase having typos in a large fraction of tickets is likely small. We plot the distribution of length and frequency of phrases (Figure 7) and then manually inspect a random subset in each quartile to derive heuristics for threshold-based filtering (Table 3).

Part-Of-Speech (PoS) Filter: The second filter is based on the seminal work of Justeson *et al.* [23]. They postulate that technical terms or domain-specific phrases have no satisfactory formal definition and can only be intuitively characterized: they generally occur only in specialized types of usage and are often specific to subsets of domains. Specifically, they conclude that most technical phrases contain only nouns and adjectives after analyzing four major technical dictionaries and subsequently provide a set of seven patterns outlined

Table 3: Thresholds for Phrase Length/Frequency filter.

Filtering Rule	Reason
Length $>$ 20 words	Likely Templates (long repeated patterns)
Single word phrases *i.e.*, unigrams	Important unigrams occur as part of a bi-gram or a tri-gram.
Frequency $<$ 10 i.e., rare words or phrases	Likely an isolated incident and not a frequently occurring problem trend
Contain numbers	Domain-specific phrases rarely contain numbers

Table 4: NetSieve converts the Justeson-Katz PoS patterns to Penn Treebank PoS patterns to filter technical phrases.

Justeson-Katz Patterns	NetSieve Patterns	Example
Adjective Noun	JJ NN[PS]*	mobile network
Noun Noun	NN[PS]* NN[PS]*	demo phase
Adjective Adjective Noun	JJ JJ NN[PS]*	fast mobile network
Adjective Noun Noun	JJ NN[PS]* NN[PS]*	accessible device logs
Noun Adjective Noun	NN[PS]* JJ NN[PS]*	browser based authentication
Noun Noun Noun	NN[PS]* NN[PS]* NN[PS]*	power supply unit
Noun Preposition Noun	NN[PS]* IN NN[PS]*	device down alert
JJ: Adjective; **NN**: Singular Noun; **NNP**: Proper singular noun; **NNPS**: Proper plural noun; **IN**: Preposition		

in Table 4. We build upon these patterns and map them to state-of-the-art Penn Treebank tagset [34], a simplified part-of-speech tagset for English, using regular expressions (Table 4). Further, this mapping allows our implementation to leverage existing part-of-speech taggers of natural language toolkits such as NLTK [29] and SharpNLP [44]. Filtering takes place in two steps: (1) each input phrase is tagged with its associated part-of-speech tags, and (2) the part-of-speech pattern is discarded if it fails to match a pattern.

Entropy Filter: The third filter uses information theory to filter statistically insignificant phrases, and sorts them based on importance to aid manual labeling. We achieve this by computing two metrics for each phrase [52]:

1. **Mutual Information (MI)**: $MI(x,y)$ compares the probability of observing word x and word y together (the joint probability) with the probabilities of observing x and y independently. For a phrase pattern, MI is computed by the following formula:

$$MI(xYz) = \log\left(\frac{tf(xYz) * tf(Y)}{tf(xY) * tf(Yz)}\right) \quad (1)$$

where xYz is a phrase pattern, x and z are a word/char-

acter and Y is a sub-phrase or sub-string, tf denotes the *term-frequency* of a word or phrase in the corpus.

2. **Residual Inverse Document Frequency (RIDF)**: RIDF is the difference between the observed IDF and what would be expected under a Poisson model for a random word or phrase with comparable frequency. RIDF of a phrase is computed as follows:

$$RIDF = -\log\left(\frac{df}{D}\right) + \log\left(1 - \exp\left(\frac{-tf}{D}\right)\right) \quad (2)$$

where df denotes the document-frequency (the number of tickets which contain the phrase) and D as the total number of tickets.

Phrases with high RIDF or MI have distributions that cannot be attributed to chance [52]. In particular, MI aims to pick vocabulary expected in a dictionary, while RIDF aims to select domain-specific keywords, not likely to exist in a general dictionary. We investigated both metrics as they are orthogonal and that each tends to separately pick interesting phrases [52].

IMPLEMENTATION: The three filters are applied as a sequential pipeline and are implemented in Python. For PoS tagging, we utilize the Stanford Log-Linear PoS Tagger [46] as an add-on module to the Natural Language Toolkit [29] and implement a multi-threaded tagger that uses the phrase length/frequency filter to first filter a list of candidate phrases for tagging.

After applying the threshold-based filtering and PoS filters on the input 18.85M phrases, RIDF and MI are computed for the remaining 187K phrases. This step significantly reduced the computational cost compared to prior work [9, 39, 45, 52], which aim to compute statistics for all n-grams. Similar to [52], we did not observe strong correlation between RIDF and MI (Figure 8 (top)). We relied solely on RIDF because most phrases with high MI were already filtered by RIDF and the remaining ones contained terms not useful in our context.

The bottom graph of Figure 8 shows the CCDF plot of RIDF which can be used to set a threshold to narrow down the phrase list for the next stage of human review. Determining the threshold poses a trade off between the completeness of the domain-dictionary and the human effort required to analyze the extracted patterns. In our prototype, we set the threshold based on RIDF such that 3% (5.6K) of the total phrases (187K) are preserved. Further, we sort these phrase patterns based on RIDF for expert review as phrases with higher values get labeled quickly. An expert sifted through the 5.6K phrases (Figure 9) and selected 1.6K phrase patterns that we consider as ground truth, in less than four hours. We observed that

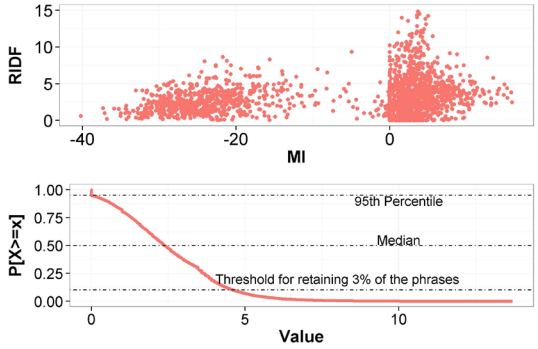

Figure 8: Absence of correlation between RIDF and MI metrics (top). Using a CCDF plot of RIDF to determine a threshold for filtering the phrases for expert review (bottom).

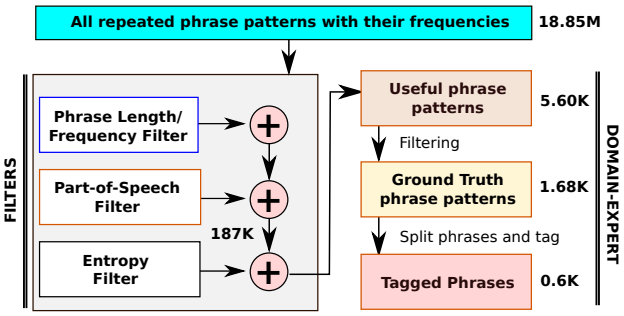

Figure 9: The pipeline of filtering phrases to determine a list of ground truth phrase patterns which are then split and tagged manually with the NetSieve-Ontology classes.

most of the discarded patterns were redundant as they can be constructed from the ground truth patterns.

While we leverage manual review to obtain the ground truth, this is a necessary step for any supervised technique. We plan to explore other techniques such as Named-Entity Recognition [35] and using domain experts for crowdsourcing [26] to automate this step.

3.3.3 Ontology Modeling

The goal of building ontology models is to determine *semantic interpretation* of important domain-specific phrases generated by the knowledge discovery stage. For instance, between the terms *slot* and *memory slot*, we are looking for the latter term with high specificity. Intuitively, we need to model an ontology where domain-specific phrases have a concrete meaning and can be combined together to enable semantic interpretation.

DESIGN: Developing an ontology model involves three steps [17, 40]: (1) defining classes in the ontology, (2) arranging the classes in a taxonomic (superclass, subclass) hierarchy and (3) defining interactions amongst the

Table 5: Classes in the NetSieve-Ontology model

Class	Sub-Class	Interpretation	Example
Entity	Replaceable	Tangible object that can be created/destroyed/replaced	Flash memory, Core router
	Virtual	Intangible object that can be created/destroyed/replaced	Port channel, configuration
	Maintenance	Tangible object that can act upon other entities	field engineer, technician
Action	Physical	Requires creating/destroying an entity	decommission, replace, rma
	Maintenance	Requires interacting with an entity and altering its state	clean, deploy, validate, verify
Condition	Problem	Describes condition that is known to have a negative effect	inoperative, reboot loop
	Maintenance	Describes condition that describes maintenance	break-fix
Incident	False Positive	State known to not have any problems	false positive, false alert
	Error	State known to cause a problem	error, exception
Quantity	Low	Describes the quantity of an entity/action	low, minor
	Medium		medium
	High		high, major
Negation	Synthetic	Uses verb or noun to negate a condition/incident/action	absence of, declined, denied
	Analytic	Uses 'not' to negate a condition/incident/action	not
Sentiment	Positive	Adds strength/weakness to an action/incident	confirm, affirmative
	Neutral		not sure
	Negative		likely, potential

classes. Note that defining an ontology is highly domain-specific and depends on extensions anticipated by the domain-expert. We designed and embedded one such ontology into NetSieve based on discussion with operators. Below, we discuss the design rationale behind the classes, taxonomies and interactions in our model.

Classes and Taxonomies: A class describes a concept in a given domain. For example, a class of entities can represent all devices (*e.g.*, routers, load balancers and cables) and a class of conditions can represent all possible states of an entity (*e.g.*, bad, flapping, faulty). A domain-expert sifted through the 1.6K phrases from previous stage and after a few iterations, identified seven classes to describe the phrases, shown in Table 5.

Taxonomic Hierarchy: To enable fine-grained problem inference, *Entity* is divided into three sub-classes: *Replaceable* denoting entities that can be physically replaced, *Virtual* denoting entities that are intangible and *Maintenance* denoting entities that can "act" upon other entities. *Actions* are sub-classed in a similar way. The rest of the classes can be considered as qualifiers for *Entities* and *Actions*. Qualifiers, in general, act as adjectives or adverbs and give useful information about an *Entity* or *Action*. In the final iteration, our domain-expert split each of the 1.6K long phrase patterns into their constituent small phrase patterns and tagged them with the most specific class that captured the phrase e.g.,
"and gbic replacement" → [(and, OMIT WORD), (gbic, ReplaceableEntity),(replacement, PhysicalAction)].

Most of these tagged phrases are domain-specific multi-word phrases and are not found in a dictionary. While the words describing Entities were not ambiguous, we found a few cases where other classes were ambiguous. For instance, phrases such as "power supply" (hardware unit or power line), "bit errors" (memory or network link), "port channel" (logical link bundling or virtual link), "flash memory" (memory reset or type of memory), "device reload" and "interface reset" (unexpected or planned), and "key corruption" (crypto-key or license-key) were hard to understand without a proper context. To address this ambiguity, we use the text surrounding these phrases to infer their intent (§3.4).

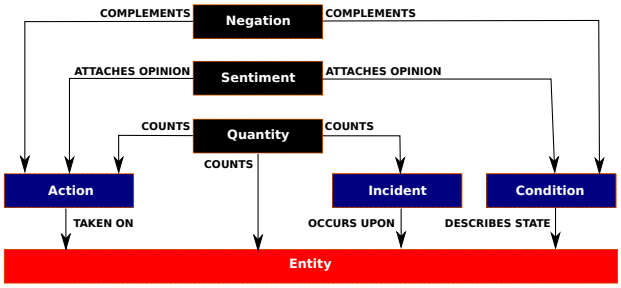

Figure 10: Ontology Model depicting interactions amongst the NetSieve-Ontology Classes.

Finally, using these mappings, NetSieve embeds a *ClassTagger* module that given an input, outputs tags for words that have an associated class mapping.

Interactions: An interaction describes relationships amongst the various classes in the ontology model. For instance, there are valid interactions (an *Action* can be caused upon an *Entity*) and invalid interactions (an *Entity* cannot describe a *Sentiment*). Figure 10 shows our model comprising interactions amongst the classes.

IMPLEMENTATION: We obtained 0.6K phrases from the 1.6K phrases in §3.3.2 categorized into the seven classes. We implemented the *ClassTagger* using a trie constructed using NetSieve knowledge base of domain-

Table 6: Concepts for the NetSieve-Ontology

Concept	Pattern	Example
Problems	[Replaceable — Virtual — Maintenance] Entity preceded/succeeded by ProblemCondition	The (device) was (faulty)
Activities	[Replaceable — Virtual — Maintenance] Entity preceded/succeeded by MaintenanceAction	(check) (device) connectivity and (clean) the (fiber)
Actions	[Replaceable — Virtual — Maintenance] Entity preceded/succeeded by PhysicalAction	An (RMA) was initiated for the (load balancer)

specific phrases, and a dictionary of their ontology mappings. The tagging procedure works in three steps. First, the input is tokenized into sentences. Second, using the trie, a search is performed for the longest matching phrase in each sentence to build a list of domain-specific keywords e.g., in the sentence "the power supply is down", both "supply" and "power supply" are valid domain keywords, but the ClassTagger marks "power supply" as the relevant word. Finally, these keywords are mapped to their respective ontology classes using the dictionary. For instance, given the snippet from §3.3.2, the *ClassTagger* will produce the following output:

We found that the (device) **/ ReplaceableEntity** <**name**> **(Power LED) / ReplaceableEntity** is **(amber) / Condition** and it is in **(hung state) / ProblemCondition**. This device has **(silver) / Condition (power supply) / ReplaceableEntity**. We need to change the **(silver) / Condition (power supply) / ReplaceableEntity** to **(black) / Condition**. We will let you know once the **(power supply) / ReplaceableEntity** is **(changed) / PhysicalAction**.

3.4 Operational Phase

The goal of this phase is to leverage the knowledge base to do automated problem inference on trouble tickets. A key challenge to address is how to establish a relationship between the ontology model and the physical world. In particular, we want to map certain interactions from our ontology model to concepts that allow summarizing a given ticket.

DESIGN: Our discussion with operators revealed a common ask to answer three main questions: (1) What was observed when a problem was logged?, (2) What activities were performed as part of troubleshooting? and (3) What was the final action taken to resolve the problem? Based on these requirements, we define three key concepts that can be extracted using our ontology model (Table 6): (1) *Problems* denote the state or condition of an entity, (2) *Activities* describe the troubleshooting steps, and (3) *Actions* capture the problem resolution.

The structure of concepts can be identified by sampling tickets describing different types of problems. We randomly sampled 500 tickets out of our expert-labeled ground truth data describing problems related to different device and link types. We pass these tickets through NetSieve's *ClassTagger* and get a total of 9.5K tagged snippets. We observed a common linguistic structure in them: in more than 90% of the cases, the action/condition that relates to an entity appears in the same sentence *i.e.,* information can be inferred about an entity based on its neighboring words. Based on this observation, we derived three patterns (Table 6) that capture all the cases of interest. Intuitively, we are interested in finding instances where an action or a condition precedes/succeeds an entity. Based on the fine granularity of the sub-classes, the utility of the concepts extracted increases i.e., *PhysicalAction was taken on a ReplacableEntity* is more important than *Action was taken on an Entity*.

This type of a proximity-search is performed once for each of the three concepts. First, the *ClassTagger* produces a list of phrases along with their associated tags. Second, we check to see if the list of tags contain an action/condition. Once such a match is found in a sentence, the phrase associated with the action/condition is added to a dictionary as a key and all entities within its neighborhood are added as corresponding values. We implemented several additional features like negation detection [15] and synonym substitution to remove any ambiguities in the inference output.

EXAMPLE: *"The load balancer was down. We checked the cables. This was due to a faulty power supply unit which was later replaced"*, is tagged as *"The (load balancer) / ReplaceableEntity was (down) / ProblemCondition. We (checked) / MaintenanceAction the (cables) / ReplaceableEntity. This was due to a (faulty) / ProblemCondition (power supply unit) / ReplaceableEntity which was later (replaced) / PhysicalAction"*. Next, a dictionary is built for each of the three concepts. Two classes are associated if they are direct neighbors. In this example, the output is the following:
[+] Problems - {down: load balancer, power supply unit}
[+] Activities - {checked: cable}
[+] Actions - {replaced: power supply unit}
In the final stage, the word "replaced" is changed into "replace" and "checked" into "check" using a dictionary of synonyms provided by the domain-expert to remove any ambiguities.

IMPLEMENTATION: We implemented our inference logic using Python. Each ticket is first tokenized into sentences and each sentence is then used for concept inference. After extracting the concepts and their associated entities, we store the results in a SQL table. Our implementation is able to do problem inference at the rate of 8 tickets/sec on average on our server, which scales to 28,800 tickets/hour; note that this time depends on the ticket size, and the overhead is mainly due to text processing and part-of-speech tagging.

Table 7: Evaluating NetSieve accuracy using different datasets. High F-scores are favorable.

Source	Dataset		Precision		Recall		F-Score		Accuracy %	
	Devices	# Tickets	Problems	Actions	Problems	Actions	Problems	Actions	Problems	Actions
Domain Expert	LB-1	122	1	1	0.982	0.966	0.991	0.982	97.7	96.6
	LB-2	62	1	1	1	1	1	1	100.0	100.0
	LB-3	31	1	1	1	0.958	1	0.978	100.0	95.7
	LB-4	36	1	1	1	1	1	1	100.0	100.0
	FW	35	1	1	0.971	0.942	0.985	0.970	97.1	94.3
	AR	410	1	1	0.964	0.951	0.981	0.974	96.4	95.1
Vendor	LB	78	1	1	0.973	0.986	0.986	0.993	97.3	98.7
	CR	77	1	1	1	0.896	1	0.945	100.0	89.6
CR: Core Routers; **LB[1-4]**:Types of Load balancers; **FW**:Firewalls; **AR**: Access Routers										

4 Experimental Results

For evaluation, we use two standard metrics from information retrieval: (1) *Accuracy Percentage* [31] computed as $\frac{TP+TN}{TP+TN+FP+FN}$ and (2) *F-Score* [32] computed as $\frac{2TP}{2TP+FP+FN}$, where TP, TN, FP, FN are true positives, true negatives, false positives and false negatives, respectively. F-Scores consider both precision and recall, and its value of 1 indicates a perfect classifier. Precision is defined as the ratio of TP and (TP+FP), and recall is defined as the ratio of TP and (TP+FN).

4.1 Evaluating NetSieve Accuracy

To test NetSieve's accuracy, we randomly divided our two year ticket dataset into training and test data. The test data consists of 155 tickets on device replacements and repairs from two network vendors and 696 tickets labeled by a domain expert while the training data comprises the rest of the tickets. We use the training data to build NetSieve's knowledge base. The domain expert read the original ticket to extract the ground truth in terms of Problems and Actions, which was then compared with corresponding phrases in the inference output.

Table 7 shows the overall results by NetSieve on the test dataset. On the expert labeled data, we observe the precision of NetSieve to be 1, minimum recall value to be 0.964 for Problems and 0.942 for Actions, F-score of 0.981-1 for Problems and 0.970-1 for Actions, and accuracy percentage to be 96.4%-100% for Problems and 94.3%-100% for Actions. These results indicate that NetSieve provides useful inference with reasonable accuracy in analyzing network tickets.

Next, we validate the inference output against data from two network device vendors that record the ground truth based on the diagnosis of faulty devices or components sent back to the vendor. Each vendor-summary reported the root cause of the failure (similar to NetSieve's Problems) and what was done to fix the problem (similar to NetSieve's Actions). We obtained vendor data corresponding to total 155 tickets on load balancers and core routers from our dataset. Since the vocabulary in the vendor data comprised new, vendor-specific words and synonyms not present in our knowledge base (e.g., port interface mentioned as 'PME'), we asked a domain-expert to validate if NetSieve's inference summary covers the root cause and the resolution described in the vendor data. For instance, if the vendor data denoted that a router failed due to a faulty supervisor engine (SUP), the expert checked if NetSieve captures "failed device" under *Problems* and "replaced SUP" under *Actions*.

The accuracy of NetSieve on the vendor data is observed to be 97.3%-100% for Problems and 89.6%-100% for Actions. One reason for relatively lower accuracy for Actions on this dataset is due to a small number of false negatives: the corrective action applied at the vendor site may differ from our ticket set as the vendor has the expert knowledge to fix problems specific to their devices.

Overall, NetSieve has reasonable accuracy of 96.4%-100% for *Problems* and 89.6%-100% for *Actions*, measured based on the labeled test dataset. We observed similar results for *Activities* and thus omit them.

4.2 Evaluating Usability of NetSieve

We conducted a user study involving five operators to evaluate the usability of NetSieve for automated problem inference compared to the traditional method of manually analyzing tickets. Each operator was shown 20 tickets selected at random from our dataset and asked to analyze the type of problems observed, activities and actions. Then, the operator was shown NetSieve inference summary of each ticket and asked to validate it against their manually labeled ground truth. We measured the accuracy, speed and user preference in the survey.

Figure 11 shows the accuracy and time to manually read the ticket versus the NetSieve inference summary across the operators. We observed average accuracy of 83%-100% and a significant decrease in time taken to manually read the ticket from 95P of 480s to 95P of 22s for NetSieve.

Table 8: Top-3 Problems/Activities/Actions and Failing Components as obtained through NetSieve's Trend Analysis

Device	Problems	Activities	Actions	Failing Components
AR	memory error, packet errors, illegal frames	verify cable, reseat/clean cable, upgrade OS	replace with spare, rma, reboot	SUP engine, cables, memory modules
AGG	device failure, packet errors, defective N/W card	verify cable, upgrade N/W card, swap cable	replace with spare, reboot, rma	cables, N/W card, SUP engine
CR	circuit failure, N/W card failure, packet errors	verify cable, verify N/W card, upgrade fiber	replace with spare, reboot, rma	cables, N/W, memory modules
ER	circuit failure, N/W card failure, packet errors	verify cable, verify N/W card, upgrade fiber	replace with spare, rma, reboot	N/W card, chassis, cables
LB	PSU failure, device rebooted, config error	verify PSU, verify config, verify cable	replace with spare, reboot, rma	PSU, HDD, memory modules
ToR	connection failure, ARP conflict, SUP engine failure	verify cable, power cycle blade, verify PSU	reboot, replace with spare, rma	cables, OS, SUP engine
FW	connection failure, reboot loop, data errors	verify config, verify connections, verify PSU	reboot, replace with spare, rma	cables, PSU, N/W card
VPN	connection failure, config error, defective memory	verify config, verify N/W card, deploy N/W card	reboot, replace with spare, rma	OS, SUP engine, memory modules

AR: Access Routers, **AGG**: Aggregation Switch; **[E/C]R**: Edge/Core Router; **LB**: Load Balancer; **ToR**: Top-of-Rack Switch; **FW**: Firewall

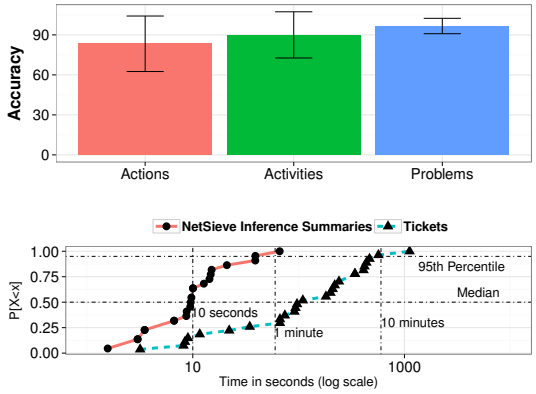

Figure 11: Accuracy obtained from the user survey (top). CDF of time to read tickets and inference summaries (bottom).

5 Deployment and Impact

NetSieve has been deployed in the cloud provider we studied to enable operators to understand global problem trends instead of making decisions based on isolated incidents. Further, NetSieve complements existing tools (e.g., inventory db) by correlating device replacements with their failures and problem root causes. A simple alternative to mining tickets using NetSieve is to ask operators for explicit feedback e.g., to build Table 8, but it will likely be biased by anecdotal or recent data.

Currently, our NetSieve prototype supports SQL-like queries on the inference output. For instance, "SELECT TOP 5 Problems FROM InferenceDB WHERE DeviceType = 'Load Balancer' would output the top-5 failure problems observed across load balancers. Next, we present how NetSieve has been used across different teams to improve network management.

Network Architecture: This team used NetSieve to compare device reliability across platforms and vendors. In one instance, NetSieve showed that a new generation of feature-rich, high capacity AR is half as reliable as its predecessor. In another instance, it showed that software bugs dominated failures in one type of load balancers. Based on grouping tickets having Problem inference of Switch Card Control Processor (SCCP) watchdog timeout for LB-2, NetSieve showed that hundreds of devices exhibited reboot loops due to a recurring software bug and were RMAed in 88% of the tickets.

Capacity Planning: This team applied NetSieve to analyze cases when network redundancy is ineffective in masking failures. Specifically, for each network failure, we evaluate if the redundancy failover was not successful [13] and then select the corresponding tickets. These tickets are then input to NetSieve to do problem inference which output the following to be the dominant problems:

1. **Faulty Cables**: The main reason was that the cable connected to the backup was faulty. Thus, when the primary failed, it resulted in a high packet error rate.
2. **Software Mismatch**: When the primary and backup had mismatched OS versions, protocol incompatibility caused an unsuccessful failover.
3. **Misconfiguration**: Because operators usually configure one device and then copy the configuration onto the other, a typo in one script introduced the same bug in the other and resulted in an unsuccessful failover.
4. **Faulty Failovers**: The primary device failed when the backup was facing an unrelated problem such as software upgrade, down for scheduled repairs, or while deploying a new device into production.

Incident Management and Operations: The incident management team used NetSieve to prioritize checking for the top-k problems and failing components while

troubleshooting devices. The operation team uses NetSieve to determine if past repairs were effective and decide whether to repair or replace the device. Table 8 shows NetSieve's inference output across different device types.We observe that while Problems show a high diversity across device types such as packet errors, line card failures and defective memory, verifying cable connectivity is a common troubleshooting activity, except for Firewalls and VPNs where operators first verify device configuration. For Actions related to Firewalls, VPNs and ToRs, the devices are first rebooted as a *quick fix* even though the failing components are likely to be bad cable or OS bugs. In many cases, we observe that a failed device is RMAed (sent back to the vendor) which implies that the network is operating at reduced or no redundancy until the replacement arrives.

Complementary to the above scenarios, NetSieve inference can be applied in several other ways: (1) prioritize troubleshooting steps based on frequently observed problems on a given device, its platform, its datacenter, or the hosted application, (b) identify the top-k failing components in a device platform and resolving them with their vendors, and (c) decide whether to repair, replace or even retire a particular device or platform by computing a total cost-of-ownership (TCO) metric.

6 Related Work

Network Troubleshooting: There has been a significant work in analyzing structured logs to learn statistical signatures [1, 10], run-time states [54] or leveraging router syslogs to infer problems [41]. Xu et al. [51] mine machine logs to detect runtime problems by leveraging the source code that generated the logs. NetSieve complements these approaches to automate problem inference from unstructured text. Expert systems [8, 25], on the other hand, diagnose network faults based on a set of pre-programmed rules. However, they lack generality as they only diagnose faults in their ruleset and the ruleset may become outdated as the system evolves. In comparison, the incremental learning phase in NetSieve updates the knowledge base to improve the inference accuracy.

Mining Network Logs: Prior efforts have focused on automating mining of network failures from syslogs [41, 53] or network logs [28]. However, these studies do not analyze free-form text in trouble tickets. Kandula et al. [24] mine rules in edge networks based on traffic data. Brauckhoff et al. [7] use association rule mining techniques to extract anomalies in backbone networks. NetSieve is complementary to these efforts in that their mining methodologies can benefit from our domain-specific knowledge base. TroubleMiner [36] selects keywords manually from the first two sentences in tickets and then performs clustering to group them.

Analyzing Bug Reports: There is a large body of work in software engineering analyzing [19, 22], summarizing [6] and clustering [5, 43] bug reports. Betternburg et al. [6] rely on features found in bug reports such as stack traces, source code, patches and enumerations and, hence their approach is not directly applicable to network tickets. Others [5, 43] use standard NLP techniques for the task of clustering duplicate bug reports, but they suffer from the same limitations as keyword based approaches. In comparison, NetSieve aims to infer "meaning" from the free-form content by building a knowledge base and an ontology model to do problem inference.

Natural Language Processing: *N*-gram extraction techniques [9, 39, 45, 52] focus on extracting all possible *n*-grams thereby incurring a high computation cost for large datasets (§3.3.1). NetSieve addresses this challenge by trading completeness for scalability and uses the dictionary built by its WLZW algorithm. Wu et al. [50] detect frequently occurring text fragments that have a high correlation with labels in large text corpora to detect issues in customer feedback. Other efforts [14, 38] achieve summarization via paragraph and sentence extraction. Much research in this area deals with properly-written regular text and is not directly applicable to our domain. In contrast, NetSieve focuses on free-form text in trouble tickets to do problem inference.

7 Conclusion

Network trouble tickets contain valuable information for network management, yet they are extremely difficult to analyze due to their free-form text. This paper takes a practical approach towards automatically analyzing the natural language text to do problem inference. We presented NetSieve that automatically analyzes ticket text to infer the problems observed, troubleshooting steps, and the resolution actions. Our results are encouraging: NetSieve achieves reasonable accuracy, is considered useful by operators and has been applied to answer several key questions for network management.

8 Acknowledgements

We thank our shepherd Kobus Van der Merwe and the anonymous reviewers for their feedback; our colleagues Sumit Basu, Arnd Christian König, Vivek Narasayya, Chris Quirk and Alice Zheng for their insightful comments; and the network operations team for their guidance and participation in our survey.

References

[1] AGUILERA, M., MOGUL, J., WIENER, J., REYNOLDS, P., AND MUTHITACHAROEN, A. Performance debugging for distributed systems of black boxes. In *ACM SIGOPS Operating Systems Review* (2003), ACM.

[2] AHO, A., AND CORASICK, M. Efficient string matching: an aid to bibliographic search. *Communications of the ACM* (1975).

[3] BEHNEL, S., BRADSHAW, R., SELJEBOTN, D., EWING, G., ET AL. Cython: C-extensions for python, 2008.

[4] BENSON, M. Collocations and general-purpose dictionaries. *International Journal of Lexicography* (1990).

[5] BETTENBURG, N., PREMRAJ, R., ZIMMERMANN, T., AND KIM, S. Duplicate bug reports considered harmful really? In *IEEE International Conference on Software Maintenance* (2008).

[6] BETTENBURG, N., PREMRAJ, R., ZIMMERMANN, T., AND KIM, S. Extracting structural information from bug reports. In *ACM International Working Conference on Mining Software Repositories* (2008).

[7] BRAUCKHOFF, D., DIMITROPOULOS, X., WAGNER, A., AND SALAMATIAN, K. Anomaly extraction in backbone networks using association rules. In *ACM Internet Measurement Conference* (2009).

[8] BRUGNONI, S., BRUNO, G., MANIONE, R., MONTARIOLO, E., PASCHETTA, E., AND SISTO, L. An expert system for real time fault diagnosis of the italian telecommunications network. In *International Symposium on Integrated Network Management* (1993).

[9] CHURCH, K., AND HANKS, P. Word association norms, mutual information, and lexicography. *Journal of Computational Linguistics* (1990).

[10] COHEN, I., ZHANG, S., GOLDSZMIDT, M., SYMONS, J., KELLY, T., AND FOX, A. Capturing, indexing, clustering, and retrieving system history. In *ACM SIGOPS Operating Systems Review* (2005), ACM.

[11] FIELDING, R. Representational state transfer (rest). *Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine* (2000), 120.

[12] FORD, D., LABELLE, F., POPOVICI, F., STOKELY, M., TRUONG, V., BARROSO, L., GRIMES, C., AND QUINLAN, S. Availability in globally distributed storage systems. In *USENIX Symposium on Operating Systems Design and Implementation* (2010).

[13] GILL, P., JAIN, N., AND NAGAPPAN, N. Understanding network failures in data centers: measurement, analysis, and implications. In *SIGCOMM Computer Communication Review* (2011).

[14] GOLDSTEIN, J., KANTROWITZ, M., MITTAL, V., AND CARBONELL, J. Summarizing text documents: sentence selection and evaluation metrics. In *International ACM SIGIR Conference on Research and Development in Information Retrieval* (1999).

[15] GORYACHEV, S., SORDO, M., ZENG, Q., AND NGO, L. Implementation and evaluation of four different methods of negation detection. Tech. rep., DSG, 2006.

[16] GOYAL, A., DAUMÉ III, H., AND VENKATASUBRAMANIAN, S. Streaming for large scale NLP: Language modeling. In *Annual Conference of the Association for Computational Linguistics* (2009).

[17] GRUBER, T., ET AL. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human Computer Studies* (1995).

[18] HEAFIELD, K. Kenlm: Faster and smaller language model queries. In *Workshop on Statistical Machine Translation* (2011).

[19] HOOIMEIJER, P., AND WEIMER, W. Modeling bug report quality. In *IEEE/ACM International Conference on Automated Software Engineering* (2007).

[20] HUANG, Y., FEAMSTER, N., LAKHINA, A., AND XU, J. Diagnosing network disruptions with network-wide analysis. In *ACM SIGMETRICS Performance Evaluation Review* (2007).

[21] JOHNSON, D. Noc internal integrated trouble ticket system. `http://goo.gl/eMZxX`, January 1992.

[22] JUST, S., PREMRAJ, R., AND ZIMMERMANN, T. Towards the next generation of bug tracking systems. In *IEEE Symposium on Visual Languages and Human-Centric Computing* (2008).

[23] JUSTESON, J., AND KATZ, S. Technical terminology: some linguistic properties and an algorithm for identification in text. *Journal of Natural language engineering* (1995).

[24] KANDULA, S., CHANDRA, R., AND KATABI, D. What's going on?: learning communication rules in edge networks. *ACM SIGCOMM Computer Communication Review* (2008).

[25] KHANNA, G., CHENG, M., VARADHARAJAN, P., BAGCHI, S., CORREIA, M., AND VERÍSSIMO, P. Automated rule-based diagnosis through a distributed monitor system. *IEEE Transactions on Dependable and Secure Computing* (2007).

[26] KITTUR, A., CHI, E., AND SUH, B. Crowdsourcing user studies with mechanical turk. In *ACM SIGCHI Conference on Human factors in Computing Systems* (2008).

[27] LABOVITZ, C., AHUJA, A., AND JAHANIAN, F. Experimental study of internet stability and backbone failures. In *IEEE International Symposium on Fault-Tolerant Computing* (1999).

[28] LIM, C., SINGH, N., AND YAJNIK, S. A log mining approach to failure analysis of enterprise telephony systems. In *IEEE Dependable Systems and Networks* (2008).

[29] LOPER, E., AND BIRD, S. Nltk: The natural language toolkit. In *Association for Computational Linguistics Workshop on Effective Tools and Methodologies for teaching Natural Language Processing and Computational Linguistics* (2002).

[30] MANBER, U., AND MYERS, G. Suffix arrays: a new method for on-line string searches. In *ACM SIAM Journal on Computing* (1990).

[31] MANI, I., HOUSE, D., KLEIN, G., HIRSCHMAN, L., FIRMIN, T., AND SUNDHEIM, B. The tipster summac text summarization evaluation. In *Association for Computational Linguistics* (1999).

[32] MANNING, C., RAGHAVAN, P., AND SCHUTZE, H. *Introduction to information retrieval*, vol. 1. Cambridge University Press Cambridge, 2008.

[33] MANNING, C., AND SCHÜTZE, H. *Foundations of statistical natural language processing*. MIT Press, 1999.

[34] MARCUS, M., MARCINKIEWICZ, M., AND SANTORINI, B. Building a large annotated corpus of english: The penn treebank. *MIT Press Computational Linguistics* (1993).

[35] MCCALLUM, A., AND LI, W. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Association for Computational Linguistics Conference on Natural Language Learning* (2003).

[36] MEDEM, A., AKODJENOU, M., AND TEIXEIRA, R. Troubleminer: Mining network trouble tickets. In *IEEE International Workshop Symposium on Integrated Network Management* (2009).

[37] MELCHIORS, C., AND TAROUCO, L. Troubleshooting network faults using past experience. In *IEEE Network Operations and Management Symposium* (2000).

[38] MITRAY, M., SINGHALZ, A., AND BUCKLEYYY, C. Automatic text summarization by paragraph extraction. *Compare* (1997).

[39] NAGAO, M., AND MORI, S. A new method of n-gram statistics for large number of n and automatic extraction of words and phrases from large text data of japanese. In *ACL Computational Linguistics* (1994).

[40] NOY, N., MCGUINNESS, D., ET AL. Ontology development 101: A guide to creating your first ontology. Tech. rep., Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, 2001.

[41] QIU, T., GE, Z., PEI, D., WANG, J., AND XU, J. What happened in my network: mining network events from router syslogs. In *ACM Internet Measurement Conference* (2010).

[42] ROUGHAN, M., GRIFFIN, T., MAO, Z., GREENBERG, A., AND FREEMAN, B. Ip forwarding anomalies and improving their detection using multiple data sources. In *ACM SIGCOMM Workshop on Network Troubleshooting: Research, Theory and Operations Practice meet malfunctioning reality* (2004).

[43] RUNESON, P., ALEXANDERSSON, M., AND NYHOLM, O. Detection of duplicate defect reports using natural language processing. In *IEEE International Conference on Software Engineering* (2007).

[44] S., D. Sharp NLP. `http://goo.gl/Im4BK`, December 2006.

[45] SMADJA, F. Retrieving collocations from text: Xtract. *MIT Press Computational linguistics* (1993).

[46] TOUTANOVA, K., AND MANNING, C. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *ACL SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora* (2000).

[47] TURNER, D., LEVCHENKO, K., SNOEREN, A., AND SAVAGE, S. California fault lines: understanding the causes and impact of network failures. In *ACM SIGCOMM Computer Communication Review* (2010).

[48] UKKONEN, E. On-line construction of suffix trees. *Algorithmica* (1995).

[49] WELCH, T. Technique for high-performance data compression. *Computer 17*, 6 (1984), 8–19.

[50] WU, F., AND WELD, D. Open information extraction using wikipedia. In *Computational Linguistics* (2010).

[51] XU, W., HUANG, L., FOX, A., PATTERSON, D., AND JORDAN, M. Detecting large-scale system problems by mining console logs. In *ACM SIGOPS Symposium on Operating Systems Principles* (2009).

[52] YAMAMOTO, M., AND CHURCH, K. Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Computational Linguistics* (2001).

[53] YAMANISHI, K., AND MARUYAMA, Y. Dynamic syslog mining for network failure monitoring. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* (2005).

[54] YUAN, D., MAI, H., XIONG, W., TAN, L., ZHOU, Y., AND PASUPATHY, S. Sherlog: error diagnosis by connecting clues from run-time logs. In *ACM SIGARCH Computer Architecture News* (2010).

[55] ZHANG, J., GAO, J., AND ZHOU, M. Extraction of chinese compound words: An experimental study on a very large corpus. In *ACL Workshop on Chinese Language Processing* (2000).

[56] ZHANG, L. N-Gram Extraction Tools. `http://goo.gl/7gGF1`, April 2012.