

Improving TCP Data Transportation for Internet of Things

Jamal A. Khan

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Computer Science and Applications

Ali R. Butt, Chair
Muhammad Shahzad
Dongyoon Lee

July 26, 2018
Blacksburg, Virginia

Keywords: Internet of Things, TCP, Routers, Router Buffers, Buffer Sizing, Cloud,
Datacenter, Reno, Cubic, Vegas, DCTCP
Copyright 2018, Jamal A. Khan

Improving TCP Data Transportation for Internet of Things

Jamal A. Khan
(ABSTRACT)

Internet of Things (IoT) is the idea that every device around us is connected and these devices continually collect and communicate data for analysis at a large scale in order to enable better end user experience, resource utilization and device performance. Therefore, data is central to the concept of IoT and the amount being collected is growing at an unprecedented rate. Current networking systems and hardware are not fully equipped to handle influx of data at this scale which is a serious problem because it can lead to erroneous interpretation of the data resulting in low resource utilization and bad end user experience defeating the purpose of IoT. This thesis aims at improving data transportation for IoT. In IoT systems, devices are connected to one or more cloud services over the internet via an access link. The cloud processes the data sent by the devices and sends back appropriate instructions. Hence, the performance of the two ends of the network *i.e.* the access networks and datacenter network, directly impacts the performance of IoT.

The first portion of the our research targets improvement of the access networks by improving access link (router) design. Among the important design aspects of routers is the size of their output buffer queue. We have developed a probabilistic model to calculate the size of the output buffer that ensures high link utilization and low latency for packets. We have eliminated limiting assumptions of prior art that do not hold true for IoT. Our results show that for TCP only traffic, buffer size calculated by the state of the art schemes results in at least 60% higher queuing delay compared to our scheme while achieving almost similar access link utilization, loss-rate, and goodput. For UDP only traffic, our scheme achieves at least 91% link utilization with very low queuing delays and aggregate goodput that is approx. 90% of link capacity. Finally, for mixed traffic scenarios our scheme achieves higher link utilization than TCP only and UDP only scenarios as well as low delays, low loss-rates and aggregate goodput that is approx 94% of link capacity.

The second portion of the thesis focuses on datacenter networks. Applications that control IoT devices reside here. Performance of these applications is affected by the choice of TCP used for data communication between Virtual Machines (VM). However, cloud users have little to no knowledge about the network between the VMs and hence, lack a systematic method to select a TCP variant. We have focused on characterizing TCP Cubic, Reno, Vegas and DCTCP from the perspective of cloud tenants while treating the network as a black box. We have conducted experiments on the transport layer and the application layer. The observations from our transport layer experiments show TCP Vegas outperforms the other variants in terms of throughput, RTT, and stability. Application layer experiments show that Vegas has the worst response time while all other variants perform similarly. The results also show that different inter-request delay distributions have no effect on the throughput, RTT, or response time.

Improving TCP Data Transportation for Internet of Things

Jamal A. Khan

(GENERAL AUDIENCE ABSTRACT)

Internet of Things (IoT) is the idea that every electronic device around us, like watches, thermostats and even refrigerators, is connected to one another and these devices continually collect and communicate data. This data is analyzed at a large scale in order to enable better user experience and improve the utilization and performance of the devices. Therefore, data is central to the concept of IoT and because of the unprecedented increase in the number of connected devices, the amount being collected is growing at an unprecedented rate. Current computer networks over which the data is transported, are not fully equipped to handle influx of data at this scale. This is a serious problem because it can lead to erroneous analysis of the data, resulting in low device utilization and bad user experience, hence, defeating the purpose of IoT. This thesis aims at improving data transportation for IoT by improving different components involved in computer networks. In IoT systems, devices are connected to cloud computing services over the internet through a router. The router acts a gateway to send data to and receive data from the cloud services. The cloud services act as the brain of IoT i.e. they process the data sent by the devices and send back appropriate instructions for the devices to perform. Hence, the performance of the two ends of the network *i.e.* routers in the access networks and cloud services in datacenter network, directly impacts the performance of IoT.

The first portion of our research targets the design of routers. Among the important design aspects of routers is their size of their output buffer queue which holds the data packets to be sent out. We have developed a novel probabilistic model to calculate the size of the output buffer that ensures that the link utilization stays high and the latency of the IoT devices stays low, ensuring good performance. Results show that that our scheme outperforms state-of-the-art schemes for TCP only traffic and shows very favorable results for UDP only and mixed traffic scenarios.

The second portion of the thesis focuses on improving application service performance in datacenter networks. Applications that control IoT devices reside in the cloud and their performance is directly affected by the protocol chosen to send data between different machines. However, cloud users have almost no knowledge about the configuration of the network between the machines allotted to them in the cloud. Hence, they lack a systematic method to select a protocol variant that is suitable for their application. We have focused on characterizing different protocols: TCP Cubic, Reno, Vegas and DCTCP from the perspective of cloud tenants while treating the network as a black-box (unknown). We have provided in depth analysis and insights into the throughput and latency behaviors which should help the cloud tenants make a more informed choice of TCP congestion control.

Dedicated to my father and mother without whom this would not have been possible.

Acknowledgments

This work is sponsored in part by the NSF under the grants: CNS 1565314, CNS 1405697, CNS 1615411, and CNS 1616273.

Contents

List of Figures	ix
List of Tables	xii
1 Introduction	1
1.1 Motivation	2
1.1.1 Router Design for IoT Access Networks	3
1.1.2 Understanding TCP in blackbox Networks of the Cloud	3
1.2 Research Contributions	4
1.2.1 Improved Buffer Sizing for Access Routers	4
1.2.2 Insights into Performance of TCP Variants in Datacenter Environments	5
1.3 Thesis Organization	6
2 Background and Related Work	7
2.1 Router Design for Congested Links	7
2.1.1 Buffer Sizing Schemes for Conventional Networks	7
2.1.2 Measurement Studies on Buffer Sizing	9
2.1.3 Active Queue Management	10
2.1.4 Limitations of Prior Art	10
2.2 Behavior of TCP in Unknown Networks	11
2.2.1 TCP Congestion Control Algorithms:	12
2.2.2 Comparative Measurement Studies:	12

2.2.3	Virtualized Congestion Control and Homogenized TCP:	13
2.2.4	Limitations of Prior Art:	14
3	Sizing Buffers for IoT Access Routers	15
3.1	Introduction	15
3.1.1	Problem Statement	16
3.1.2	Proposed Approach	17
3.1.3	Key Contributions	18
3.2	Exploratory Study	18
3.3	Buffer Size Calculation	21
3.3.1	Fraction of Flows with Given Number of Dropped Packets	22
3.3.2	Average Congestion Window Size after Congestion Event	24
3.3.3	Output Buffer Size	24
3.3.4	Buffer Sizing for Cubic	26
3.4	Parameter Selection	27
3.4.1	Percentage of Long Flows	27
3.4.2	Number of Packet Drops in a Congestion Event	29
3.5	Buffer Sizing for UDP Traffic	29
3.5.1	Buffer Sizing for TCP and UDP combined	32
3.6	Experimental Evaluation	33
3.6.1	Buffer Size vs. Link Utilization and Latency	34
3.6.2	Comparison of Buffer Sizes	35
3.6.3	Comparison of Link Utilization, Latency, Loss, & Goodput	37
3.6.4	UDP only Traffic	38
3.6.5	TCP+UDP Traffic	39
3.7	Chapter Summary	41
4	Characterizing TCP in the Cloud for Cloud Tenants	42
4.1	Introduction	42

4.2	Data Collection	44
4.2.1	Choice of platform	44
4.2.2	TCP Performance Logging	45
4.2.3	Why only TCP?	45
4.2.4	Choice of TCP Variants	45
4.3	iPerf Bechmarking	46
4.3.1	Experiment Design and Topology	47
4.3.2	Results and Discussion	47
4.4	Jmeter Bechmarking	59
4.4.1	Results and Discussion	60
4.5	Chapter Summary	66
5	Conclusions and Future Work	67
5.1	Summary	67
5.2	Future Directions	70
5.2.1	Goal Based TCP Congestion Control: A Reinforcement Learning Approach	70
5.2.2	Middle-Box Channels: Virtual Congestion Control And Choice of TCP	71
	Bibliography	72

List of Figures

1.1	High level view of IoT architecture	1
3.1	The setup and topology used to study the assumptions of prior art	19
3.2	Time-series of congestion window sizes of three Raspberry Pis	19
3.3	Percentage of short flows in IoT traffic	20
3.4	Block diagram of a typical IoT implementation	22
3.5	Average # of packets successfully inserted into the output buffer before the first drop vs. # of flows	28
3.6	Median of the number of packets dropped during each congestion event vs. the number of active flows	29
3.7	Buffer size vs. Utilization	34
3.8	Buffer size vs. Delay	34
3.9	Buffer size for 0% short flows	36
3.10	Buffer size for 15% short flows	36
3.11	Buffer size for 30% short flows	36
3.12	Buffer size comparison 0% short flows	36
3.13	Buffer size comparison 15% short flows	36
3.14	Buffer size comparison for 30% short flows	36
3.15	Link Utilization for 0% short flows	36
3.16	Link Utilization for 15% short flows	36
3.17	Link Utilization for 30% short flows	36
3.18	Queuing Delay for 0% short flows	36

3.19	Queuing Delay for 15% short flows	36
3.20	Queuing Delay for 30% short flows	36
3.21	Link Loss for 0% short flows	37
3.22	Link Loss for 15% short flows	37
3.23	Link Loss for 30% short flows	37
3.24	Link Goodput for 0% short flows	37
3.25	Link Goodput for 15% short flows	37
3.26	Link Goodput for 30% short flows	37
3.27	% Link Utilization for UDP flows	39
3.28	Queuing Delay for UDP flows	39
3.29	% Link Loss for UDP flows	39
3.30	Aggregate Goodput for UDP flows	39
3.31	% Link Utilization for mixed flows	40
3.32	Queuing Delay for mixed flows	40
3.33	% Link Loss for mixed flows	41
3.34	Goodput for mixed flows	41
4.1	iPerf Experiment Topology	47
4.2	Throughput over time	48
4.3	Round trip time over time	48
4.4	Throughput distribution	49
4.5	Round trip time distribution	49
4.6	Throughput trends with intermittent gaps	52
4.7	Throughput RTT Characteristics	53
4.8	Throughput variation	54
4.9	Round trip time variation	54
4.10	TCP congestion window distribution	55
4.11	Throughput variance across batches	56
4.12	RTT variance across batches	56

4.13 Throughput Poincaré map	57
4.14 RTT Poincaré map	57
4.15 Throughput Lyapunov exponents	58
4.16 RTT Lyapunov exponents	58
4.17 Throughput for different file sizes	61
4.18 RTT for different file sizes	62
4.19 Throughput for different file sizes	63
4.20 RTT for different file sizes	64
4.21 Response Time distribution for different file sizes	65

List of Tables

3.1	Buffer sizes for mixed traffic experiments	40
4.1	Throughput fit parameters	50
4.2	Round trip time fit parameters	51

Chapter 1

Introduction

Internet of Things (IoT) is the idea that every device around us is connected. These devices continually collect and communicate data for analysis at a large scale in order to enable better end user experience, resource utilization and device performance. Therefore, data is central to the concept of IoT and the amount being collected is growing at an unprecedented rate as the number of connected device continue to increase [23]. In 2016, the number of connected devices was estimated to be around 400 Million and is predicted to increase upto 18 Billion by 2020 [22]. Current networking systems and hardware *are not fully equipped to handle influx of data at this scale* which is a serious problem because it can lead to erroneous interpretation of the data resulting in *low resource utilization and bad end user experience* defeating the purpose of IoT. To understand the scope of the problem, it is important to understand the architecture of IoT and the components involved. Figure 1.1 gives a birds eye view of the various components involved in the IoT architecture.

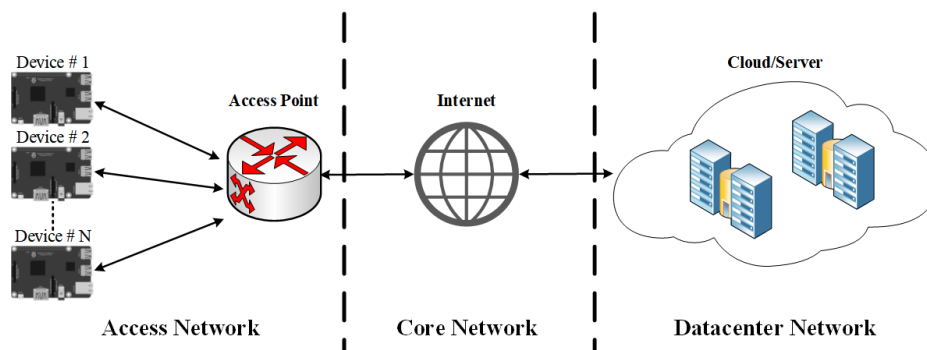


Figure 1.1: High level view of IoT architecture

In typical IoT systems, devices (embedded computers) are connected to sensors and/or actuators directly. These devices obtain measurements from sensors and send those measurements through the internet to an appropriate application service deployed in the cloud. The devices connect to the internet typically via an access point *i.e.* a router. The applications deployed

in the cloud process the measurements and send appropriate instructions which the devices execute. Notice that the IoT architecture spreads over the entire network but two portions of it interact with the architecture components directly:

1. ***Access networks*** where the devices physically live.
2. ***Datacenter networks*** where the applications reside.

Owing to the direct nature of interaction of the IoT devices and applications with the above mentioned portions of the network, their performance is tied with the efficiency of the data communication within the networks. Circling back to the problem *i.e.* lack of ability of current networking systems to handle the influx of IoT data, ***if current networking systems are to be improved in order to better handle this influx, both the access networks and the data center networks need to be improved.*** Consider the former of the two portions *i.e.* access networks, two things that can affect network performance are 1) the Network Interface Cards (NIC) of the devices, or 2) the access router. Current generation low-end embedded devices *e.g.* raspberry pi [8] have NICs that are capable of handling 10/100 M Ethernet. Hence, if the IoT devices are well provisioned, network performance is contingent on router design and performance. Coming onto the latter *i.e.* datacenter networks, when IoT developers buy infrastructure from cloud providers to deploy the IoT applications, they have little to no control over how the Virtual Machines (VM) are networked to one another. Furthermore, they have no knowledge about the network hardware capabilities. These factors, when compounded, make it difficult for the developers to choose a suitable data transportation protocol. Choosing the correct one has considerable impact on network performance [10, 17, 27, 29, 30, 39]. Taking into account the discussion above, the problem at hand is two pronged in nature and so, this thesis too takes a two pronged approach to the solution.

Thesis Objective: Improve network performance to better handle IoT data via a two pronged approach: 1) Improve router design in order to enable better communication of data between IoT devices and cloud applications, and 2) Provide a systematic method or guideline to enable better choice of communication protocols in environments when the network is a black box.

1.1 Motivation

Inline with the objective of the thesis, the work too has been done in two parts and in this section we provide the motivation for the two projects.

1.1.1 Router Design for IoT Access Networks

As discussed earlier, in typical IoT systems, IoT devices are connected to sensors and/or actuators, and to one or more remote cloud services through the internet. To connect to cloud services, the IoT devices access the internet through an access router. Due to significant differences between IoT networks and conventional networks, such as in terms of traffic type, medium access protocols, and device heterogeneity, vendors have already started developing access routers that are tailored for IoT networks, such as Cisco's 829 router [2] and Norton's Norton Core Router [6]. While there are several design aspects of an access router that impact an IoT system, such as the number and type of wireless/wired interfaces, one of the most important design aspects is the size of the output packet buffer of the interface of the access router that connects to the access link, *i.e.*, the internet.

Selecting an appropriate size of the output buffer of the IoT access router is crucial because it directly impacts two key performance metrics of the IoT system: 1) access link utilization (*i.e.*, the percentage of time the link is under use), and 2) latency (*i.e.*, the time a packet takes to go from sender to receiver). If the buffer is too large, utilization increases but latency increase as well and as a result the throughput drops. Conversely, if the buffer is too small, latency falls but utilization falls as well. The loss-rate increases and as a result throughput falls.

Problem Statement: Design a scheme to systematically select the appropriate size of the buffer based on traffic behavior and TCP semantics in order to achieve high buffer utilization, high throughput, low latency and low loss-rate.

1.1.2 Understanding TCP in blackbox Networks of the Cloud

Cloud computing for several years now, has been the defacto paradigm for providing computing services for all myriad of applications. As noted earlier, the emergence and subsequent explosive growth of IoT has made the importance of cloud even more concrete. Applications that process data sent by IoT devices reside here. Due to the enormity of the data influx, efficient communication of data between machines in the cloud is of crucial importance. Therefore, it is important that the protocols being used to do so are chosen carefully. If the protocol is not chosen through careful consideration, it may result in increased latency and reduced throughput.

From the *cloud tenants'* (people who deploy applications in the cloud) perspective choosing the correct protocol is difficult because of the lack of knowledge of and control over the data center network. Note that when *cloud tenants* buy Virtual Machines (VM) *e.g.* Elastic Compute Cloud (EC2) instances from a cloud service provider like AWS, they have little control over where the instances are placed and almost no control over how they are networked to one another. The EC2 instances may end up being hosted on the same physical machine, or

on different machines in the same rack, or on machines in different racks. Furthermore, the network between two or more pairs of instances requested and allocated at the same time, may or may not be homogeneous. Some may be a single hop away from a given machine while others may take several hops to reach that same machine. The traffic generated between the said instances passes through network equipment having varying capabilities such as switch bandwidth, Explicit Congestion Notification (ECN) capability etc., which again are unknown to the tenant. This gap in knowledge of the network topology and its capabilities makes it tough to choose a protocol. TCP is the most popular protocol in the cloud because of its adaptability to network variability and it accounts for more than 90% of the traffic [31]. Therefore, the question then becomes which TCP variant to choose? and how to choose it?. Choosing the correct TCP is important because it impacts the throughput, round trip time (RTT) and response time of the applications deployed by the tenants. All of these metrics, especially response time, directly impact user experience of the application and hence, have an effect on the popularity thereof. Correct choice of TCP may also lead to reduction in the overall amount of network traffic sent between the VMs, which in turn would result in reduction in revenue spent on network communication and hence, possibly, increase profits.

Problem Statement: Conduct a measurement study to characterize the throughput, RTT and response time behaviors of TCP Cubic, Reno, Vegas and DCTCP as well provide insights into the behaviors based on the design semantics of the TCP variants.

1.2 Research Contributions

Through application of our knowledge of TCP semantics and traffic behavior, we have built up on and furthered the research from [12, 20, 52] and [9, 27, 30, 39, 47], to enable better data transport in both access networks and datacenter networks respectively. Our investigations have led to two key solutions 1) A novel probabilistic buffer sizing scheme for access routers handling IoT traffic and, 2) Improved guidelines for selection of TCP congestion control variants based on empirical evidence collected from a cloud service.

1.2.1 Improved Buffer Sizing for Access Routers

Based on the discussion in section 1.1.1, we know that selecting an appropriate size of this output buffer is crucial because it directly impacts two key performance metrics of an IoT system: 1) access link utilization and 2) latency. Therefore, we have developed a novel probabilistic scheme for calculating the size of the output buffer of IoT access routers. The model ensures that the access link stays fully utilized as long as the IoT devices have enough data to send and at the same time, significantly lowers the average latency experienced by the IoT packets. To calculate this buffer size, we first theoretically model the average TCP

congestion window size of all IoT devices, and eliminate three key assumptions of prior art that do not hold true in IoT TCP traffic, as demonstrated by our motivational measurement study. These assumptions include: 1) the sawtooth time-series of TCP congestion window of each flow is uniform, 2) TCP traffic is comprised primarily of long flows, and 3) TCP traffic arriving at the access router can be paced. Our results from an extensive set of simulations show that for IoT traffic, buffer size calculated by the state of the art schemes results in at least 60% higher queuing delay compared to our scheme while achieving almost similar access link utilization, loss-rate, and goodput as our scheme. For UDP only traffic our scheme achieves at least 91% link utilization with very low queuing delays and aggregate goodput that is approx. 90% of link capacity. Finally, for mixed traffic scenarios we show that our scheme performs better than either TCP only or UDP only traffic scenarios achieving higher link utilization than either, as well as low delays, low loss-rates and aggregate goodput that is approx 94% of link capacity.

1.2.2 Insights into Performance of TCP Variants in Datacenter Environments

To bridge the gap in knowledge as described in section 1.1.2, in this work we have focused on characterizing TCP Cubic, Reno, Vegas and DCTCP from the perspective of cloud tenants while treating the network as a black box which should help the tenants make a better choice as to which TCP variant to deploy. We have conducted experiments on two layers 1) transport layer and 2) application layer. For the transport layer, we have studied network behavior isolated from application induced behavior, by generating raw TCP traffic via iPerf which is a popular traffic generation tool. From the results we have observed that, 1) piecewise behavior exists for long lived flows which have an exponential period followed by a steady state period, 2) Vegas has the highest steady state throughput and lowest steady state RTT for long lived flows, 3) Reno has the least stable throughput over time for any given flow and its RTT behavior is identical to DCTCP, 4) Cubic has the highest (worst) average RTT and least stable RTT, and 5) the performance of the AWS network is similar and stable regardless of the time of day. For the application layer experiments we have used jMeter, another traffic generation tool, to generate HTTP traffic. The experiments from this layer show that 1) the piecewise behavior disappears or is greatly mitigated if the flows are not long lived, 2) Vegas has the worst throughput if the flows are not long lived whereas other variants deliver similar performance, 3) Vegas has the worst response time while all other variants perform similarly and, 4) different inter-request delay distributions have almost no effect on the throughput, RTT, or response time.

1.3 Thesis Organization

The rest of the thesis is organized as follows. We first present the related works and their shortcomings in Chapter 2. In Chapter 3 we present the first project targeting access networks and buffer sizing of routers therein. We detail the derivation of the scheme and a performance comparison with the state-of-the-art schemes. Chapter 4 shifts the focus to datacenter networks. We explain the experiment design and instrumentation tools followed by analysis and explanation of the behaviors and trends observed. Finally, Chapter 5 concludes the thesis, provides an overall summary and discusses the future directions.

Chapter 2

Background and Related Work

In this chapter, we provide background required for various aspects of our thesis. This thesis focuses on improving performance of access networks by focusing on router design and data center networks by providing an in depth analysis of TCP profiles. This chapter summarizes the state of the art research work that is closely related to the themes described. We also highlight the shortcomings of the related works to emphasize the need, novelty and effectiveness of our work. Inline with the objective of the thesis, the following two subsections will discuss the background of the two projects separately.

2.1 Router Design for Congested Links

To the best of our knowledge, while work has been done on router buffer design, no prior work addresses this problem for IoT access routers. In the following subsection we will first describe three lines of work that are related to our project and in the next subsection we will highlight the shortcomings thereof.

2.1.1 Buffer Sizing Schemes for Conventional Networks

Sizing output buffers of routers is not a new problem and has been studied for quite a while. The first seminal piece of work dates back to 1994 [52] by Villamizar and Song. They performed an experimental study using 1, 4, and 8 long TCP flows on a 40Mbps link. Based on their observations, they proposed that to guarantee 100% utilization of any given link, the size of the output buffer of a router's interface connected to that link should equal the product of the link bandwidth and the average RTT, *i.e.*, the time elapsed between sending a packet and receiving its ACK.

Morris followed the same trend and performed two simulation studies on a 10Mbps link with

1500 TCP flows and a 25ms lower bound on the RTT [40, 41]. He observed that the buffer size calculated using the delay bandwidth product rule proposed in [52] is sufficient only when the number of flows is small. With larger number of flows, when the amount of data arriving at the router exceeds the product of the outgoing link bandwidth and the lowest possible RTT, the senders start experiencing highly variable delay and higher packet drops at the output buffer despite high link utilization. As a quick solution to this problem, he suggested that the size of the output buffer should be made proportional to the maximum number of active flows. Through simulations, he showed that the buffer size 10 times as large as the number of flows provided a good tradeoff between link utilization and packet drop rate.

Appenzeller *et al.*, however, made an conflicting observation in [12] compared to the observations presented by Moris. This was the second seminal work that later came to be known as the standford model for buffer sizing due to its popularity. The authors showed that the amount of buffer calculated in [52] was actually more than what is required for TCP traffic seen in the real-world. They noted that one of the assumptions that Villamizar and Song made in their work, namely, the sawtooth time-series of the congestion windows of all senders are synchronized, was unrealistic, and caused the calculated buffer sizes to be larger than required. They showed that the amount of buffer actually required to keep the link utilization near 100% is equal to that calculated by Villamizar and Song divided by the square root of the number of flows, when the number of flows is large; and this is a widely accepted result in the community for TCP traffic in conventional networks [36]. The key limitation of Appenzeller *et al.*'s work, when considered for IoT TCP traffic, is that even though they did not assume that the sawtooth time-series of the congestion windows of senders are synchronized, they still made the assumption (although not explicitly stated in their paper) that the sawtooth time-series of the congestion windows of all senders are uniform. Unfortunately, this assumption does not hold, especially, in IoT TCP traffic, as we will show through experiments in Section 3.2. Furthermore, although the authors discussed the presence of short flows in the real-world traffic, they argued that the number of short flows is very small compared to the long flows, and thus, do not contribute to buffering requirements. Therefore, they did not take short flows into consideration while theoretically calculating the buffer size. While their argument might hold for traffic on traditional networks, it does not hold in the case of IoT networks because a significant portion of the flows in IoT traffic are short flows.

Dhamdhare *et al.*, unlike the work before them, took a probabilistic approach to calculate the output buffer [20]. This was a fresh take and they showed that the buffer requirement in the presence of multiple TCP flows depends on the harmonic mean of the RTTs of all senders. They also showed that buffer requirement depends on the correlation of packet losses across different flows. Due to the probabilistic nature of their approach, the buffer sizing formula that they proposed is rather complex. To keep the theoretical development tractable, they made the same two assumption as [12].

Gorinsky *et al.* were the first to propose a flow proportional scheme that could be used for

protocols other than TCP. They proposed to set the output buffer size large enough such that it can hold twice as many packets as the number of incoming flows [26]. The objective of this approach is primarily to reduce the packet loss rate at the output buffer, and does not take the link utilization and latency into account. As the size of buffer calculated using this method increases linearly with the increase in the number of flows, the queueing delay experienced by the packets increases greatly as the number of incoming flows increases. Long queueing delays are undesirable for duty-cycled IoT devices and real-time IoT applications.

Similarly other works kept on emerging, each with a new sizing scheme based on observations from the traffic, with some schemes clearly in conflict with others. Enachescu *et al.* proposed that the buffer size for routers in internet core can be made as small as the logarithm of the sender's congestion window size while keeping the link utilization high if the TCP traffic arriving at the routers is paced (either with the help of senders or due to the over-provisioned network core) [21]. Beheshti *et al.* extended the work in [21] to calculate buffer sizes for optical packet-switched routers with the objective of keeping the buffer sizes very small at around 20-100 packets only (optical routers can only have small buffers) [13]. Similar to [21], [13] also requires the TCP traffic to be paced to make the buffer sizes small, and relies on the large bandwidth of core links compared to the access links to do this pacing. In [14], Beheshti *et al.* showed how to build a network from routers with tiny buffers as long as the TCP traffic is well paced. Unfortunately, pacing the TCP traffic, either by asking the sender to space out its packets or by over-provisioning the bandwidth of the access link, is not possible in IoT systems due to the energy and cost constraints, respectively.

2.1.2 Measurement Studies on Buffer Sizing

Researchers have also done measurement studies related to buffer sizing. For example, Prasad *et al.* empirically showed that if the ratio of the output capacity of the router to its input capacity is less than 1, then the buffer size required is just a few dozen packets, whereas, when this ratio is greater than 1, the required buffer size increases significantly [44]. In [15], Beheshti *et al.* provided an experimental evaluation and comparison of the buffer sizing schemes mentioned above in various different scenarios. Along similar lines, [11] considers a simple model of TCP with drop-tail buffer management, and studies tradeoffs between buffer size (delay) and throughput. Through simulations they compare the the bandwidth delay product buffer sizing scheme to smaller buffer sizes and conclude that smaller buffers are adequate to maintain high utilization in practice. Work from this line doesn't directly influence our research but still provides insights into the dynamics of TCP semantics and the role they play in buffer sizing.

2.1.3 Active Queue Management

Another approach that researchers have explored to control link utilization, latency, and loss rate is known as active queue management [37, 38, 42, 43]. The key idea behind active queue management techniques is to induce early packet drops and keep the output buffer from getting full and thus, avoid global loss synchronization. The two approaches, *i.e.*, buffer sizing and active queue management, are complementary to each other and have their pros and cons. Active queue management techniques can help avoid bursty losses but lead to lower link utilizations while buffer sizing techniques can keep the link utilization at a desired level but suffer from bursty losses. We envision that a truly desirable technique would merge active queue management techniques and buffer sizing techniques such that the resulting technique keeps the link utilization high and at the same times, experiences reduced loss bursts. The possibility of merging these two types of techniques is a good topic for exploration in future work.

2.1.4 Limitations of Prior Art

Although, researchers have been working on the problem of calculating the optimal buffer size for routers in *conventional networks*, the assumptions that the prior buffer sizing schemes make about the TCP traffic and the network characteristics, while true in conventional networks, do not hold true in IoT systems. Therefore, the problem of buffer sizing must be revisited for IoT systems. In this subsection we describe three important assumptions, either or all of which are made by prior schemes, and explain why these assumptions do not hold true in IoT systems.

Assumption 1 - Uniform Sawtooth Congestion Window: The first assumption is that the TCP congestion window size of *each* sender follows a *uniform* sawtooth time-series [12, 20, 40, 41]. The sawtooth shape results from the famous additive-increase & multiplicative-decrease congestion avoidance method of TCP. A *uniform* sawtooth time-series is a sawtooth time-series wherein any given crest or trough has the same value as any other crest or trough, respectively. For the sawtooth time-series of a flow to be uniform, that flow must never experience more than one packet drop at a time, and the packet drop should happen exactly when the congestion window size attains the same value as its value at the time of the previous most recent packet drop. While we observed from our experiments (described in Section 3.2) that the congestion window sizes of IoT devices make sawtooth time-series, the sawtooth time-series are never uniform. The reason being that the TCP flows mostly experience variable number of consecutive packet drops and packet drops do not always occur at a fixed value of the congestion window size. It is imperative to remove this assumption because it leads to under-provisioning of the output buffer in some schemes [20] and over-provisioning in some others [12, 52].

Assumption 2 - Negligible Effect of Short Flows: The second assumption is that the TCP flows of the majority of the devices are *long flows*, *i.e.*, the flows are persistent and always stay in congestion avoidance mode [12, 20, 40, 41]. This isn't true for IoT systems because TCP flows of IoT devices are frequently *short flows*, *i.e.*, the flows are either non-persistent, or in slow start mode, or both. This happens because IoT devices are often resource constrained and thus, kill their TCP connections and sleep soon after transmitting the current data and set up new connections when they have new data to send. Consequently, the TCP traffic generated by a typical IoT system contains a good mix of both long and short flows. It is imperative to remove this assumption as well because the size of the output buffer calculated using this assumption leads to the over-provisioning of the output buffer because by treating short flows as long flows, the amount of total traffic theoretically expected to arrive at the access router is greater than the amount of traffic that actually arrives.

Assumption 3 - TCP Traffic Pacing: The third assumption is that the traffic arriving at the router can be smoothed by asking the senders to pace their TCP traffic, *i.e.*, instead of sending all packets allowed by the congestion window in a single burst, space the packets out over an entire round trip time (RTT) [13, 14, 15, 21]. While smoother non-bursty traffic resulting from TCP pacing reduces the buffer size required to keep the access link utilization high, and thus reduces the latency, unfortunately, TCP pacing is not suitable for the energy-constrained IoT systems because it requires the IoT devices to stay awake the entire time. This requirement is contrary to one of the primary goals in the design of energy-constrained IoT systems, *viz.*, keep the IoT devices asleep as long as possible by performing sensing/actuation and data transmission/reception as quickly as possible. A caveat worth mentioning here is that when any TCP traffic reaches the internet core, it automatically gets paced on the core links due to the significantly larger bandwidths of the core links compared to the access links. Consequently, the TCP traffic arriving at the core routers is already well paced [13]. However, as the bandwidth of the access link is actually much smaller than the bandwidths of the links between the IoT devices and the IoT access router (as they are on a local area network), such automatic pacing does not occur for the traffic arriving at the IoT access routers.

2.2 Behavior of TCP in Unknown Networks

In the following subsections we describe research work related to our second project which focuses on datacenter networks. There are three lines of work, out of which the first two are directly related where as the third has been a major inspiration and is also an applicability case for our research.

2.2.1 TCP Congestion Control Algorithms:

The first body of works that directly relates to our measurement study is the work done on TCP variants. Ever since the introduction of the original Van Jacobson congestion control algorithm, researchers have proposed several variants that aim to improve network stability, bandwidth fairness, allocation, resource utilization etc. These works relate to our measurement study because they detail the performance of the proposed variants under different conditions. Three such pieces of work on TCP variants relate directly to our study. TCP Cubic was proposed in [29] by Xu and Rhee as an improvement to TCP BIC [54]. Cubic was design for high speed networks with an aim to simplify the BIC congestion control as well as to improve TCP-friendliness and RTT-fairness. The paper showed that Cubic could maintain a fair share of the bandwidth when running in the same network as other TCPs in both low RTT and high RTT networks. It was also shown that Cubic shares the bandwidth more fairly between different flows sharing the same link as compared to other TCP variants. Second up is TCP Vegas [17], it was proposed by Brakmo et al. and is another attempt at congestion control that claimed to have between 37% and 71% better throughput compared to TCP Reno. Vegas was the first attempt at developing a delay-based congestion control mechanism as opposed to a loss-based scheme. Due to it's fine grained time sampling and hence better throughput estimates, it was shown to have between 20% and 50% less losses than TCP Reno. It was also shown to be more fair in sharing bandwidth among flows than TCP Reno. Note that this study did partially explore the effect of transfer size/ file size on the performance metrics of the variants but the size was limited to 1 Mbyte. Third and lastly is Data center TCP (DCTCP) [10], it's a newer variant proposed by Alizadeh et al. designed specifically with data centers in mind. It uses neither loss-based nor delay-based schemes to estimate congestion in the network. It relies on explicit congestion notification (ECN) packet from the network hardware to get an explicit signal of the extent of congestion. The results show DCTCP to be incredibly fair in sharing bandwidth among flows on the same link when compared to other TCPs, achieving both higher aggregate throughput and lower aggregate loss. However DTCP does not co-exist well with other TCPs in the same network. In case of congestion or loss, it decreases its congestion window much slower than conventional TCP variants and so the throughput reduction in DCTCP will be much slower. As a result DCTCP gains a larger share of the capacity compared to conventional TCP traffic traversing the same path.

2.2.2 Comparative Measurement Studies:

The second line of related works are comparative studies such as [47], [39], [27], [30], [9]. In [47] Rao et al. have done a comparative study of TCP Cubic, HTCP, STCP performance for wide area data transfers in high performance computing (HPC) infrastructures using both simulations and physical testbeds. They have observed that 1) slow-start followed by well sustained throughput leads to concave regions in throughput profile, 2) large buffers and

multiple parallel streams expand the concave regions in addition to improving the throughput, and 3) stable throughput dynamics, indicated by a smoother Poincaré map and smaller Lyapunov exponents, lead to wider concave regions. The authors have also developed a coarse throughput model for the TCPs which produces results inline with the observations above. Mo et al. have done an analytical comparison of TCP Reno and Vegas in [39]. The contributions are two fold, first the authors propose improvement to the Vegas standard and second they show that Vegas is superior to Reno in using the network resources. In particular they have shown that TCP Reno is less fair or too aggressive such that TCP Vegas connection does not receive a fair share of bandwidth in the presence of a TCP Reno connection unless the buffer sizes are extremely small. They also show that TCP Vegas is less affected by propagation delays as compared to Reno. Similar work has been on Westwood+, New Reno, and Vegas has been by Grieco and Mascolo [27]. Their simulation results have shown that TCP Westwood+ is both friendly towards New Reno and it improves the bandwidth sharing fairness with New Reno. Westwood+ also improves the utilization of wireless links as compared to both Vegas and New Reno even in the presence of uniform or bursty losses. They also show that TCP Vegas is not able to grab a proportional share of the bandwidth when sharing a link with New Reno which is inline with the results presented in [39]. Work by Hasegawa et al. in [30] again comments on improvement of fairness between Vegas and Reno. They propose changes in the Vegas congestion control algorithm and Random Early Detection (RED) algorithm to improve the performance. The modified version achieves better fairness and resolves the problem observed in [39]. Finally in [9] Abdeljaouad et al. investigate the goodput, intra- and inter-protocol fairness of TCP Compound, Cubic and New Reno. Their results show that over wired links TCP Cubic outperforms TCP Compound and New Reno in the presence of reverse traffic. They also show for wired links, as measured by Jain's fairness index, that TCP compound is the most fair of the variants studied. However, When the connections are wireless, all the variants are fair.

2.2.3 Virtualized Congestion Control and Homogenized TCP:

The third line of work though not directly related, relates to the cloud tenants. Works such as [32] and [19] try to homogenise the TCP in a data center in order to improve the network performance *i.e.* throughput, fairness, RTT and response times of all applications. Both the papers propose the addition of a middle box translation layer in the hypervisors that translates from the VM controlled TCP variant to a hypervisor controlled TCP variant. Since the hypervisor is in control of the cloud providers and hence, the network administrators as well, a single TCP variant can be deployed throughout the data center. The reason why these works are related to our study is that while these works detail how to translate TCP while achieving minimal overhead, they do little to motivate what TCP to translate to. Our effort are directed to provide TCP performance profiles that would help the administrators to make a better choice of the TCP variants. It is to be noted here that our study is mainly aimed at cloud tenants and not network administrators.

2.2.4 Limitations of Prior Art:

Perspective Matters: As discussed above, a lot of work has been done on TCP but a recurring trend to be noted is that *such studies are done from the cloud provider’s or network administrator’s perspective i.e.* how will a certain variant of TCP would benefit the entire datacenter or the entire network. They preclude the tenant and tenant specific needs. Metrics such as fairness and friendliness are of no concern to the tenant because the tenant neither knows of nor has control over what other TCP variants are being used on the same subnet.

Knowledge of the Network Topology and Hardware: Previous work on the TCP protocols themselves such as Reno, Cubic [29], DCTCP [10], Vegas [17] *etc.*, do explain the performance of the proposed variants in detail and under varied scenarios but do so in a controlled setting where the network between data sources and data sinks is explicitly configured. Comparative pieces of work on TCP such as [47], [39], [27], [30] and [9] also rely on the knowledge of the network topology. In [47] the authors have attempted to profile the performance of TCP variants in high performance computing (HPC) environments but the study has been done in a controlled environment where either the network was already known or explicitly configured. Mo et al. compare TCP Vegas and Reno in [39] but the study is from the network administrators perspective and inspects metrics such as TCP fairness and the (in)compatibility of the two TCP variants. It again relies on simulations and an explicitly configured network. Work by Grieco et al. in [27] follows the same trend as [39] *i.e.* the study is from network administrator’s perspective, relies on majorly simulations and measures aspects such as throughput, fairness and friendliness. [30] again follows along the same lines as [39] and [27]. Finally [9] does the same thing.

Limitations of Simulators: A lot of these works also rely on simulators such as Network Simulator 3 (NS3), OMNet++, SimPy *etc.*, which may not necessarily reflect real world scenarios despite the effort to do so. For example, [45] points out the limitations of NS3 in terms of credibility, validation, scalability and lack of upper layer functionality. Similarly in [18] Chaudhry et al. comment that NS2 abstracts a large amount of the network layer and below, which leads to big discontinuities when transitioning from simulation to experiment.

Loss of User Control: Works like [32] and [19] attempt to bridge that gap by homogenizing TCP across entire data-centers in a bid to provide better performance to both the user applications and the datacenter at the same time. However the deployment of such solutions, again, is in the hands of the cloud provider and not the tenant. Furthermore, while [32] and [19] do talk of homogenizing TCP across datacenters, they do little to explain which variant to translate to under varying conditions and application needs. Since, a “*one size fits all TCP variant*” does not exist at the moment, translating all variants to a single one might hurt application performance for some tenants while trying to achieve the datacenter wide performance objectives. This, by nature of the consequence mentioned, might be a difficult route for cloud providers to take.

Chapter 3

Sizing Buffers for IoT Access Routers

3.1 Introduction

With the advent of the Internet of Things (IoT), the number of devices connecting to the internet is growing at an unprecedented rate [23]. These devices, often referred to as IoT devices, are usually small embedded computers that implement the TCP/IP network protocol stack, have one or more network interface cards, and have general purpose input/output pins through which they can interface with external sensors, actuators, and even other IoT devices. Examples of such commercially available IoT devices include, but are not limited to Raspberry Pi [7] and Arduino [1]. In typical IoT systems, these IoT devices are connected to sensors and/or actuators directly and to one or more remote cloud services through the internet. The IoT devices obtain measurements from sensors either periodically or when some predefined event(s) take place and send those measurements to an appropriate cloud service over the internet, using TCP for reliability (*e.g.*, several famous IoT application layer protocols, such as MQTT and XMPP, use TCP). The cloud service processes the measurements and sends appropriate instructions, such as changing the state of an actuator, back to the IoT devices, which the devices execute.

To connect to cloud services, the IoT devices access the internet through an access router. Due to significant differences between IoT networks and conventional networks, such as in terms of traffic type, medium access protocols, and device heterogeneity, vendors have already started developing access routers that are tailored for IoT networks, such as Cisco's 829 router [2] and Norton's Norton Core Router [6]. While there are several design aspects of an access router that impact an IoT system, such as the number and type of wireless/wired interfaces, one of the most important design aspects is the size of the output packet buffer of the interface of the access router that connects to the access link, *i.e.*, the internet. In the rest of this chapter, we will call this buffer, the output buffer.

Selecting an appropriate size of the output buffer of the IoT access router is crucial because

it directly impacts two key performance metrics of the IoT system: 1) access link utilization (*i.e.*, the percentage of time the link is under use at full capacity), and 2) latency (*i.e.*, the time a packet takes to go from sender to receiver). *If the buffer is under-provisioned, i.e.*, the size of the output buffer is too small, the access link utilization decreases because when IoT devices decrease their TCP congestion window sizes on experiencing packet losses and take an intermittent pause from sending new packets, the small output buffer drains quickly and the access link stays idle until the IoT devices resume sending the packets. *If the buffer is over-provisioned, i.e.*, the size of the output buffer is too large, the latency experienced by the packets of the IoT devices increases because during an intermittent increase in the number of packets arriving at the access router, the packets that arrive and enter the buffer when the buffer already has a large number of packets waiting in it, experience large queueing delays. The lower access link utilization and higher queueing delays can lead to decrease in the throughput and/or increase in the power consumption of the IoT devices. The increase in the power consumption stems from the fact that IoT devices are often duty-cycled to conserve power, *i.e.*, they wake up either periodically or when some predefined events occur, perform their tasks, and return to sleep. When the access link utilization is low, these devices have to stay awake longer to send their data, and thus consume more power. Similarly, when queueing delays are large, these devices have to stay awake longer while waiting for the acknowledgements of their packets.

Optimizing the access link utilization and latency is extremely important for IoT systems. Increasing the access link utilization is important because IoT systems lease access links of required bandwidths from internet service providers to access the internet, and this bandwidth is expensive. The price is even higher for IoT systems deployed in remote locations, such as \$1,200 per sensor for a cellular-based soil moisture measuring system [3]. Therefore, the access bandwidth must be fully used by keeping the access link fully utilized at all times. Reducing the latency is important for real-time IoT systems, such as smart power grids that acquire and distribute power based on the dynamically changing demands of various parts of a city in real time. A small increase in latency in such IoT systems may cause violations of service level agreements and result in significant financial and functional losses [16, 53]. We conclude that the size of the output buffer of access routers must be carefully calculated due to its impact on the two key performance metrics of IoT systems.

3.1.1 Problem Statement

In this chapter, our objective is to calculate the size of the output buffer of IoT access routers that ensures that the access link utilization stays close to 100% as long as the IoT devices send enough data and at the same time, significantly lowers the average latency experienced by the packets of the IoT devices.

3.1.2 Proposed Approach

In this chapter, we calculate the output buffer size for IoT access routers that ensures that the access link utilization stays close to 100% and at the same time, significantly lowers the average latency experienced by the packets. In calculating the optimal buffer size, we do not make any of the three assumptions as mentioned above. Before we describe our approach to calculate the size of the output buffer of IoT access router, we first define the term *congestion event*.

The primary purpose of an output buffer is to store packets during the time the amount of data arriving at the access router from the IoT devices exceeds the bandwidth of the access link. If the duration for which the amount of incoming data exceeds the access link bandwidth is long, then the output buffer can become full, causing the access router to start dropping any subsequently arriving packets until some space in the output buffer opens up again. A congestion event is defined as the event where the IoT access router drops some packets due to the output buffer being full, such that the time difference between any consecutive drops in any given flow and any consecutive drop across flows is less than RTT. The motivation behind choosing RTT as the threshold is that TCP takes at least one RTT to react to packet drops.

With this definition of the congestion event, we calculate the size of the output buffer of access routers in three steps. In the first step, we calculate the probability that any given TCP flow will experience a drop of d consecutive packets during a congestion event. Using this probability, we derive an expression that quantifies the fraction of all TCP flows that each experience a drop of d consecutive packets during this congestion event. In the second step, we use this expression to calculate the average congestion window size across all flows, including both long flows and short flows, after the congestion event. In calculating this average congestion window size, we take two important semantics of TCP Reno into account: 1) every time TCP detects drop of a new packet of a given flow, it decreases the size of the congestion window of that flow by a factor of 2, and 2) TCP detects drop of at most one packet every RTT. Consequently, any flow that experiences a drop of d consecutive packets decreases the size of its congestion window by a factor of 2^d in d RTTs and stops sending any new packets until the amount of its outstanding data falls below its new congestion window size. Finally, in the last step, we use the average congestion window size calculated in the second step to calculate the *minimum* size of the output buffer, such that the buffer does not go empty during the time some IoT devices that experience packet drops pause sending new packets. We focus on calculating the *minimum* buffer size because the average latency increases monotonically with increasing buffer size (we will empirically show this in the evaluation section). By calculating the *minimum* buffer size that keeps the access link utilization near 100%, we ensure that the average latency is minimized.

3.1.3 Key Contributions

In this chapter, we make following three key contributions: 1) We present detailed theoretical development to calculate the size of the output buffers of IoT access routers to keep access links fully utilized and latency low. 2) We have implemented a real IoT system comprised of a cluster of 15 Raspberry Pi 3 IoT devices [7] and a configurable NETGEAR FS750T2 switch [4], to study the assumptions of prior art in IoT TCP traffic. 3) We present the implementation, evaluation, and comparison of our scheme with prior art through extensive simulations. Our results show that for IoT traffic, the buffer sizes calculated by our scheme not only keep the link utilization high but also reduce the latency by about 50% compared to the buffer sizes calculated by the prior schemes.

3.2 Exploratory Study

In this section, we present our observations from two sets of experiments that show that the first two assumptions, described in Section 2.1.4, do not hold in IoT traffic. We do not empirically study the third assumption because it is not an assumption about the properties of traffic due to the congestion control mechanism of TCP, rather it is about the properties of traffic due to the way the senders schedule the packet transmissions. Next, we first present the set of experiments to study the first assumption, *i.e.*, congestion window's saw-tooth is uniform, and after that, present the set of experiments to study the second assumption, *i.e.*, percentage of short flows is negligible.

Assumption 1: Congestion Window's Saw-tooth is Uniform

In the first set of experiments, we took 15 Raspberry Pi 3s and installed Raspbian Linux on them. The reason for choosing Raspberry Pis is that hardware configurations of IoT devices vary widely depending on the developer. Therefore, we have chosen the most general single board compute units available. We placed a `printk` statement in the `net/ipv4/tcp_input.c` file of Linux kernel version 4.9 to log the value of the `snd_cwnd` variable, *i.e.*, the congestion window size. We compiled this kernel and used it in the Raspbian Linux. We physically connected these 15 Raspberry Pi 3s to a server through a NETGEAR FS750T2 switch, as shown in Figure 3.1. All ports of the switch were configured to operate full duplex at 10Mbps. We deployed an application layer process on each Raspberry Pi that initiates a TCP connection with the server and pumps data into that connection at a rate of 1Mbps. Figure 3.2 plots the time-series of the sizes of congestion windows of TCP flows of three randomly chosen Raspberry Pis. We observe from this figure that although all three time-series follow a sawtooth pattern, the sawtooth is not uniform due to the variable number of packet drops experienced by the TCP flows at different time instants. This is contrary to the assumption made by several prior schemes, such as [12, 20, 40, 41], that the sawtooth

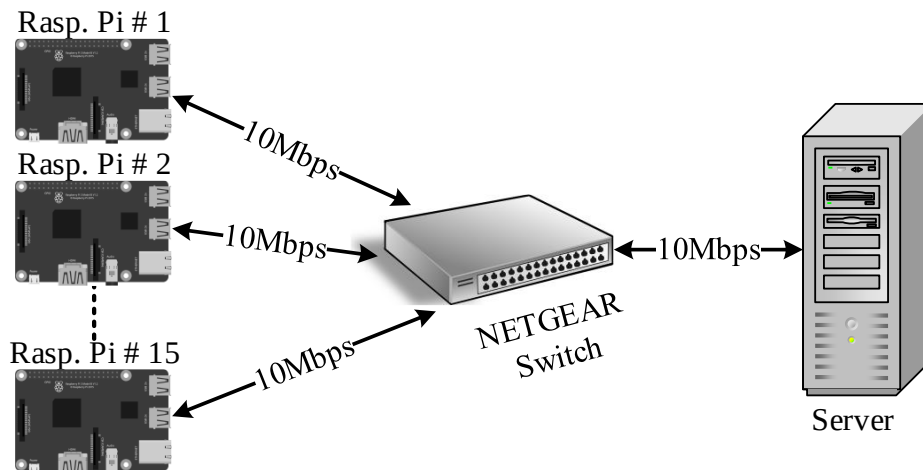


Figure 3.1: The setup and topology used to study the assumptions of prior art

time-series of the congestion window size of each TCP flow is always uniform. The reason why the (in)validity this assumption matters is that most buffer sizing schemes including ours, are based on modeling of the congestion window. In particular, our scheme is based on modeling the average congestion window before and after congestion takes place at the router queue. The non-uniformity or uniformity would change the average congestion window after the congestion event and this may lead to longer or shorter wait times depending on whether the average value increases or decreases. Since the duration of pausing the senders changes, the duration in which no packets are sent out change as well and so, the buffer requirements change. The extent of the change would depend on the change in the average value of the congestion window. Therefore, we deemed it important to do away with the simplifying assumption of uniform sawtooth congestion window.

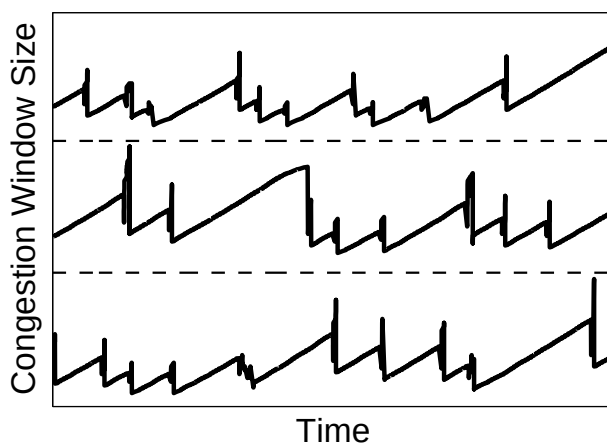


Figure 3.2: Time-series of congestion window sizes of three Raspberry Pis

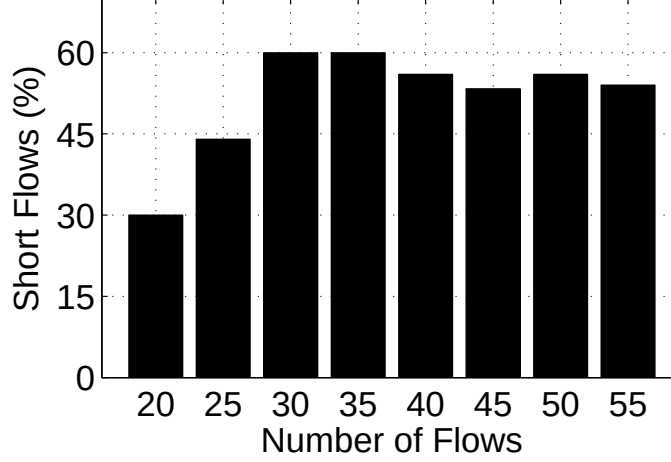


Figure 3.3: Percentage of short flows in IoT traffic

Assumption 2: Percentage of Short Flows is Negligible

In the second set of experiments, we performed NS-3 [5] simulations to study the validity of the second assumption. For this set of experiments, we used NS-3 simulations and not the real Raspberry Pis so that we can study this assumption in the scenarios where the number of IoT devices sending data is large. Since, we are using a simulation, we have taken care that the scenario simulated is representative of the real world. We simulated the topology shown in Figure 3.1 with varying number of IoT devices using access link bandwidth of 40Mbps, one way link-latency between IoT devices and server of 20.02 ms, and MTU of 1500 bytes. The 0.02 ms part of the delay comes from the one way delay between the IoT and switch, while the 20 ms comes from the delay between the switch and the server. The delay between the devices and the switch is small because they are mostly just a single hop away and on the same subnet while the 20 ms was empirically determined using speed tests. We programmed the application layer process on each simulated IoT device to generate packets with exponentially distributed inter-arrival times such that the aggregate rate at which the data arrived at the switch was equal to the access link bandwidth. The reason for choosing the exponential distribution is that the combined effect of these exponential distributions will be to generate a Poisson arrival process for the packets at the switch port. This is inline with the fact that queues or buffers are birth and death processes which are Poisson in nature. Each IoT device generated a single flow at a time. Note that the IoT devices, especially the ones that are constrained by energy, would prefer to send their data in a non-periodic / event based manner *i.e.* send data only when changes in data are sensed. In between these bursts of sending packets, the devices will most probably be in low power mode and have their radios turned off. The duration of having the radios off and the asynchronous nature of data transmission is captured by using an exponential distribution for inter-packet arrival time. For the same energy constraints as well as resource constraints, the devices would go to sleep mode and terminate their TCP connections if their corresponding

application layer processes do not provide their TCP layers a packet to send out within a certain amount of time after sending out the previous packet. Keeping the connection alive would require state maintenance and hence, would consume resources. We kept this time fixed at twice the inter-arrival time of packets. Any flow that terminates while it is still in TCP's slow-start mode is considered a short flow; all other flows are considered long flows. Hence, if we take into account that 1) the simulation scenario is catering for the turning off of radios and terminating connections, 2) the data transmission is asynchronous and, 3) over all packet arrival is a Poisson process, then we can say that the proposed simulation scenario is representative of the real world. We ran our simulator multiple times using a different number of devices during each run and measured how many flows generated by the IoT devices turned out to be short flows. Figure 3.3 shows a bar chart of the percentage of short flows in the traffic that the IoT devices generated during the simulations. We observe from this figure that this percentage is large for any number of IoT devices. This is contrary to the scenario considered by prior schemes, such as [12, 20] both of which separately discuss short flows and develop a separate buffer sizing scheme for them. However, in their results they show that the the buffer sizing is dictated by long flows and so the formula developed for just long flows is viable for use. Hence, the end effect is that short flows are ignored in the sizing scheme.

3.3 Buffer Size Calculation

Recall that in IoT systems, IoT devices connect and communicate with a cloud service to achieve the full potential of the IoT system. To reach the cloud service, each IoT device forwards its data over a wireless/wired link to an IoT access router. The access router then forwards that data to the internet over the access link, and the internet routes it to the destination cloud service, as shown in Figure 3.4. Note that “today, the core of the internet is over-provisioned with high speed links that experience little congestion” [36]. The bottleneck links are almost always the access links. In Figure 3.4, there are two access links, one that connects the IoT access router to the internet and the other that connects the data center hosting the cloud service to the internet. Again, today, the data center access links are over-provisioned to provide high quality of service to customers. Thus, it is the access link that IoT devices access the internet through that is the bottleneck, and thus needs carefully sized output buffer at the IoT access router.

Using this typical IoT implementation as reference, in this section, we derive an expression to calculate the minimum size of the output buffer that keeps the access link fully utilized. As different IoT implementations can differ from each other in various ways, such as the amount of data and the number of IoT devices, the expression to calculate the buffer size will have several tunable parameters. In this section, we focus only on the derivation of the expression to calculate the buffer size. We will describe how to select the values of the tunable parameters in Section 3.4. Next, we first derive an expression to calculate the fraction

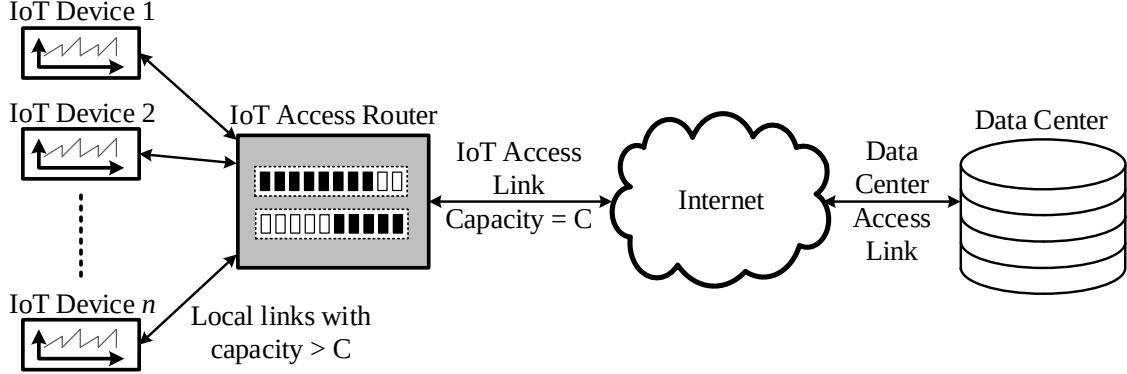


Figure 3.4: Block diagram of a typical IoT implementation

of all flows that experience a given number of packet drops during a congestion event. Using this expression, we calculate the average congestion window size across all flows after the congestion event. Finally, we use this average congestion window size to calculate the size of the output buffer.

3.3.1 Fraction of Flows with Given Number of Dropped Packets

Let n represent the total number of flows that are sending packets to the IoT access router during a given congestion event. Let $\mathcal{P}_T$ be the random variable that represents the total number of packets that arrive at the router during the congestion event out of which, the router drops l packets. Let f_i represent the i^{th} flow, where $1 \leq i \leq n$, and $\mathcal{P}_{f_i}$ be the random variable that represents the number of packets of this i^{th} flow that arrive at the router during the congestion event. Let $\mathcal{D}_{f_i}$ be the random variable for the number of packets of flow f_i that the router drops during the congestion event. The random variable $\mathcal{D}_{f_i}$ follows a binomial distribution $\mathcal{D}_{f_i} \sim \text{Binom}(l, \mathcal{P}_{f_i}/\mathcal{P}_T)$, and is calculated as:

$$\Pr \{ \mathcal{D}_{f_i} = d \} = \binom{l}{d} \left(\frac{\mathcal{P}_{f_i}}{\mathcal{P}_T} \right)^d \left(1 - \frac{\mathcal{P}_{f_i}}{\mathcal{P}_T} \right)^{l-d} \quad (3.1)$$

Note that eq. 3.1 implies that the packets are dropped in an independent fashion and that different number of packets may be dropped in different sessions. The reason for independence is that if the packets are in flight, then whether one is dropped is simply dictated by whether the router buffer is full or not. In typical IoT deployments, all IoT devices that connect to a cloud service through a given access router perform the same sensing/actuation tasks and are also often deployed by the same vendor. Consequently, the sizes of their flows follow identical distributions. This implies that the distribution of the number of packets of any given flow arriving at the router during the given congestion event is the same across all flows, *i.e.*, $\forall i, j \in [1, n], \mathcal{P}_{f_i} = \mathcal{P}_{f_j}$, which in turn means that $\forall i, j \in [1, n], \mathcal{D}_{f_i} = \mathcal{D}_{f_j}$ during that congestion event. Furthermore, one flow dropping a packet will have no effect on the TCP

congestion control semantics of another flow which means that the flows are also independent of one another. Hence, flows in an IoT deployment can be approximated using independent identical distributions i.i.d. Using this observation, we represent the number of packets of any given flow among the n flows arriving at the router during the congestion event with random variable $\mathcal{P}$ and the number of packets that the router drops of any given flow among the n flows during the congestion event with random variable $\mathcal{D}$. Thus, Eq. (3.1) becomes:

$$Pr\{\mathcal{D} = d\} = \binom{l}{d} \left(\frac{\mathcal{P}}{\mathcal{P}_T}\right)^d \left(1 - \frac{\mathcal{P}}{\mathcal{P}_T}\right)^{l-d} \quad (3.2)$$

As the total number of packets that arrive at the router during the congestion event is the sum of the number of packets of all flows that arrive at the router during the congestion event, $\mathcal{P}_T = \sum_{i=1}^n \mathcal{P} = n\mathcal{P}$. Thus, Eq. (3.2) becomes:

$$Pr\{\mathcal{D} = d\} = \binom{l}{d} \left(\frac{1}{n}\right)^d \left(1 - \frac{1}{n}\right)^{l-d} \quad (3.3)$$

Note that in the equation above, we approximated the ratio of the random variable $\mathcal{P}$ with itself with 1. This is reasonable, because the expected value of the ratio of a random variable with itself is 1 and the variance of the ratio of a random variable with itself is 0, *i.e.*, $E[\mathcal{P}/\mathcal{P}] = 1$ $\text{Var}(\mathcal{P}/\mathcal{P}) = 0$, as we show next. Based on the results derived in [51], the expected value and variance of the ratio of any two random variables $\mathcal{A}$ and $\mathcal{B}$ are given by the following two equations.

$$\begin{aligned} E\left[\frac{\mathcal{A}}{\mathcal{B}}\right] &= \frac{E[\mathcal{A}]}{E[\mathcal{B}]} - \frac{\text{Cov}(\mathcal{A}, \mathcal{B})}{E^2[\mathcal{B}]} + \frac{E[\mathcal{A}]}{E^3[\mathcal{B}]} \text{Var}(\mathcal{B}) \\ \text{Var}\left(\frac{\mathcal{A}}{\mathcal{B}}\right) &= \frac{\text{Var}(\mathcal{A})}{E^2[\mathcal{B}]} - \frac{2E[\mathcal{A}]}{E^3[\mathcal{B}]} \text{Cov}(\mathcal{A}, \mathcal{B}) + \frac{E^2[\mathcal{A}]}{E^4[\mathcal{B}]} \text{Var}(\mathcal{B}) \end{aligned}$$

Substituting $\mathcal{A} = \mathcal{B} = \mathcal{P}$ in the equations above and doing some simple algebraic manipulations, it is straightforward to see that $E[\mathcal{P}/\mathcal{P}] = 1$ and $\text{Var}(\mathcal{P}/\mathcal{P}) = 0$.

Let $\mathcal{I}_d$ be the indicator random variable for any given flow whose value is 1 if the given flow experiences a drop of exactly d packets. Let $\mathcal{N}_d$ be the random variable that represents the fraction of all flows that experience a drop of exactly d packets. Thus,

$$\mathcal{N}_d = \frac{\sum_{i=1}^n \mathcal{I}_d}{n} = \mathcal{I}_d \quad (3.4)$$

As $\mathcal{I}_d$ is a bernoulli random variable, $E[\mathcal{I}_d] = Pr\{\mathcal{I}_d = 1\} = Pr\{\mathcal{D} = d\}$. Applying the expectation operator on Eq. (3.4) and substituting the value of $Pr\{\mathcal{D} = d\}$ from Eq. (3.3), we get the following equation to calculate the expected fraction of all flows that experience a drop of exactly d packets during a congestion event, where $0 \leq d \leq l$.

$$E[\mathcal{N}_d] = \binom{l}{d} \left(\frac{1}{n}\right)^d \left(1 - \frac{1}{n}\right)^{l-d} \quad (3.5)$$

3.3.2 Average Congestion Window Size after Congestion Event

Let W_b represent the average size of the congestion window across all n flows right before the congestion event started. Similarly, let $W_a^{(\xi)}$ represent the average size of the congestion window across all n flows ξ RTTs after the start of the congestion event. During the congestion event, on average, the fraction of flows that experience a drop of d packets will be $E[\mathcal{N}_d]$. As a TCP Reno sender reduces its congestion window size by 50% on detecting each packet drop and as a TCP Reno sender detects only one packet drop per RTT, all flows that experience a drop of d packets will reduce their window sizes by a factor of 2^d after d RTTs since the start of the congestion event. Note that, on average, $E[\mathcal{N}_0]$ fraction of all n flows will not experience any packet drops, and will therefore increase their window sizes. If a flow that experiences no packet drop during the congestion event is in slow start phase, it will double its window size within one RTT. Similarly, if a flow that experiences no packet drop during the congestion event is in congestion avoidance phase, it will increase its window size by the maximum segment size (MSS) within one RTT. Let α represent the expected fraction of flows among the n TCP flows that are long flows, *i.e.*, are in congestion avoidance phase. Thus, the fraction of short flows, *i.e.*, the flows in the slow start phase, is $1 - \alpha$. Based on this discussion, the average congestion window size across all flows approximately ξ RTTs after the start of the congestion event is calculated as:

$$W_a^{(\xi)} = \left(\alpha(W_b + \xi \text{MSS}) + (1 - \alpha)2^{\xi-1}W_b \right) E[\mathcal{N}_0] + \sum_{d=1}^{\xi} \frac{W_b}{2^d} E[\mathcal{N}_d] \quad (3.6)$$

The term $(\alpha(W_b + \xi \text{MSS}) + (1 - \alpha)2^{\xi-1}W_b) \times E[\mathcal{N}_0]$ in the equation above captures the increase in the congestion window size contributed by the flows that did not experience packet drops. Note that this term explicitly takes into account the contributions from both long flows and short flows separately by using the terms $\alpha(W_b + \xi \text{MSS})$ and $(1 - \alpha)2^{\xi-1}W_b$, respectively. The term $\sum_{d=1}^{\xi} \frac{W_b}{2^d} E[\mathcal{N}_d]$ captures the decrease in the average congestion window size contributed by the flows that experienced packet drops. Note that this term takes the contributions from both long flows and short flows into account because on experiencing a packet loss, both types of flows reduce the sizes of their congestion windows by 50% every RTT.

3.3.3 Output Buffer Size

Let C represent the capacity of the IoT access link, *i.e.*, the bottleneck link. Thus, the maximum aggregate bandwidth achievable by all IoT devices is C . Let T represent the average RTT experienced by the packets generated by the IoT devices minus any queueing delay in the IoT access router's output buffer. At the time instant immediately before the congestion event starts, the buffer must be full (otherwise, the congestion event would not have occurred). Furthermore, at this time instant, the total amount of data sent by all IoT

devices that is still unacknowledged is nW_b . This unacknowledged data is either residing in the buffer of the IoT access router or accommodated by the delay bandwidth product CT . Let B represent the size of the output buffer. Thus,

$$nW_b = B + CT \quad \Rightarrow \quad W_b = (B + CT)/n \quad (3.7)$$

Just under one RTT since the start of the congestion event, the TCP flow of any IoT device whose packets were dropped either times-out or receives triple duplicate ACKs, and thus detects a packet drop. On detecting the drop, TCP halves the congestion window size of that flow. As the window size limits the number of unacknowledged packets of the flow out in the network, the flow is allowed to have a larger number of unacknowledged packets before a packet drop is detected compared to after the packet drop is detected. Thus, the flow has more unacknowledged packets than it is currently allowed to have, and it must pause while it waits for the ACKs for those excess unacknowledged packets. As our objective is to keep the access link fully utilized, the output buffer of the access router must not go empty while some flows are paused. As we are interested in the minimum size for the output buffer, we consider the case when new packets start to arrive from the paused flows at the same moment when the last byte is drained from the buffer. As the sending rate of a TCP flow is equal to the ratio of its window size to the RTT its packets experience, the average sending rate of all n TCP flows ξ RTTs after the congestion event is $nW_a^{(\xi)}/T$. In order to keep the access link fully utilized, at the moment the last byte drains out of the buffer, the smallest value of this average sending rate must equal the bandwidth of the access link. The expected value of the number of RTTs at which the smallest value of this average sending rate occurs is $\xi = \sum_{d=0}^l dE[\mathcal{N}_d] = \frac{l}{n}$. This is intuitive because after l/n RTTs have passed since the congestion event, on average, half the flows that had stopped increasing their amounts of outstanding packets in response to the packet drops during the congestion event have recovered from their respective losses and have started to increase their amounts of outstanding data by increasing their congestion window sizes. Consequently, after l/n RTTs, the decrease in the amount of outstanding data due to the flows that have not yet recovered is less than the increase in the amount of outstanding data of the flows that have recovered. Thus, to keep the access link fully utilized, at the moment the last byte drains out of the buffer, the smallest value of this average sending rate, *i.e.*, $nW_a^{(l/n)}/T$, must equal the bandwidth of the access link.

$$nW_a^{(l/n)}/T = C \quad \Rightarrow \quad W_a^{(l/n)} = CT/n \quad (3.8)$$

Substituting the values of W_b and $W_a^{(l/n)}$ from Eqs. (3.7) and (3.8), respectively, into Eq. (3.6) and solving for B , we get the following closed form solution to calculate the buffer size B .

$$B = \frac{CT \left[1 - \Psi - E[\mathcal{N}_0] \left[(1 - \alpha)2^{\frac{l}{n}} - \alpha \right] \right] - \alpha l \text{MSS} E[\mathcal{N}_0]}{\Psi + E[\mathcal{N}_0] \left[(1 - \alpha)2^{\frac{l}{n}} - \alpha \right]} \quad (3.9)$$

where $\Psi = \sum_{d=1}^{l/n} \frac{\binom{l}{d} \left(\frac{1}{n}\right)^d \left(1 - \frac{1}{n}\right)^{l-d}}{2^d}$ and $E[\mathcal{N}_0] = \left(1 - \frac{1}{n}\right)^l$.

3.3.4 Buffer Sizing for Cubic

We have used Reno as a case study for buffer sizing but our derivation method can be adapted to for different congestion control variants. In this section we give an example of how to modify the derivation for a popular congestion control variant called TCP Cubic. The key change lies in recognizing how the congestion window grows and how it reduces on packet losses. For TCP Cubic, the congestion window follows the following semantics:

$$W \rightarrow \beta W \quad (3.10)$$

where $\beta \in (0, 1)$ *i.e.* the window reduces by a factor of β on packet loss. Similarly for window growth we have that:

$$W \rightarrow S(T - K)^3 + W \quad (3.11)$$

where T is RTT for the purpose of our derivation, S is the scaling factor and $K = \left[\frac{\beta W}{S} \right]^{1/3}$. Now that we have the congestion window change semantics, eq.3.6 which relates the congestion window before and after the congestion event can be adapted as follows:

$$W_a^\xi = \xi E[\mathcal{N}_0] [S(T - K)^3 + W_b] + \sum_{d=1}^{\xi} \beta^d W_b E[\mathcal{N}_d] \quad (3.12)$$

Here, $\xi E[\mathcal{N}_0] [S(T - K)^3 + W_b]$ represents the portion of flows that don't experience any packet drops and hence, increase the window size. $\sum_{d=1}^{\xi} \beta^d W_b E[\mathcal{N}_d]$, represents the portion of flows that experience atleast 1 packet drop and upto ξ packet drops and as a result decrease the congestion window size. Borrowing expressions from the previous sections, we have that:

$$\begin{aligned} W_a &= \frac{CT}{n} \\ W_b &= \frac{CT + B}{n} \\ \xi &= \sum_{d=0}^l d E[\mathcal{N}_d] = \frac{l}{n} \\ E[\mathcal{N}_d] &= \binom{l}{d} \left(\frac{1}{n} \right)^d \left(1 - \frac{1}{n} \right)^{l-d} \end{aligned}$$

Substituting these back into eq. 3.12, we get:

$$\begin{aligned} & \frac{l}{n} \left(1 - \frac{1}{n} \right)^l \left[\frac{B + CT}{n} + S \left(T - \left(\frac{\beta(B + CT)}{nS} \right)^{1/3} \right)^3 \right] \\ & + \sum_{d=1}^{\xi} \beta^d \frac{CT + B}{n} \binom{l}{d} \left(\frac{1}{n} \right)^d \left(1 - \frac{1}{n} \right)^{l-d} - \frac{CT}{n} = 0 \end{aligned} \quad (3.13)$$

Unfortunately a closed form solution doesn't exist, therefore eq. 3.13 has to be solved numerically.

3.4 Parameter Selection

To calculate the value of buffer size B using Eq. (3.9), we need the values of the following six parameters: 1) C : bandwidth of the access link, 2) n : number of flows, 3) MSS: maximum segment size, 4) T : average round trip time minus any queuing delays, 5) α : percentage of long flows among the n TCP flows, and 6) l : number of packets dropped by the access router during a congestion event. The value of C is a constant in any given IoT system and the value of n depends on the number of IoT devices and the number of flows each IoT device generates at any given time. Both these values are provided by the network administrator. The value of MSS is defined by TCP. The value of T is dictated by the internet and can be measured by simply timing pings over a few days. Next, we describe how we calculate the values of α and l .

3.4.1 Percentage of Long Flows

Recall from Section 3.2 that IoT devices, especially the energy-constrained ones, go to sleep and terminate their TCP connections if their corresponding application layer processes do not provide a packet to send within τ seconds of providing the previous packet. Let q represent the minimum number of packets that a TCP flow must send to be called a long flow. Thus, any given flow of any given IoT device will be a long flow only if the inter-arrival time between each consecutive pair of its first q packets is less than τ . Let $\mathcal{T}_{ii}$ be the random variable for the inter-arrival time between consecutive packets in any given flow. Let P_{sf} represent the probability that any given flow is a short flow. As α is the probability that any given flow is long flow, $P_{sf} = 1 - \alpha$. Next, we calculate P_{sf} .

A flow is a short flow if the inter-arrival time between any pair of its first q consecutive packets exceeds τ . Thus,

$$P_{sf} = Pr \{ \mathcal{T}_{ii} > \tau \} \sum_{i=0}^{q-2} \left(1 - Pr \{ \mathcal{T}_{ii} > \tau \} \right)^i \quad (3.14)$$

The right hand side of the equation above is essentially the CDF of a geometric distribution with parameter $Pr \{ \mathcal{T}_{ii} > \tau \}$. Using the well-known expression for the CDF of geometric random variables [5], the equation above simplifies into:

$$P_{sf} = 1 - \left(1 - Pr \{ \mathcal{T}_{ii} > \tau \} \right)^{q-1} = 1 - \left(Pr \{ \mathcal{T}_{ii} \leq \tau \} \right)^{q-1} \quad (3.15)$$

As $\alpha = 1 - P_{sf}$, thus $\alpha = \left(Pr \{ \mathcal{T}_{ii} \leq \tau \} \right)^{q-1}$. To calculate α using this equation, we need the values of τ , q , and the parameters and distribution of $\mathcal{T}_{ii}$. The parameters and distribution of $\mathcal{T}_{ii}$ are provided by the IoT system administrator. The administrator knows these because in typical IoT applications, IoT devices either poll sensors periodically at a predetermined rate or adopt event driven polling at predetermined events. Similarly, the value of τ is also

provided by the system administrator because he/she knows the hard-coded rules that the IoT devices follow when going to sleep. Next, we describe how to select a value for q .

A TCP flow goes into congestion avoidance mode as soon as its amount of unacknowledged data, which is equal to the size of the congestion window, crosses the slow start threshold [36]. Next, we calculate the number of packets after which a flow enters the congestion avoidance mode with high probability. For this, we used NS-3 and simulated the topology shown in Figure 3.1 with varying number of IoT devices, access link bandwidth of 1Gbps, one way link-latency between IoT devices and server of 20.02 ms, and MTU of 1500 bytes. The application layer process on each IoT device generated packets with exponentially distributed inter-arrival times with mean of 0.4ms. Each IoT device generated a single flow at a time. We ran our simulator for different number of devices n , and created a log entry for each dropped packet at the output buffer. Next, from the log generated corresponding to each value of n , for all $i \in [1, n]$, we calculated the number of packets q_i of the flow generated by the i^{th} IoT device that were successfully inserted into the buffer by the IoT access router before the first packet of this flow was dropped. The motivation behind using the number of packets before the first drop, *i.e.*, q_i , is that until the first drop, the flow is in slow start mode and q_i is the largest number of unacknowledged packets a flow can have while still in slow start. During the remaining lifetime of the TCP flow, with high probability, the slow start threshold is less than the amount of data contained in q_i packets. Figure 3.5 shows a bar chart where each bar represents the average of all q_i values, *i.e.*, $\sum_{i=1}^n q_i/n$, for each value of n that we used during simulations. We observe from this figure that this average value lies in the range [4, 7]. Therefore, in our evaluations, we select $q = 7$ because it ensures that the majority of flows that have more than 7 packets will have entered the congestion avoidance mode, and can thus be called long flows. To select an appropriate value of q for any arbitrary IoT system, the designer of that IoT system can perform a similar simulation study using the values of C , T , MSS, MTU, n , and the parameters of the distribution of inter arrival times of the packets that are specific to his/her IoT system.

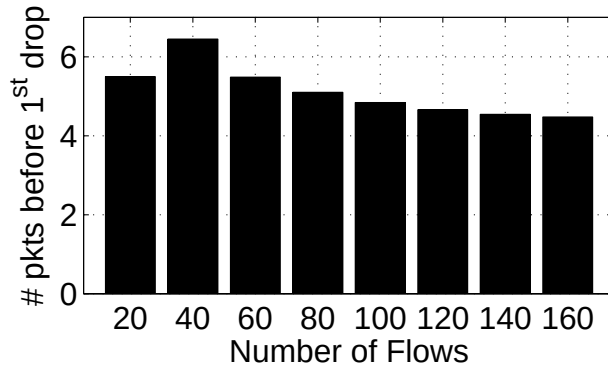


Figure 3.5: Average # of packets successfully inserted into the output buffer before the first drop vs. # of flows

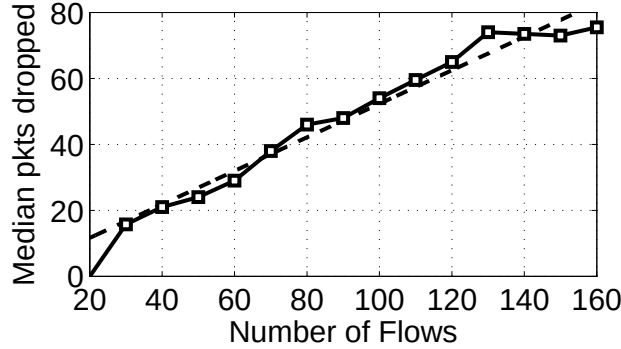


Figure 3.6: Median of the number of packets dropped during each congestion event vs. the number of active flows

3.4.2 Number of Packet Drops in a Congestion Event

To calculate the value of l , we used the logs of the dropped packet at the output buffer of the switch from the simulations described in Section 3.4.1. During those simulations, every time we created a log entry for a dropped packet at the output buffer, we also stored the time-stamp of the packet drop. From the log of time-stamps collected during each run of the simulation, we measured the median value of the number of packets dropped during each congestion event. Figure 3.6 plots the median number of packets dropped during each congestion event for different number of concurrently active flows (*i.e.*, the number of IoT devices sending data simultaneously). We observe from this figure that with the increase in the number of flows, the median number of packets dropped during each congestion event also increases almost linearly. Due to the linear trend, we use a straight line approximation to obtain a relationship between l and n as $l = \xi n$, as shown by the dotted line in Figure 3.6. Based on our empirical observations from these simulations, the value of ξ lies in the range $[0.4, 0.6]$. To select an appropriate value of ξ for any arbitrary IoT system, the administrator/designer of that IoT system can perform a similar simulation study using the values of C , T , MSS , MTU , n , and the parameters of the distribution of inter arrival times of the packets that are specific to his/her IoT system.

Summary: In summary, the IoT administrator directly provides the values of C , MSS , n , and T . To calculate the values of α and l , the IoT administrator/designer performs the two simulation studies described in Sections 3.4.1 and 3.4.2 using the appropriate distribution that the application layer processes in his/her IoT system follow when generating packets.

3.5 Buffer Sizing for UDP Traffic

Buffer sizing for UDP traffic is much simpler than for TCP. The reason being that when a UDP flow suffers a packet loss, it does not pause the sending of packets. Since there is no

period where the senders are explicitly paused, the only time when packets are not being sent is when there is no data for the senders to transmit. Therefore in order to derive the buffer requirements, the worst case scenario needs to be considered where the inter-packet delay of all the senders overlaps producing a window of time where no packet transmission takes place. Once the delay is determined, calculating the buffer size is straightforward *i.e.* the buffer size will simply be:

$$B = Cd \quad (3.16)$$

where:

B : Buffer size

C : Output rate of the router (capacity of access link)

d : Duration in which no packets are transmitted

The value of d to be used here depends on the distribution of the inter-packet delay of packets arriving at the router. Hence, we can use the CDF of the said distribution to calculate the value d such that the probability of seeing that value of d is very low.

$$P = CDF[d, \mu_d, \sigma_d] \quad (3.17)$$

where:

P : probability of seeing a delay value

μ_d : average delay

σ_d : standard deviation of delay

For example, if buffer utilization of atleast 95% is desired, then we can set P to be 0.05 so that we design the buffer for a delay value that is only 5% likely to occur. If the buffer is designed for this 5% value, then for the 95% of the time the delay will be smaller than the calculated value of d . Therefore, for 95% of the time the buffer will not go empty because it has been designed for a larger delay value.

Taking an example of an exponential inter-packet delay distribution, we have that:

$$P = \frac{1}{\mu_d} e^{-d/\mu_d} \quad (3.18)$$

where:

P : probability of seeing a delay value

μ_d : average delay value

d : delay value

Notice that we have actually used the PDF function for exponential distribution and not the CDF. This works because the exponential distribution is a monotonic function. Buffer size calculate using a certain value of P will hold for all value $P' \leq P$. It is very important to note here that the steps that follow can be performed for an arbitrary distribution of choice and that we have used the exponential distribution as an example. However, if the

distribution of choice is not monotonic the CDF must be used as we demonstrate later with the Gaussian distribution. Solving for d we get:

$$d = \mu_d \ln \left[\frac{1}{\mu_d P} \right] \quad (3.19)$$

Plugging this value back into eq. 3.16, we can calculate the buffer size as:

$$B = C \mu_d \ln \left[\frac{1}{\mu_d P} \right] \quad (3.20)$$

As discussed earlier as well, in typical IoT deployments, all IoT devices that connect to a cloud service through a given access router perform the same sensing/actuation tasks and are also often deployed by the same vendor. Consequently, the sizes of their flows follow identical distributions. Hence the flows will be i.i.d. Therefore, if we represent the first moment of the the arrival rate of packets as $E|R|$, then by the strong law of large numbers we will have that the average arrival rate of the combination of inbound flows will, with probability 1, converge to $E|R|$ *i.e.*:

$$\frac{R_1 + R_2 + \dots + R_n}{n} \rightarrow E|R| \quad (3.21)$$

The average delay of the combined distribution will then be $\mu_d = \frac{1}{E|R|}$ and we know from the i.i.d phenomena that $\mu_d^* = \frac{1}{E|R|}$ where μ_d^* is the average inter-packet delay of a single flow. Hence we have that:

$$\mu_d^* = \mu_d$$

Substituting back into 3.20 we get:

$$B = C \mu_d^* \ln \left[\frac{1}{\mu_d^* P} \right] \quad (3.22)$$

If the same process were to be repeated using Gaussian distribution, we would have the following:

$$P = \frac{1}{2} \left[1 + \operatorname{erf} \left(\frac{d - \mu_d}{\sigma_d \sqrt{2}} \right) \right] \quad (3.23)$$

where $\operatorname{erf}(x)$ denotes the Gaussian error function.

Solving for d we get:

$$d = \mu_d + \sqrt{2} \sigma_d \operatorname{erf}^{-1}(2P - 1) \quad (3.24)$$

where $\operatorname{erf}^{-1}(x)$ denotes the inverse error function and is defined in the interval $x \in [-1, 1]$ which is alright because $P \in [0, 1]$ and $(2P - 1) \in [-1, 1]$.

Plugging this d back into eq. 3.16 we get:

$$B = C \left[\mu_d + \sqrt{2} \sigma_d \operatorname{erf}^{-1}(2P - 1) \right] \quad (3.25)$$

Now, we know that if $X_1, X_2, \dots, X_n$ are mutually independent normal random variables with means $\mu_1, \mu_2, \dots, \mu_n$ and variances $\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$, then the linear combination:

$$X^* = \sum_{i=1}^n a_i X_i$$

will follow a normal distribution $\mathcal{N}(\mu_d^*, (\sigma_d^*)^2)$ such that:

$$\begin{aligned} \mu_d^* &= \sum_{i=1}^n c_i \mu_i \\ (\sigma_d^*)^2 &= \sum_{i=1}^n c_i^2 \sigma_i^2 \end{aligned}$$

Again, using the fact that IoT flows follow similar distribution *i.e.* $[\mu_i = \mu_j = \mu_d \ \sigma_i = \sigma_j = \sigma_d \ \forall i, j]$, the above equations become:

$$\mu_d^* = \sum_{i=1}^n c_i \mu_i = n \mu_d \quad \Rightarrow \quad \mu_d = \frac{\mu_d^*}{n} \quad (3.26)$$

$$(\sigma_d^*)^2 = \sum_{i=1}^n c_i^2 \sigma_i^2 = n \sigma_d^2 \quad \Rightarrow \quad \sigma_d = \frac{\sigma_d^*}{\sqrt{n}} \quad (3.27)$$

Plugging these back into eq. we get the final expression as:

$$B = C \left[\frac{\mu_d^*}{n} + \sqrt{\frac{2}{n}} \sigma_d^* \text{erf}^{-1}(2P - 1) \right] \quad (3.28)$$

3.5.1 Buffer Sizing for TCP and UDP combined

In realistic scenarios where the traffic contains from both TCP and UDP flows, the buffer can be sized using the two schemes derived above separately. If high buffer utilization is the only goal then using the fact that 1) UDP flows don't pause transmission of packets after experiencing a loss and, 2) the only delay is the one from the inter-packet delay distribution, the buffer only needs to be sized for TCP flows. If the buffer is sized for TCP flows then it will have been designed to not go empty for a window of time that is longer than what the inter-packet delay distribution of UDP flows is likely to produce. However, this argument holds only under the assumption that delay from inter-packet packet delay distribution d_{udp} is much smaller than the time it takes for TCP sender to recover from packet loss and start transmission of packets again. In case of absence of this assumption and having only high buffer utilization as the design goal, a solution would be to have the following:

$$B = \text{MAX}(B_{\text{TCP}}, B_{\text{UDP}}) \quad (3.29)$$

where:

B_{TCP} : Buffer size calculated from 3.9 using $n = f_{\text{TCP}}N$

B_{UDP} : Buffer size calculated from 3.22 or 3.28 using $n = f_{\text{UDP}}N$

N : Total number of flows inbound to the link.

f_{TCP} : fraction of total flows that are TCP.

f_{UDP} : fraction of total flows that are UDP.

However, this scheme will probably result in excessive packet loss for UDP flows because the buffer has been designed to cater to only a fraction of the flows. If only B_{TCP} is used to size the buffer, then the buffer has been sized to handle only $f_{\text{TCP}}N$ flow to which the remaining portion $f_{\text{UDP}}n$ are excessive. The UDP flow do not pause their packet transmission and hence, will make the buffer overflow almost all of the time. The same will be observed when only B_{UDP} is used to size the buffer. The UDP flows will experience high packet loss. A simple and elegant solution to this is to use both formulas and use the aggregate buffer size as follow:

$$B = B_{\text{TCP}} + B_{\text{UDP}} \quad (3.30)$$

where:

B_{TCP} : Buffer size calculated from 3.9 using $n = f_{\text{TCP}}N$

B_{UDP} : Buffer size calculated from 3.22 or 3.28 using $n = f_{\text{UDP}}N$

This will ensure that the buffer is large enough such that the UDP flows cannot overflow it and that TCP flows will get a share of the buffer space.

3.6 Experimental Evaluation

In this section, we present the experimental evaluation of our buffer sizing scheme and its comparison with prior schemes. While no prior work exists on calculating buffer sizes for IoT access routers, we still compare our buffer sizing scheme with the schemes proposed by Appenzeller *et al.* [12], Dhamdhere *et al.* [20], and Gorinsky *et al.* [26]. Next, we first empirically present the effect of the buffer size on the access link utilization and on the latency experienced by the packets of IoT devices in travelling to the server. After that, we compare the buffer sizes calculated by our scheme using Eq. (3.9) and by the schemes proposed in [12], [20], and [26]. Finally, we present and compare the access link utilization, latency experienced by packets in travelling from IoT devices to the server, packet loss rate, and good put achieved by IoT devices in various scenarios resulting from the buffer sizes calculated by our scheme and by the three prior schemes.

Due to the lack of any real-world benchmark IoT traffic traces that would be suitable for our evaluations, we performed our experiments through NS-3 simulations. In each experiment, we simulated the topology shown in Figure 3.1 for three minutes of simulated time with different number of IoT devices using access link bandwidth of 40Mbps, MTU of 1500 bytes,

one way latency from the IoT devices to the access router of $20\mu s$, and one way latency from the access router to the server of $20ms$. Note that the queueing delay experienced by any given packet inside the access router is dictated by how full the output buffer is at the time the packet entered the output buffer. The application layer process on each simulated IoT device generated packets with exponentially distributed inter-arrival times, where the rate R at which the process sent packets to the TCP layer was different in different experiments. To produce traffic of γ Mbps on the physical link from an IoT device to the access router, the application layer process sends messages of size $MTU - 40$ bytes to the TCP layer at the rate of $R = \lceil \gamma / MTU \rceil$. The 40 byte difference between the size of the message sent by the application layer and the MTU is to accommodate the 40 byte TCP/IP headers. We will mention the value of γ for each experiment as we describe that experiment. Each IoT device generated a single flow at a time.

3.6.1 Buffer Size vs. Link Utilization and Latency

To empirically study how the buffer size affects the access link utilization and the latency experienced by the packets of IoT devices, we ran our NS-3 simulations using $n = 40$ IoT devices, where each IoT device produced traffic of $\gamma = 1$ Mbps on the physical link. Thus, the aggregate traffic arriving at the access router was approximately 40Mbps, which is equal to the bandwidth of the access link. We ran our simulations multiple times, where in each simulation, we used a different buffer size in the range $[2 \times MTU, 2 \times CT/\sqrt{n}]$, where $CT/\sqrt{n}$ is the buffer size proposed in [12]. Figure 3.7 plots the average utilization of the access

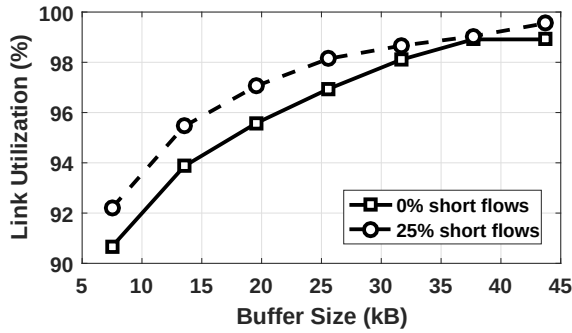


Figure 3.7: Buffer size vs. Utilization

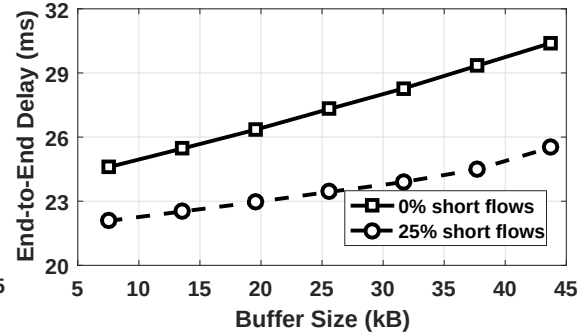


Figure 3.8: Buffer size vs. Delay

link for different buffer sizes when all flows are long and also when 25% of the flows are short. We observe that as the buffer size increase, the link utilization increases because the buffer holds more data and thus goes empty less frequently when some IoT devices pause after experiencing a packet drop. From this observation, one might conclude that to keep the link utilization high, one could simply over-provision the output buffer. This would be problematic, especially for the latency sensitive applications, because the increase in buffer

size increases the queuing delay experienced by the packets. Figure 3.8 plots the average latency experienced by all packets in travelling from the IoT devices to the server during the experiments from which we obtained Figure 3.7. We observe from this figure that as the buffer size increases, the latency experienced by packets also increases.

3.6.2 Comparison of Buffer Sizes

Next, we compare the buffer size calculated by our scheme using Eq. (3.9) with the buffer sizes calculated by [12],[20] and [26]. Figures 3.9, 3.10 and 3.11 plot the buffer sizes for number of flows ranging from $n = 10$ to 100 for three different IoT traffics, one containing 0% short flows, the second containing 15% short flows and the last one containing 30% short flows respectively, respectively. We observe from these three figures that as the percentage of short flows in the IoT traffic changes, the buffer sizes calculated by [12] and [20] change because both [12] and [20] calculate the buffer size based on only the number of long flows in the traffic. Contrary to [12] and [20], the buffer size calculated by [26] does not change as the percentage of short flows in the IoT traffic changes because it calculates the buffer size based on the total number of flows, irrespective of whether the flows are long or short.

We make three additional and important observations from Figures 3.9, 3.10 and 3.11. First, as the number of IoT devices increases, the required buffer sizes calculated by Eq. (3.9) as well as by [12] and [20] decrease. This happens because with the larger number of IoT devices, when TCP flows of some devices pause after experiencing packet drops, there is a higher probability that at any given time instant, TCP flows of some other IoT devices will be sending data to the IoT access router and keeping the link utilized. Note that after a certain value of the number of flows, [20] starts increasing the buffer size linearly. This behavior is due the piecewise definition of the model in [20] which after a certain number of flows, shifts the goal from maximization of link utilization to minimization of loss-rate. The buffer size calculated by [26] shows exactly the opposite behavior compared to other schemes as it linearly increases the buffer size with increase in the total number of flows to keep the packet drop rate low. Second, as the percentage of short flows increases, the buffer size calculated by our scheme decreases because when a short flow experiences a packet drop, due to smaller congestion window size, it pauses for a shorter duration compared to when a long flow experiences a packet drop. Therefore, short flows resume sending their packets more quickly compared to long flows, and a smaller buffer size suffices. Note that in conventional networks, the duration of pause depends on both RTT and congestion window size. However, in the case of IoT systems, the RTTs of all IoT devices in a given IoT system are very similar, and therefore, the duration of pause for any given TCP flow is primarily dictated by its congestion window size. Third, the buffer sizes calculated by [12], [20] and [26] in the presence of short flows are significantly larger compared to the buffer sizes calculated by Eq. (3.9), because [12] and [20] consider only the number of long incoming flows to calculate the buffer size and thus, *implicitly* assume that all flows will pause for longer durations, which leads to over-provisioning the buffer. Figures 3.13 and 3.14 plot the ratio of the buffer size calculated

by each prior scheme with the buffer size calculated by our scheme. We observe from this figure that prior schemes over provision the buffer sizes by several times in comparison to the buffer sizes calculated by our scheme. This over-provisioning implies that [12], [20], [26] can keep the access link utilization higher compared to our scheme, but at the cost of a significantly larger latency, as we empirically demonstrate in the next section.

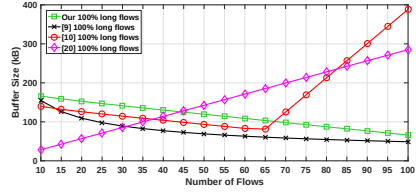


Figure 3.9: Buffer size for 0% short flows

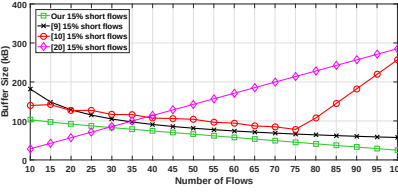


Figure 3.10: Buffer size for 15% short flows

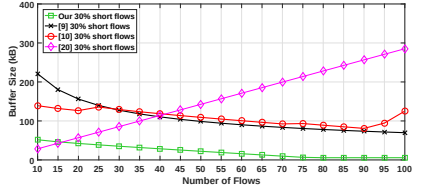


Figure 3.11: Buffer size for 30% short flows

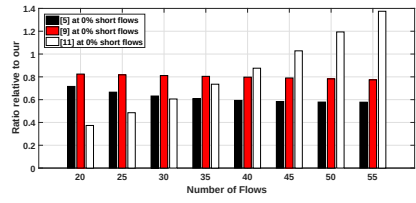


Figure 3.12: Buffer size comparison 0% short flows

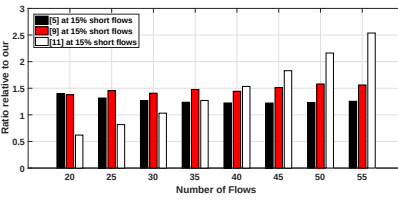


Figure 3.13: Buffer size comparison 15% short flows

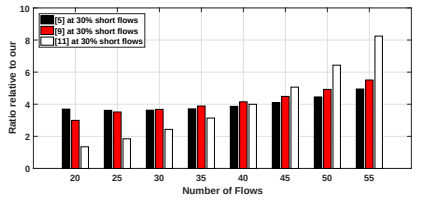


Figure 3.14: Buffer size comparison for 30% short flows

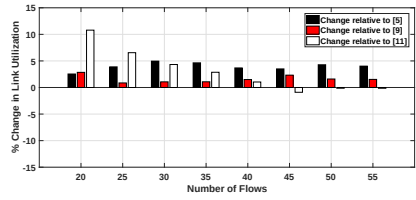


Figure 3.15: Link Utilization for 0% short flows

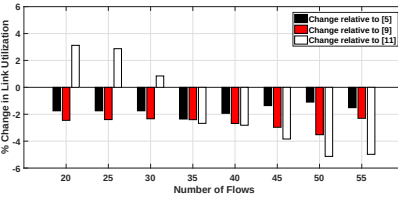


Figure 3.16: Link Utilization for 15% short flows

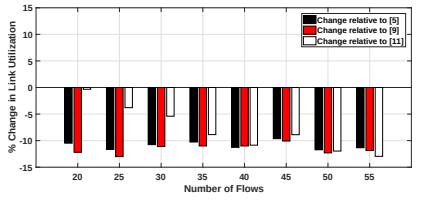


Figure 3.17: Link Utilization for 30% short flows

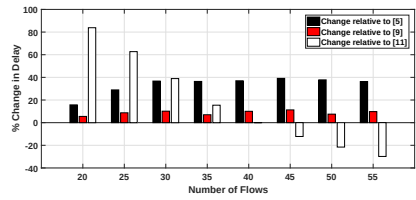


Figure 3.18: Queuing Delay for 0% short flows

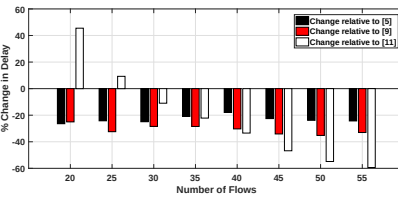


Figure 3.19: Queuing Delay for 15% short flows

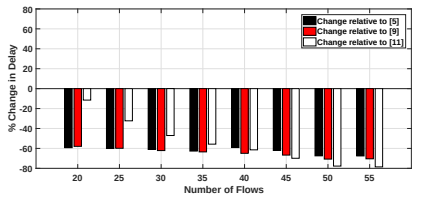


Figure 3.20: Queuing Delay for 30% short flows

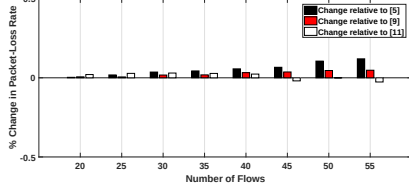


Figure 3.21: Link Loss for 0% short flows

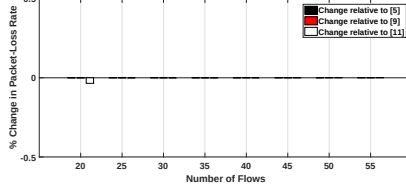


Figure 3.22: Link Loss for 15% short flows

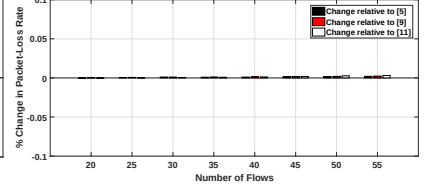


Figure 3.23: Link Loss for 30% short flows

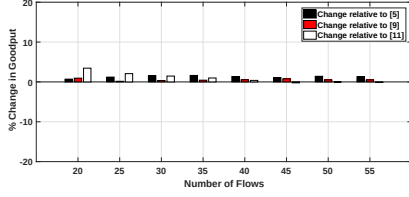


Figure 3.24: Link Goodput for 0% short flows

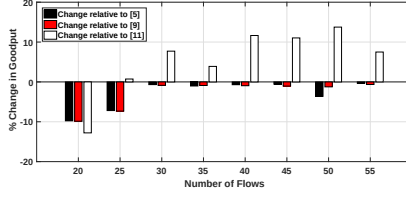


Figure 3.25: Link Goodput for 15% short flows

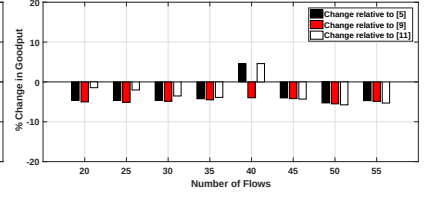


Figure 3.26: Link Goodput for 30% short flows

3.6.3 Comparison of Link Utilization, Latency, Loss, & Goodput

Next, we compare link utilization, latency, packet loss rate, and goodput resulting from the buffer sizes calculated using Eq. (3.9) and those calculated by [12], [20] and [26]. For this, we performed NS-3 simulations where each IoT device produced traffic of $\gamma = \frac{40}{n}$ Mbps on the physical link, where n represents the total number of flows. Hence the link was congested, regardless of the number of flows. We varied the number of IoT devices from 20 to 55 and generated three different sets of IoT traffics, one containing no short flows, the second containing 15% short flows and the last one containing 30% short flows. The later two sets of IoT traffic are more important because IoT device are more likely to generate short flows due to their energy constraints.

Figure 3.15 plots the percentage change in the average link utilization resulting from the buffer size calculated by our scheme compared to prior schemes for IoT traffic that contained no short flows. We observe from this figure that the link utilizations resulting from the buffer sizes calculated by [12], [20] and [26] are slightly lower compared to our scheme because the buffer sizes calculated by our scheme are slightly larger than prior schemes when all flows in the traffic are long flows. Figures 3.16 and 3.17 plot the percentage change in the average link utilization for IoT traffic that contained 15% short flows and 30% short flows, respectively. As prior schemes significantly over estimate the buffer sizes in the presence of short flows, the link utilizations resulting from the buffer sizes calculated by prior schemes are slightly higher compared to our scheme.

While prior schemes achieve higher link utilization for IoT traffic that contained 15% short flows or 30% short flows, this higher utilization comes at the cost of significantly higher latency. Figures 3.18, 3.19 and 3.20 plot the percentage change in average queueing delay

experienced by the packets when using the buffer size calculated by our scheme compared to when using the buffer size calculated by prior schemes. We observe from these figures that for the IoT traffic with 15% and 30% short flows, which are more representative IoT traffic, the buffer sizes calculated by prior schemes increase the queuing delay by an average of 25% and 60%, respectively, compared to our scheme. Such large queuing delays deteriorate the performance of realtime and streaming IoT applications.

Finally if we look at Figures (3.21, 3.22, 3.23) and (3.24, 3.25, 3.26), which plot the change in the packet loss rate and goodputs, respectively, resulting from the buffer sizes calculated by our scheme compared to the prior schemes, we see that the packet loss rates and goodputs of all schemes are almost the same. Thus, we conclude that our scheme reduces the buffer size significantly, which leads to 25% to 60% lower queueing delay for IoT traffic with negligible impact on the remaining three performance metrics, *i.e.*, link utilization, loss rate, & goodput.

3.6.4 UDP only Traffic

Moving onto the second scenario of UDP only traffic, the experiments follow the same topology as before *i.e.*, we have performed NS-3 simulations where each IoT device produced traffic of $\gamma = \frac{40}{n}$ Mbps on the physical link, where n represents the total number of flows. Hence, the link was congested, regardless of the number of flows and we varied the number of IoT devices from 20 to 55. The minimum buffer size calculated by eq. 3.22 for $P = 0.05$ and the same traffic conditions as the ones used in section 3.6.3, comes out to be very small because the inter-packet delay is tiny *i.e.* 0.08 packets or 120 Bytes. Since the Buffer size was very small we have used $B = \min(4MSS, B_{UDP})$ where B_{UDP} is the buffer size calculated from eq. 3.22. $4MSS$ turns out to be greater and hence is the buffer size value used.

Figure 3.27 plots the percentage link utilization versus the number of flows used in the experiment. It can be observed that the average value across all flows is hovering around 91%. Since UDP flows do not stop upon experiencing a packet drop, it is reasonable to expect the utilization to be high regardless of the number of flows inbound on the link. Consistently high values in Figure 3.27 confirm this intuition. Figures 3.28, 3.29 and 3.30 plot the queuing delay, loss rate and aggregate goodput values against the number of flows used in the experiment, respectively. Observe from Figure 3.28 that the queuing delay remains low *i.e.* less than 2ms and is decreasing as the number of flows increase. The reason is that the buffer size is tiny *i.e.* the smaller the buffer the less time it takes for a packet to enter the queue and then exit it. Another positive observation can be made from Figure 3.30, the aggregate goodput is close to the total link capacity of 40 Mbps. So while delay is decreasing, the goodput and link utilization remain high which indicates good performance. However, a downside of focusing just on utilization is that the small buffer size leads to high loss rates at the router buffer *i.e.* above 2% as plotted in Figure 3.29. The trend is increasing as the the number of flows increase. While it may not be an issue for applications that deal with non-sensitive data or bulk data transfer, it will most certainly be a problem for interactive

applications and application that have sensitive data *e.g.* medical IoT devices. This can be mitigated by using a large buffer, for example, we could deploy a flow propotional scheme that increases the buffer size as the number of flows increases similar to one one proposed by Gorinsky *et al.* [26]. Lower loss rate though, would come at the cost of higher delay due to large buffer size. This is a trade off that is inevitable. Hence, we repeat here again that buffer sizing for UDP flows is not as complicated as for TCP if link utilization is the only metric being optimized.

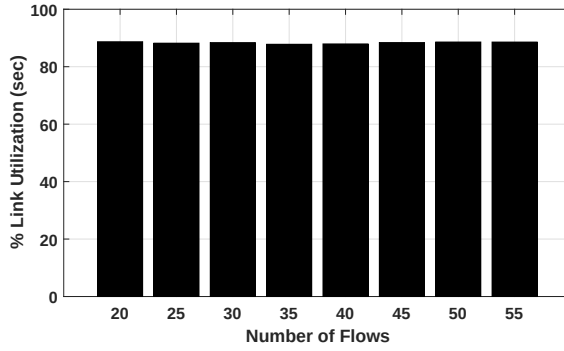


Figure 3.27: % Link Utilization for UDP flows

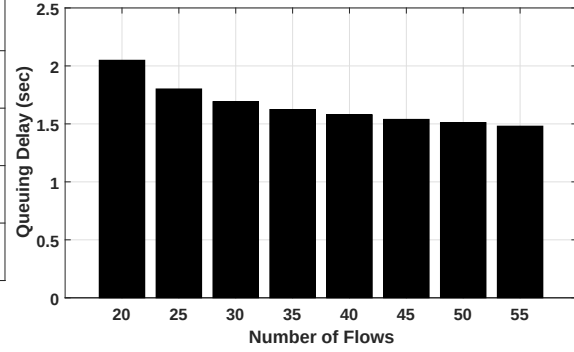


Figure 3.28: Queuing Delay for UDP flows

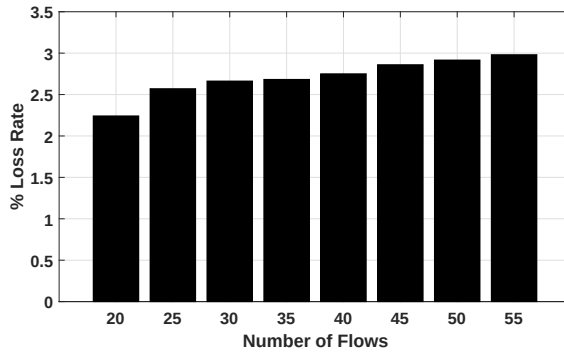


Figure 3.29: % Link Loss for UDP flows

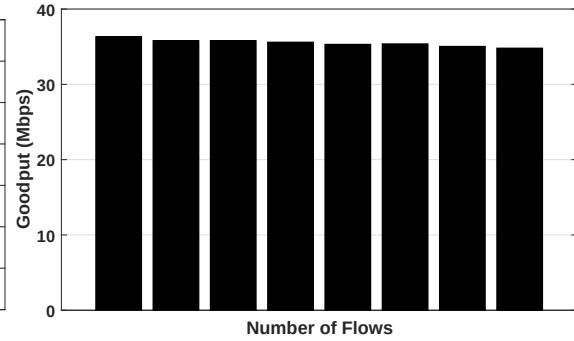


Figure 3.30: Aggregate Goodput for UDP flows

3.6.5 TCP+UDP Traffic

Finally, moving onto the last and most realistic scenario, we now inject both TCP and UDP flows into the network. We have used a slightly modified version of the topology used in section 3.6.3. The access link capacity is still fixed at 40Mbps and each flow still generates $\frac{40}{n}$ Mbps of traffic regardless of whether it is a TCP or UDP flow. However, this time the

	0%	15%	30%
80:20	141604	82424	33969
50:50	157806	95697	44775
20:80	184259	119804	66645

Table 3.1: Buffer sizes for mixed traffic experiments

number of total flows is fixed at 40 and the ratio of TCP to UDP flows is varied. We have simulated three flow mixes 1) 80% TCP flow and 20% UDP flows, 2) 50% TCP flow and 50% UDP flows, and 3) 20% TCP flow and 80% UDP flows. We have also tested these three flows mixes for three percentages of TCP short flows *i.e.* 0%, 15% and 30% short flows in TCP traffic. This simply means that for each traffic mix *e.g.* the 80% TCP, 20% UDP mix we have performed three experiments in which 0%, 15% and 30% of the TCP flow were short flows. Table 3.1 details the buffer sizes used in the experiments. Each row indicates a different flow mix *e.g.* the 80:20 row indicates that 80% of the 40 flows are TCP and 20% of them are UDP flows. Each column indicates the percentage of short flows in the TCP flows *e.g.* the 15% means the 15% of all TCP flows are short.

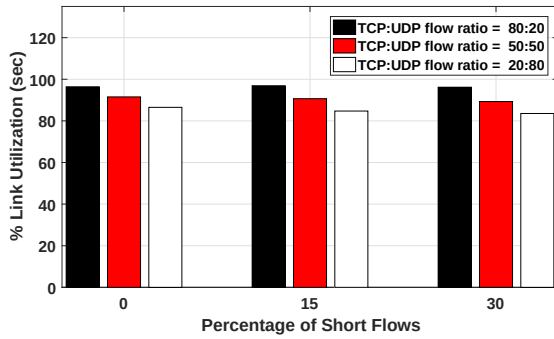


Figure 3.31: % Link Utilization for mixed flows

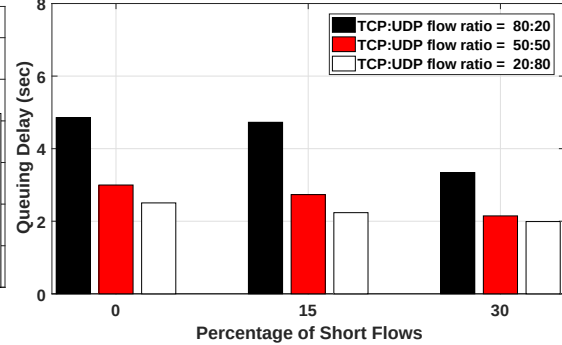


Figure 3.32: Queuing Delay for mixed flows

Figure 3.31 plots the percentage link utilization for both the different traffic mixes and short flow percentages. It can be seen that the link utilization value are very high (above 80%) regardless of the flow mix or short flow percentage of TCP traffic. Infact utilization numbers are higher for mixed traffic because while the TCP flows try to avoid congesting the link, UDP flows do not stop transmission of packets upon packet loss. This shortens the duration in which there are no packets inbound to the link. The only moments when the router buffer goes empty is when the router sends out packets from (empties) the buffer faster than the packets are coming in. Queuing delay is plotted in Figure 3.32. It can be seen that for 0% short flows, the queuing delay is higher and the reason is that our scheme realizes that when there are no short flows, the average congestion window will be larger before a congestion event and so the pause of transmission of packets will be proportionally larger as well. Therefore,

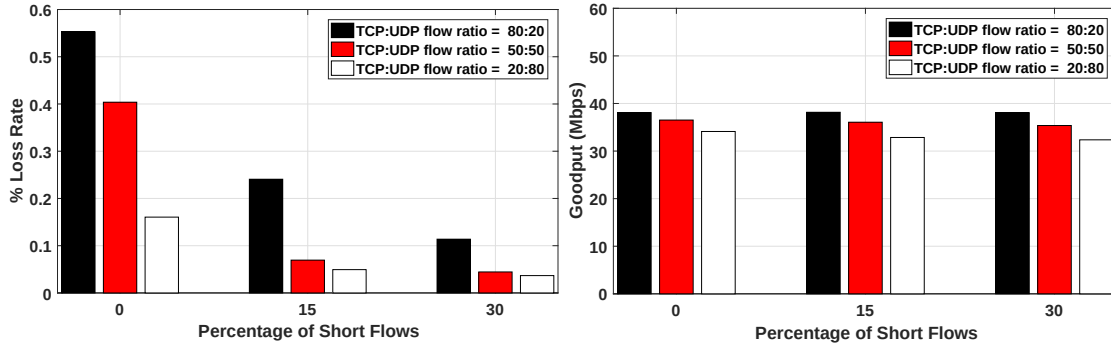


Figure 3.33: % Link Loss for mixed flows Figure 3.34: Goodput for mixed flows

a larger buffer is required in order to avoid it going empty. The bigger the buffer the longer it takes for a packet to enter and then exit the queue and hence higher the queuing delay. This requirement reduces as the number of short flows increases which is why the delay decreases as well. Furthermore, as the percentage of TCP flows reduces, delay reduces and the reason is that the buffers become smaller as detailed in Table 3.1. Also notice that the loss rates reported in Figure 3.33 are lower than those reported for UDP only Figure 3.29. The reason again is the larger buffer sizes. The larger size ensures that the buffer is always entirely full *i.e.* most of the time there is space to accommodate new inbound packets into the buffer. Finally, observe from Figure 3.34 that the aggregate goodput of the TCP and UDP flows combined is near link capacity of 40 Mbps indicating good performance.

3.7 Chapter Summary

In this chapter, we have presented a model to calculate the size of the output buffer for IoT access routers. The key technical depth of this work is in the theoretical modeling of the size of the TCP congestion window after the congestion event, and its use in calculating the output buffer size. We have identified three key assumptions of prior art and shown through both real world experiments and NS-3 simulations that they do not hold true in IoT traffic. We have presented the experimental evaluation of our buffer sizing scheme and its extensive comparison with the state of the art schemes proposed in [12], [20] and [26]. Our results show that for IoT traffic, [12], [20] and [26] over-estimate the buffer size by at least two times the required value, due to which, IoT flows experience over at least 60% higher queuing delay compared to when using the buffer size calculated by our scheme. For UDP only traffic our scheme achieves at least 91% link utilization with very low queuing delays and aggregate goodput that is approx. 90% of link capacity. Finally, for mixed traffic scenarios we show that our scheme performs better than either TCP only or UDP only traffic scenarios achieving higher link utilization than either as well as low delays, low loss-rates and aggregate goodput that is approx 94% of link capacity.

Chapter 4

Characterizing TCP in the Cloud for Cloud Tenants

4.1 Introduction

Over the recent years cloud computing has become the defacto paradigm for fulfilling the computing needs of a myriad of applications ranging from streaming services to learning management systems like Canvas from Instructure. It has also been highly successful at providing on-demand compute infrastructure for applications like graph processing and deep learning [48]. Several popular cloud providers, for exmaple Amazon Web Services (AWS), offer customized machine learning and deep learning services to train, test and deploy models at scale in the cloud [49][25][35]. Furthermore, with the emergence and the subsequent explosive growth of Internet of Things (IoT), the importance of cloud has become more concrete. The reason being that while IoT devices gather data from sensors, they mostly do not process it on device, rather they forward it to one or more applications which are often deployed in the cloud. The cloud applications in turn, after processing the received data, send appropriate instructions to the IoT devices, *e.g.* changing states of actuators *etc..* Hence the cloud is arguably the brain of IoT. Since sending data over networks is a crucial aspect in all of these transactions, it is important that the protocols being used to do so are chosen carefully. If the protocol is not chosen through careful consideration, it may result in increased latency and reduced throughput, as we show in results later, both of which would result in loss of profits.

We use the following two terms throughout the chapter: *cloud tenants* and *customer*. We define *cloud tenant* as people and/or organizations that use cloud for getting infrastructure and have one or more applications deployed on VMs with which customers interact. We define *customers* as people who interact with and use the applications deployed by cloud tenants in the cloud. When *cloud tenants* buy Virtual Machines (VM) *e.g.* Elastic Compute

Cloud (EC2) instances from a cloud service provider like AWS, they have little control over where the instances are placed and almost no control over how they are networked to one another. The EC2 instances may end up being hosted on the same physical machine, or on different machines in the same rack, or on machines in different racks. The network topology would probably be the simplest when the EC2 instances are hosted on the same host machine and in this case they should be a single hop away from one another. However in the latter two cases, the machines may be single hop or more away from one another. Furthermore, the network between two or more pairs of instances requested and allocated at the same time, may or may not be homogeneous, *i.e.* that not all machines would be the same number of hops away from one another. Some may be a single hop away from a given machine while others may take several hops to reach that same machine. The traffic generated between the said instances passes through network equipment having varying capabilities such as switch bandwidth, Explicit Congestion Notification (ECN) capability etc., which again are unknown to the tenant. This gap in knowledge of the network topology and its capabilities makes it tough to choose one protocol over another. TCP is the most popular protocol in the cloud because of its adaptability to network variability and accounts for more than 90% of the traffic [31]. Therefore, the question then becomes which TCP variant to choose? and how to choose it?. Choosing the correct TCP is important because it impacts the throughput, round trip time (RTT) and response time of the applications deployed by the tenants. All of these metrics, especially response time, directly impact user experience of the application and hence, have an effect on the popularity thereof. Correct choice of TCP may also lead to reduction in the overall amount of network traffic sent between the VMs, which in turn would result in reduction in revenue spent on network communication and hence, possibly, increase profits.

In this work we have focused on profiling TCP from the cloud tenants perspective while treating the network inside the cloud as a black box as it is for the tenants. We have conducted two sets of tests, one at the transport layer and the other at the application layer and have drawn inferences and conclusions. The results should help guide the tenants in making a better choice of TCP congestion control based on their needs. For the transport layer metrics we have focused on the throughput and RTT characteristics. In particular we have analyzed trends in throughput and RTT over time within experiment runs as well as across experiment runs. We have used a popular traffic generation tool called iPerf to generate traffic between VMs and `tcp-probe` [24][34] to collect packet level traces. Throughput and RTT were then calculated using these packet level traces. Our results show the existence of a piecewise trend in both throughput and RTT that is exponential in the first phase and linear in the second phase. While all TCP variants behave in a very similar manner in the exponential phase, they show difference in performance metrics in the steady state phase. For example, TCP Vegas outperforms all three variants in terms of both average throughput and average RTT. While the difference in throughput is small, the RTT of TCP Vegas is at least 9 times lower than the other three variants. We have also observed that there is no difference in average throughput and average RTT by time of day *i.e.* regardless of the time the AWS network offers similar performance to the VMs.

For the application layer experiments, we have used HTTP as a case study because it operates on top of TCP/IP. We have used Apache Jmeter for generating HTTP GET requests for files of different sizes ranging from 10K Bytes to 1G Bytes. Again, we have used `tcp-probe` to collect packet level traces for analysis. However, for this set of experiments we have also studied response time. We define response time to be the time difference between when a client sends a request to a server and when the request is completed. Again, we have analyzed the over time trends in throughput, RTT and response time within an experiment as well across experiments. We have also studied the effects of file size on the three metrics. Our results highlight an interesting difference between the transport layer and the application layer trends over time *i.e.* the piecewise behavior of the throughput and the RTT was not observed in the application layer experiments. Another interesting observation was the difference in throughput between a client that was pushing requests at the maximum possible rate and one that was generating requests based on a delay distribution. The latter achieved a lower throughput than the former even though the delay distribution for the latter was set such that both the scenarios should have the same average throughput. Results also show that for file sizes less than or equal to 10K Bytes, TCP Reno achieves a marginally higher throughput while maintaining similar response time as compared to the other variants. Another observation is that while TCP Vegas maintains similar throughput as compared to the other variants, the response time is higher (worse) for all file sizes. Throughput and RTT stability within an experiment run also decreases as the file size increases for all TCP variants.

We plan on releasing our network traffic data comprising of more than 133 hours of TCP traffic and 36 hours of HTTP traffic collected on Amazon’s public cloud AWS.

4.2 Data Collection

4.2.1 Choice of platform

The provider we have chosen is Amazon Web Services (AWS) The reason being that when cloud tenants need to deploy their application, they require Infrastructure as a service (IaaS), *i.e.* they require access to virtual machines. AWS provides IaaS and is the biggest shareholder *i.e.* 35% of the market share, in the public cloud market [28]. Since, we want to study the blackbox network, all other factors that may become a bottle neck need to be abstracted out. Hence, adequate CPU, Memory and IO performance is required. To this end we have used t2.large instances [50] in all of our experiments. These instances are provisioned with 2 vCPUs and 8 GB of RAM and therefore, are an ideal choice so as to not introduce any artifacts into TCP behavior.

4.2.2 TCP Performance Logging

For micro benchmarks we were primarily interested in instrumenting behavior of TCP itself and not the application. Hence, we have logged TCP at the transport layer in the Linux kernel. To this end, we have made use of a Linux module called `tcp-probe` [24] [34], which uses the linux k-probe. It, inserts a hook into the TCP receive path and logs data at the packet level. More concretely, the hook is inserted in `tcp_rcv_established()` in `/net/ipv4/tcp_input.c`. Since the logging takes place in the receive path, we naturally insert the probe into the source machines which are generating the traffic. The advantage of logging at the traffic source is that it allows for accurate monitoring of the congestion window and sender round trip time (SRTT) directly and simultaneously. Effective Bandwidth can be calculated as well, though it requires non-trivial processing of `tp → snd_nxt` i.e. next sequence number to be sent and `tp → snd_una` i.e. the last unacknowledged sequence number. Here, `tp` refers to Linux struct `tcp_sock` which is constructed from the `sock` struct.

An alternative method which we tried is through `printk()` statements inserted into the kernel itself. Specifically, we tried to extract fields from the `sock`, `tcp_sock` and `tcphdr` structs being used inside the `tcp_transmit_skb()` function in `net/ipv4/tcp_output.c` and the `tcp_event_data_rcv()` function in `net/ipv4/tcp_input.c`. However, to get out any useful amount of information, the `printk` ring buffer size has to be increased significantly and this method requires a kernel recompile. Though, this method can log at all source and destination ports for all source and destination IPs which `tcp-probe` cannot, `tcp-probe` is the easier one of the two to use when logging at a given port. Another drawback is that `printk` generates a large overhead which introduces measurement bias. Therefore, we have used `tcp-probe`.

4.2.3 Why only TCP?

In this work we focus on TCP traffic only due to multiple reasons. First, TCP applications (HTTP and HTTPS) are very dominant in cloud data center accounting for more than 90% of cloud traffic [31]. Second, the uncertainty of delay and BDP, different architectures, and virtualization has led many researchers to come up with different variants of TCP with different congestion control mechanism. Hence, The selection of correct TCP variant is important for the cloud tenants in order to optimize network performance for their application. Finally, UDP being a best effort protocol does not offer control to the tenants to dynamically adapt to the network variability.

4.2.4 Choice of TCP Variants

In this study we are focusing on four variants of TCP 1) Cubic, 2) Reno, 3) DCTCP, and 4) Vegas. The reasons for the choice of these variants are their design characteristics and

popularity. In terms of design there are three distinct approaches that TCP algorithms use to estimate congestion in the network 1) loss based schemes which use duplicate ACKs to detect losses and estimate congestion in the network, 2) delay based schemes which use estimates of throughput to estimate congestion, and 3) Explicit congestion notification (ECN) packet capable algorithms rely on explicit feedback from the network hardware. Based on this, Cubic was chosen because it uses a loss based scheme and is known to have been optimized for high bandwidth-delay product. Another reason is that it is the default congestion control variant in Linux. Reno is the classic additive increase multiplicative decrease (AIMD) congestion control algorithm that has been very widely used and is also a loss based scheme. The difference in Reno and Cubic lies in the growth and reduction of the congestion window size. DCTCP is a relatively newer variant and it is of the type that relies on explicit feedback from the network hardware instead of perceived losses to estimate congestion. Furthermore it has been optimized for data center environments and is popular because of its good handling of the incast problem in data centers. Vegas unlike the other three is delay based TCP and relies on the difference between estimated throughput and actual throughput, both of which are based on estimates of RTT. Its goal is to keep the delay as low as possible and hence is optimized for keeping RTT low.

4.3 iPerf Bechmarking

This section presents the transport layer experiments. The aim is to estimate the capability/limits of the the AWS network and understand the bandwidth-delay characteristics of the TCP variants. Since the network under question in AWS is unknown, the first thing we did was to measure the best possible performance that could be derived from it. Here performance refers to the maximum and average throughput as well as the minimum and average RTT that the network offers. Hence, the two questions which would help estimate this performance are 1) what are the throughput characteristics of different TCP variants?, and 2) what are the RTT characteristics of different TCP variants? In particular we will look into the trends in throughput and RTT over time, average value over an experiment run and the variation of the average values between runs. We will also inspect the variation and stability of the throughput and RTT over an experiment run.

An important point to note here is that our intention is to study the effect of the TCP variants under isolation from application behavior or application induced artifacts. Therefore inorder to abstract the application, we have used iPerf which is a popular TCP and UDP traffic generation tool. The reason why application behavior and application induced artifacts are abstracted is because, we have used iPerf to generate traffic at the maximum possible rate. Hence, instead of workloads with certain traffic characteristics, iPerf was saturating the available bandwidth. The topology used and experiment design are explained in the following subsection.

4.3.1 Experiment Design and Topology

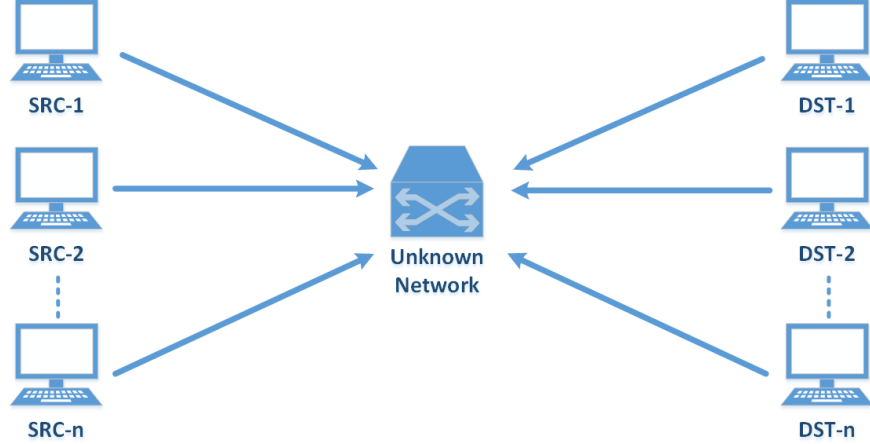


Figure 4.1: iPerf Experiment Topology

For the experiments a simple source-sink topology was used as shown in figure 4.1. Iperf was deployed on one end in client mode as a traffic generator, and on the other in server mode. Traffic was sent only between client-server pairs $\{SRC_i, DST_i\}$ *i.e.* client SRC_i sent data only to DST_i . The said traffic was generated at the maximum possible rate with the TCP congestion control of choice. The server side used the same TCP variant as the client side.

In each experiment run, traffic was sent from the iPerf client to the iPerf server for a duration of 600 seconds during which all of the packets on the client (traffic generator) side were logged. One of the goals was to explore if the performance afforded by the AWS network changes over time. Hence, to reliably capture the changes in trends and behavior over time, we spaced out the experiments on two levels, 1) over different parts of a day and 2) over multiple days. On a given day the experiments were split into five sections, in each of which four experiment runs were performed for selected TCP variants. These experiments were then repeated over a course of ten days. To summarize, for each TCP variant we repeated the experiment 200 times ($4 \times 5 \times 10$) for 600 seconds, producing in total 120000 seconds = 2000 minutes $\approx$ 33.33 hours of traffic per TCP variant. Therefore, in total we have generated more than 133 hours of TCP traffic.

4.3.2 Results and Discussion

In this section we present the results of the aforementioned experiments and the discussion thereof. We begin by showing the trends observed in throughput and RTT one at a time. Based on an observation of the piecewise trend of throughput and RTT, we then present a smaller set of experiments which explore AWS's behavior of resetting the piecewise trends. Next we move onto average value analyses for throughput and RTT over an experiment

run. We also comment on the variance of average throughput and average RTT between experiment runs. After that, we explore the variance in average throughput and average RTT by the time of day in order to observe the consistency of TCP performance and AWS network behavior or lack thereof. Finally we comment on the stability of throughput and RTT over the duration of an experiment by borrowing techniques from chaos theory literature. We use poincaré plots to comment on stability of throughput and RTT and then use variation in lyapunov exponents to characterize the results of the poincaré maps.

Throughput and RTT Over Time

The objective of this section is to analyze the trends in throughput or RTT over time in experiments if present. The aim is to see if these trends in throughput and RTT obey some policy and also to see how stable or erratic the change in them are. To this end observe Figures 4.2 and 4.3 which plot the throughput in Mbits and RTT in μsec over time. The x-axis represents time during an experiment run in seconds. The trace for each TCP in both the graphs was generate by averaging values at the same timesteps over all runs. More concretely, for a given TCP, each experiment produces a trace of logs, the length of which varies a little for each experiment. For the purpose of analyses these traces are down-sampled to 600,000 data points. Hence, each experiment log has 600,000 data points over 600 seconds which means each data point represents the value for 1 millisecond. The shape of this set of vectors is $600,000 \times 200$ because recall that for each TCP variant we have performed 200 runs. We average the set to produce a single vector of shape $600,000 \times 1$.

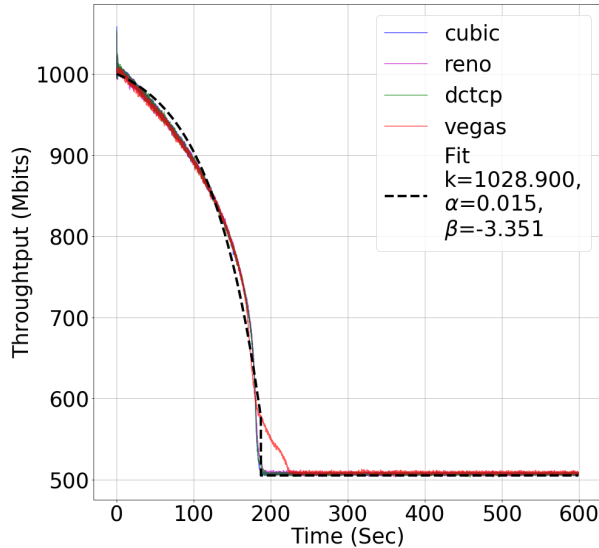


Figure 4.2: Throughput over time

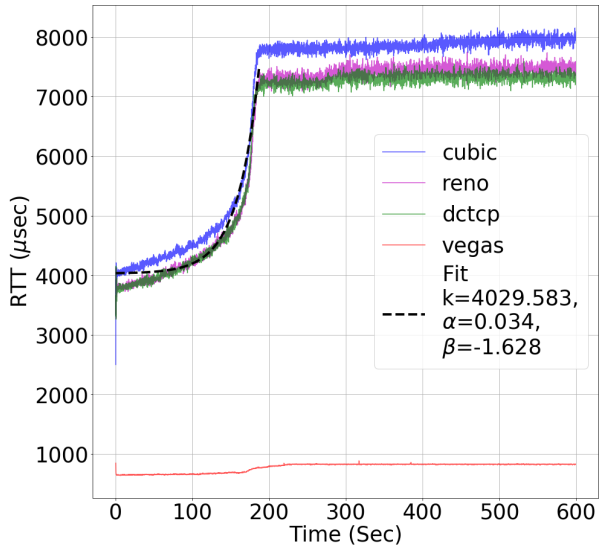


Figure 4.3: Round trip time over time

An interesting thing to note in Figures 4.2 and 4.3 is the piece-wise behavior present in both the throughput and RTT over time. The trend has an exponential and a stable period. For RTT though, TCP Vegas doesn't follow the same trend *i.e.* it doesn't have the four fold

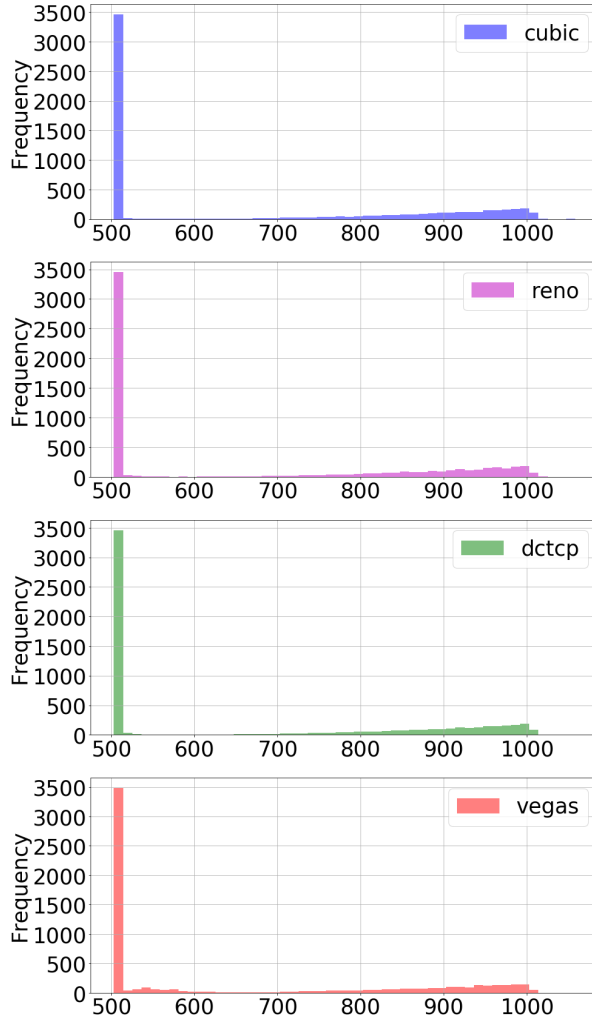


Figure 4.4: Throughput distribution

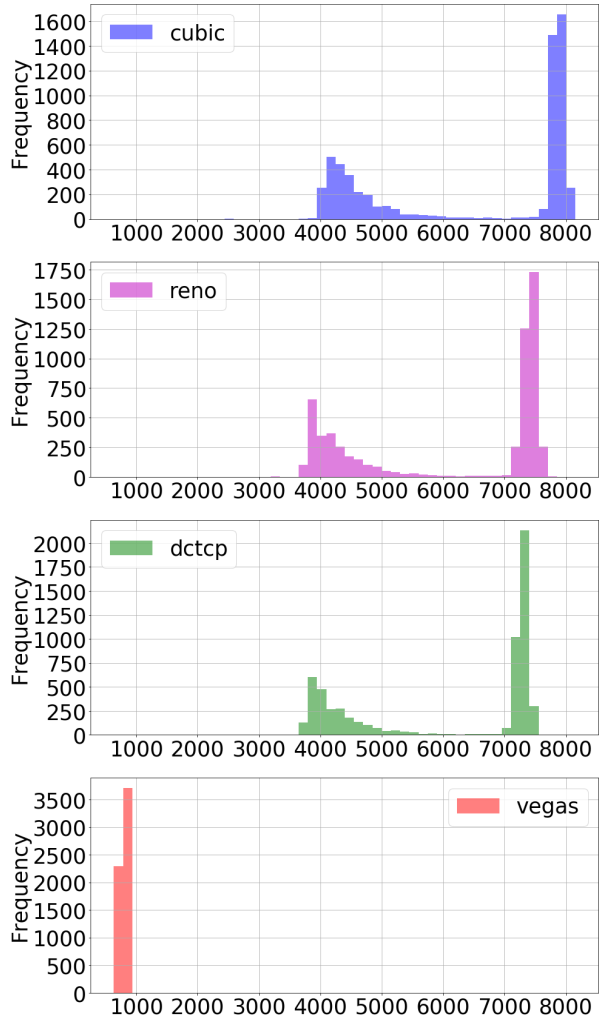


Figure 4.5: Round trip time distribution

exponential increase period. It does increase but only about less than two times. The reason for such behavior is the semantics of TCP Vegas *i.e.* since it is a delay based TCP and has been designed to keep RTT as low as possible. Figures 4.4 and 4.5 which plot the histogram of bandwidth in Mbits/sec and RTT in μsec respectively by using the vector of length 600,000 mentioned earlier. These histograms tell the same story because the distribution is bi-modal (having 2 peaks) which indicates that there are two distinct behaviors present. From Figure 4.5 we can observe that the non-steady-state portion of the RTT trend has a roughly exponential distribution centered around $4000\mu\text{sec}$ - $4500\mu\text{sec}$ for all the variants except TCP Vegas. A similar trend can be seen for throughput from Figure 4.4 though it is hard to observe because of the scale of the Figure. This claim of exponential distribution can be confirmed by fitting exponential functions to the time-series curves in Figures 4.2 and 4.3.

The exponential portion of the throughput is a concavely decreasing function of the form:

$$\Theta = k_1 - e^{\alpha_1 t - \beta_1} \quad (4.1)$$

where t is the time during the experiment, Θ is the throughput and k_1, α_1, β_1 are learnable(fittable) parameters

For the sake of convenience we have set the latter part of Θ to simply be a constant $B \approx 500$. The parameters fitted used least squares fitting as well as the R-squared values for each TCP variant are detailed in table 4.1. R-squared is a statistical measure of how close the data points are to the fitted regression line. It is also known as the coefficient of determination and we are going to denote it by R^2 from this point onward for ease of use. R^2 values are always between 0 and 1 *i.e.* $R^2 \in [0, 1]$. $R^2 = 1$ denotes a perfect fit indicating that the model explains all the variability and $R^2 = 0$ indicates that none of the variability is explained by model. Figure 4.2 also plots the curve generated by using the averaged values of the parameters in dotted black line style and it can be observed that roughly all TCP variants follow the trend. The reason for this piecewise behavior is not TCP semantics, but the network policies EC2. AWS implements the mechanism similar to the one detailed in [33] It is a standard traffic policing mechanism that allows a limiting transfer rate with bursting. Based our experiments in the next section we have estimated that: 1) tokens are allotted every 60 seconds, 2) the Peak Information Rate (PIR) is around 1 GBits, and 3) the Committed Information Rate (CIR) is around 0.5 Gbits.

TCP Variant	k_1	α_1	β_1	R^2
Cubic	1012.94	1.585×10^{-2}	-3.099	0.976
Reno	1006.65	1.657×10^{-2}	-2.956	0.977
DCTCP	1008.02	1.662×10^{-2}	-2.965	0.976
Vegas	1087.99	9.256×10^{-3}	-4.382	0.979
Average	1028.90	1.458×10^{-2}	-3.351	N/A

Table 4.1: Throughput fit parameters

RTT shows the converse trend where, the increase is exponential of the form:

$$\Omega = k_2 + e^{\alpha_2 t - \beta_2} \quad (4.2)$$

where t is the time during the experiment Ω is the RTT and k_2, α_2, β_2 are learnable(fittable) parameters

The parameters fitted used least squares fitting for RTT and the corresponding R^2 are detailed in table 4.2. Though the R^2 values is slightly lower for Ω as compared to Θ , they are still high enough for the fit to considered good. Note that the later part for RTT unlike throughput, is not simply a constant. The reason is queueing at different stages in the network, for example, queueing in network stack of VM, queueing in the network interface card (NIC) in the host machine, queueing in the network switches, queueing in the routers and so on. Each such queue adds queueing delay *i.e.* the time required to enter and exit a queue, which increases the RTT.

TCP Variant	k_2	α_2	β_2	R^2
Cubic	4220.30	3.568×10^{-2}	-1.455	0.960
Reno	3941.47	3.370×10^{-2}	-1.723	0.961
DCTCP	3926.97	3.391×10^{-2}	-1.707	0.962
Vegas	642.32	1.846×10^{-2}	-1.167	0.951
Average	4029.58	3.443×10^{-2}	-1.628	N/A

Table 4.2: Round trip time fit parameters

Resetting the Piecewise Behavior

The experiments performed for section 4.3.2 used traffic from a single flow for the entire duration of the experiment. This setup provides no information on whether the mechanism that forced the bandwidth to drop and RTT to increase will reset if a new flow is started. The reason why the possibility of resetting the trends is being studied is because it may be useful to the cloud tenant. The exponential portion of the curve possesses higher throughput and lower RTT and the longer a VM operates in this region the higher the performance it could extract from the AWS network. Therefore, if the tenants could load balance the traffic between different VMs, they could operate for a longer duration in the exponential region.

To this end we performed a small separate set of experiments where the 600 second experiment was split up into 100 seconds portions and for each portion the existing flow was terminated and after some pause a new one started. We varied the duration of pause between the 100 second portions from 0 seconds to 100 seconds. We have tested all TCP variants but are presenting the results of TCP Cubic because the behavior is consistent across the variants.

Figure 4.6 plots the bandwidth observed for the all the pause values. The x-axis represents experiment time *i.e.* since there were 6 portions of 100 seconds each the total time of the axis 600. The time of the gaps has been removed from plotting for ease of comparison. The y-axis represents the throughput in Mbits. If we observe the throughput profile of the 0 sec experiment in Figure 4.6, we can see that the trend is the same as was observed earlier with the single flow 600 sec experiment. This piecewise trend persists as long as gap value is less than 60 sec. At around a gap value of 60 seconds, the throughput starts recovering and a decaying sawtooth trend can be observed. If the gap value is increased further then a clearer steady sawtooth pattern can be observed where during each section the throughput decays according to eq (4.1) and then resets for the next section. RTT shows a similar trend, the difference being that RTT increases over the 100 second duration and an inverted sawtooth pattern is observed. Due to space restriction we haven't shown the figure for RTT. From these results we conclude that it is not possible to load balance the traffic to extend operation in the exponential portion because it is highly unlikely for any VM to receive no traffic for 60 seconds or more even with a large number of VMs.

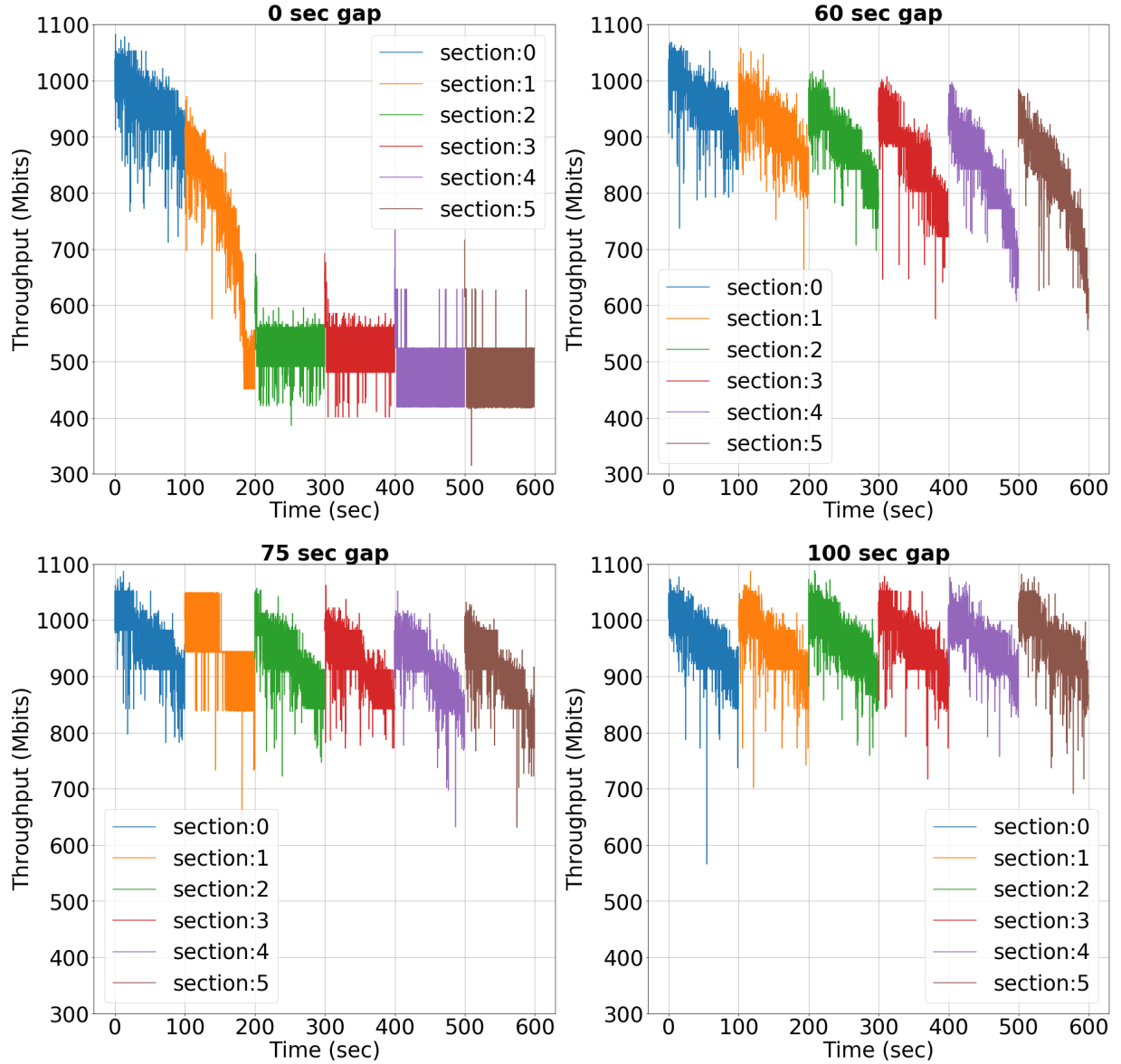


Figure 4.6: Throughput trends with intermittent gaps

Steady State Avg. Throughput and RTT

We have demonstrated that if a new connection is established within 60 seconds of the termination of an existing one then the throughput and RTT profiles mimic the behavior of a single long lived connection. Therefore, it is highly probable that most of the instances (VMs) would operate in the steady state portion of the profile for majority of the time. Hence, the objective of this section is to study the average throughput and average RTT dynamics of the steady state portion. The scatter plot in Figure 4.7 provides a high level view of the average Throughput-RTT characteristics. The y-axis and x-axis represent throughput

in Mbits/sec and RTT in μsec , respectively. Each data point represents one experiment run and is generated using the average throughput and average RTT seen over the steady state section. Hence there are as many data points in the scatter plot as there are experiments.

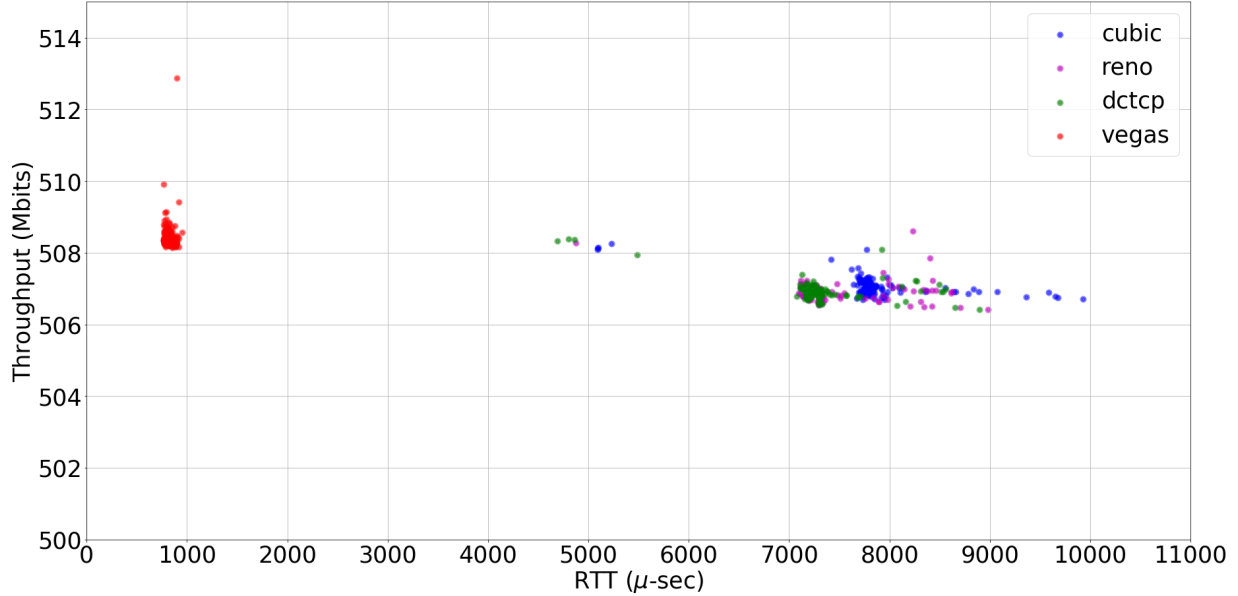


Figure 4.7: Throughput RTT Characteristics

Two interesting observations can be made directly. First, the average throughput in the steady state is similar for all TCP variants and second, the round trip time for Vegas is almost an order of magnitude lower than the other three variants. The reason, as stated earlier, is the design semantics of TCP Vegas *i.e.* it uses a delay based scheme to handle congestion and so is designed to keep RTT as low as possible.

Figures 4.8 and 4.9 contain throughput and RTT boxplots respectively. For any box, the solid orange line represents the median while the dotted green line represents the mean. Referring to Figure 4.8 Vegas has the overall highest throughput and the variance between the experiment runs is similar for all the variants. While TCP Cubic does outperform Reno and DCTPC in terms of throughput, the difference is negligible. In terms of RTT however, as shown in Figure 4.9, the variance for Vegas is the lowest with almost no outlying data points. The other three variants do have outlying points and in both directions indicating relatively less predictable behavior when compared to TCP Vegas. Reno and DCTCP perform comparably, while Cubic is evidently the worst both in terms of mean RTT and outliers.

The throughput and RTT behavior seen in Figures 4.7, 4.8 and 4.9 can be explained based on the semantics of how the variants handle their congestion window and bytes in flight. TCP Cubic is a loss based TCP and changes its congestion window based on the packet loss observed and it is known to have been optimized for high bandwidth-delay product. Therefore, it tends to be aggressive with its congestion window and has higher number of

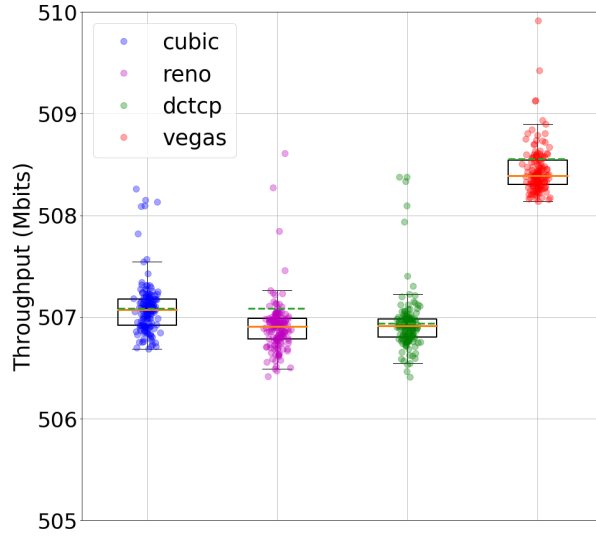


Figure 4.8: Throughput variation

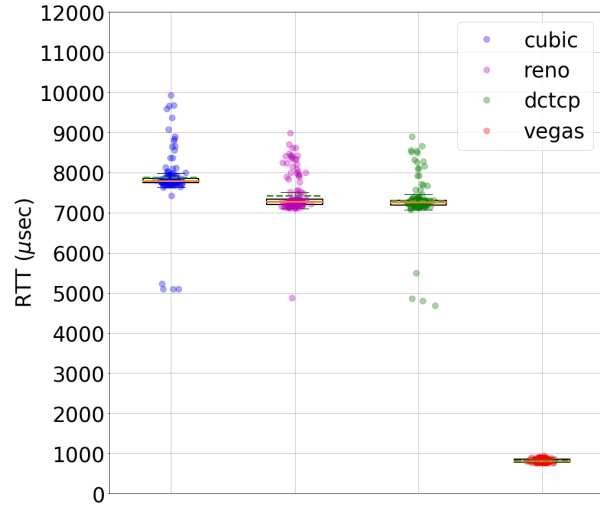


Figure 4.9: Round trip time variation

packet in flight as compared to Reno. This aggression leads to higher RTTs because it is not in the design goals of this TCP variant to maintain a low delay. TCP Reno is much like Cubic in terms of congestion window growth but in addition it is more aggressive in window reduction. Under the same network conditions, this translates to Reno having a larger spread in its congestion window distribution as compared to Cubic. These TCP design semantics can be confirmed by observing the congestion window distribution as plotted in Figure 4.10. The x-axis represents the window size in KBits and the y-axis represents the frequency with which values in any bucket were seen. For these histograms, a common bucket size of 75 Kbits was used to allow for enough granularity for the distributions to be comparable with the naked eye. It can be observed from Figure 4.10 that TCP Cubic has a higher average congestion window than the other three variants and that TCP Reno has a distribution similar to TCP Cubic but with a wider spread indicating aggressive window reduction policy. In terms of performance, the higher average congestion window should result in a wider spread in average throughput of experiments, which is exactly what we observe from Figure 4.8. TCP Cubic has higher average throughput and wider spread indicating higher variance. For RTT, the higher average congestion window should result in higher RTT and the extent of the spread should be similar. The observations in Figure 4.9 verify this because we can see that TCP Cubic not only has higher average value but also has similar number of outliers as compared to TCP Reno. We are not certain whether the network hardware of AWS EC2 is ECN capable or not, the absence of this ECN capability would explain the striking similarity in the distributions of DCTCP and TCP Reno. The similarity in the congestion windows of DCTCP and TCP Reno imply that the congestion control algorithms are behaving in a similar manner. Since the algorithms are behaving in a similar manner, we should expect very similar throughput and RTT dynamics and in fact, Figures 4.7, 4.8 and 4.9 show that we do. For both the throughput and RTT, we can see that DCTCP and TCP Reno have similar average values, similar width of box and whiskers as well as similar number of outliers.

Finally TCP Vegas is a delay based TCP has been optimized for keeping the delay small, which is why we observe from Figure 4.9 that it has average RTT that is almost an order of magnitude lower *i.e.* $9.3\times$ lower than the other variants.



Figure 4.10: TCP congestion window distribution

Variance by Time of Day

Since from a cloud tenant's perspective the performance of their deployed application should be stable throughout the day, therefore, the difference in throughput and RTT at different times during the day should be similar. Recall that to capture the difference or lack thereof in average throughput and average RTT at different times of day, we had spread out the experiments over 5 times in a day. Let's call these 5 time periods, batches. In this section we analyze the throughput and RTT across the batches for each TCP variant.

Figures 4.11 and 4.12 show the boxplots for Throughput in Mbits and RTT in μsec for each TCP variant. There are 5 boxes per TCP each indicating one of the batches. The data points inside the boxes represent average value of the steady state portion of each experiment run.

Note that the 5 batches were evenly spaced out during the time interval of 9:00 AM and 10:00 PM. Each batch had 4 experiment runs per TCP variant and each experiment was 600 seconds long. This process was repeated over a course of 10 days producing 40 datapoints for each box.

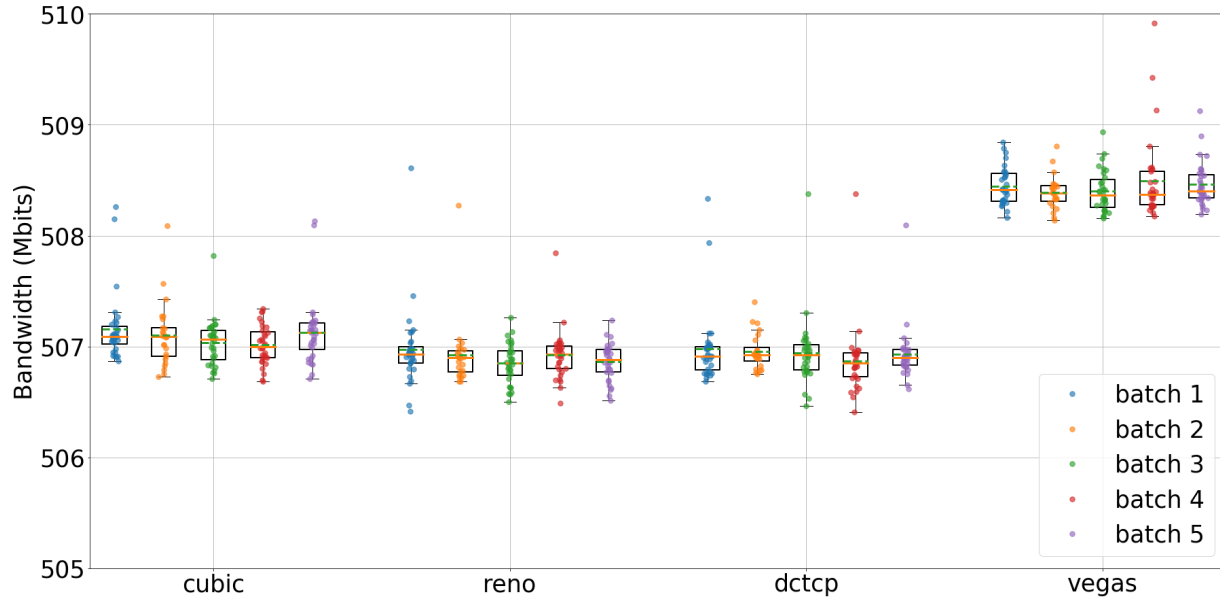


Figure 4.11: Throughput variance across batches

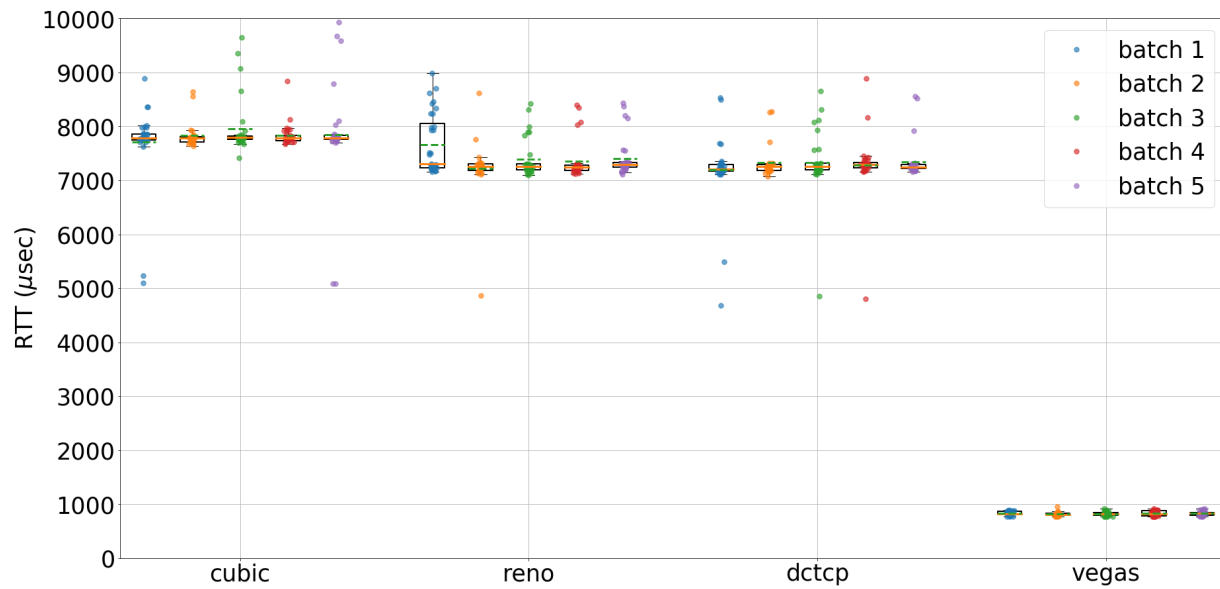


Figure 4.12: RTT variance across batches

From Figure 4.11 we can see that for TCP variants, the throughput variation between the

batches are comparable. This means that they demonstrate the same level of performance regardless of the time of day. The same can be observed for round trip time from Figure 4.12. With the exception of reno batch 1, the behavior of the batches within a TCP variant is very similar. Hence it is safe to conclude that within a day both the throughput and the RTT of any two flows is very similar.

Stability of RTT and Throughput

The objective of this section is to analyse the stability of the throughput and RTT *i.e.* within an experiment run how drastic are the changes between a time step and the subsequent ones. For this purpose, we have use Poincare maps as shown in Figures 4.13 and 4.14 which plot the Poincare maps for Throuput in Mbits and RTT in μsec , respectively. A Poincaré Map is the result of a mapping M from d dimensional space to a d dimensional space, *i.e.* $M : \mathbb{R}^d \rightarrow \mathbb{R}^d$, that generates a vector X_t for each input such that $X_{t+1} = M(X_t)$. In our case the sequence $X_0, X_1 \cdot X_n$, for Figure 4.13 is the throughput times series and for Figure 4.14, it is the RTT time series. For convenience we denote Throughput as Θ_t and RTT as Ω_t , where t is the used to indicate the value at time step t . The Throughput and RTT used here are the same as the ones used in section 4.3.2. Simply put, such plots are used to view the next value a_{t+1} in a time series as a function of the current value a_t .

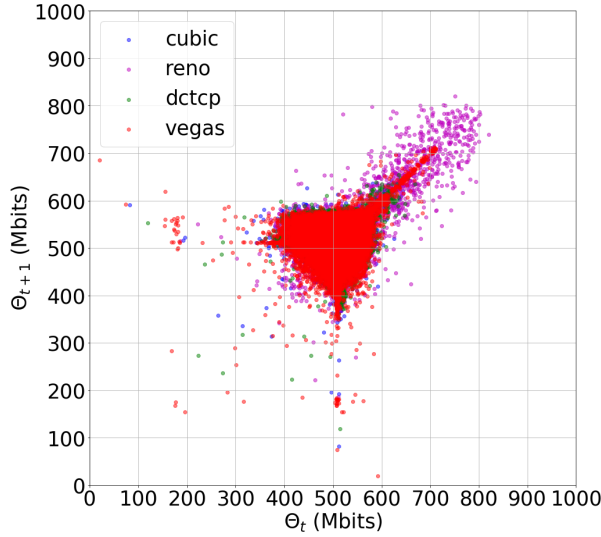


Figure 4.13: Throughput Poincaré map

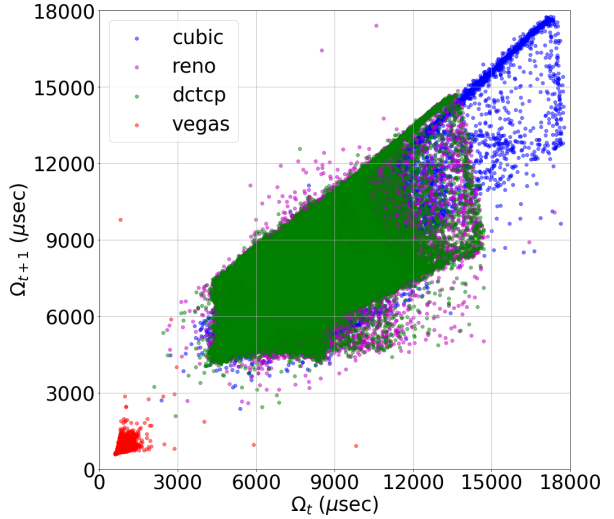


Figure 4.14: RTT Poincaré map

According to [46], an ideal TCP trace should form curves in the poincaré map which are 1-D *i.e.* lines and not clusters of different shapes. However, our results show the formation of clusters. We can see from Figure 4.13 that the cluster of TCP cubic spreads more than the other TCP variants. While it does achieve higher throughput values, it varies alot. TCP Vegas also varies but majority of its data points are clustered with TCP Cubic and DCTCP.

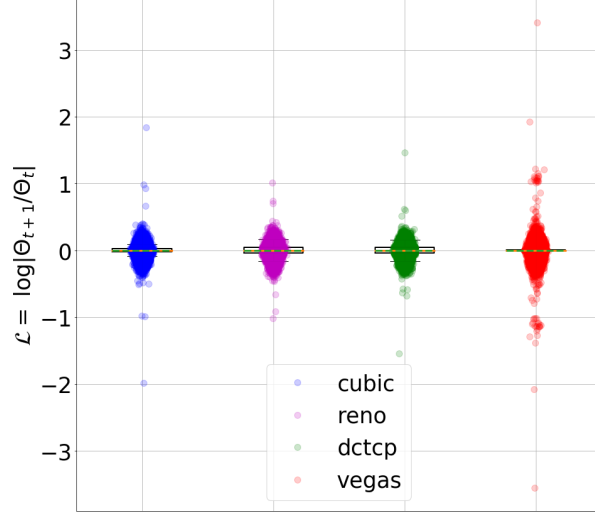


Figure 4.15: Throughput Lyapunov exponents

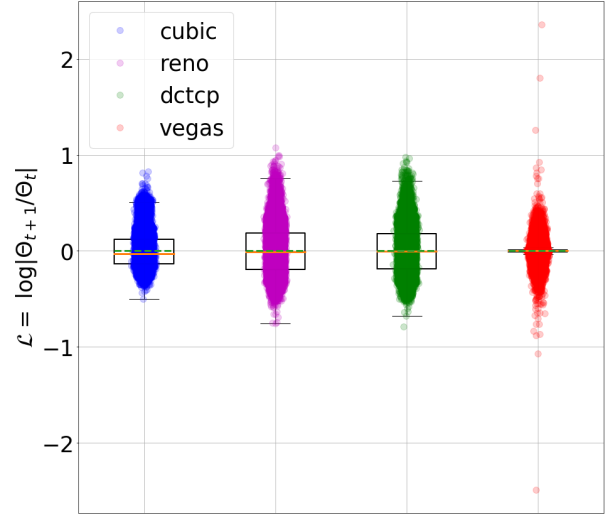


Figure 4.16: RTT Lyapunov exponents

This pattern of clustering indicates that in terms of throughput predictability *i.e.* guessing the next value given current value, Vegas, Cubic and DCTCP outperform TCP Reno. Reno has the largest variation in throughput. Looking at Figure 4.14, we can see that TCP Cubic has a much greater spreading cluster than the other variants *i.e.* it starts at around the same point as compared to Cubic and DCTCP plus it goes to a higher point. This is inline with the average RTT results that were reported in Figure 4.9 *i.e.* TCP cubic not only has a higher average RTT but variation of the RTT is highest as well. Since TCP Vegas is a delay based TCP which has been optimized to maintain low delay, we can see from Figure 4.14 that its cluster is not only centered closer to the origin but also has a much smaller spread. Therefore in terms of RTT predictability TCP Vegas outperforms all the other variants, TCP Cubic performs the worst while Reno and DCTCP behave very similarly.

This behavior can be confirmed via *Lyapunov exponents* which are used to characterize the poincaré maps. It is defined as:

$$\mathcal{L} = \log \left| \frac{dM}{dX} \right| = \log \left| \frac{X_{T+1}}{X_T} \right|$$

where, X_T is original vector and X_{T+1} is the shifted vector. In terms of dynamics, negative Lyapunov exponents indicate stability while positive values indicate exponential change or possibly chaotic behavior. Figures 4.15 and 4.16, show the box plots for the calculated Lyapunov coefficients for throughput and RTT for each TCP variant. The value of the data points in both Figure 4.15 and 4.16 are either negative or very small (approx. less than 1) if positive, it is safe to conclude that both throughput and RTT demonstrate fairly stable dynamics. If the boxes are observed carefully in Figure 4.15, the box of TCP Reno is largest and this indicates that Reno has the least stable throughput profile. From Figure 4.15 the

converse for TCP Vegas *i.e.* for RTT TCP Vegas has the least spread of Lyapunov exponents and hence is the most stable variant interms of RTT profile.

4.4 Jmeter Bechmarking

Having established the baseline for what the best performance for different TCP can be in the cloud, the next step is to see whether this performance is translated when using a protocol built on top of TCP *i.e.* HTTP. We will also explore the effect of different workload characteristics: file size, inter-request time distribution, and the interaction of the two. The possible effect of file size is being studied because for key value stores such as Cassandra and object stores such as S3, the items being stored may be of different sizes or within a certain range of sizes and certain TCP variants might provide better throughput, RTT and/or request response time for certain file sizes. We define response time to be the time difference between when a client sends a request to a server and when the request is completed. We study response time of the TCP variants because it directly impacts the interactive experience of the users.

For the purpose of testing the effect of filesize we have opted for the use of HTTP over TCP/IP. The high level goal is to make HTTP GET requests for different file sizes and then observe the resulting traffic and TCP behavior. Hence, for simplicity and tractability, a single client and a single server topology was used as shown in 4.1. It is the same topology as the one used for iPerf Benchmarking. A client machine was generating GET request to a website on a server machine hosting the files. On the server side we have deployed a website running atop the LAMP stack. The website consists of a few blog pages and different downloadable files which are accessed via links. The files sizes used are: 10K, 100K, 1M, 10M, 100M, 1G. On the client side Apache Jmeter, is used to generate HTTP request streams with the desired inter-request time distributions. There are two scenarios for the experiments that we have done:

Scenario 1: Pushing GET requests at maximum rate possible. In this scheme the next GET request is generated after the response to the previous one has been completed. The goal of this experiment design is to see the best possible performance that each TCP variant could achieve for each of the file sizes mentioned above. We have performed 5 experiment runs per TCP variant for each file size. Since there are 4 TCP variants, 6 file sizes and 5 runs per TCP per file size, in total $4 \times 6 \times 5 = 120$ experiment runs were done. Each experiment run was approximately 300 seconds long and the HTTP connection type was keep-alive. In total, for this experiment, we have generated $120 \times 300 = 36000$ seconds = 600 minutes = 10 hours of HTTP traffic.

Scenario 2: Pushing GET requests according to a delay distribution. In this scheme the next GET request is generated regardless of whether the the response to the previous one has

been received or not. We've used 4 delay distributions: 1) constant delay, 2) gaussian random delay, 3) poisson random delay, and 4) uniform random delay. The distributions are centered to have a mean such that max possible traffic can be pushed so that a mean throughput similar to scenario 1 can be observed. Similar to the previous scenario, for each TCP and each delay distribution 5 experiment runs were performed per file size. However the file sizes were limited to 1M, 10M, 100M and 1G. Hence, for 4 TCP variants, 4 distributions, 4, file sizes and 5 experiment runs per TCP per distribution per size, we have in total $4 \times 4 \times 4 \times 5 = 320$ runs. Each experiment run was approximately 300 seconds long and the HTTP connection type was keep-alive. For this experiment we have generated $320 \times 300 = 96000$ seconds = 1600 minutes ≈ 26.67 hours of HTTP traffic.

4.4.1 Results and Discussion

In this section we present the combined results from both of the scenarios. Though the second scenario focuses on inter request delay distributions, we have combined the results of both scenarios because there is little to no difference in the observed results. However, at certain points we do comment on differences observed that originated from distribution of traffic.

Throughput and RTT Over Time

The objective in this section is the same as it's counterpart in the iPerf Benchmarking section *i.e.* to see if any trends in throughput and RTT emerge over time, if they obey some policy and how stable or erratic the behavior is. Figures 4.17 and 4.18 plot the throughput in Mbits and the RTT in μsec over time. The trace for each TCP in the graphs was generated by averaging values at the same timesteps over all runs. More concretely, for a given TCP, each experiment produced a trace that was roughly of the same length. So, for the sake of processing, each trace was down-sampled from it's original length to 300,000 data points. This corresponds to 1000 data points per second *i.e.* each data point in Figures 4.17 and 4.18 represents one millisecond. This vector was generated for each experiment and then averaged over 5 experiments to produce a single vector of shape 300,000.

From Figure 4.17 we can observe that the throughput for 10K Byte file size is much lower for all TCP variants as compared to the others. The reason for such low numbers is that throughput is determined by both the number of requests and filesize that was requested. While more GET requests for 10K byte filesize can be completed per second than for example, 100K Bytes filesize, there is a limit to how many requests can be processed per second by the CPU. Hence, the CPU becomes a bottleneck *i.e.* a ceiling on the number of requests that can be processed per second and as a result overall throughput is reduced. It is safe to conclude that for smaller file sizes ($\leq 10\text{K Bytes}$) such phenomenon will be observed. The extent of the reduction will depend on the capability of the CPU.

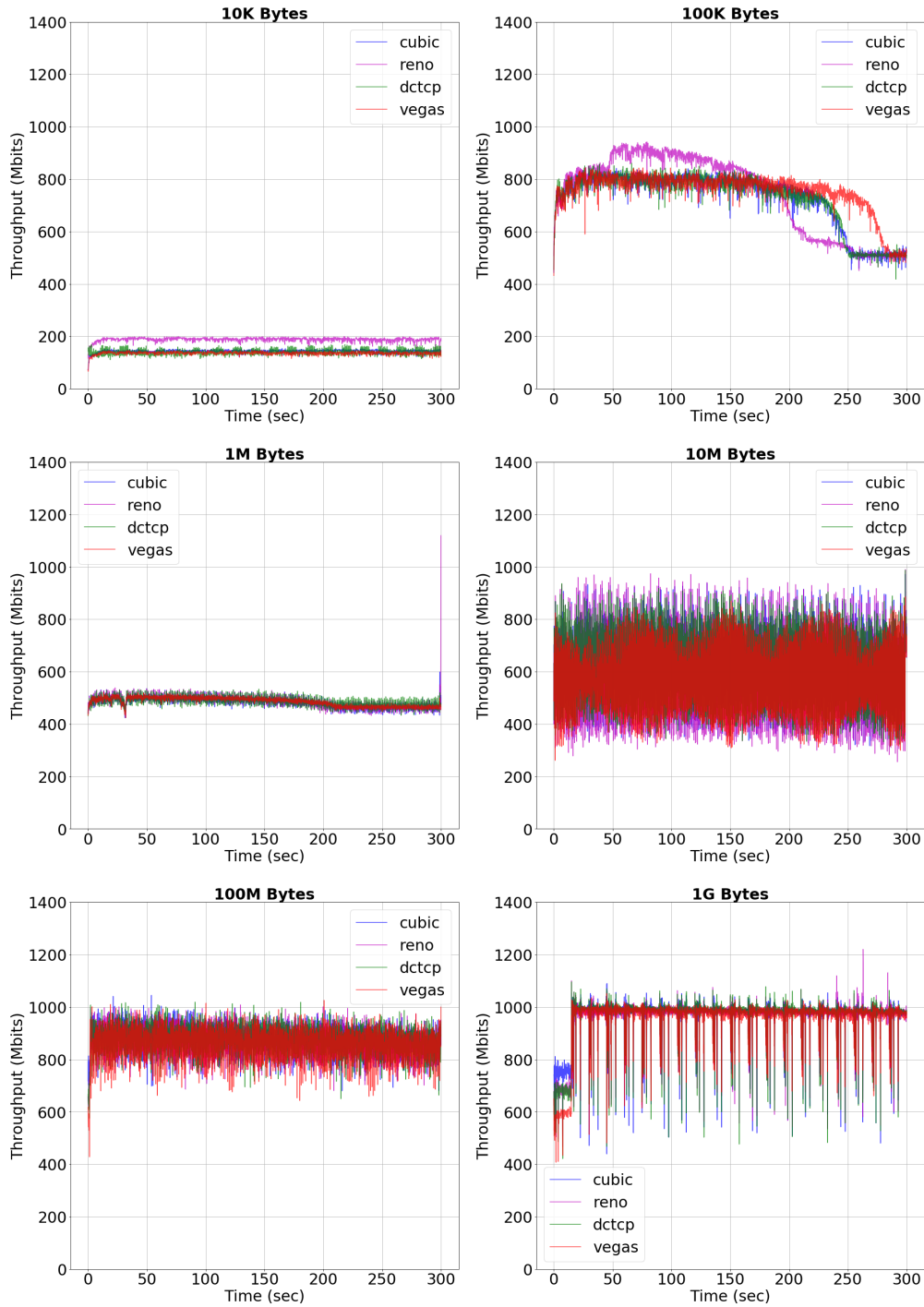


Figure 4.17: Throughput for different file sizes

The second observation is the absence of the piecewise trends in Figures 4.17 and 4.18 for both throughput and RTT that was observed in the iPerf Benchmarking experiments. Except for 100K Byte throughput and RTT trace in Figure 4.17 no piecewise trend to be observed.

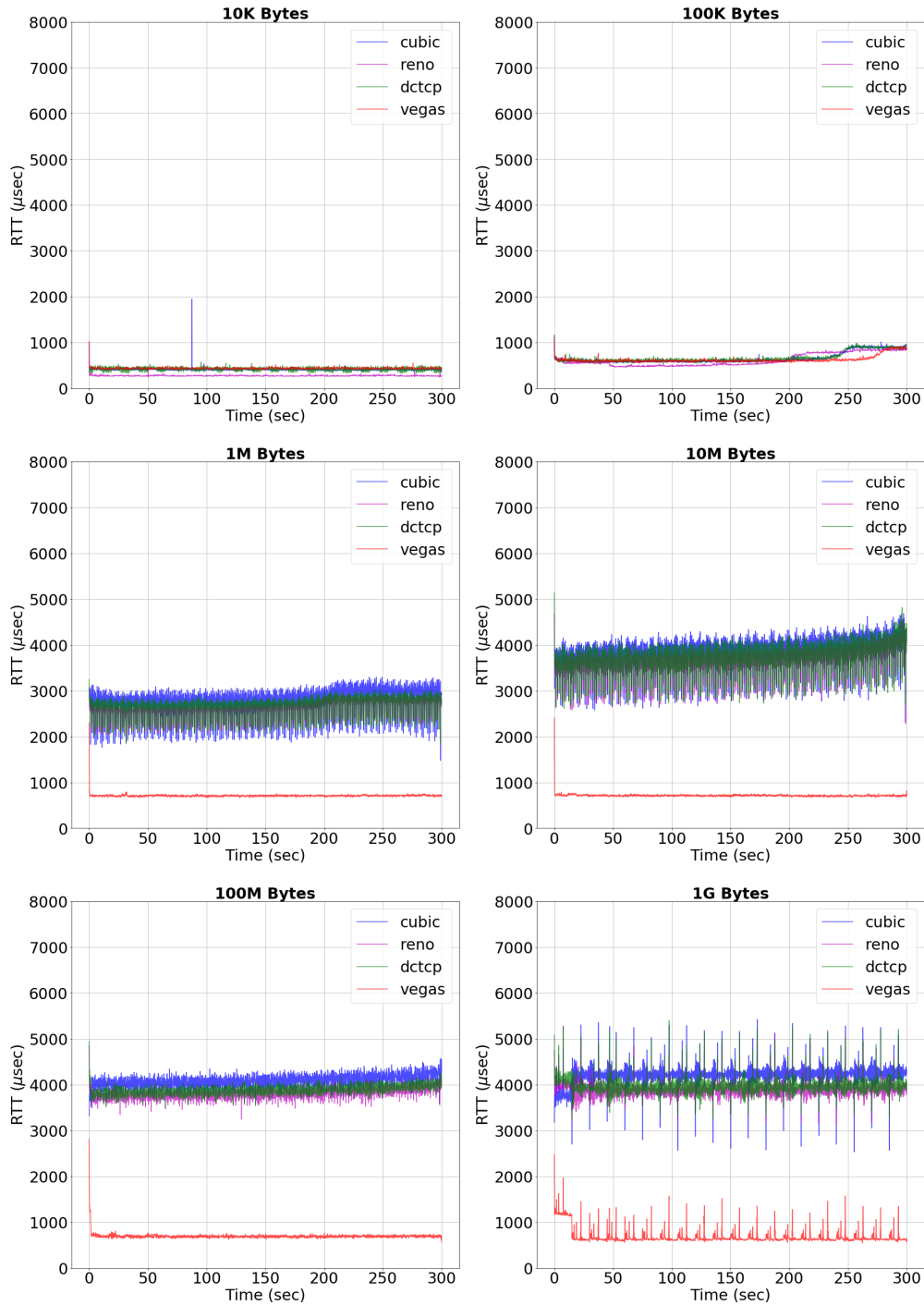


Figure 4.18: RTT for different file sizes

The third observation from Figure 4.17 and 4.18 is the reducing stability of throughput and RTT as the file size increases. By stability we mean, the magnitude of change in value between two subsequent time steps. If the change is large then the behavior is unstable and

vice versa when it's small. For all TCP variants, 1G, 10M and 100M show the least amount of stability.

Average Throughput, RTT and Response Time

Since the piecewise behavior of throughput and RTT is not present for all filesizes, the average value will be calculated by using values from the entire duration of the experiment. Hence, the objective of this section is to analyze the mean values and variation of throughput and RTT between experiment runs. The results will illustrate the consistency or lack thereof in performance of the TCP variants between one run and another. Figures 4.19 and 4.20 show the box plots for throughput in Mbits and RTT in μ sec. The x-axis represents the file size and and boxes are grouped into six groups each representing a file size. The box plot has been made using the same method as in Figure 4.12 where instead of batches we have file sizes.

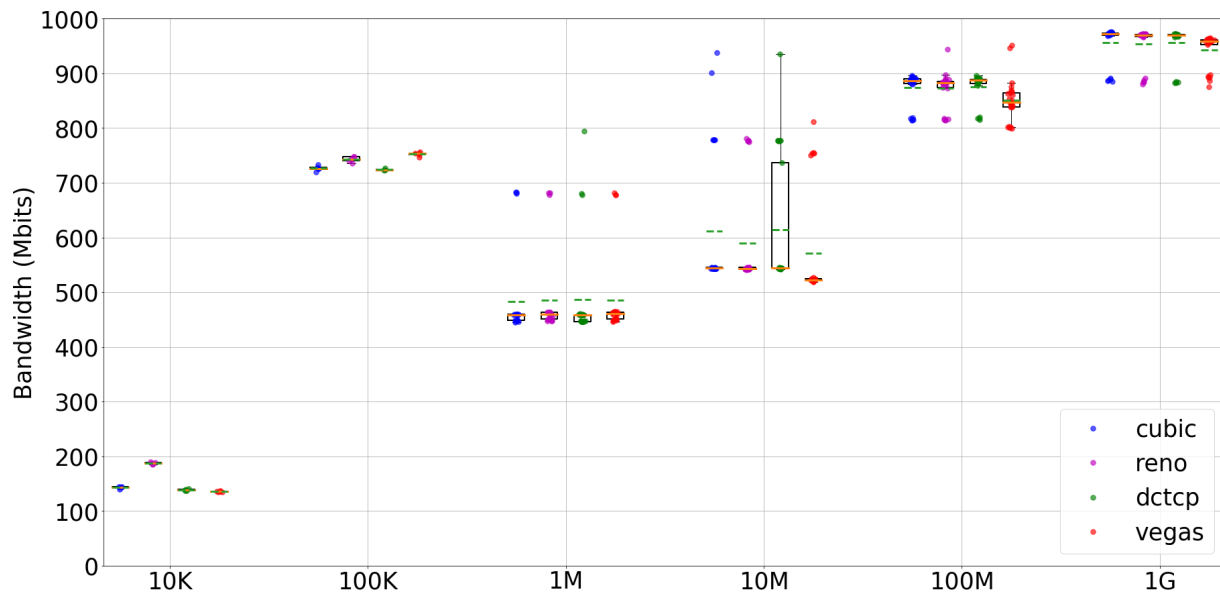


Figure 4.19: Throughput for different file sizes

One simple observation from Figure 4.19 is that throughput increases as the file size increases. The reason for the decrease in throughput for small file sizes, as mentioned earlier as well, is the that the CPU becomes the bottleneck *i.e.* it cannot process requests fast enough. An interesting thing to note for file sizes 1M Bytes and above is that the data points are split into two sections, those that fall inside the box or are close to the whiskers and those that are far outside the box. The reason this happens only for filesizes 1M Bytes and above is that scenario 2 experiments were performed only for filesizes 1M Bytes and above. The points that indicate higher throughput originate from the scenario 1 *i.e.* experiments which were pushing traffic at maxium possible rate. Even though the distributions were centered to have average

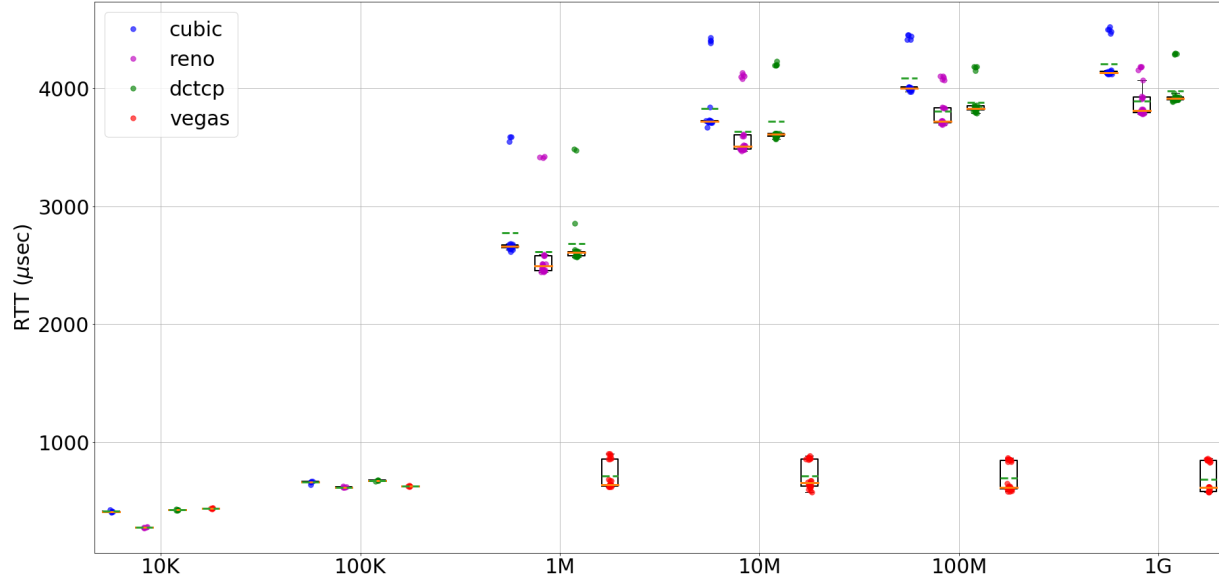


Figure 4.20: RTT for different file sizes

throughput rates similar to scenario 1, the average throughput for the scenario 2 experiments is lower. Also the distributions of inter request time have no effect when compared to each other *i.e.* whether the requests follow a gaussian distribution or a poisson distribution, the average throughput remained the same.

From Figure 4.20 the same can be said for RTT except for TCP Vegas. For 10K Byte and 100K Byte file sizes the difference is negligible. For 1M Bytes and above we see the same split in data points as was observed in Figure 4.19. Apart from that the difference in performance of the variants for different file sizes is negligible. Consistent with earlier observations, TCP Vegas has the lowest RTT. Cubic on the other hand still comes out to be the worst variant in terms of RTT.

Another important aspect to consider from the cloud tenant's perspective, as mentioned earlier, is the response time. Figure 4.21 plots the a normalized histogram of the response time for all TCP variants for different file sizes. The x-axis represents the response time value and the y-axis represents the frequency with which a certain value was seen.

For filesize of 1M Bytes and less all of the TCPs are equally responsive. However, beyond 1M Bytes, we start seeing the difference. TCP Vegas tends to have the highest average response time of the three variants *i.e.* it's distribution is centered further out, while the other three variants behave in a similar manner. Higher response time of TCP Vegas can be attributed to the fact that it tries to operate in the pre-congestion zone and the resulting bandwidth-delay product is low. Hence, while TCP Vegas tries not to cause congestion in the network, it is too conservative in terms of of it's congestion window and cuts the rate too often. While it does manage to maintain a marginally higher throughput as compared to other variants,

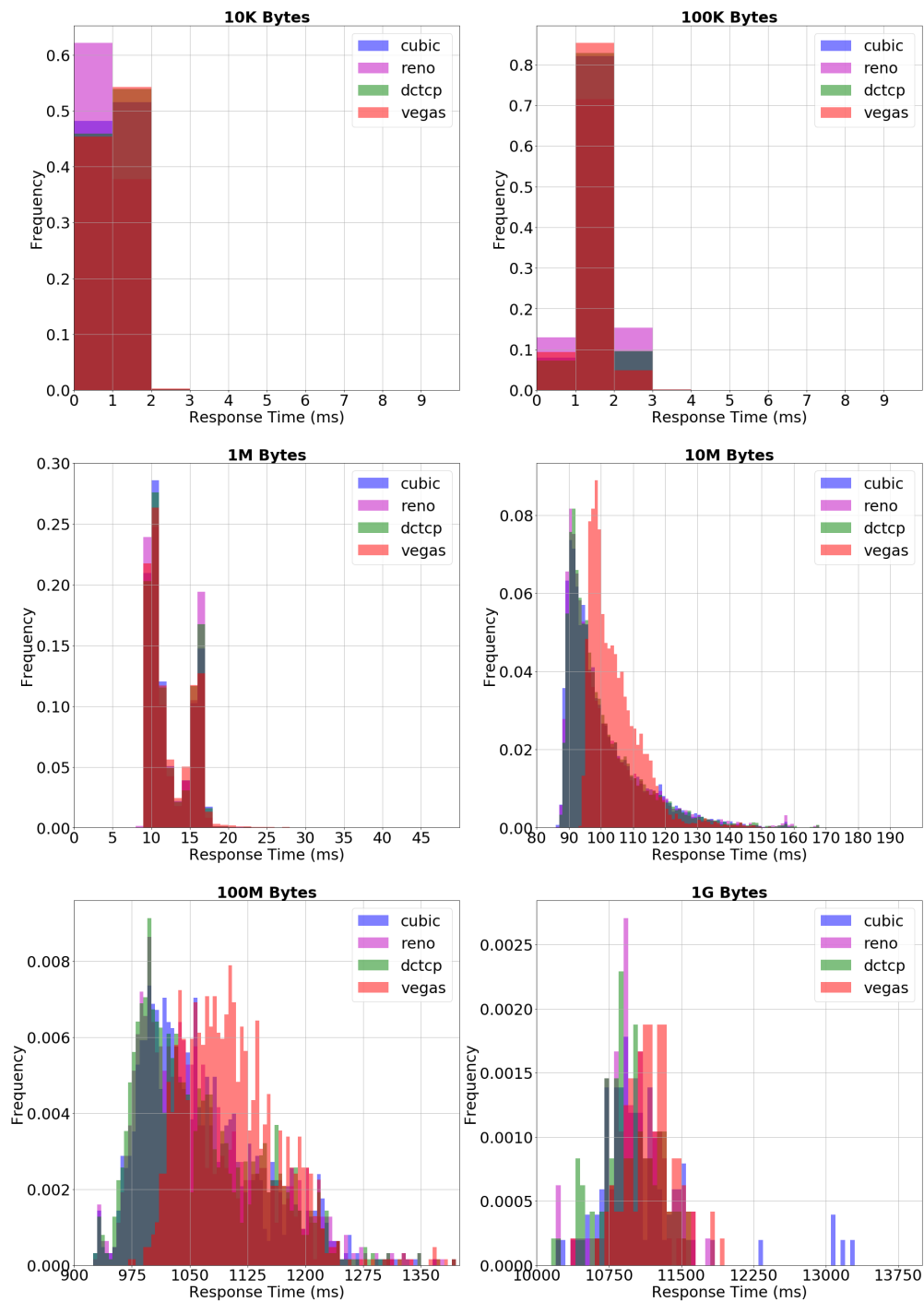


Figure 4.21: Response Time distribution for different file sizes

response time suffers.

4.5 Chapter Summary

In this chapter we have presented measurement profiles for TCP Cubic, Reno, Vegas, and DCTCP from a cloud tenant's perspective while treating the AWS network as a blackbox. We have shown that throughput and RTT have piecewise behavior for long lived connections and that Vegas has the highest average throughput in the steady state. All other variants have similar throughput. Vegas consistently has the lowest RTT in the experiments and Cubic has the highest RTT. Reno is the least stable in terms of throughput and Cubic has the most variation in RTT both in an experiment and across experiments. File size affects throughput *i.e.* if the size is too small the CPU becomes a bottleneck. Throughput of all variants is similar regardless of file size or inter request delay distribution. All variants perform equally well in terms of response time except Vegas which has the highest (worst) response times. The results should help the tenants make a better and more informed choice of the TCP variants.

Chapter 5

Conclusions and Future Work

The key goal of this thesis is to improve data transportation for Internet of Things. We have successfully applied knowledge of TCP semantics and traffic behaviors to enable better data transport in both edge networks and datacenter networks. Our investigations have led to two key solutions 1) A novel probabilistic buffer sizing scheme for access routers sitting in edge networks for IoT Deployments and, 2) Improved guidelines for selection of TCP congestion control variants based on empirical evidence collected from a cloud service. Building up directly on work from [12, 20, 52], the former details a scheme and formula to size router output buffers which enable improved performance *i.e.* latency, while demonstrating similar levels of buffer utilization and packet-loss, all without the change of TCP protocols or change of networking hardware or infrastructure. The latter builds up on [9, 27, 30, 39, 47] and provides detailed analysis of TCP profiles collected from a real deployment of VMs in a cloud service, to explain the varied performance of TCP congestion control variants under different traffic patterns. It provides explanation into the behaviors observed and is intended as a guide for choosing TCP variants based on traffic patterns.

5.1 Summary

The increasing popularity of IoT and the accompanying tsunami of data has made it crucial to improve data transportation methods because current systems are not equipped to handle the influx of data at this scale. IoT devices live on the edge and as such their application's performance is directly impacted by the performance of the access link that they are connected to. One of the most crucial factors dictating the performance at the access link is the design of routers. Among the important design aspects of these routers is their size of their output buffer queue. Selecting an appropriate size of the output buffer is crucial because it directly impacts two key performance metrics of the IoT system: 1) access link utilization (*i.e.*, the percentage of time the link is under use at full capacity), 2) latency (*i.e.*,

the time a packet takes to go from sender to receiver) and, 3) the packet loss-rate (fraction of packets dropped by the router output queue). *If the buffer is under-provisioned, i.e.,* the size of the output buffer is too small, the access link utilization decreases because when IoT devices decrease their TCP congestion window sizes on experiencing packet losses and take an intermittent pause from sending new packets, the small output buffer drains quickly and the access link stays idle until the IoT devices resume sending the packets. *If the buffer is over-provisioned, i.e.,* the size of the output buffer is too large, the latency experienced by the packets of the IoT devices increases because during an intermittent increase in the number of packets arriving at the access router, the packets that arrive and enter the buffer when the buffer already has a large number of packets waiting in it, experience large queueing delays. The lower access link utilization and higher queueing delays can lead to decrease in the throughput and/or increase in the power consumption of the IoT devices.

Therefore, in our first work we present a probabilistic method to calculate the size of the output buffer for IoT access routers. Through extensive literature review, we identify three key assumptions of prior art such as [12, 13, 14, 15, 20, 21, 26, 40, 41] and show through both real world experiments based on raspberry Pis as well as NS-3 simulations that they do not hold true in IoT traffic. The key technical depth lies in the theoretical modeling of the size of the TCP congestion window before and after the congestion event, and its use in calculating the output buffer size. We first model packet drops at the router output queue as a binomial distribution by using arbitrary distributions of number of incoming packets. We then derive the expression for the expected value of the proportion of flows that will drop a given number of packets in a congestion event. By modeling the growth and reduction of the average congestion window of senders (devices) for the appropriate fraction of flows, after a congestion event occurs and then relating these to the average congestion window before the congestion event, we set the framework for the derivation of the buffer size. This expression also factors in the fraction of flows that are in slow start and those that are congestion avoidance mode explicitly. We then state two expressions that relate the average congestion window before and after the congestion event to the minimum required buffer size before and after the congestion event. We solve the resulting equations for the buffer size and provide a closed form formula that can be evaluate based on a few network and traffic parameters known to the network administrators. We then show that that this frame work is general and can be adapted to different congestion control variants by deriving a buffer sizing expression for TCP Cubic. Furthermore, we show that buffer sizing for UDP is much simpler than for TCP by deriving a buffer sizing formula for UDP only traffic. Finally we merge the TCP and UDP schemes to provide a more general mixed traffic sizing scheme that can be used in realistic scenarios. We then present the experimental evaluation of our buffer sizing scheme and its extensive comparison with the state of the art schemes proposed in [12], [20] and [26]. Our results show that for IoT traffic, [12], [20] and [26] over-estimate the buffer size by at least two times the required value, due to which, IoT flows experience over atleast 60% higher queueing delay compared to when using the buffer size calculated by our scheme. Hence, our scheme delivers lower latency while maintaining similar or better buffer utilization and packet loss-rate. For UDP only traffic our scheme achieves at least 91% link

utilization with very low queuing delays and aggregate goodput that is approx. 90% of link capacity. Finally, for mixed traffic scenarios we show that our scheme performs better than either TCP only or UDP only traffic scenarios achieving higher link utilization than either as well as low delays, low loss-rates and aggregate goodput that is approx 94% of link capacity.

We then shift focus from access networks to datacenters networks because the cloud is arguably the brain of IoT and hence, if IoT systems are to have good performance, the communication between VMs in the cloud must be improved. We again focus on improvement of transportation using TCP because it accounts for 90% of the traffic in the cloud. When *cloud tenants* *i.e.* people who provide application services and use the cloud for meeting their infrastructure needs, buy Virtual Machines (VM) *e.g.* Elastic Compute Cloud (EC2) instances from a cloud service provider like AWS, they have little control over where the instances are placed and almost no control over how they are networked to one another. The EC2 instances may end up being hosted on the same physical machine, or on different machines in the same rack, or on machines in different racks. Furthermore, the network between two or more pairs of instances requested and allocated at the same time, may or may not be homogeneous, *i.e.* that not all machines would be the same number of hops away from one another. The underlying network equipment also has varying capabilities such as switch bandwidth, Explicit Congestion Notification (ECN) capability etc., which are unknown to the tenant. This gap in knowledge of the network topology and it's capabilities makes it tough to choose one TCP variant over another? Choosing the correct TCP is important because it impacts the throughput, round trip time (RTT) and response time of the applications deployed by the tenants. All of these metrics, especially response time, directly impact user experience of the application and hence, have an effect on the popularity thereof. Correct choice of TCP may also lead to reduction in the overall amount of network traffic sent between the VMs, which inturn would result in reduction in revenue spent on network communication and hence, possibly, increase profits.

Therefore, we focus on profiling TCP from the cloud tenants perspective while treating the network inside the cloud as a black box as it is for the tenants. We conduct two sets of tests, one at the transport layer and the other at the application layer and provide insights and conclusions. The results should help guide the tenants in making a better choice of TCP congestion control based on their needs. For the transport layer metrics we focus on the throughput and RTT characteristics. In particular we analyze trends in throughput and RTT over time within experiment runs as well as across experiment runs. We use a popular traffic generation tool called iPerf to generate raw TCP traffic between VMs and `tcp-probe` to collect packet level traces. Throughput and RTT are then calculated using these packet level traces. Our results show the existence of a piecewise trend in both throughput and RTT that is exponential in the first phase and linear in the second phase. While all TCP variants behave in a very similar manner in the exponential phase, they show difference in performance metrics in the steady state phase. For example, TCP Vegas outperforms all three variants in terms of both average throughput and average RTT. While the difference in throughput is small, the RTT of TCP Vegas is atleast 9 times lower than the other

three variants. Furthermore, there is no difference in average throughput and average RTT by time of day *i.e.* regardless of the time the AWS network offers similar performance to the VMs. For the application layer experiments, we use HTTP as a case study because it operates on top of TCP/IP. We use Apache Jmeter for generating HTTP GET requests for files of different sizes ranging from 10K Bytes to 1G Bytes. `tcp-probe` is again used to collect packet level traces for analysis. Again, we have analyzed the over time trends in throughput, RTT and response time within an experiment as well across experiments. We also study the effects of file size on the three metrics. Our results highlight an interesting difference between the transport layer and the application layer trends over time *i.e.* the piecewise behavior of the throughput and the RTT is not observed in the application layer experiments. Another interesting observation is the difference in throughput between a client that is pushing requests at the maximum possible rate and one that is generating requests based on a delay distribution. The latter achieved a lower throughput than the former even though the delay distribution for the latter was set such that both the scenarios would have the same average throughput. Results also show that for file sizes less than or equal to 10K Bytes, TCP Reno achieves a marginally higher throughput while maintaining similar response time as compared to the other variants. Furthermore, while TCP Vegas maintains similar throughput as compared to the other variants, the response time is higher (worse) for all file sizes. Finally, Throughput and RTT stability within an experiment run decreases as the file size increases for all TCP variants.

5.2 Future Directions

5.2.1 Goal Based TCP Congestion Control: A Reinforcement Learning Approach

Traditional congestion control schemes are based on explicitly crafted window alteration formulas *i.e.* formulas determine how much the congestion window grows or reduces and how often it does so. The main idea of this project would be to do away with explicit formulas in favor of a goal based approach. This is where Reinforcement Learning (RL) plays a key role. RL is a form of machine learning in which an agent(subject) learns to achieve objectives based on reward functions(feedback) from it's environment. Over several iterations the agent learns how to achieve it's goal or atleast get closer to it while adapting to changes in the environment via reward feedback. Hence, in networking or protocol design in particular, the idea is to treat the computer managing a connection as the agent, the network as the environment and perception of packet loss as feedback. The goal could be what the manufacturer or the user desires; to name a few, TCP bandwidth fairness, low link congestion, low packet loss, high throughput *etc.* The feedback is very important here and plays the most vital role in shepherding the agent towards achieving the goal. ECN capable equipment which have emerged recently, send explicit notifications of congestion and as such

are a perfect fit for this scenario because they provide explicit feedback. The end result would be a universal TCP protocol that would learn to adapt to environments that it is deployed to, be it lossy networks, or networks with high number of connections per link, or incast scenarios prevalent in datacenters. The method of validation and performance comparison would be 1) direct comparison with existing TCP protocols at achieving goals that the said protocols were design to achieve in a variety of environments, 2) having the universal TCP mimic the behavior of existing protocols and evaluating the degree of similarity and, 3) computation resources required to maintain to run the congestion control algorithm. To the best of my knowledge this would be the first work of it's kind in protocol design and so, it posses novelty as well as high chance of applicability and impact if executed well.

5.2.2 Middle-Box Channels: Virtual Congestion Control And Choice of TCP

Recently there have been successful and efficient attempts to homogenize TCP congestion control across entire datacenters, for example, AC/DC [32] and Virtualized congestion control [19]. Since both of the schemes revolve around efficient translation of the client version of TCP to cloud provider version of TCP, they posses the mechanism that could allow for efficient network QoS control. Hence, the idea is to provide communication channels of different quality between machines much in the same way as AWS provides different EC2 instances types, each optimized for a certain task. The research work and value lies in an extensive empirical measurement study using the virtual congestion control toolkits. Applications of different sort *e.g.* File Systems such as HDFS, Key Value Stores such as Cassandra, Stream proccessing frameworks such as Storm *etc.* would need to be tested with different TCP types. Once the application behaviors and performance on different TCP variants is known, the channels can be created.

This would also allow for traffic optimization from the cloud providers perspective because unlike the cloud tenant, they already have knowledge of the network hardware and would now know the coarse behavior of the traffic. For example, optimization could be made in terms of which TCP types are mutually compatible *i.e.* fair to each other in terms of bandwidth sharing and latency, and machines operating with those TCPs could be grouped in the same racks. Hence, when customers demand certain types of TCPs their VMs could be shifted to the racks optimized for the demanded TCP variant.

Bibliography

- [1] Arduino uno. <https://www.arduino.cc/en/Main/arduinoBoardUno>.
- [2] Cisco 829 industrial integrated services routers. <http://www.cisco.com/c/en/us/products/routers/829-industrial-router/index.html>.
- [3] Remote control: Lora offers a cheaper link to the iot. <http://www.reuters.com/article/us-tech-communications-lora-idUSKCN10E2TE>.
- [4] Netgear fs750t2 switch. <https://www.netgear.com/business/products/switches/smart/FS750T2.aspx>.
- [5] Ns-3: a discrete-event network simulator. <https://www.nsnam.org/>.
- [6] Norton core router. <https://us.norton.com/core>.
- [7] Raspberry pi 3 model b. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [8] Raspberry pi 3. URL <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [9] I. Abdeljaouad, H. Rachidi, S. Fernandes, and A. Karmouch. Performance analysis of modern tcp variants: A comparison of cubic, compound and new reno. In *2010 25th Biennial Symposium on Communications*, pages 80–83, May 2010. doi: 10.1109/BSC.2010.5472999.
- [10] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review*, volume 40, pages 63–74. ACM, 2010.
- [11] L. L. H. Andrew, T. Cui, J. Sun, M. Zukerman, K. T. Ko, and S. Chan. Buffer sizing for nonhomogeneous tcp sources. *IEEE Communications Letters*, 9(6):567–569, Jun 2005. ISSN 1089-7798. doi: 10.1109/LCOMM.2005.1437373.

- [12] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. *SIGCOMM Comput. Commun. Rev.*, 34(4):281–292, August 2004. ISSN 0146-4833. doi: 10.1145/1030194.1015499. URL <http://doi.acm.org/10.1145/1030194.1015499>.
- [13] N. Beheshti, Y. Ganjali, R. Rajaduray, D. Blumenthal, and N. McKeown. Buffer sizing in all-optical packet switches. In *2006 Optical Fiber Communication Conference and the National Fiber Optic Engineers Conference*, pages 3 pp.–, March 2006. doi: 10.1109/OFC.2006.215714.
- [14] N. Beheshti, Y. Ganjali, A. Goel, and N. McKeown. Obtaining high throughput in networks with tiny buffers. In *2008 16th International Workshop on Quality of Service*, pages 65–69, June 2008. doi: 10.1109/IWQOS.2008.13.
- [15] Neda Beheshti, Yashar Ganjali, Monia Ghobadi, Nick McKeown, and Geoff Salmon. Experimental study of router buffer sizing. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, pages 197–210, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-334-1. doi: 10.1145/1452520.1452545. URL <http://doi.acm.org/10.1145/1452520.1452545>.
- [16] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.
- [17] Lawrence S. Brakmo and Larry L. Peterson. Tcp vegas: End to end congestion avoidance on a global internet. *IEEE Journal on selected Areas in communications*, 13(8):1465–1480, 1995.
- [18] Rachna Chaudhary, Shweta Sethi, Rita Keshari, and Sakshi Goel. A study of comparison of network simulator-3 and network simulator-2. *IJCSIT) International Journal of Computer Science and Information Technologies*, 3(1):3085–3092, 2012.
- [19] Bryce Cronkite-Ratcliff, Aran Bergman, Shay Vargaftik, Madhusudhan Ravi, Nick McKeown, Ittai Abraham, and Isaac Keslassy. Virtualized congestion control. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 230–243, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4193-6. doi: 10.1145/2934872.2934889. URL <http://doi.acm.org/10.1145/2934872.2934889>.
- [20] A. Dhamdhare, H. Jiang, and C. Dovrolis. Buffer sizing for congested internet links. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 2, pages 1072–1083 vol. 2, March 2005. doi: 10.1109/INFCOM.2005.1498335.
- [21] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Part iii: Routers with very small buffers. *SIGCOMM Comput. Commun. Rev.*, 35(3):83–90, July 2005. ISSN 0146-4833. doi: 10.1145/1070873.1070886. URL <http://doi.acm.org/10.1145/1070873.1070886>.

- [22] Erricson. Internet of things forecast. URL <https://www.ericsson.com/en/mobility-report/internet-of-things-forecast>.
- [23] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1:14, 2011.
- [24] The Linux Foundation. Tcp probe, July 2016. URL <https://wiki.linuxfoundation.org/networking/tcpprobe>.
- [25] Google. Cloud ai: Fast, large scale and easy to use ai services. URL <https://cloud.google.com/products/machine-learning/>.
- [26] Sergey Gorinsky, Anshul Kantawala, and Jonathan Turner. Simulation perspectives on link buffer sizing. *SIMULATION*, 83(3):245–257, 2007. doi: 10.1177/0037549707081179. URL <https://doi.org/10.1177/0037549707081179>.
- [27] Luigi A. Grieco and Saverio Mascolo. Performance evaluation and comparison of west-wood+, new reno, and vegas tcp congestion control. *SIGCOMM Comput. Commun. Rev.*, 34(2):25–38, April 2004. ISSN 0146-4833. doi: 10.1145/997150.997155. URL <http://doi.acm.org/10.1145/997150.997155>.
- [28] Synergy Research Group. Cloud market keeps growing at over 40share, October 2017. URL <https://www.srgresearch.com/articles/cloud-market-keeps-growing-over-40-amazon-\still-increases-share>.
- [29] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.
- [30] G. Hasegawa, K. Kurata, and M. Murata. Analysis and improvement of fairness between tcp reno and vegas for deployment of tcp vegas to the internet. In *Proceedings 2000 International Conference on Network Protocols*, pages 177–186, 2000. doi: 10.1109/ICNP.2000.896302.
- [31] Keqiang He, Alexis Fisher, Liang Wang, Aaron Gember, Aditya Akella, and Thomas Ristenpart. Next stop, the cloud: Understanding modern web service deployment in ec2 and azure. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 177–190. ACM, 2013.
- [32] Keqiang He, Eric Rozner, Kanak Agarwal, Yu Jason Gu, Wes Felter, John Carter, and Aditya Akella. Ac/dc tcp: Virtual congestion control enforcement for datacenter networks. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 244–257. ACM, 2016.
- [33] J Heinanen and R Guerin. Rfc 2697—a single rate three color marker. *The Internet Society*, 1999.

- [34] Stephen Hemminger. Tcp probe source coe. URL https://elixir.bootlin.com/linux/v4.15.18/source/net/ipv4/tcp_probe.c.
- [35] IBM. Watson machine learning: Create, train, and deploy self-learning models. URL <https://www.ibm.com/cloud/machine-learning>.
- [36] James F Kurose and Keith W Ross. *Computer networking: a top-down approach*, volume 7. Addison-Wesley Reading, 2016.
- [37] Long Le, Jay Aikat, Kevin Jeffay, and F. Donelson Smith. The effects of active queue management on web performance. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 265–276, New York, NY, USA, 2003. ACM. ISBN 1-58113-735-4. doi: 10.1145/863955.863986. URL <http://doi.acm.org/10.1145/863955.863986>.
- [38] Vishal Misra, Wei-Bo Gong, and Don Towsley. Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 151–160, New York, NY, USA, 2000. ACM. ISBN 1-58113-223-9. doi: 10.1145/347059.347421. URL <http://doi.acm.org/10.1145/347059.347421>.
- [39] J. Mo, R. J. La, V. Anantharam, and J. Walrand. Analysis and comparison of tcp reno and vegas. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1556–1563 vol.3, Mar 1999. doi: 10.1109/INFCOM.1999.752178.
- [40] R. Morris. Tcp behavior with many flows. In *Proceedings of the 1997 International Conference on Network Protocols (ICNP '97)*, ICNP '97, pages 205–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8061-X. URL <http://dl.acm.org/citation.cfm?id=850935.852431>.
- [41] R. Morris. Scalable tcp congestion control. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 3, pages 1176–1183 vol.3, Mar 2000. doi: 10.1109/INFCOM.2000.832487.
- [42] T. J. Ott, T. V. Lakshman, and L. H. Wong. Sred: stabilized red. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1346–1355 vol.3, Mar 1999. doi: 10.1109/INFCOM.1999.752153.
- [43] Teunis J Ott, TV Lakshman, and Larry H Wong. Sred: stabilized red. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1346–1355. IEEE, 1999.

- [44] R. S. Prasad, C. Dovrolis, and M. Thottan. Router buffer sizing for tcp traffic and the role of the output/input capacity ratio. *IEEE/ACM Transactions on Networking*, 17(5):1645–1658, Oct 2009. ISSN 1063-6692. doi: 10.1109/TNET.2009.2014686.
- [45] Sebastian Rampfl. Network simulation and its limitations. In *Proceeding zum Seminar Future Internet (FI), Innovative Internet Technologien und Mobilkommunikation (IITM) und Autonomous Communication Networks (ACN)*, volume 57, 2013.
- [46] Nageswara SV Rao, Jianbo Gao, and Leon O Chua. On dynamics of transport protocols over wide-area internet connections. In *Complex Dynamics in Communication Networks*, pages 69–101. Springer, 2005.
- [47] Nageswara SV Rao, Qiang Liu, Satyabrata Sen, Don Towlsey, Gayane Vardoyan, Raj Kettimuthu, and Ian Foster. Tcp throughput profiles using measurements over dedicated connections. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pages 193–204. ACM, 2017.
- [48] Steffen Rendle, Dennis Fetterly, Eugene J Shekita, and Bor-yiing Su. Robust large-scale machine learning in the cloud. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1125–1134. ACM, 2016.
- [49] Amazon Web Services. Amazon sagemaker: Build, train, and deploy machine learning models at scale. URL <https://aws.amazon.com/sagemaker/>.
- [50] Amazon Web Services. Amazon ec2 instance types, May 2018. URL <https://aws.amazon.com/ec2/instance-types/>.
- [51] Muhammad Shahzad and Alex X Liu. Every bit counts: fast and scalable rfid estimation. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 365–376. ACM, 2012.
- [52] Curtis Villamizar and Cheng Song. High performance tcp in ansnet. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, October 1994. ISSN 0146-4833. doi: 10.1145/205511.205520. URL <http://doi.acm.org/10.1145/205511.205520>.
- [53] Zhitao Wan, Ping Wang, Jing Liu, and Wei Tang. Power-aware cloud computing infrastructure for latency-sensitive internet-of-things services. In *Computer Modelling and Simulation (UKSim), 2013 UKSim 15th International Conference on*, pages 617–621. IEEE, 2013.
- [54] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524. IEEE, 2004.