Edge Based SDN Solution for Handling Increasing IoT Data

Aditya Saroja Hariharakrishnan, Abhirami Shankar
Department of Electrical Engineering, University of Colorado Boulder
Aditya.SarojaHariharakrishnan@colorado.edu, Abhirami.Shankar@colorado.edu
https://github.com/AdvNetSys-EdgeComputing/Edge-Based-SDN-Solution-for-Handling-Increasing-IoT-Data

***Abstract*—In today's world, IoT has emerged to be one of the industries that is generating extremely large amount of information due to wide-area deployments of IoT devices. The varied types of Iot devices and multiple heterogeneous networking environments need to co-exist, thus demanding management of such environments and related volume of data. Given the different classes of IoT traffic and estimated volume of excess data, the current approaches proposed for IoT information management may not be sufficient. Most of the ideas today for managing IoT data involves storage and processing of the data in the Cloud, which not only places more computational load on the cloud to handle the data, but also introduces latency, jitter, bandwidth consumption and delay. In order to reduce these critical parameters and enable data abstraction at the network level, we propose an Edge based SDN solution to handle IoT data. In this paper, we create the functionality of processing the data at the edge, and an SDN controller is used to retrieve the data-specific functions from the cloud. When IoT raw data is received by the edge, the type of service the data needs in order to convert it to human readable data is identified by analyzing the header of the IoT raw data. Once the type of service has been identified, the edge acts as an SDN controller and retrieves only the data-specific function from the cloud, caches it and serves similar type of IoT data. As and when a new type of service is identified by the edge, the data-specific function is retrieved from the cloud. Through this, IoT data does not have to travel all the way to the cloud and back, thereby reducing negative factors mentioned above.**

***Index Terms*—SDN Controller, Edge processing, IoT Gateway, Caching, MQTT**

I. Introduction

With the world turning into a technologically smart age, Internet of Things has recently gained more popularity due to the kind of services it offers. Smart homes, home-automation and security are some of the distinguishing features of IoT that explains its demand all over the world today. In the near future, IoT devices will be deployed on a wide scale, with every device processing large amount of IoT data. In order to support this kind of an environment, we require to have networking architectures that not only support but also allow computation of such data. The realization and deployment of such an environment poses significant technical and design challenges. The next-generation devices would operate on a wireless, dynamic and heterogeneous environment since most of the devices are expected to support mobility. Due to the already existent legacy devices, various kinds and classes of IoT devices would have to co-exist in the same environment. Thus, storage of information and computational power would be some of the basic requirements, not only for information processing but also to maintain and manage the path via which information flows from the providers to the consumers respectively.

In terms of the design, many ideas proposed today involve collection of IoT data, storage as well as processing in the Cloud. This kind of approach may not be feasible due to the burden it induces on the cloud itself. The cost involved in storing such large amounts of data would be a serious point of concern. Also, the time it takes for the data to reach the cloud, process it in the cloud and then return back to the consumer would cause significant latency and delay. This kind of approach would involve storage and processing of all the IoT data in one go rather than considering filtering out of only required parts of data from IoT raw data and processing the same.

IoT applications would require some kind of middle-ware to allow inter-operability in case of various kinds of protocols being used by different devices for machine-to-machine communication. Middle-ware support is required such that data optimization or hand-off in case of switching from one wireless technology to another can be performed automatically . The biggest challenge that many cloud service providers are facing today is physically provisioning bandwidth for their increasing network demands. As they want to expand their network and provide service to more users, the bandwidth concern becomes significant. In order to have enough bandwidth to maintain the quality existing users are experiencing as well as provide the same or better quality to new users, the only choice that providers have left is to physically deploy more servers in their network which can handle the new requests. This hits on cost, capital and various other factors.

The challenges presented so far demand some kind of novel layered architecture to allow support for various heterogeneous IoT functionality, with varied requirements for every application and device. These challenges can be addressed using SDN approach which provides an abstraction between the data and control plane thus enabling data distribution as well as data processing at the same time. SDN could act as a centralized stack which can provide bandwidth on demand without having to worry about adding additional hardware, CPU processing, storage requirements, memory, quality of service etc. An SDN controller can provide a layered architecture thus providing flexibility to accommodate dynamic IoT data and providing the required resources to specific IoT applications in and on-demand basis.

In this paper, we propose an edge based SDN solution that addresses the excess of IoT data management by filtering and processing the data at the edge itself such that the data can

be processed and sent back from the gateway closest to the source that sends the data. The major idea of the paper is to have an SDN controller that receives user requests for specific IoT services, retrieves only the specific function required for the IoT service from the cloud, deploys that function onto the gateway and allows processing of data and distribution from the gateway itself, thus reducing the load on the Cloud and latency compared to sending all the data to the Cloud and back.

SDN could be very helpful in addressing IoT excess data issues due to various reasons:

1) SDN separates the data and control plane and provides abstraction of low-level functionalites from high-level network services. This kind of separation makes it much easier for managing the network without having to worry about the dependency between the data and control plane.
2) The configuration flexibility offered by SDN will allow efficient resource sharing and dynamic allocation of resources thus saving Bandwidth.
3) SDN controller allows provisioning of bandwidth on demand without having to add physical devices to the existing infrastructure.
4) SDN provides a perfect balance between centralized control through the Controller and decentralized operations.

II. IoT Applications In Urban Environment

As new systems and applications are being developed, the need to deal with solutions that can enable IoT device to device communication as well as service IoT requirements in a timely manner becomes essential.

Consider a case where cellular devices need to communicate with each other. In the near future, this will put a large burden on the network as the number of IoT devices keep increasing. The use of MQTT could be very helpful for efficient machine-to-machine communication. MQ Telemetry Transport involves the use of and MQTT Broker in order facilitate communication between the client which subscribes for information and the server which is responsible for publishing the information. Thus, the broker will handle the session, connection establishment and connection tear down between the client and server without the devices having to communicate directly with each other. MQTT is a very light-weight protocol that allows usage of limited bandwidth as well as efficient distribution of information. In this way, devices can communicate only when required and devices with constrained resources like IoT sensors etc. can communicate without having to worry about bandwidth. QoS can be achieved using MQTT by defining the quality level each application requires so that the messages can be exchanged as specified by the quality (best effort or with error control and re-transmission). MQTT provides the flexibility to every application that uses it, to specify

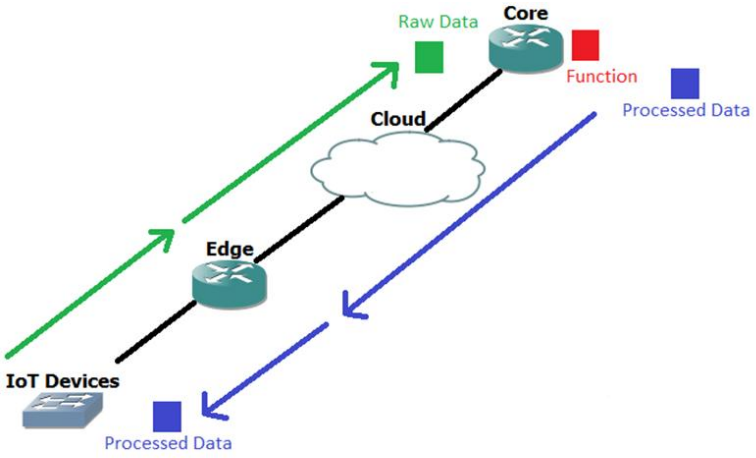

Fig. 1. Proposed system (edge based solution for increasing IoT data)

the QoS requirements it needs according their functional and environmental requirements.

In this paper, we will show the key differences between the current system existing for IoT data processing, the load it adds onto the network, latency and bandwidth concerns etc. and how the edge based solution would prove to be better in terms of bandwidth provisioning and resource allocation. We will also demonstrate test results achieved by using both the systems and a possible future case where there would be millions of IoT devices requiring IoT data processing. We will demonstrate how using the edge processing system would prove to have significant advantage when it comes to providing better quality and lower latency.

III. Architecture Overview

This section focuses on the key differences in the existing architecture for processing IoT data v/s the solution proposed by the paper.

A. Current System

As displayed in Fig.2, the current system for processing IoT data involves the data traveling all the way to the provider core, getting stored and converted to usable data at the core and then traveling all the way back to the user. This not only adds burden to the network but also degrades the quality that the consumer experiences since it adds latency since the data has to travel all the way to the cloud and back.

B. Proposed System

The edge based solution does not require the data to travel all the way to the provider core and back. Instead, the system we propose filters Iot data and segregates it into similar type of service categories by identifying the type of service looking at the header attached to the data. Once the type of service for

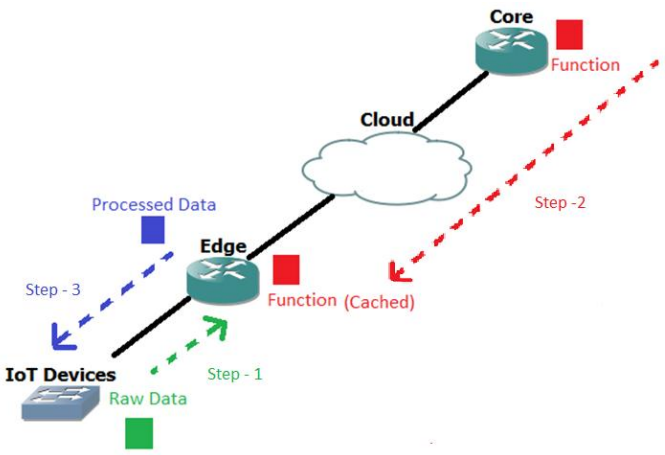

Fig. 2. Proposed system (edge based solution for increasing IoT data)

the IoT data has been identified, the edge acts as a controller and retrieves only the data-specific function from the provider core.

Caching is an important feature that has been imlemented in the proposed system. The basic idea behind caching is to reduce latency, provide better quality of service to the user and avoid unnecessary load on the network. Rather than trying to retrieve the function every single time from the Provider core, we identify the type of service requested by the user at the edge, retrieve the data-specific function from the core and cache the function at the edge for a few seconds. During the period of time that the cache is valid for, any data that equires the same function for processing is serviced through the cache itself. Once the cache expires the function is retrieved again from the core, given that the edge encounters the similar kind of raw data. If the edge ceases to receive requests for the same type of service, the function is not retrieved from the core even after cache expires thus freeing up some memory space at the edge.

When a new type of service arrives at the edge, the data-specific function is retrieved form the cloud and serviced at the edge. This means, data-specific functions are retrieved from the cloud only on demand when the edge receives requests to process a certain type of data. Thus, there is not much storage concerns or load induced on the edge either. The only requirement for the edge is to have a controller that can request functions from the provider edge on demand and segregate similar type of traffic by looking at the header.

IV. Evaluation Setup

To evaluate our proposed system, we develop a setup with a local machine acting as an IoT data and traffic simulator, another local machine acting as the edge and a third machine on the cloud, acting as the IoT core. The IoT core machines are present in two locations, one within Colorado and the other in Wisconsin, in order to obtain a fair average.

A. Customer Edge

At the customer edge, we use a python script to generate data as an IoT temperature/pressure sensor would. This data would be sent to MQTT Broker running on the same machine or a different machine. For our system, we used the same machine. The MQTT broker runs as a daemon on the machine and any data which gets generated by the python code gets sent to 127.0.0.1:1833. Also, IoT data gets generated every 5 seconds. This data gets stored in an output file, which is evaluated either at the edge or at the core. The PAHO library by Eclipse is used as a client side implementation. In our implementation, the edge IP is hard-coded on the customer end. In real life applications, this would not be the case and rather, DHCP would be used to provide the IP addresses. Also, in real time cases, the Iot data would typically use hand-held mobile devices or wireless access points as next hops after which they would go on to the provider edges. We also inserted our own bits into the MQTT headers so that IoT data belonging to various vendors can be differentiated and analyzed on an as-per basis.

B. IoT Core

At our IoT cores, we have the various functions that can convert the raw Iot data into user required data. In our setup, we have python scripts that emulate a temperature function which analyses the temperature readings provided by the IoT sensor at the edge and converts it into calculated output. Another function converts the pressure readings to analyzed data. The core also runs a script that checks for any IoT data that comes in at all times and if it detects any, checks the header and runs the python functions if the headers match. This is to keep the evaluated results as real time as possible. The python functions also calculate their computation time for analysis.

C. Provider Edge

The provider edge, in our setup, performs two roles. While analyzing the current system, the provider edge simply checks if there is incoming IoT raw data and if so, forwards it over to the IoT core. While analyzing our proposed system, the provider edge examines the header of the incoming MQTT data, checks the vendor, contacts the vendor's Autonomous system, gets the computational function, computes the user required data, and sends the data over to the user, while at the same time caching the function received from the IoT core. The caching system is a primary core of our system as that is the advantage our system proposes over other systems. With caching, the next time IoT data is received with similar headers and requirements, the provider edge does not have to retrieve the function from the IoT core. This reduces latency and bandwidth utilization, thus freeing traffic that otherwise could

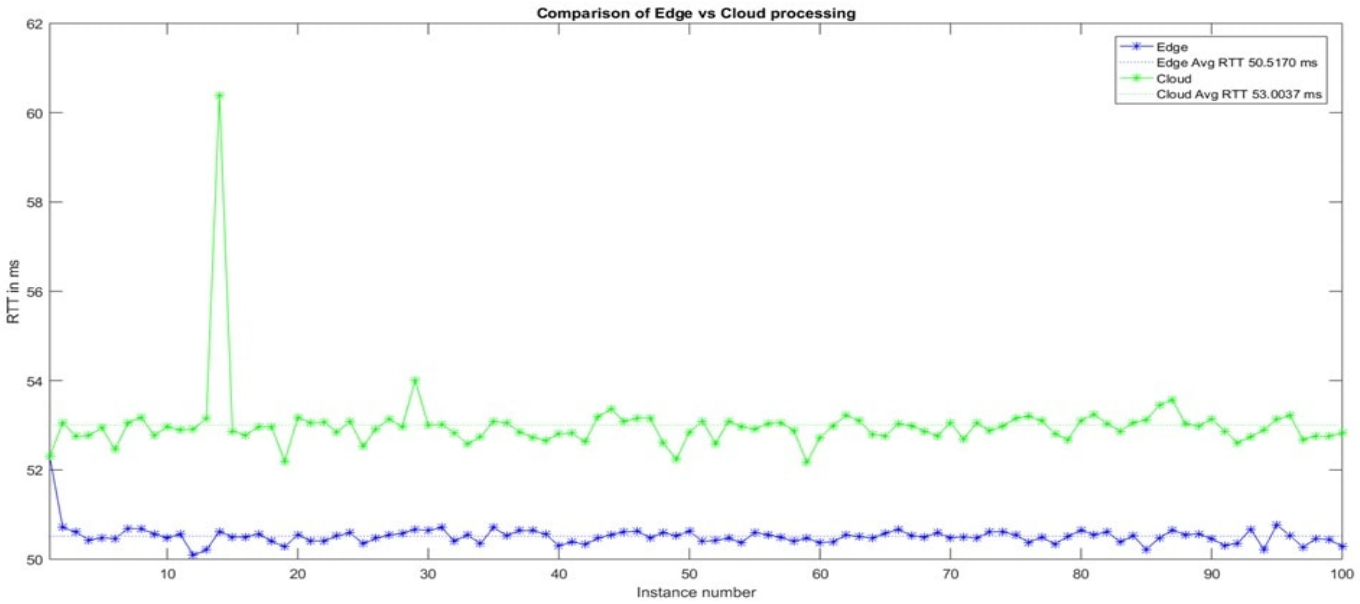

Fig. 3. Comparison of RTT for computation at edge vs computation at IoT Core

have choked the entire route. Caching introduces overheads and requirements which are addressed later.

When the provider edge sees a new IoT data with a new header, it again asks the corresponding IoT core for the function and gets it. By this process of caching and reusing, various advantages are seen over conventional IoT processing. The cache is timed so as to not over-utilize the provider edge's resources. In our setup, all the provider edge's functionality are done using shell scripts.

V. Evaluation and Results

To measure the latency we use Wireshark. Wireshark running on the IoT customer edge measures the difference of the time for the first outgoing raw IoT data packet and the first incoming processed IoT data. This is done for both systems; the conventional and our proposed system. Using MATLAB, the comparison plots are developed.

As seen in the Fig. 3, the x axis is represented by the data packet at that instant and the y axis is the round trip time. The blue curve represents the RTT it takes form from the customer edge to the provider edge, get computed, and return back to the customer edge. It is a representation of our system. The green curve on the other hand represents the conventional system where the data goes all the way to the IoT core, gets converted into usable data and returns back to the customer. For this evaluation we just use the cloud in Colorado as it produces the results more linearly. The Wisonsin cloud results were very erratic.

Observing the results, we do not note much of a difference in the RTT for our proposed system and the current system. The peak during the first instance in our proposed system is the provider edge retrieving the function from the IoT cloud. There is an average difference of RTT by 3 ms.

IoT data is estimated to grow exponentially. This has not been considered in our developed system for deployment due to hardware constraints. When we have such a high volume of IoT data, we expect the RTT curve to grow exponentially as various other factors that did not affect our previous graph, such as bandwidth allocation and RSVP mechanisms come into play. The data gets queued and gets executed using various queuing mechanisms. This leads to backlog of data and high resource overload. This is aptly depicted in Fig. 4. The x axis denotes the volume of data and the y axis denotes the RTT in milliseconds. Initially for lesser quantity of data, the current system and our proposed system do not give much of an RTT difference. But clearly, as data volume increases further we can see a very visible difference between the efficiency of our system and the proposed system.

VI. Scope for Development

1) Bandwidth management. IoT devices may not have to send out data at all instances. So by managing the time and bandwidth allocated for a particular IoT device or a group of IoT devices, we can conserve bandwidth.
2) Currently, the function is cached using simple cache mechanisms with a timeout, and retrieved from the stored location as and when needed. When there is a large amount of data with large user base, then the way the functions are recovered and used make a huge difference in performance. To achieve this flow control mechanisms have to be implemented.
3) Implementing a distributed system for load handling at the ISP edges for increased efficiency.

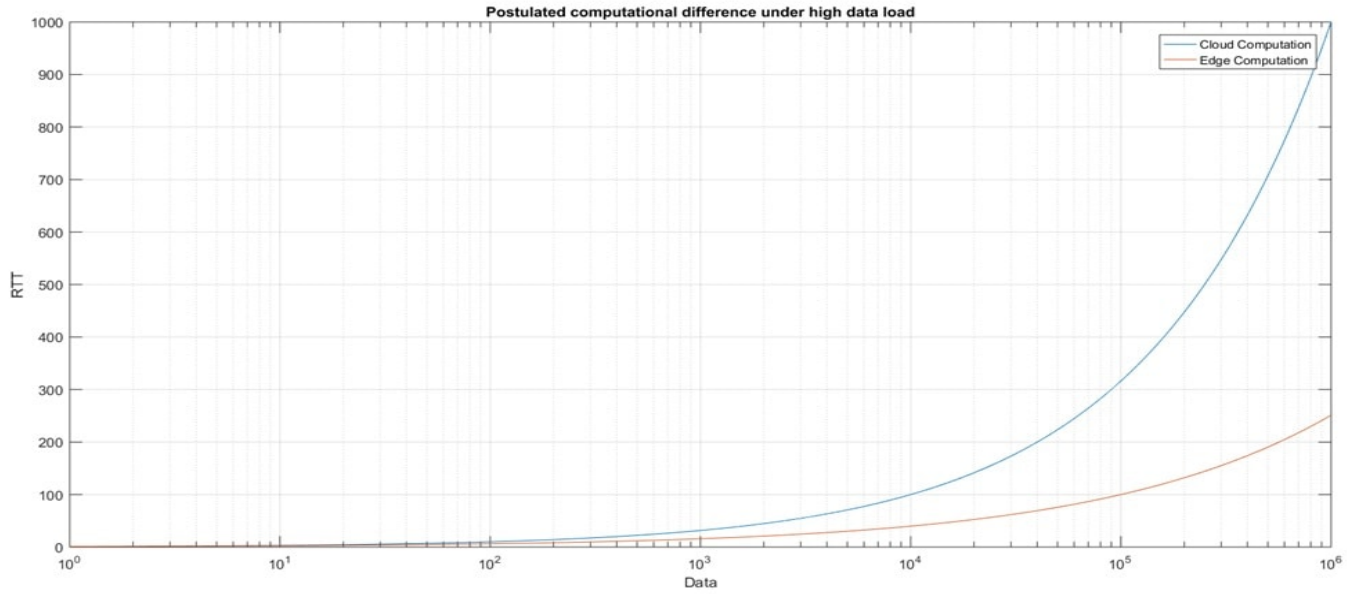

Fig. 4. Postulated comparison of RTT for computation at edge vs computation at IoT Core

VII. Implementation Hurdles

1) The first most obvious hurdle we face is the caching of the function within the provider edge. Functions are proprietary content of the IoT vendor and before such content can be stored on the edge, there needs to be a set of agreements forged between the providing ISP and the IoT vendor.
2) The above hurdle has an extension. IoT devices are highly mobile and as such if they were to roam between the provisioning areas of different ISPs, then every ISP might need to have a service level agreement with the IoT provider in order to achieve goals of optimal resource utilization and lesser latency. This might not be possible or at the best case, might take a long time to get implemented.
3) Such a system would reduce the load on the IoT core provider, but increases computation and resource load on the ISP's edge proportionally. This calls for some sort of compensation for the ISP since they would be handling disproportionate load.
4) Balancing out load across various edges of the ISP also becomes a problem, especially when it comes to vast quantity of data.

VIII. Conclusion

In short, by bringing the computation to the edge, by controlling the allocation of bandwidth, and by caching the function received from the IoT core, we can achieve increased performance, which is highly relevant when the volume of IoT data increases, as it is estimated to within a span of few years. To demonstrate this we have developed a customer edge-provider edge-IoT core system which performs both approaches; the traditional one where raw IoT data goes all the way to the cloud, gets converted to user required data, and comes all the way back, and our proposed system where IoT data stops at the edge, the edge retrieves functions to convert it to user required output and does all the computation at the edge. We have performed latency measurements for both these approaches and have determined our approach to have a better performance, under limited load. We have also postulated a case where other overhead such as queuing and caching play a much bigger role. We estimate our system to have a higher performance in such a scenario too.

IX. Acknowledgments

We would like to extend our thanks to Dr. Eric Keller, head of the Next Generation Networks (NGN) group, and Assistant Professor with the ECEE Department at the University of Colorado Boulder. We would also like to thank our peers in Advanced Network Systems for their valuable support and feedback.

References

[1] Zhijing Qin et al., "A Software Defined Networking Architecture for the Internet-of-Things" *NOMS 2014 - 2014 IEEE/IFIP Network Operations and Management Symposium*
[2] Mauro Tortonesi, James Michaelis, Alessandro Morelli, Niranjan Suri, Michael A. Baker, "SPF: An SDN-based Middleware Solution to Mitigate the IoT Information Explosion" *2016 IEEE Symposium on Computers and Communication (ISCC)*
[3] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman, "Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area" *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*
[4] Y. Zheng et al., "Urban Computing: Concepts Methodologies and Applications" *ACM Transactions on Intelligent Systems and Technologies*