IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 21, 2019, accepted April 22, 2019, date of publication April 30, 2019, date of current version May 14, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2914067

Design and Implementation of an Open Source Framework and Prototype For Named Data Networking-Based Edge Cloud Computing System

REHMAT ULLAH[1], MUHAMMAD ATIF UR REHMAN[1], AND BYUNG-SEO KIM[2], (Senior Member, IEEE)

[1]Department of Electronics and Computer Engineering, Hongik University, Sejong 30016, South Korea
[2]Department of Software and Communications Engineering, Hongik University, Sejong 30016, South Korea

Corresponding author: Byung-Seo Kim (jsnbs@hongik.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) through the Korean Government under Grant 2018R1A2B6002399, and in part by the International Research & Development Program of the National Research Foundation of Korea (NRF) through the Ministry of Science and ICT under Grant NRF-2018K1A3A1A39086819.

ABSTRACT Named data networking (NDN) and edge cloud computing (ECC) are emerging technologies that are considered as the most representative technologies for the future Internet. Both technologies are the promising enabler for the future Internet such as fifth generation (5G) and beyond which requires fast information response time. We believe that clear benefits can be achieved from the interplay of NDN and ECC and enables future network technology to be much more flexible, secure and efficient. In this paper, therefore, we integrate NDN with ECC in order to achieve fast information response time. Our framework is based on N-Tier architecture and comprises of three main Tiers. The NDN is located at the Tier1 (Things/end devices) and comprises of all the basic functionalities that connect Internet of Things (IoT) devices with Tier 2 (Edge Computing), where we have deployed our Edge node application. The Tier 2 is then further connected with Tier 3 (Cloud Computing), where our Cloud node application is deployed at multiple hops on the Microsoft Azure Cloud machine located in Virginia, WA, USA. We implement an NDN-based ECC framework and the outcomes are evaluated through testbed and simulations in terms of interest aggregation, round trip time (RTT), service lookup time with single query lookup time and with various traffic loads (load-based lookup time) from the IoT devices. Our measurements show that enabling NDN with edge computing is a promising approach to reduce latency and the backbone network traffic and capable of processing large amounts of data quickly and delivering the results to the users in real time.

INDEX TERMS Internet of things, edge computing, fog computing, distributed computing, caching, named data networking, framework, offloading, latency, scalability, multi-access edge computing, future Internet.

I. INTRODUCTION

The future Internet of Things (IoT) is evolving very fast which will potentially connect billions and trillions of edge devices [1]. These devices could generate massive amount of data at a very high speed and some of the applications may require very low latency [2]. The conventional cloud only model may short fall due to centralized computation, storage, and networking in a small number of datacenters. Moreover, the relative long distance between the edge devices and the remote data-centers is vulnerable to the real time applications.

Industrial Internet of Things (IIoT), such as manufacturing systems, smart grids, oil and gas systems, often require end-to-end latencies of a few millisecond between the sensor and the control node. Moreover, IoT applications, such as vehicle-to-vehicle communications, vehicle-to-roadside communications, drone flight control applications, Augmented reality (AR) and virtual reality (VR) applications, and online gaming, may require latencies below a few tens of milliseconds. Some wearable camera applications, or industrial monitoring and controlling applications require response time to be as low as 10-50 ms [3]. Moreover, the number of IoT devices is exponentially increasing and it is predicted that by year 2020 these devices will generate 2.3 trillion gigabytes of data each day [4]. Such massive increment of devices and data

The associate editor coordinating the review of this manuscript and approving it for publication was Jun Wu.

will result in a significance load on the Internet. Therefore, it is important that most of the data should be processed by the edge node first. After processing and filtering data at the edge node, further the data should be sent to the cloud data center. Similarly, comparing with the servers in cloud data centers, most of the IoT devices or sensors at the edge tier are resource constrained in terms of computing power and battery capacity [5]. Therefore, the complex and intense computation are not appropriate for these devices. Instead, most of such computation and data processing tasks can be offloaded to ECC, which could save energy on IoT devices and get the tasks done faster.

For such resource constrained and massive number of IoT devices the traditional centralized cloud computing would be alarming due to the large data volume generated and the long distance between clients and the backend data-centers. Therefore, Edge Computing can provide effective ways to overcome many limitations of the cloud only model by distributing computing, control, storage, and networking functions closer to end user devices. However, the traditional model (IP approach) is still the dominant solution, where the end-to-end communication is managed between the IoT data sources and fixed purpose-built servers deployed at the network edges [6]. Although edge may accommodate all such enormous data, it cannot fulfill the end to end latency requirement of future networks such as AR and VR.

For future Internet the demand for various services such as multimedia IoT, autonomous vehicle service, and location-based service is expected to increase. It is expected that the next generation real time applications should be capable of processing large amounts of data quickly and delivering the results to the users in real time through various network environments that change due to mobility rather than a static environment [7]. However, the static network environment based on the existing IP and Server-Client approach, the inflexible communication environment in the mobility environment, the network overload due to the massive traffic transmission through the fixed path, the remote clouding and the occurrence of an enormous delay arising from a series of processes such as transmission, processing, and data reception of real-time data, which causes degradation of the real time service performance in the network in the future, has been recognized as an impediment to the development of a combined technology of NDN and ECC.

In order to solve the problems of real-time services and to improve the performance of real time services, the framework to combine well-known innovative future network technologies: NDN and ECC are needed. NDN offers network layer content and services, increased caching, built-in mobility, built-in security, multihoming in case of heterogeneous wireless environment and so on. In addition, edge computing offers computation, storage, and increased caching capabilities in close proximity of end users/devices, thereby reducing the end to end latency. Edge computing removes the cloud dependency for content processing and removes the end to end mapping. We believe that the two technologies will bring an enormous performance improvement to the real-time applications that require response time of less than 1ms. In this paper, we proposed an open source framework, in which we leverage and exploit the features of both technologies (NDN and ECC) in order to empower the future networks. Since the main goal of edge computing in IoT is to reduce the backbone network traffic on cloud data centers, to reduce latency and increase the bandwidth. Therefore, it is possible to bring the content and services more closer to the end users via NDN.

As a summary, the major contributions of this paper include:

1) We propose an open source framework integrating NDN and ECC via N-Tier architecture with 7 layers at Edge Tier and 6 layers at the Cloud Tier.
2) We propose.NET web API for RESTful web services in order to communicate IoT devices with the Things Tier, Edge computing Tier and Cloud computing Tier.
3) In order to integrate NDN with Edge computing Tier and Cloud computing Tier, we make changes in the official ndnSIM codebase.
4) Development of an Edge node application and Cloud node application from the scratch.
5) A prototype implementation and extensive experimentation to validate and evaluate our propose framework. This prototype is not dedicated to our propose framework evaluation. Instead anyone can use it for possible edge computing ideas.
6) Open source all the source code to the research community for reuse and further improvement.

The rest of the paper is organized as follows. We present the background studies of edge computing and NDN in the Section 2. In Section 3 we present related works. Section 4 presents the proposed framework. In Section 5 we present the testbed experimental setup and results and discussion. Finally, we conclude the paper in Section 6.

II. BACKGROUND AND MOTIVATION

A. EDGE COMPUTING

Edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network. Here the ''Edge'' means any computing and network resources along the path between data sources and cloud data centers. The rationale of edge computing is that computing should happen at the proximity of data sources. Sometimes the term ''Edge computing'' is interchangeably used with ''Fog computing''. However, the Edge computing focus more toward the Things side, while Fog computing focus more on the infrastructure side [2]. Edge computing pushes computing power to the edges of network, instead of centralized cloud. The data analytics happens very close to the devices and sensors. Therefore, edge computing thus results in lower delay and high speed of task execution [7]. The core idea of Edge computing is to bring the resources at the network edge such as computation and bandwidth. The storages resource is moved closer to the IoT devices to reduce the backbone

data traffic and the response latency, and to facilitate the resource-intensive IoT applications. Edge computing has relatively smaller computation capacity compared to cloud computing, however, takes advantage of short access distance, flexible geographical distribution, and relatively richer computational resource than mobile device(s).

Various edge computing proposals has been presented in the literature under the umbrella of edge computing paradigm. Initially Mobile Cloud Computing (MCC) was introduced in the edge computing initiative. MCC [8] offers offloading mechanism for the mobile devices by integrating mobile Internet, mobile computing, and cloud computing into a combined system. The major purpose of MCC is offloading tasks from a mobile device to the cloud servers in order to overcome the storage and computation limitation of mobile devices. Moreover, the mobile devices also have the issue of battery drainage. Therefore, the objective of MCC is to save energy and prevent the factors that cause battery drainage.

Cloudlet [9] has been introduced as an extension of MCC and is considered as one of the key enabling technologies for MCC. Offloading to the cloud is not always a solution because of high wide area network (WAN) latencies. Real-time applications need low latency and may not be achieved by offloading tasks to the cloud. Tasks running on mobile devices may require high computation and lower latency. Such limitations of mobile device cannot fulfill these requirements. Therefore, to provide computation power and to meet lower latency requirements, these tasks can be offloaded to the Cloudlet instead of Cloud servers. Cloudlet is kind of small cloud server located between mobile device and central cloud. Cloudlets are placed nearby to mobile devices with single hop proximity and works as virtual-machine (VM).

CISCO proposed Fog computing model [10] for the better management of the Clouds by enabling services, applications, and content storage in close vicinity to mobile end users. In the Fog computing paradigm, data processing happens locally rather than being sent to the Cloud servers. Fog computing supports offloading, caching, location awareness, and mobility information. It has many advantages for applications that are delay sensitive.

European Telecommunications Standards Institute (ETSI) coined the term Mobile Edge Computing (MEC) [11] with the aim to push computational power into Radio Access Network (RAN) and to leverage the virtualization of software at the radio edge. In order to realize the power of location, now both fixed and mobile networks are accepting MEC. Therefore, the MEC acronym no longer refers to "*Mobile Edge Computing*" and instead stands for "*Multiple-access Edge Computing*" [12]. The motivation behind MEC is the proliferation of smart phones and the traffic generated from the phones. According ETSI, MEC can reduce the latency and can provide location awareness to mobile users. Requirements such as bandwidth (higher), latency (lower) and mobility should be met in future mobile networks such as 5G and beyond. Therefore, to fulfill such requirements both the RAN and core network should be optimized to serve billion of devices. Furthermore, Edge servers address issue of congestion at the core network. The reason is that most of the traffic is processed locally instead sending to the backbone networks.

1) DIFFERENCES BETWEEN EDGE COMPUTING PROPOSALS

The terms such as Edge, Fog, MEC, Cloudlet and MCC all are the umbrella concepts of Edge computing. However, the implementation speciation differs at different aspects for each proposal.

MEC provides access within RAN instead of core WAN to minimize latency and to decrease the energy consumption. However, Cloudlets and Fog computing provides the services to offload the subscriber's tasks. The specifications of MEC states that MEC servers should be co-located with the cellular network base station. On the other hand, Fog servers are generally provided by private environment such as shopping malls and coffee shops etc. However, they can be deployed as routers and gateways in Internet service provider (ISP) infrastructure. Cloudlet can be deployed in a distributed way. However, there is not any exact location or vendor defined for the deployment of Cloudlets. MEC server is reachable via third generation/Long-Term Evolution (3G/LTE) base station. However, Fog servers and Cloudlets are accessible via wireless access point (AP) whose coverage area is much smaller than 3G/LTE. Mostly the Cloudlet study emphasis on Wi-Fi as an access technology. However, Cloudlet is not limited and can be applied in other wireless technologies too.

Due to technological enhancement in cellular networks, the operators emphasis on MEC technology. In other words, future cellular networks are more commonly referred to MEC instead of any other Edge Computing proposal. Hence, for cellular networks, MEC is the de-facto Edge computing technology. The reason is that Cloudlet and Fog computing have short range communication such as Bluetooth and Wi-Fi and hence cannot reach to the level of MEC.

The number of devices and users are also different in each proposal. Fog computing addresses the use cases of IoT and vehicle-to vehicle (V2V) communication. For this reason, the users and devices in Fog computing is expected higher than the Cloudlet. However, Cloudlet covers the IoT devices but not V2V communication. The users of MEC is much smaller because MEC only focuses on subscribers and providers in cellular environment. Furthermore, the server density of MEC servers is limited to the base stations. Whereas the Cloudlet can be installed at any public place such as coffee shops, shopping malls and restaurants etc. Cloudlets mostly uses wireless local area network (WLAN) as an access technology. Therefore, the density of Cloudlet is much higher than other Edge computing proposals. Fog server(s) deployment is average and cannot be installed everywhere like Cloudlet [12].

2) NEED OF EDGE COMPUTING

The necessity of bringing content and services closer to end users comes from various factors that drive the evolution of edge computing paradigm. The challenges of resource constrained IoT devices can be addressed by providing local computing storage and the networking resources. The data generated by the end users will be processed at the edge nodes and only small portion of data will be sent to the cloud for further processing. Therefore, the backbone traffic and the networking load will be reduced. The motivations behind edge computing paradigm are as follows:

1) QoS and Latency: Even though edge devices are powerful, most of them are lacking enough capacity for fulfilling the delay sensitive requirements. Cloud computing provides resource enriched technology infrastructure for constrained devices with huge computation and storage capacity. However, most of the devices in IoT i.e. wearable devices are delay sensitive. In legacy cloud paradigm, the accessibility of all these devices is done via WAN, which creates delay, and hence conventional cloud cannot deal with the problems such as the mobility, and real-time requirements. The latency-sensitive applications demanding computation power and memory resources that cannot be built satisfactorily using cloud services, which can be many network hops away from user locations. Computational resources are required at the edge of the network to meet high QoS. As an example, in autonomous vehicles the generated data of camera need to be processed instantly to meet the real-time requirements of QoS [13]. User experience is affected by the centralized servers of cloud due to limited Internet bandwidth and WAN delay. The overall latency can be reduced if the servers are deployed closer to user devices. The benefit of servers closer to the users are the high local-area network (LAN) bandwidth and a smaller number of hopes. Moreover, edge computing provides caching, storage and computation capabilities in close proximity of end users/devices thereby reduces end to end latency. Edge computing offers such benefits without requiring deployment in the core and remove cloud dependency for content processing.

2) Minimization of Core Network Traffic: In the conventional cloud computing approach, all the traffic flows from devices through core network to reach cloud servers. The content as well as the context require processing and storage which may not achievable on mobile devices. Therefore, it is done on the cloud that result in higher response time and increased backbone traffic. In case of IoT billions of devices may generate a huge amount of raw data to be processed and stored. According to [14], 15 petabytes of traffic has been generated per month. Sending all the traffic to the cloud servers may result in congestion on cloud servers, since cloud servers have limited capacity. To optimize bandwidth utilization and to reduce traffic on the core network, the traffic should be handled at the edge servers. Traffic from billions of devices can be handled at edge servers to prevent congestion and latency problems. Therefore, edge computing paradigm can play a meaningful role in traffic reduction on the core network [15].

3) Scalability: It is predicated that the end user devices will reach million and billions that may create a serious scalability challenge. Therefore, sending such a massive amount of data to the centralized servers create congestion within the data-centers. As a result, cloud computing may fall short in the context of scalability for applications and data. The virtualization of edge servers can bring an opportunity to support scalability. If any of the edge server becomes congested, then the request can be distributed to other edge servers in proximity and so on. The burden on the cloud servers can be reduced by processing data at the edge servers, since smaller amount of traffic will be forwarded to the cloud servers.

B. NAMED DATA NETWORKING

Named Data Networking (NDN) is an architecture under the umbrella of Information-Centric Networking (ICN) paradigm where various architectures are being proposed such as DONA, NetInf, CCN, NDN, MobilityFirst, PURSUIT, CONVERGENCE, and COMET [16]. Named data networking (NDN) paradigm is gaining more popularity and is widely accepted in the research community due to its simple communication architecture [17]. In this paper, therefore, we are also using the NDN architecture. In the NDN architectures the content is treated as the first-class citizen rather than host and names are used for network layer communication instead IP addresses. The predecessor of NDN is the Content-Centric Networking (CCN) architecture which was originally proposed by Van Jacobson as a project initiated by Palo Alto Research Center (PARC) [18]. The main idea behind CCN is that the Consumer(s) send Interest packet(s) containing the names of the requested content and the Data packet(s) flow back, carrying the named and secured content or chunks of the content, following the same path through which the Interest packets were sent. The content provider, or any other network node with a copy of the requested content can reply with the requested content along with additional authentication and data-integrity information, along the path. Furthermore, caching on each path node is enabled depending on the caching policy of the node.

Named-data networking (NDN) is an enhanced version of the CCN architecture. Similar to CCN, NDN also follows the interest/data packet combination to obtain any particular data. Each node maintains three types of data structures: (1) a Content Store (CS), which is capable of caching data temporarily; (2) a Pending Interest Table (PIT), which retains the records of unsatisfied Interest packets (3) a Forwarding Information Base (FIB), which traverses Interest packets toward the data providers.

The main architectural difference between NDN and CCN is that CCN (which is the previous implementation of NDN), the CS proceeds the PIT search. However, in NDN PIT is checked first and then CS is checked. However, in pure ICN and in most of the literature, the CS lookup is the very first operation after the arrival of an Interest packet. The main

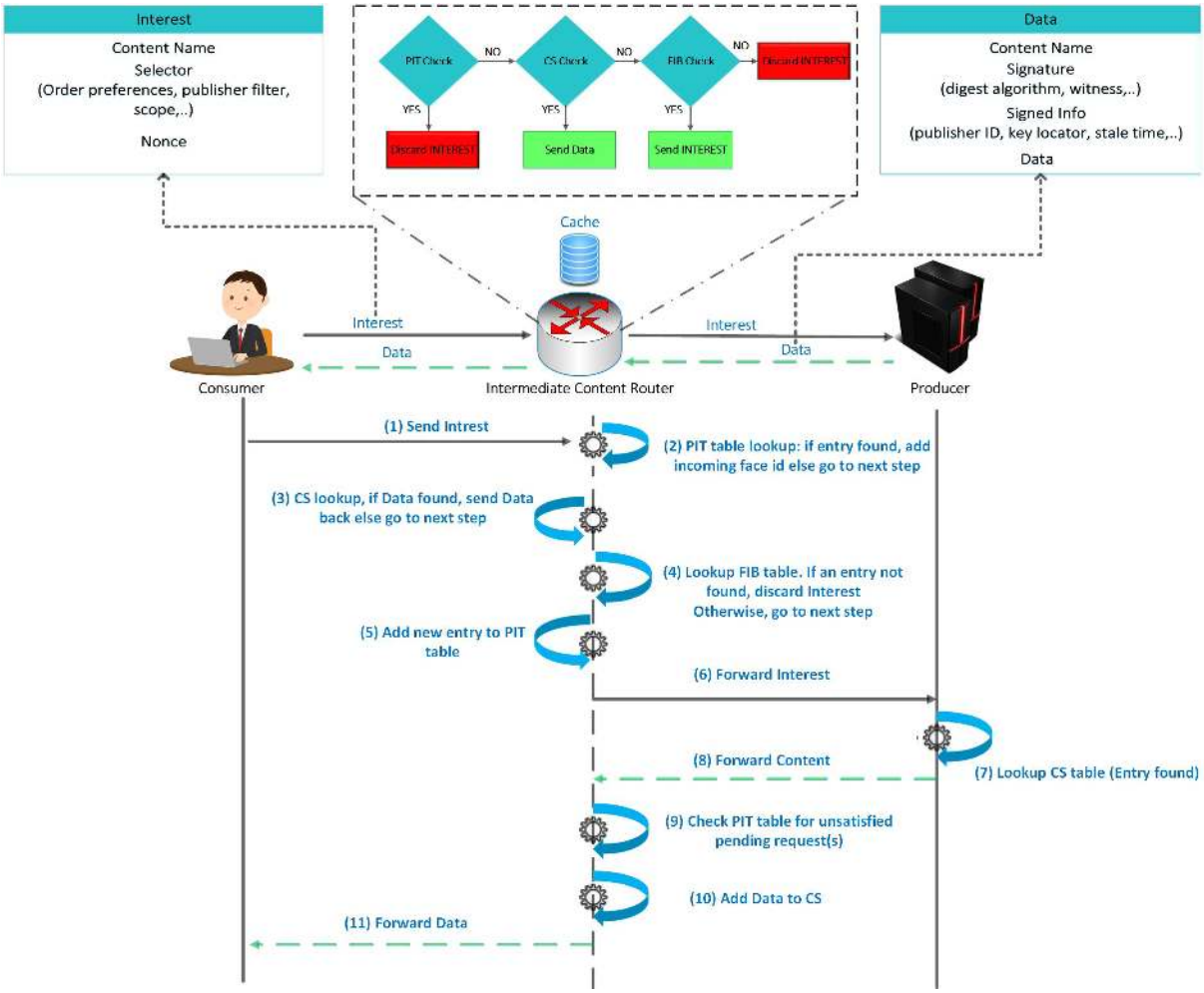

FIGURE 1. NDN communication process.

reason is to overcome the lookup delay of CS since the PIT is considered much smaller than the CS. [19].

Figure 1 depicts the NDN communication flow. When a consumer node wants to access specific content, it sends an Interest packet containing the name of the content. The consumer node uses the content name and its selectors (the content attributes, resolutions, filters, nonce etc.) in the Interest message to request the content object. When some relay node(s) receives the Interest packet, it checks its PIT. If an entry is found in PIT (but with a different receiving face or with a different NONCE or Interest payload ID), the node updates the existing PIT table by adding a new incoming interface entry and discards the Interest for further processing. Otherwise, the content is searched in the CS using name and selectors; if match is found, the node sends the data back to the consumer via the same interface from which the Interest is received. If there are more than one content matches, the selectors are used to precisely identify the content object. Otherwise, a new PIT entry is created and the Interest packet is sent further via interface(s) stored in the FIB. If the FIB search returns more than one next hops, then the Interest is forwarded based on the forwarding strategy, i.e., send the Interest to all, best path, or alternatively to the paths based on their ranks, etc. If no FIB entry is found, the NDN discard the Interest message.

After receiving an Interest packet, the provider node will forward the Data packet which carries content as the payload, the name that identifies it, and additional information to verify and validate the content and the data message. This additional information includes the signature (cryptographic), the identification information of the publisher or signer, etc. When the Data packet is coming back, the received node first checks it's PIT table. If some entry is matched, it forwards the data to the interface(s) from where the corresponding Interests received. Moreover, it stores the data in the CS and delete the existing PIT entry. However, the data storing is dependent on the caching policy, i.e., all caching, no caching, popularity-based caching, etc. Note that the PIT entry is also purged when the life-time of the PIT entry is expired. If there is no entry in PIT table then the received Data packet is considered as unsolicited Data packet (data which is not requested) and it is discarded due to security reasons. The Data packet follows the "breadcrumb" i-e the same route of PIT entries till it reached to the consumer(s).

Detailed processing steps of an Interest packet and a Data packet are depicted in Figure 1.

C. WHY NDN WITH EDGE COMPUTING?

In today's Internet design the communication happens between fixed entities due to host oriented approach (TCP/IP). However, due to rise of IoT and real-time applications such as AR, VR and Tactile Internet, it becomes challenging in terms of mobility, scalability, security and network management. NDN is a promising paradigm that is based on named based communication rather than host address. In this paper, we leverage the NDN features to facilitate edge computing functionalities for seamless and efficient edge computing. Since NDN is an emerging technology for distributing contents to users, which makes contents directly addressable and routable in networks. Therefore, NDN-based Edge computing will improve the efficiency of content distribution. The Edge computing is another novel paradigm that moves the control of cloud services to the network edge devices. Both NDN and Edge computing can improve the communication performance by reducing the distance between users and services. Thus, the integration of both technologies (NDN and Edge computing) will be a great opportunity for 5G networks and beyond. There are many expected benefits resulting from the integration of NDN in edge computing which are described as follows:

1) Interest Aggregation: NDN provide the interest aggregation feature natively. In NDN, when a content router receives an Interest packet that cannot be satisfied through its local CS, it creates a PIT entry and forwards the Interest if there is no entry for the same content name in its PIT. If a PIT entry already exists for the same content the Interest is aggregated at the router and is not forwarded. The benefits of Interest aggregation in NDN architectures has been that network and server loads can be drastically reduced by controlling similar Interests, and the end-to end latency can be reduced by integrating caching with Interest aggregation [20]. Therefore, we are using the interest aggregation feature of NDN for edge computing. We control the traffic generated from end users/things to the edge device via NDN interest aggregation feature. All the incoming requests for the same content will be aggregated in the PIT table and will not be forwarded further to the edge node. However, in case of IP (without using NDN), since there is no PIT table. Therefore, all the interests will be forwarded to the edge node which increases the network and server loads. Without aggregation, all the traffic from the things Tier will be sent to the edge Tier which is quite a challenging and alarming situation for the case of IoT where billion of devices will send the traffic to the edge and the cloud. Such aggregation of request reduces the traffic load on the edge node and the cloud data center as well.

2) Native in-network caching and computation reuse: Nodes in NDN caches the data and even results of functions/services and make them available to other consumers without performing the computation again and again [21]. Moreover, the distance between the cloud and edge networks can be several hops, which may result in a significant delay. NDN over edge computing can achieve latency requirements by providing data and services that are close to end users via caching the content and results as well. Therefore, NDN not only provides content caching but also functions/code caching in order to avoid re-request the content or re-execute the function/code.

3) Location-independent naming: Hierarchical user-friendly Uniform Resource Identifier (URI) like names uniquely identify a content (e.g., a movie, a picture, a song) as well as context (location, identity etc), independently of the identity/locator (i.e., the IP address) of the node generating/hosting it. Therefore, NDN is not bound to specific address of content, service or context. In NDN those all reflect the named pieces of content [22].

5) In-network security: In NDN protection and trust are implemented at the packet level, rather than at the channel level. By design, NDN offers native support for security, which are still not effectively available in the host-centric paradigm. The self-certify names model of ICN enable to verify the binding between public key and self-certify name in distributed system without relying on a third party. This can reduce the security risk of involving a third party. However, it is difficult to maintain the centralized key management infrastructures such as Central Authority (CA), especially in the constrained IoT. The reason is large communication and computational overheads incurred due to complex trust chain of certificate verifications. Recently *Jun Wu et al.* propose an anonymous distributed key management scheme based on CL-PKC specifically for Space Information Network (SIN) in order to overcome the security issues [25]. Authors designed a distributed key management system model for key exchange services. Since authors scheme is based on the certificateless public key cryptosystem, therefore, it can avoid the problems of complicated certificate management. However, this was specifically designed for SIN and may be used for NDN based edge computing. In [23] authors proposed a scheme for information centric social networks (ICSN) and claimed that the existing schemes for the conventional social networks cannot fulfill the requirements of ICSN. Therefore, a fog computing-based content-aware filtering method for security services, FCSS, is proposed in information centric social networks [23]. They introduced fog computing in IC-SN, and the content aware filtering scheme is proposed for security services. Such edge computing based ICN solutions can be introduced in many NDN-based edge computing applications. Moreover, CL-PKC is one of the areas to be explored for NDN based edge computing applications.

6) Built-in Mobility support: Consumer Mobility is a built-in feature of NDN due to receiver driven and connectionless data communication nature. When an end device moves to a new location, it simply needs to re-express request for its interested data. However, support for producer mobility still a research problem in NDN. Some networks are highly mobile, such as vehicular networks (VNs). Therefore, if mobile users can only receive the content from the

provider (original server), then the connection can be lost during the mobility of users. Due to interruptions in connectivity, users will again make a request from the original server. In NDN the mobility feature (consumer) is inherently supported. The devices can directly communicate using service names instead of specific host such as Netflix.com or Youtube.com. Services are provided by the network and does not rely on end to end communication. NDN caching provides a copy of the content to all users and hence mobile users no longer make requests of the original server. Therefore, if the mobile users keep moving in network as in Vehicular Network, then content can be obtained from the nearest cache instead of going to the original server. Hence, a reduction in delay and support for mobility is achieved [24].

D. WILL NDN AND EDGE COMPUTING CO-EXIST AND WORK TOGETHER?

We argue that the combination of NDN and edge computing would speed up content retrieval. However, NDN in edge computing poses many challenges to network's infrastructure and architecture. The issue arises because of two different architectures. Inter-networking schemes with existing architecture of edge computing are necessary to make both paradigms interoperable. Since the traditional Internet is based on TCP/IP (host oriented) network model. Therefore, to replace the TCP/IP model with NDN is impractical. However, there is a way to deploy the ICN partially or overlay such as ICN over IP or IP over ICN. In our framework we are not replacing the TCP/IP model. Instead we use both approaches to work efficiently together where required. The complete details on how both technologies work together in our framework has been provided in Section IV.

III. RELATED WORKS ABOUT ICN OVER EDGE COMPUTING

Only limited number of works have been devoted to NDN over edge computing. Moreover, some works have also been proposed that can act as a bridge between IP and NDN networks. *Trossen et al.* [26] proposed an architecture that enables IP applications to communicate with an NDN network via Network Attachment Point (NAP). Similarly, *Refaei et al.* [27] proposed a general purpose, extensible IP-to-NDN gateway that translates between NDN and IP packets. The translation is based on predefined rules. *Susmit et al.* proposed IPoC [28], a protocol that can enable a transition to ICN in mobile networks by encapsulating and forwarding IP traffic over an ICN core. *Wu et al.* investigated the issues for incrementally deploying NDN in LANs using dual-stack switches that support both NDN and IP protocols [29]. However, all these protocols didn't specify the edge computing with NDN. In the literature some of the support for virtualization of services for edge computing have been provided, such as Docker [30], Amazon Lambda [31] or serverless computing technologies such as unikernels [32]. These technologies are useful for edge computing, since it provides encapsulation of functions into self-contained software components, executable on edge nodes and totally independent of its deployment structure. Specifically, about ICN with edge computing, authors in [21] proposed a pioneering scheme call ''Named Function Networking'' (NFN) for the extension of NDN to the edge computing. In NFN, the name field of interest packet carry the name of the content as well as expressions for named functions. The network is in-charge of computing the result and resolving the forwarding plane of NDN. However, NFN is constrained by the number of services/functions it can support. In many scenarios, nodes require more sophisticated processing, custom code and libraries, which is difficult to express only through simple expressions and acquiring additional function code. In [33] Named Function as a Service (NFaaS) has been proposed that supports more sophisticated processing with lightweight virtual machines in the form of named unikernels. The unikernels are actually the functions/codes. In NDN the content is cached in the CS, whereas, in NFaaS an additional data structure call Kernel Store (KS) has been introduced. Every node contains Kernel Store that stores the unikernels. The KS is responsible not only for storing functions, but also for making decisions on which functions to run locally. Since the Kernel Store has lots of functions and which functions to download locally to the node is calculated by score function. The score function scores all the popular function that is requested more frequently and the main goal of score function is to identify the unikernels/functions that are worth downloading locally into the node's memory. In [34] authors have extended the NDN architecture to turn the network edge into a dynamic computing in the IoT domain. The proposed scheme performs distributed in-network IoT data processing at the network edge, by relying on NDN augmentation and named computations. This scheme also performs the dynamic execution of services, according to the interests popularity function. In the proposed scheme authors have performed minor modification to legacy NDN and used a naming scheme that identifies IoT contents and services without affecting the NDN routing. In [35] authors have proposed an NDN based scheme call NDNe (NDN at the edge) that supports cloudification at the edge. In NDNe the existing NDN packet is extended and names are used to address, not only ''contents'', but also to identify different types of cloud service (e.g., storage, computation, etc.)

On the contrary, we propose a framework that works realistically with IP and NDN network. In order to do that we use the.Net Web API services and proposed our own API for carrying the end users request to the edge node and cloud node.

IV. PROPOSED FRAMEWORK

Our proposed framework is based on N-Tier architecture and comprises of 3 main tiers: Tier 1, Tier 2 and Tier 3 and represents Things/end devices, Edge computing and cloud computing respectively. Tier 2 and Tier 3 is then further divided into layers. Since the framework comprises of Tiers and layers, therefore, firstly, we need to clarify the differences

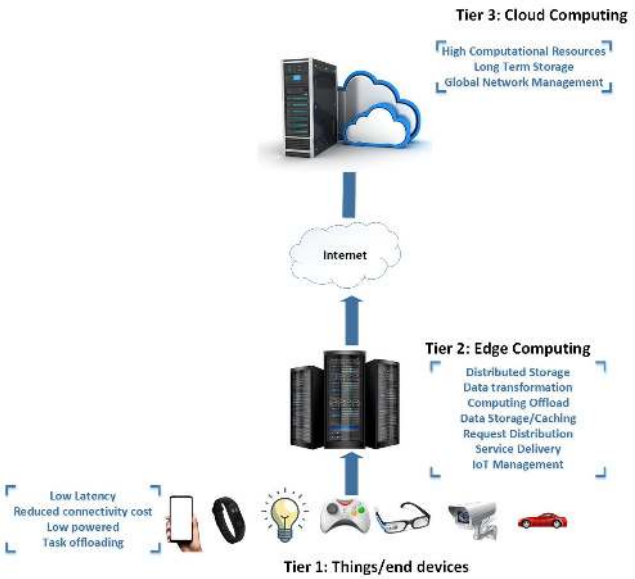

FIGURE 2. N-Tier architecture.

between the two terms in N-Tier architecture: Tier and layer and are described as follows:

A. TIER AND LAYER

Tier usually means the physical deployment of computers or devices. Usually an individual running server is one Tier. Several servers may also be counted as one Tier, such as Cloud servers and edge servers. By contrast, layer usually means the logical components of software that are grouped mainly by functionality. Layered approach has many advantages and is a good way to achieve N-Tier architecture. Each layer may run in an individual Tier. However, multiple layers may also be able to run in one Tier. A layer may also be able to run in multiple Tiers.

In this paper, first we introduce the 3-Tier concept so that the readers can understand layers in each Tier easily. After that a comprehensive detail of each layer is presented. The simplest of N-Tier architecture of our proposed framework typically contain following Tiers listed from the top level to the low level: Cloud computing Tier, Edge computing Tier and Things Tier, as depicted in Figure 2. Edge computing Tier is then further divided into the following layers listed from top level to low level: application programming interface (API) layer, web layer, service layer, data access layer (DAL), component layer, model layer and unit test layer. Furthermore, the cloud computing layer is also divided in all the aforementioned layers except the web layer.

1) TIER1: THINGS/END DEVICES

This Tier is involved with end user(s) or IoT device(s) directly. There may be several different types of IoT devices/users coexisting, such as sensor nodes, end user's smart hand-held devices (smart phone, smart watch, smart vehicles etc). These device(s) request for data and services and are connected at single hop to the edge Tier.

2) TIER2: EDGE COMPUTING

The edge computing Tier comprises of fog servers and provide delay constrained service request to end users. This Tier is located at one hope distance from the Things Tier. The edge computing Tier is not like the actual cloud computing Tier, however, provides medium number of service request with low latency.

3) TIER3: CLOUD COMPUTING

The cloud computing Tier is the top most Tier and comprises of traditional cloud servers and has enough storage and computing resources. This Tier is located at multiple hops from the Things Tier and edge computing Tier. However, this Tier comes with a cost of higher latency to the end users. All the three Tiers are illustrated in Figure 2.

B. ARCHITECTURE DETAILS

Figure 3 shows a detailed architectural diagram of the proposed framework. In the following subsections we have provided the detail of our proposed framework as follows:

1) TIER1: NAMED DATA NETWORKING

All the IoT devices which we call end user devices located at the things layer. At this layer we have used the NDN technology where all the communication is content centric, and content are requested at the network layer using interest names. All the end users/IoT devices communicate locally via NDN network. If there are some tasks that cannot be handled by the NDN, then it is forwarded to the edge device. The edge device send back the computations/services to the end users subject to the policies of edge device. In addition, if edge device is not able to perform computation on these requests then the edge will forward the requests to the Cloud data center, where further processing will take place. To aid better understanding to the readers, we explain the NDN Tier (Things/IoT devices) at first then edge Tier and cloud Tier is presented. Subsequently, we provide the complete architectural details of proposed edge Tier and cloud Tier in the subsequent sections.

At the Things Tier, without loss of generality, we have used a simple scenario of three nodes i-e consumer node, relay node and the producer node. Since the producer node act as a gateway node between the NDN network and edge node, therefore, we are using the term gateway node for producer node throughout this paper. In this scenario, the consumer node send request for content and/or services via relay node. If the relay node has the services/content cached before, then it will send back to the user as the conventional NDN architecture does. Otherwise the request for content/service will be forwarded to the gateway node. The gateway node possibly has the content and also maintains a list of services. In order to better differentiate the gateway node of NDN, edge node and the cloud node, we assume that in our framework the gateway node maintains a limited number of services for end users. The edge node is power full than gateway node of NDN

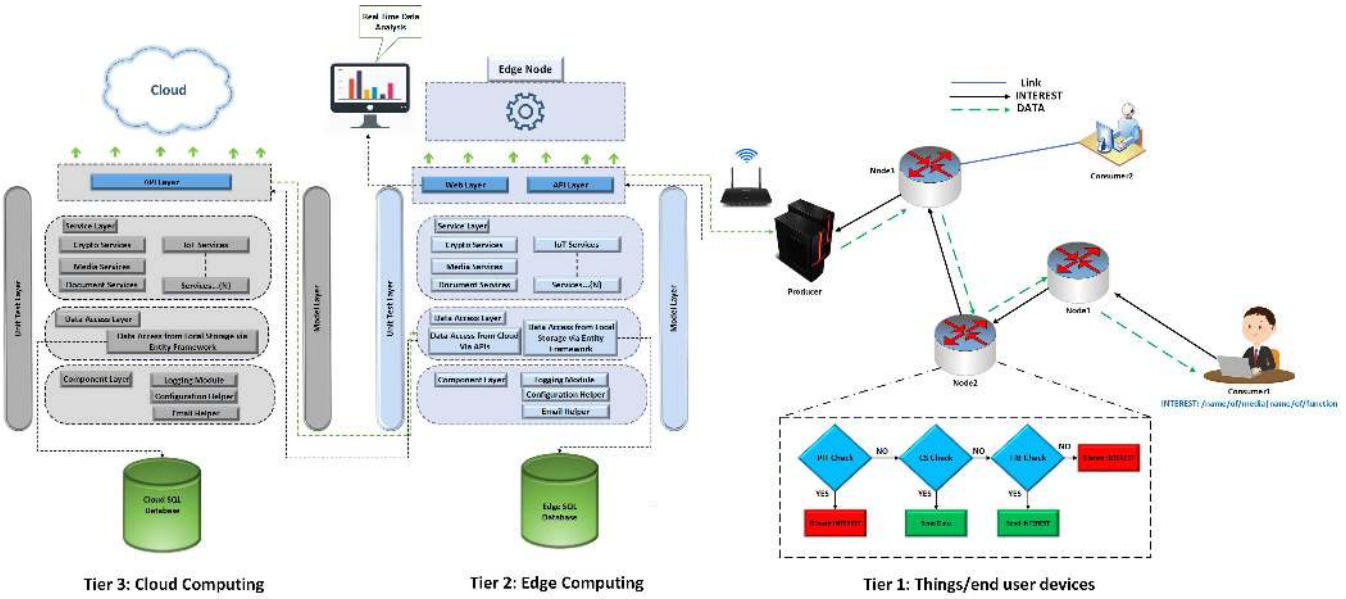

FIGURE 3. An architectural diagram of the proposed framework.

where a large amount of services is listed and similarly the cloud is resource enrich platform for all type of resources. The end user when request for data or services then interest and data exchange in NDN between consumer and router nodes will be as follows:

Step 1. The end user(s) or device(s) send request for data or service to the nearest content router (CR).

Step 2. Nearest CR discovers if the requested content/services exists in the router CS through NDN discovery process.

Step 3. If the content/services exist in the CR, the request may be satisfied, and the data/service may be sent back to the user(s).

Step 4. If the content/service is not found in the CR, the request is forwarded to the gateway node.

Step 5. Now when the gateway node receives interests from end user(s) in the NDN, it processes the received interests and respond with data/services back to the user (subject to the data and service availability). It is also possible that the end user devices are requesting less computing tasks/services and the gateway node is capable to provide such services. Therefore, all such less computing requests for data/services would be handled at the gateway node.

As the devices in IoT could be million or even billions, and many devices may request for content or services. In that case, some requests could be identical for many users (i-e same request would be sent by many end users). Therefore, we exploit the NDN aggregation feature in the NDN network. The same request from many end users will not be forwarded in the network. Instead it will be aggregated at the NDN relay nodes or at the gateway node which substantially reduce the traffic on the gateway as well as at the edge and cloud node. Moreover, the in-network caching will also alleviate the interest forwarding and the devices will access the data from nearby nodes in the network. Exploiting such feature of NDN results in lower delay, better mobility management, in network caching, and security of services and the content. The usage of NDN enhances the data communication efficiency and minimize access to the cloud and edge since all the subsequent data access does not need to go to the cloud computing Tier and can be accessed from NDN nodes and edge node as well. The subsequent requests for the same content will be aggregated and filtered by NDN routers. The services and data are brought at very close vicinity of end users.

It is also important to mention that the gateway node might not have the requested services, processing power or computation resources for some complex tasks. Therefore, the gateway node will try to benefit from the edge node which is much powerful than the gateway node of the NDN network. In that case, in order to avoid failure of request or non-availability of data/services, the NDN gateway node will communicate with the edge node which is outside of the NDN network. At this point, we need to switch from NDN network to IP network of the edge node. In our framework, the gateway node runs both NDN and IP. It is also to be noted that the gateway node could be multiple in number, since in dense IoT environment a single gateway node could be a bottleneck for end user devices. Therefore, for dense scenario, we also deployed multiple gateway nodes in our framework for the communication with end users and edge device as well.

For communication with the IP based edge device, we have used Web Application Programming Interface (Web API). In our framework, the web API connects the NDN network with the edge computing device located at Tier 2 (Edge computing). We have defined specific layers for each kind of services at the edge node. All such layers are discussed in the edge Tier in the subsequent section as follows:

2) TIER2: EDGE COMPUTING

In our proposed framework, the Tier 2 represents the edge computing where our edge node application is deployed. We further divide the edge computing Tier into seven different layers and each layer is discussed in detail as follows:

1) Layer 1: API Layer The term API stands for "Application Programming Interface". APIs can provide access to hardware and software resources and allow developers to save time by taking advantage of a platform's implementation. In our implementation we have used RESTful API which stands for "Representational State Transfer" and it is an architectural pattern for creating an API that uses Hypertext Transfer Protocol (HTTP) as its underlying communication method. Almost every device that is connected to the Internet already uses HTTP; it is the base protocol that the Internet is built on. HTTP is a request and response system where an interested user sends a request to an endpoint and the endpoint responds. REST uses HTTP requests, responses, verbs and status codes for the communication purpose. Any IoT device can consume HTTP such as toasters, cars and sensors etc. Following are the several key implementation details with HTTP that we have used in our framework for the communication between end user IoT devices and the edge devices.

Resources: REST uses addressable resources to define the structure of an API. The resources in our framework typically represent the services (i.e. crypto services, media services, document services and IoT services etc). We include a few services in our prototype and more IoT services could be added according to the requirements.

Request Verbs: The verb describes what to do with the resource. End nodes in IoT issue a verb to instruct the edge node for performing some action. There are many verbs available such as GET, POST, PUT and DELETE. For example, a GET verb gets data about an entity, POST verb creates a new entity, DELETE verb delete an entity and update verb update the existing entry on the edge node. The end user can issue GET requests for services such as 192.168.72.1:82/api/services that will returns a list of services on the edge node (Edge node IP address: 192.168.72.1). However, to retrieve a specific service from the edge node, the end user or IoT devices use query string parameters with an API. For example, it could be something like 192.168.72.1:82/api/Services? Media=video which returns all the video services to the end users. The detailed explanation of the our proposed web API URL structure is presented in the follow up subsection *(Proposed.NET Web API)*.

Request Headers: The request headers contain additional instructions that are sent with the request for services or data.

Request Body: The request body contains the data that is sent with the request. For example some functions/services require some data such as conversion of media file (i- e mp4 to mp3) or document conversion (i-e word to pdf), therefore, such data is typically sent as the request body.

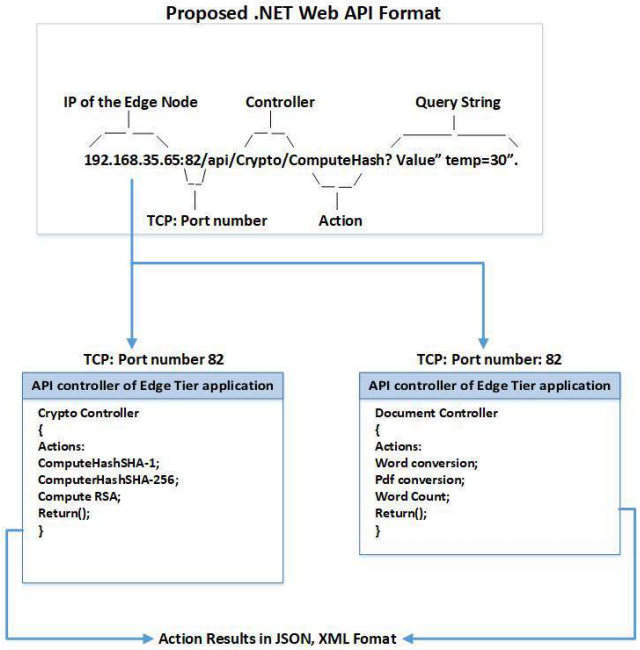

FIGURE 4. Proposed.NET web API for IoT services.

Response Body: When the request is received at the edge node then the edge node respond with a service or any computation. The response body contains the services which is requested by the end users.

Response Status codes: When the response is sent back to the end users then the end users should be informed with the status of the requested services. For example, the services which is requested by the end users is not available or might not have any resources that is requested, then these codes are issued with the relevant response and give the details on the status of the request.

We have tabulated some of the key resources which we have used in our framework and has been shown in details in Table 1.

Proposed.NET Web API: For the communication of IoT devices with the edge node and cloud node, we have used.NET Web API which is an easy way to implement a RESTful web service. All the requests from NDN network comes in the form of an API and hit the Web API (Microsoft ASP.NET Web API 2.0) layer of the edge node. This API layer provides a kind of interface to the requests coming from NDN network. For APIs deployment we are using Microsoft Internet Information Services (IIS) which host websites, web applications and services needed by users or developers. Web API uses the controller and action concepts and the resources are mapped directly to controllers. There could be different controllers such as Crypto, Document, Media, IoT etc. The structure of our proposed.NET web API is depicted in Figure 4 and is described as follows:

Our proposed API comprises of an IP, port number, Controller, action, and the value. The values could be multiple in some cases. When the end user/IoT device(s) wants to request

TABLE 1. An example of REST resources, verbs, expected outcome and response codes for the proposed framework.

Resource	Verb	Expected Outcome	Response Code
/Crypto Services	GET	A list of all services in the edge device	200/Done
/Sensor Data	POST	Creation of a new Sensor Data	200/Created
/Crypto/ComputeHash? Value= 30	GET	Compute Hash of values whos temprature is 30	200/Ok
/Service/Compression (a service which does not exist)	GET	Some error message	404/Not Available
/Media Services	PUT	An update to the Media file	200/Found
/IoT Services	DELETE	Deletion of Sensor Data	200/Deleted
. . . Services N	DELETE/POST/ GET/UPDATE/PUT	Relevant Messages	Relevant Response Code

some service from edge node then it send an API request to the edge node with all the information embedded for that service in an API.

Let's explain the complete procedure via an API request from end node which we have implemented in our framework and is depicted in Figure 4. To aid better understanding we considers an API of Crypto Controller such as 192.168.72.1:82/api/Crypto/ComputeHash? Value=30. The first part of an API represents the IP address where the edge node is located. The IP address "192.168.72.1" is the IP of our edge node (located in our Lab: Broadband Convergence Network Laboratory, Hongik University, Sejong Campus, Republic of Korea). Once the request arrives at the edge node via IP address "192.168.72.1" then the Transmission Control Protocol (TCP) fetch the port number of the specific application where the API application is deployed. For example, in our case we have used port number 82 for our API application on server. If the port number field is left void, then the port number-80 will be assigned by default. Now once the request reaches at specific port number then the API controller part is checked. Typically, APIs are held within a '/api/' route which helps to distinguish an API controllers from another non-API (such as web controller) in case web application and an API are running on same TCP port. In the proposed Edge Tier API application, the API controller is the main class, which is a base class for our Edge Tier API controller classes such as crypto controller, document controller, media controller and IoT controller etc. The controller defines the service types on which some possible actions might be performed. For example, in our framework we have used a crypto controller where the main actions are ComputeHash1, ComputeHash256, and Compute RSA etc. If the crypto controller is called, then further actions are performed such as ComputeHash1, ComputeHash256, and Compute RSA etc. Once the action is performed then the execution handler is called, and the results are then returned in Extensible Markup Language (XML) format or JavaScript Object Notation (JSON) format via response body to the IoT end user. There could be multiple actions with same name (action overloading) in each Controller. Therefore, the value field in an API will distinguish the multiple actions. For example, in Figure 4 we show the document controller. The document controller comprises of various actions such as word converter, pdf converter, word count etc.

Layer 2: Web Layer The web layer shows a graphical user interface for real time data display in our framework. In our case, it shows the real-time statistics of requests from end users to edge Tier and from edge Tier to the cloud Tier. We can see the actual requests with GUI via web layer that are coming from the Things Tier to the Edge Tier. For future work, we have planned to provide the real time statistics and graphs of related performances metrics such as end to end latency, RTT, Traffic load, backbone traffic reduction etc (since our work is prototype, therefore the real time graphs are still under development and it is open for any possible contribution from the research community).

Layer 3: Service Layer This layer coordinates data and services between an API layer and the Data Access Layer (DAL) (which is explained in the next subsection). The API layer send requests to the service layer and the service layer accept all the request from an API layer. The service layer contains all the business logics of the services on which the decision might be taken. In our prototype, we just mentioned services like crypto services (to perform some crypto related services such as ComputeHash etc), media services (such as audio/video conversion etc.), document services (such as pdf/word conversion and word count etc.) and IoT services. It is to be noted that this is a prototype implementation and the framework is open to implement any kind of service in the services layer. After receiving the requests from an API layer, the service layer checks the available services. If the request from end user devices are computational requests such as document conversion or media conversion, then the services layer will decide to coordinate with the component layer where the computations will be performed. The component layer is explained in next subsections in detail. However, if the service layer received POST request (insert soma data), GET request (retrieve some data), DELETE or UPDATE request, then the services layer will coordinate with the DAL. The service layer in coordination with DAL will decide whether to retrieve the data from local storage of edge node or cloud node. For example, if the data is available in the local storage, then the DAL will provide the data from local storage to the services layer and then the service layer will send data back to an API layer which then send it to Things Tier users. Since the edge node is not powerful as the cloud node. Therefore, there might be cases where the data might not be available at the edge node and need to be fetched from the cloud node.

In that case, the DAL will fetch the data from the cloud via cloud API. The service layer will then request to the DAL to bring the data from the cloud node. Moreover, for the POST request(s) the service layer will take decisions whether to store the data in local database of edge node or cloud data base. In our framework, the services layer performs the following major operations:

- Processing requests from the end user IoT devices via an API layer.
- Coordination between DAL and an API layer.
- Accessing data from the DAL via Local storage of edge node and the cloud storage.
- Making logical decisions upon the request(s) received from the end user device(s).
- Performing computations and calculations on various services such as crypto services, documents services, media services and IoT services with the help of component layer.
- Sending the data/services to an API layer in order to reach the data to end users.

Layer 4: Data Access layer (DAL) The DAL is the layer where data management occurs. Typically using a database such as SQL, MySQL, Oracle MongoDB etc. In our implementation we have used SQL data base. The code modules in DAL is triggered based on two major decisions; local storage and cloud storage. The service layer instructs the DAL whether to store the data in the local storage or cloud storage. By storage we mean the SQL database. The DAL then stores the results locally or send to the cloud subject to the application requirement and edge policies. In order to communicate with the local SQL database, we are using Microsoft Entity Framework (an underlying communication framework between database and an application). The Entity framework automate all the database related activities for our application. Automated database commands are generated for reading or writing data in the database and executing them for end users. Entity Framework provides relevant libraries and execute the relevant query in the database to create results for our application. Generally, the Entity framework uses three different approaches for communication with the database such as Code First, Database First and Model First. In the Code First approach, the classes are defined first and then Entity Framework comprehend the conceptual model of tables and related schemas of database. In the Database First, the tables and all other schemas are defined first and then Entity Framework defines the classes. While Model First uses a Visual Designer to define the conceptual model, which can then generate the classes to be used in the application. According to Microsoft, the Model First approach is less used approach and other two approaches are used often by the developers. In our implementation, we have used the Database First approach.

Layer 5: Unit Test Layer We have provide a unit test layer in order to ease the testing of our framework. That means if one's wants to add a new service(s) or to change some services in the framework according requirements, then the unit test layer will perform the testing of such service(s). The process will not be repeated from the beginning in order to test the newly added service. However, it will be checked directly from the unit test layer. Let's explain the unit test layer of the proposed framework as follows:

In each layer we need to write some codes for some kind of service(s) to be requested. As the API layer provides communication between Things Tier, Edge Tier and Cloud Tier. Therefore, first we need to write a code for specific service(s) to be requested. For instance, an end user wants to request some service i-e document service from the edge Tier. Therefore, the request will be sent to the edge Tier via an API comprising of IP of the edge node, controller, action and the request query as discussed in an API layer. The API layer can access service layer, DAL, and the component layer. These all layer have the accessibility to each other. Now if a developer wants to change some code in any specific layer, then the developer needs to go to the specific layer to change the specific code. In order to test the code whether it works correctly or not, an API request from the web browser or Postman (Postman is a tool for performing testing with an API) with specific API will be sent. This request will be forwarded to each layer and the newly added code will be tested. It is a hectic job and time consuming. Why not to make a separate layer and just to use that layer for testing the code in each layer. Therefore, we have used the unit test layer to perform the testing of newly added or updated code in each layer. For that we only need to make a sample code in the unit test layer. This unit test layer provides the testing of code in each layer during development of code for services to be used in IoT scenarios or some other scenarios. We argue that if we use the unit test layer then there are 95% chances that an API would be working correctly, whereas, in case of no unit test layer we are not sure whether the test would be successful or not. With unit test layer we write a function or other block of application code and create unit tests that verify the behavior of the code in response to the correct cases of input data. With test driven approach in this framework, we have provided a facility to create the unit tests before testing the actual code. Using unit testing we are able to promptly catch any bug introduced due to the update or change. In case we don't use the unit testing in place, we need to write the code, fire up the GUI and provide a few inputs that hopefully hit the code. However, if we have unit testing in place, then we write the test, write the code and run the test. Debugging becomes very easy. If a test fails, only the latest changes need to be debugged.

Layer 6: Component Layer In the component layer, we have provided those code modules that are shared among different layers such as helpers (configuration helper, email helper and log helper) and extensions etc. This layer is also open to contribution and to add any kind of relevant helper such as custom type conversion helper, extension helpers, and security helper. In this layer, users are provided with configuration helper class to configure the edge/cloud node

configuration directly instead of going to the web.config file of the edge Tier. By doing so the re-deployment of the modified Dynamic Link Library (DLL) files can be avoided. In our first version of edge and cloud codebase, we have provided 3 helper classes, 1) Configuration Helper, 2) Email Helper, and 3) Log Helper. Email helper provides the email services to the administrators (administrator can be anyone who is managing or monitoring the edge or cloud and his/her email address will be provided in web.config file at the time of deployment) and the administrators will be informed about all the activity of edge Tier. Logging is a means of tracking events that happen when some application (edge or cloud) runs. The application developer adds logging calls to their code to indicate that certain events have occurred and is described by a descriptive message. Therefore, we have provided a logging module in the component layer to ease the effort of understanding. Currently this logging module have two types Debug and Exception. For future we have planned to add more types such as Info and Warning Logs (this logging module is also open to contribution).

Layer 7: Model Layer The model layer comprised of different types of classes used for carrying data among different layers. A model typically represents a real-world object that is related to the problem or domain space. However, in programming, we create classes to represent them. These entity classes known as models, have some properties (defining their behavior) in a particular domain space. For instance, in our edge application if one wants to save the Load_Request time in database then the Load_Request_Time Model must be defined which may include different properties such as Id, Request_Arrival_Time, Request_Completion_Time, Requesting_Node_Name and Requesting_Node_Type. Our model layer contains 3 different types of models 1) Custom Models, 2) Database Models, and 3) Database Partials. Custom models may consist on those classes which may use in custom requirements. For example, we may use these classes for the computational purposes. These classes may or may not represents the database schema. Classes in database model represent the Tables in the database. Each database model class map with a single table in a database and each of its property map with the column name. Database partials are replica of database model however these classes are used for the validations such as required validation, length validation, data type validation, and extension validations.

3) TIER 3: CLOUD COMPUTING

The cloud node comprises of all the layers which are detailed in the edge Tier except the web layer and the subpart of the data access layer where request is sent to the cloud for data access (since we don't need to get data farther than the cloud). We deployed our cloud node at multiple hops at the Microsoft Azure Windows server 2016 and for the cloud storage we have used SQL 2017 developer version. Any database can be used in this framework for storage such as Oracle, My SQL, MongoDB etc. In the evaluation section we have provided the complete details about the cloud server specifications.

V. PROTOTYPE AND EVALUATION

In this section, we first introduced our experimental testbed which is built on our NDN based edge computing framework. Then we measured the possible performance metrics.

A. ENVIRONMENTAL SETUP

The testbed includes Edge node machine, access point, and Microsoft Azure Cloud server machine. The detailed hardware and software configurations are as follows:

For NDN network we generate requests from ndnSIM to the edge node. In order to do that we modified some of the code in *ndn-producer.cpp* file and *ndn-producer.hpp* file. Our custom function which generates the request is using *boost/asio.hpp* library and it's class is *asio::ip::tcp*. ndnSIM is running on Linux using VMware. The VMware has 8 GB RAM and 4 core CPU.

Edge application is hosted at one hop from NDN network on a system with specifications of 16 GB RAM, core-i7, 4710HQ-CPU and @ 2.40 Ghz core. We developed our edge application using.NET framework. For the deployment of edge application, we have used Internet Information Services (IIS). IIS is a flexible, general-purpose web server provided by Microsoft. The IIS web server accepts requests from remote client computers and returns the appropriate response. This basic functionality allows web servers to share and deliver information across local area networks, such as corporate intranets, and WANs such as the Internet. A web server can deliver information to users in several forms, such as static webpages coded in HTML; through file exchanges as downloads and uploads; and text documents, image files, JSON, XML and more.

For cloud application we have used Microsoft Azure Window Server 2016 Data Centre with specification of 8 GB RAM, Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30Ghz 2.29Ghz. Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems. Cloud application is also developed in.NET framework and deployed using IIS server. For the database of the edge node and the cloud node we have used SQL 2017 developer version. Our NDN based edge computing testbet is shown in Figure 5.

It is to be noted that we have used a limited version of Cloud VM with limited amount of resources for testing our system. However, these resources were enough for our experiments. Since we need the Cloud VM just for testing the latency related measurements and to show the real cloud behavior at multiple hops. Therefore, high amount of resources were not necessary for our evaluations. We testify our system with lower amount of Cloud VM resources and that were enough for our measurements. The system is

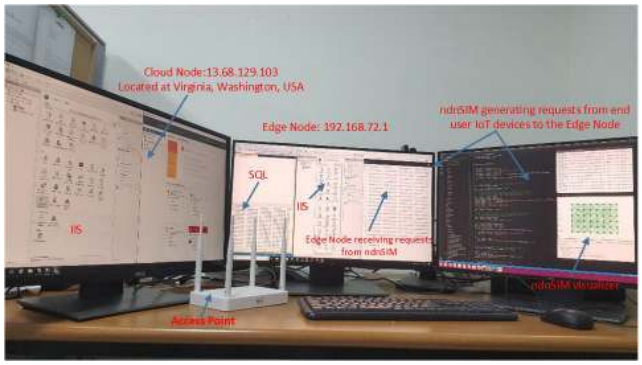

FIGURE 5. NDN based edge cloud computing testbed.

completely scalable; however, it depends on the user requirements. If one's wants to scale the system at large scale then they could scale it easily just by buying the Cloud VM according the requirements.

B. RESULTS AND DISCUSSION

Edge computing and NDN promises to increase the performance of several applications by using data locality. Moreover, it is also able to relieve the core network by addressing the increasing bandwidth demands caused by the increase of services, data volume, and IoT devices. In the following subsections we evaluated the relevant possible performance metrics of NDN and edge cloud computing.

1) INTEREST AGGREGATION IN SPARSE ENVIRONMENT

In this experiment, we have exploited the interest aggregation feature of NDN. First, we conduct the experiment with simple IP (without NDN). That means if we don't use NDN then how much interest packets would be forwarded to the edge node. By number of interest packets we mean the traffic generated from end users/things to the edge device. In case of IP all the requests from users is forwarded to the Edge Tier. In order aid better understanding we consider a simple topology of 4 nodes in NDN network which is connected with the edge node at most one hop and with cloud machine at multiple hops. Out of 4 nodes in our topology, we set 2 nodes as consumer nodes, 1 as a relay node and 1 as a producer node which also act as gateway node for the edge device. The consumer 1 starts sending requests when the simulation starts. Consumer 2 started sending request just after 10 ms after the simulation start time. The reason to send request after 10 ms for consumer 2 is to check the behavior of PIT entries and interest aggregation at the relay node. Since those requests in forwarded via relay node to the gateway node and then from the gateway node it is forwarded to the Edge node. Therefore, the relay node start aggregation for all the consequent incoming interests in the PIT table in order to reduce the traffic on the edge node. All the incoming requests for the same content will be aggregated in the PIT table and will not be forwarded further to the edge Tier. However, in case of IP, since there is no PIT table. Therefore, all the requests will be forwarded to the edge Tier.

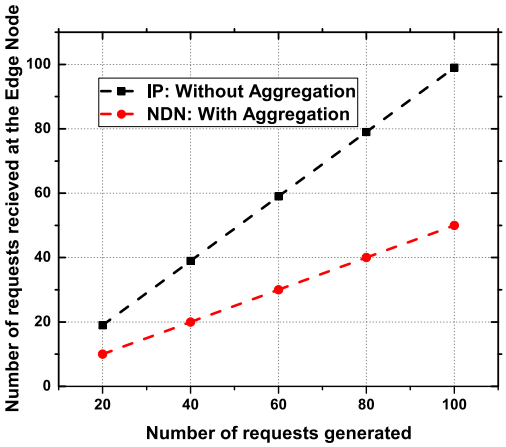

FIGURE 6. Number of request received at the edge node as a function of number of requests generated.

Figure 6 depicts the number of requests generated as a function of request received at the Tier 2 (edge node). It is evident from Figure 6 that for 20 number of interest packets only 10 packets are forwarded to the edge node, while the rest of 9 requests were aggregated in the PIT table of the relay node. Since the second consumer started sending interest packets after 10 ms therefore, total 19 request were generated over all. Whereas in case of no aggregation all the 19 requests are forwarded to the edge node. It is to be noted that both consumers have interest frequency of 1 interest /sec with a total simulation time of 10 seconds. The consumer 1 sends 10 requests in 10 seconds which make entries in the PIT table. Now when the consumer 2 sends the same 10 requests with a delay of 10 ms then all the interest packets are not forwarded further. Instead they are aggregated in the PIT table of the relay node. Therefore, only 10 interest packets are forwarded only. We have observed that 47.73 % of traffic is accommodated at the relay node via interest aggregation feature of NDN. Without aggregation 100% of the traffic is forwarded to the edge node. The same is the case for 40, 60, 80 and 100. From this experiment we have concluded that if we use the aggregation feature of NDN, then we can reduce a large amount of incoming traffic towards the edge node. Without aggregation, all the traffic from the things Tier will be sent to the edge Tier which is quite a challenging and alarming situation for the case of IoT where billion of devices will send the traffic to the edge and the cloud.

2) INTEREST AGGREGATION IN DENSE ENVIRONMENT

In this experiment we measured the interest aggregation of NDN with Edge computing using dense environment. In order to check the effectiveness of interest aggregation, we run the same topology for a dense environment with higher interest rate. Figure 7 depicts the behavior of interest aggregation with various number of interest packets rate. Initially, 2000 interest packets are generated by both consumers (i-e consumer 1 and consumer 2). However, only 1288 interest packets were forwarded to the edge node. That means 35.6% of traffic is reduced at the edge node using PIT interest aggregation feature of NDN at the relay node.

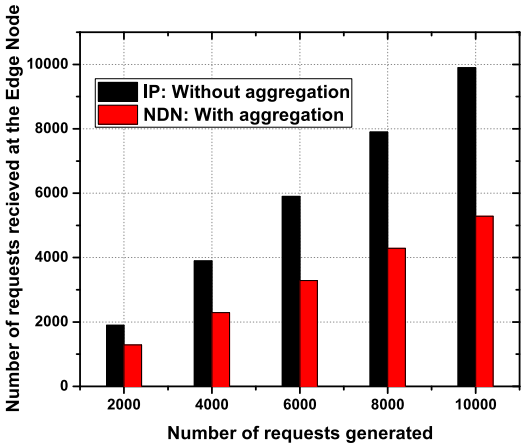

FIGURE 7. Number of request received at the edge node as a function of number of requests generated.

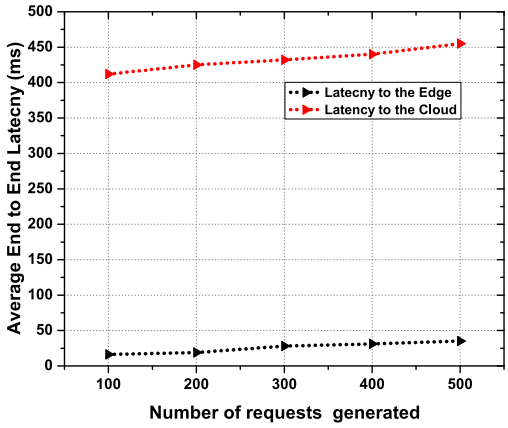

FIGURE 8. Average end to end latency as a function of number of requests generated.

Moreover, when we sent 10,000 number of interest packets then 5288 number of interest packets were forwarded to the edge node. That means 47.12% of traffic is accommodated at the edge node and 52.88% traffic were aggregated at the NDN relay node. From this experiment, we observed that in case of IoT, NDN is a promising approach with edge computing to reduce the backbone traffic on the edge Tier and cloud Tier as well.

3) EDGE VS CLOUD LATENCY

We investigated latency performance measurements of our cloud application hosted on Microsoft Azure cloud and an edge application hosted in our Lab at one hop. The latency measurements are based on Round Trip Time (RTT). We sent various number of interest packets per second from the NDN network using ndnSIM simulator. The end devices/IoT nodes send interest packets to the gateway node of NDN where the gateway node is connected to the edge device at one hop distance. Furthermore, our edge device is further connected to the Microsoft Azure Cloud machine (IP: 13.68.129.103) located in Virginia, Washington, USA. All the requests are forwarded from the IoT devices to the gateway node, and then from gateway node to the edge node and cloud machine. In order to calculate RTT, we first notice the request send time (when the request is sent from IoT device to the edge and cloud) and the request return time (when the request is returned from the cloud and edge node). All such information are traced in *RTT_tracefile.txt* and the average end to end latency were measured using the *Tracehelper.exe* tool (which we have developed in.Net framework). Figure 8 depicts the collected latency measurements. It is evident from Figure 8 that the latency has a strong relation with the physical distance of the edge node and the cloud node. Since the edge node is at one hop distance from the end users/IoT devices where the requests are generated. Therefore, the latency is very minimal. By increasing the number of interest packets per second, the latency is changing very slightly. The reason is the one hop physical location of the edge node. However, the latency to the cloud is very high and shows minimum latency of 408 ms at 100 number of requests from end users/IoT devices. The reason is that the cloud is located at multiple hops and far from end users, thereby resulting in a significant increase in average end to end latency. This study shows that the edge computing is a promising approach for end user devices to satisfy their requests with lower response time. By sending all the traffic to the cloud Tier not only increase the backbone traffic on cloud data center, but also might result in high end to end latencies which is not suitable for the real time application such as AR and VR. According to our measurements, a latency lower than a 10 ms is achieved with edge node. We have noticed an approximate average end to end latency of 8 ms from end user devices to the edge device and almost 2 ms processing delay at the edge node which might be sufficient for most of the low-latency application requirements.

4) SINGLE QUERY LOOKUP TIME

In this test we have performed the measurement of a single resolution query from NDN to the edge node and cloud node as well. That means a single gateway node in the NDN network has been used which send services requests to the edge Tier.

a) Edge Service Lookup Time: Figure 9 depicts the edge service lookup time for varying interest rate in relation to different amounts of edge node entries (1K, 10K, 20K and 50K). As in the case of IoT, various IoT devices may generate various data with different rates, therefore, we varied the number of interests per second from 10 to 90 with an interval of 20. We have checked the behavior of edge node when different number of requests received from IoT devices at the edge node. For instance, each second 10 interest packets are generated from a single gateway node and each interest carries a single query that is checked in 1K, 10K, 20K and 50K. The same is repeated for 30, 50, 70 and 90 in relation to the edge node entries. It can be observed that the lookup time for 1K entries is 8 ms and is increases slowly when the interest rate is increasing. The linear behavior is due to the increase in load of services request from IoT device while searching in the edge entries. It is also noticed that the lookup

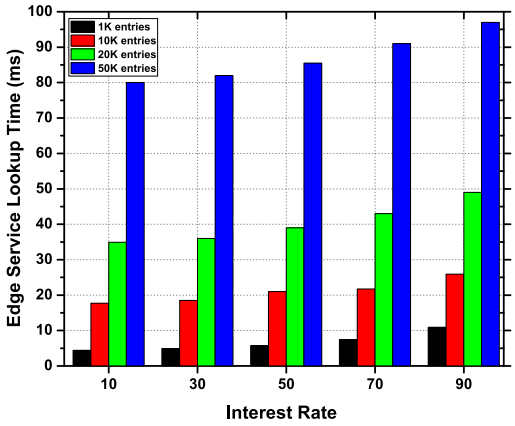

FIGURE 9. Edge service lookup time as a function of interest rate.

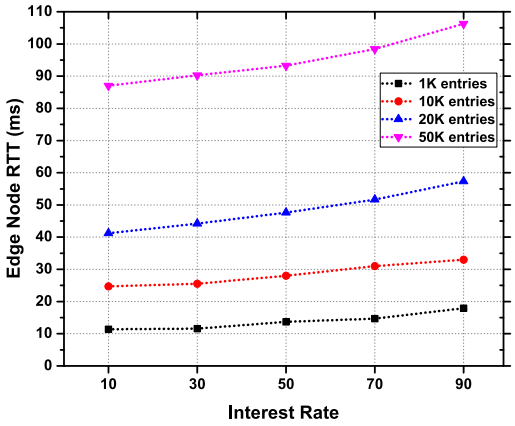

FIGURE 10. Edge node RTT as a function of interest rate.

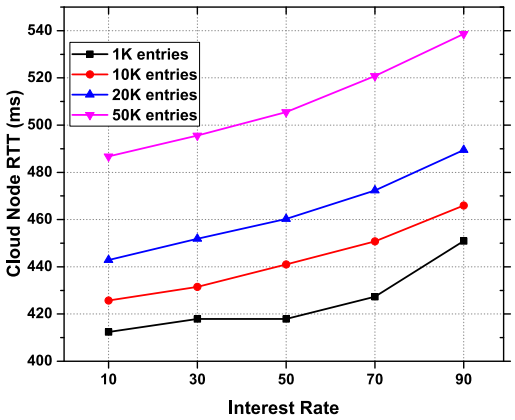

FIGURE 11. Cloud node RTT as a function of interest rate.

time for 10K, 20K and 50K increased much higher when the single interest against various number of interest packets is checked. The edge service lookup time for 50K is much higher comparing to other edge entries, since 50K comprises of higher number of services entries and the lookup time suddenly increased for looking a single query in 50K entries of the edge node. From this experiment we have observed the behavior of the edge node with lookup time against various number of edge entries. This shows that the edge computing promises a very good lookup time comparatively with the cloud computing which is presented in the next subsections.

b) Edge Node Round Trip Time: In this test, we have conducted measurements of the edge node device in terms of RTT under various Interest rate for various number of edge device entries. We have checked the total RTT from Tier 1 (IoT devices) to Tier 2 (Edge node) and then from Tier 2 to Tier 1 while checking for the entries in the edge device. The RTT comprises of the network latency form Tier 1 to the Tier 2 and from Tier 2 to Tier 1 including the processing delay at Tier 2. RTT is calculated by sending various number of interests per second from the NDN network using ndnSIM simulator. In order to calculate RTT, we first notice the request send time (when the request is sent from user to the edge and cloud) and the request return time (when the request is returned from the cloud or edge node) as discussed before. The IoT devices are sending interest packets from NDN network which is then forwarded via gateway node to the edge node. After processing the request from IoT devices, the edge node then returns the response to the IoT devices and the total RTT is calculated.

In Figure 10, for 10 interest packets per second the total RTT for 1K entries to the edge node is approximately 11 ms and 13 ms for 30 interest per second. The RTT increases linearly for 1K entries, since we are increasing the interest rate. Similarly, for edge entries 10K, 20K, and 50K, the RTT increment is higher. We have observed that for 50K edge entries the RTT is very high. The reason is that processing is quite higher because 1 interest is checked in the 50K entries and then the response is then sent back to the IoT devices.

c) Cloud Node Round Trip Time: In this test we have investigated the RTT performance measurements of our cloud application hosted on Microsoft Azure cloud with various interest rate for various cloud entries. It is evident from Figure 11 that the latency has a strong relation with the physical distance of the cloud. Since edge is at one hop distance from the users where the requests are generated. Therefore, the latency is very minimal as also shown in Figure 8. The latency to the cloud is much higher and shows minimum RTT of 408 ms at 10 number of requests per second for a single query search. For 50K entries the RTT exceed 480 ms which is quite higher RTT for real time applications. The reason is that the cloud is located at multiple hops and far from end users, thereby resulting in high RTT.

5) LOAD BASED LOOKUP TIME

In this experiment, we have conducted the measurements of the edge service lookup time under various load conditions. In the previous test, the scenario was not dense, therefore, interest rate was limited to 90. However, there may be cases when the network is very dense and the IoT devices requesting services in a dense manner. Therefore, we have considered an ultra-dense network scenario for this evaluation. In this test, multiple gateway nodes are deployed, where each gateway node start sending requests towards edge node at the same time (approximately).

a) Edge Service Lookup Time: Figure 12 depicts the behavior of edge node by varying the load of parallel requests

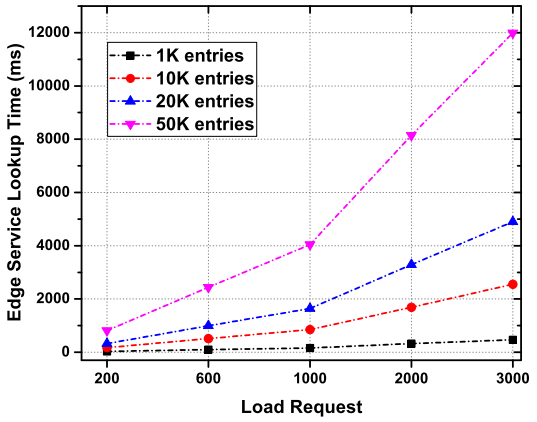

FIGURE 12. Edge service lookup time as a function of load request.

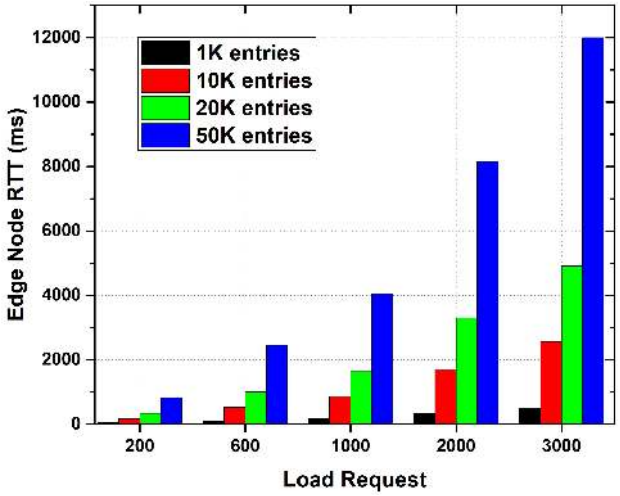

FIGURE 13. Edge node RTT as a function of load request.

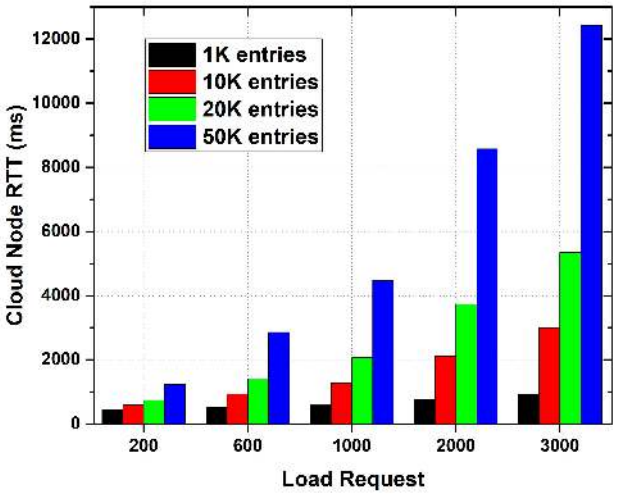

FIGURE 14. Cloud node RTT as a function of load request.

from 200 to 3000 in relation to different amounts of edge node entries (1K, 10K, 20K and 50K). The edge service lookup time for 200 number of parallel requests shows lookup time of 32.95 ms for 1K entries. Similarly, for 600 number of parallel requests the lookup time of 172.31 ms was noted for 1K entries. Similarly, for 10K, 20K, and 50K the edge service lookup time is increasing higher and higher and we have observed a significant increase in the lookup time for 50K with various number of parallel requests. If we compare Load based lookup time with the single query lookup time, then this lookup time is very high. In the single query the lookup time for 1K edge node entries were 8 ms whereas in the load-based lookup it is 32.95 ms. This shows a huge difference when we have used sparse and dense network. As in IoT networks millions and billions of devices will be connected and all such devices may send a massive amount of data. Therefore, it is a best design choice to process data at the edge node. If the data is sent to the cloud, then it may result in a high amount of Lookup time at the cloud Tier and is not a good practice for many IoT applications.

b) Edge Node Round Trip Time: Figure 13 shows the evaluation of RTT with various load conditions and different edge device entries. We have checked the total RTT from Tier 1 to Tier 2 and then from Tier 2 to Tier 1 while checking for the entries in the edge device. The behavior of edge node is checked by varying the load of parallel requests from 200 to 3000. The edge node RTT for 200 number of parallel requests shows RTT of 40.90 ms for 1K entries. The 40.90 ms includes the processing delay at the edge node and an average RTT to the edge node. As soon the interest load request is increased, RTT for edge entries 10K, 20K, and 50K also increased. For 50K edge entries with 200 number of requests the RTT of 821.11 ms was noted. The reason is that processing is quite higher because 1 interest is checked in the 50K entries and then the response is then sent back to the IoT devices. Apparently results of Figure 12 and Figure 13 looks similar, but in fact the values are slightly different in both plots. For instance with 50K edge node entries and with 3000 number of parallel requests, the edge service lookup time is approximately 12000 ms (Figure 12), whereas in the edge node RTT (Figure 13) it is approximately 12,011 ms. The reason is that the Figure 13 includes the average RTT (approximately 11 ms) to the edge node in addition to the processing delay at the edge node.

c) Cloud Node Round Trip Time: We have also checked the Cloud RTT with various load request for various entries. Figure 14 depicts that RTT to the cloud is much higher and shows a minimum latency of 439 ms at 200 number of parallel requests. The 439 ms includes the processing delay (approximately 31 ms) at the cloud node and an average RTT to the cloud node (approximately 408 ms). As soon we increase the number of parallel requests, the RTT is increasing higher and higher. For 50K entries with 3000 parallel requests the average RTT of 1223.11 ms was noted. That means that if we increase the load requests from end users then the RTT could reach to a very high level and it is very alarming and vulnerable for many IoT applications. The edge RTT, cloud RTT and edge services lookup time clearly shows that sending the traffic to the cloud is not a good practice for future IoT networks.

Moreover, we have also noticed that the use of NDN with ECC also creates some issues with the basic NDN design parameters. These design parameters of ndnSIM should be changed if one's wants to use NDN with ECC. For instance, ndnSIM provides default design parameters of 4 seconds for maximum RTT and the ndnSIM community argues that the 4 seconds PIT time is enough for the data packet to come back. However, if we use the same design parameter with edge cloud system then the requests might not be satisfied with in 4 seconds. Invoking computation at the edge device(s) may take relatively long than a simple Interest and Data interaction. Therefore, most of the packets might be still on the way and the NDN PIT entry will be purged and will result in a heavy traffic losses. Such losses results in retransmissions and create congestion in the network. Furthermore, many real time applications such as AR/VR may not afford the PIT time with 4 seconds. Therefore, such design parameters should be checked while working with NDN and ECC system.

VI. CONCLUSIONS

In this paper, we have proposed and discussed the design, implementation and evaluation of NDN based ECC framework. In order to solve the challenges of real-time services and to improve the performance of real time services, the framework combines well-known innovative technologies: NDN and ECC. NDN offers network layer content and services, increased caching, built-in mobility, built-in security, and interest aggregation. In addition, edge computing offers computation, storage, and increased caching capabilities in close proximity of end users/devices, thereby reducing end to end latency and backbone traffic. In our proposed open source framework, we have leveraged and exploited the features of both technologies in order to empower the future networks. We have implemented NDN based ECC framework prototype and conducted extensive experiments which show that NDN with ECC reduces the backbone traffic on the edge node and cloud node as well. Similarly, edge service lookup time, edge node RTT and cloud node RTT is evaluated for various load request from end users. We have observed that the data should be processed in the vicinity of end users with enabling technologies for future Internet such as NDN and edge computing. Although clouds are rich in resources, it cannot fulfill the end to end latency requirement of future networks such as AR and VR due to long physical distance. From experimental results, we have concluded that the NDN and edge computing are promising technologies to reduce the latency, backbone network traffic and can facilitate the resource-intensive and latency sensitive IoT applications.

A NOTE ON REPRODUCIBILITY

We make our implementation open and publicly accessible to the research community. Our framework prototype implementation can be forked from our GitHub repository.[1] [2]

[1]https://github.com/atifrehman/NEC
[2]https://github.com/rehmatkhan/NEC

REFERENCES

[1] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, Feb. 2018. doi: 10.1109/JIOT.2017.2767608.
[2] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016. doi: 10.1109/JIOT.2016.2584538.
[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016. doi: 10.1109/JIOT.2016.2579198.
[4] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra. (2018). "Edge cloud offloading algorithms: Issues, methods, and perspectives," [Online]. Available: https://arxiv.org/abs/1806.06191
[5] J. Pan, L. Ma, R. Ravindran, and P. TalebiFard, "HomeCloud: An edge cloud framework and testbed for new application delivery," in *Proc. 23rd Int. Conf. Telecommun. (ICT)*, Thessaloniki, Greece, May 2016, pp. 1–6. doi: 10.1109/ICT.2016.7500391.
[6] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018. doi: 10.1109/MCOM.2018.1701233.
[7] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Commun. Surveys Tuts.*, vol. 16, pp. 393–413, 1st Quart., 2014.
[8] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges," *Future Gener. Comput. Syst.*, vol. 78, pp. 680–698, Jan. 2018. doi: 10.1016/j.future.2016.11.009.
[9] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009. doi: 10.1109/MPRV.2009.82.
[10] F. Bonomi, R. Milito, J. Zhu, R. Milito, and J. Zhu, "Fog computing and its role in the Internet of things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, Helsinki, Finland, Aug. 2012, pp. 13–16.
[11] (2016). Mobile-Edge Computing Initiative, Eur. Telecommun. Stand. Inst., Sophia Antipolis, France. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing
[12] R. Ullah, S. H. Ahmed, and B. Kim, "Information-centric networking with edge computing for IoT: Research challenges and future directions," *IEEE Access*, vol. 6, pp. 73465–73488, 2018. doi: 10.1109/ACCESS.2018.2884536.
[13] G. Li, J. Wu, J. Li, K. Wang, and T. Ye, "Service popularity-based smart resources partitioning for fog computing-enabled industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4702–4711, Oct. 2018.
[14] "Cisco visual networking index: Global mobile data traffic forecast update," Cisco Syst., Inc., San Jose, CA, USA, Tech. Rep. 1486680503328360, Mar. 2017, pp. 2016–2021. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/serviceprovider/visual-networking-index vni/mobile-white-paper-c11- 520862.html
[15] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2359–2391, 4th Quart., 2017.
[16] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," in *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, Jul. 2012. doi: 10.1109/MCOM.2012.6231276.
[17] R. A. Rehman and B.-S. Kim, "LOMCF: Forwarding and caching in named data networking based MANETs," in *IEEE Trans. Veh. Technol.*, vol. 66, no. 10, pp. 9350–9364, Oct. 2017. doi: 10.1109/TVT.2017.2700335.
[18] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol.*, New York, NY, USA, Dec. 2009, pp. 1–12. doi: 10.1145/1658939.1658941.
[19] S. H. Ahmed, S. H. Bouk, M. A. Yaqub, D. Kim, H. Song, and J. Lloret, "CODIE: Controlled data and interest evaluation in vehicular named data networks," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3954–3963, Jun. 2016. doi: 10.1109/TVT.2016.2558650.
[20] A. Dabirmoghaddam, M. Dehghan, and J. J. Garcia-Luna-Aceves, "Characterizing Interest aggregation in content-centric networks," in *Proc. IFIP Netw. Conf. (IFIP Networking) Workshops*, May 2016, pp. 449–457.

[21] M. Sifalakis, B. Kohler, C. Scherb, and C. Tschudin, "An information centric network for computing the distribution of computations," in *Proc. 1st ACM Conf. Inf.-Centric Netw.*, New York, NY, USA, Sep. 2014, pp. 137–146. doi: 10.1145/2660129.2660150.
[22] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, "A survey of naming and routing in information-centric networks," *IEEE Commun. Mag.*, vol. 50, no. 12, pp. 44–53, Dec. 2012.
[23] J. Wu, M. Dong, K. Ota, J. Li, and Z. Guan, "FCSS: Fog computing based content-aware filtering for security services in information centric social networks," *IEEE Trans. Emerg. Topics Comput.*, to be published. doi: 10.1109/TETC.2017.2747158.
[24] W. Shang *et al.*, "Named data networking of things (Invited Paper)," in *Proc. IEEE 1st Int. Conf. Internet Things Design Implement. (IoTDI)*, Berlin, Germany, Apr. 2016, pp. 117–128. doi: 10.1109/IoTDI.2015.44.
[25] Y. Liu, A. Zhang, J. Li, and J. Wu, "An anonymous distributed key management system based on CL-PKC for space information network," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kuala Lumpur, Malaysia, May 2016, pp. 1–7. doi: 10.1109/ICC.2016.7510841.
[26] D. Trossen, M. J. Reed, J. Riihijärvi, M. Georgiades, N. Fotiou, and G. Xylomenos, "IP over ICN—The better IP?" in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun./Jul. 2015, pp. 413–417.
[27] T. Refaei, J. Ma, S. Ha, and S. Liu, "Integrating IP and NDN through an extensible IP-NDN gateway," in *Proc. 4th ACM Conf. Inf.-Centric Netw. (ICN)*, New York, NY, USA, Sep. 2017, pp. 224–225. doi: 10.1145/3125719.3132112.
[28] S. Shannigrahi, C. Fan, and G. White, "Bridging the ICN deployment gap with IPoC: An IP-over-ICN protocol for 5G networks," in *Proc. Workshop Netw. Emerg. Appl. Technol.*, New York, NY, USA, Aug. 2018, pp. 1–7.
[29] H. Wu *et al.*, "On incremental deployment of named data networking in local area networks," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, May 2017, pp. 82–94.
[30] Docker Inc. (2017). *The Docker Project Page*. [Online]. Available: https://www.docker.com/
[31] Amazon Web Services. (2017). *Amazon Lambda project page- Run Code, Not Servers—Serverless Computing*. [Online]. Available: https://aws.amazon.com/lambda/
[32] A. Madhavapeddy *et al.*, "Unikernels: Library operating systems for the cloud," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 461–472, Mar. 2013.
[33] M. Król and I. Psaras, "NFaaS: Named function as a service," in *Proc. ACM Conf. Inf.-Centric Netw.*, Berlin Germany, Sep. 2017, pp. 134–144.
[34] M. Amadeo, C. Campolo, A. Molinaro, G. Ruggeri, "IoT data processing at the edge with named data networking," in *Proc. 24th Eur. Wireless Conf.*, May 2018, pp. 1–6.
[35] M. Amadeo, C. Campolo, and A. Molinaro, "NDNe: Enhancing named data networking to support cloudification at the edge," *IEEE Commun. Lett.*, vol. 20, no. 11, pp. 2264–2267, Nov. 2016.

REHMAT ULLAH received the B.S. and M.S. degrees in computer science (major in wireless communications and networks) from COMSATS University, Islamabad, Pakistan, in 2013 and 2016, respectively. He is currently pursuing the Ph.D. degree in computer engineering with the Broadband Convergence Networks Laboratory, Department of Electronics and Computer Engineering, Hongik University, South Korea. His major interests are in the field of information-centric networking (ICN) specifically mobile/wireless content-centric/named data networking (CCN/NDN), edge computing, the Internet of Things (IoT), 5G and the future Internet architectures. Moreover, he is an ACM Student Member and serves as a regular reviewer for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the *IEEE Wireless Communications Magazine*, the *IEEE Communications Magazine*, *Transactions on Emerging Telecommunications Technologies*, *Future Generation Computer Systems* Journal (Elsevier), and the IEEE ACCESS Journal. He served as a reviewer and/or TPC for various international conferences and workshops including ACM MobiHoc (Los Angeles, USA), IEEE CCNC (Las Vegas, USA), VTC2018-Spring (Porto, Portugal), and SIGCSE 2019 (Special Sessions: Minneapolis, Minnesota, USA).

MUHAMMAD ATIF UR REHMAN received the B.S. degree in electronics and communication from The University of Lahore, Lahore, Pakistan, in 2013, and the M.S. degree in computer science from the COMSATS Institute of Information Technology, Islamabad, Pakistan, in 2016. He is currently pursuing the Ph.D. degree with the Broadband Convergence Networks Laboratory, Department of Electronics and Computer Engineering, Hongik University, South Korea. From 2013 to 2018, he was working as a Software Engineer and Architect in leading IT companies in Pakistan. His major interests are in the field of information centric wireless networks, named data networking, edge computing, software defined networking, the Internet of Things, and 5th generation communication. His responsibilities were to write well design, testable, and efficient code, collaborate with other team members to determine functional and non-functional requirements for new software or application, provide technical guidelines to other team members and ensure software meets all requirements of quality. He worked closely with quality assurance team to deliver high quality and reliable products.

BYUNG-SEO KIM (M'02–SM'17) received the B.S. degree in electrical engineering from In-Ha University, In-Chon, South Korea, in 1998, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Florida, in 2001 and 2004, respectively. His Ph.D. study was supervised by Dr. Y. Fang. From 1997 to 1999, he was with Motorola Korea Ltd., Paju, South Korea, as a Computer Integrated Manufacturing (CIM) Engineer in Advanced Technology Research and Development (ATR&D). From 2005 to 2007, he was with Motorola Inc., Schaumburg, IL, USA, as a Senior Software Engineer in networks and enterprises. His research focuses in Motorola Inc., were designing protocol and network architecture of wireless broadband mission critical communications. From 2012 to 2014, he was the Chairman with the Department of Software and Communications Engineering, Hongik University, South Korea, where he is currently a Professor. His work has appeared in around 167 publications and 22 patents. His research interests include the design and development of efficient wireless/wired networks including link-adaptable/cross-layer-based protocols, multi-protocol structures, wireless CCNs/NDNs, mobile edge computing, physical layer design for broadband PLC, and resource allocation algorithms for wireless networks. He served as the Member of the Sejong-city Construction Review Committee and Ansan-city Design Advisory Board. He served as the General Chair for 3rd IWWCN 2017, and the TPC member for the IEEE VTC 2014-Spring and the EAI FUTURE2016, and ICGHIC 2016 2019 conferences. He served as the Guest Editor for special issues of the *International Journal of Distributed Sensor Networks* (SAGE), the IEEE ACCESS, and the *Journal of the Institute of Electrics and Information Engineers*. He is an Associate Editor of the IEEE ACCESS.