Chapter One

Introduction

1.1 Background to the Study

Window-Based Congestion Control is a synthetic best ride, in which, up to 85% of the traffics are controlled by the Transmission Control Protocol{TCP} which are used for transmission, e-mail, browsing, file sharing, e.t.c. The Congestion regulator or control algorithms allows diverse applications to share network bandwidths efficiently without overloading or oversubscribing the links. There will be a congestion when a network link is overloaded by several applications, in which, the buffers will fill up and the packets will be plunged and retransmitted.

Latency is important for applications with a user interface like Web-based search and interactive services, but goodput is important for back-end systems like e-mail or Gmail backups. There are a variety of approaches to share on network bandwidth in order to better adapt to the mix of applications; they correspond to various objectives algorithms for congestion control such as shortest-flow-first, max-min fairness, and so on. When the network is operational, severely congested, this means, when there is a strong demand for bandwidth and numerous applications may wish to use the same lines at the same time, congestion control becomes critical. If the speed of the network delay are both in high level, this is crucial since buffers will quickly fill up before the user can react. Consider two flows that will share the single 200Gb/s blockage link with 225KB of the buffering and a 200-second Real Time Technology Solution (RTT). The buffers are filled in 20 seconds of line rate traffic, however the servers can only alter their rates on a 200-second, the scale of the time that refers to how far it takes in order to assess and

respond to traffic congestions. It may be difficult to control congestion reactively when there is a long reaction time in comparison to the prepared buffer capacity.

This problem only grows worse as RTTs go longer if the link(l) capacity increases, as in a Wide Area Network(WAN), without a matching increase in buffer size[1].

These are especially true in data communication centers this days, where a distinctive search workload movement or flow would complete in less than 10 RTTs on an unloaded 100Gb/s network. Even if there are a few competing flows, it should only take a few dozens of RTTs to complete. Fast congestion control becomes more crucial as link capacity increase and more flows become short-lived. Lastly, that time taken to mitigate congested window should not be affected by the network traffic dynamic nature; To put it another way, a traffic optimization technique must be able to react quickly and this must be quickly as possible in the scenarios, in which a very great number of the flow or load begin at the same time, or ON and OFF traffic, for which internet usage ranges from 1 to 0, as it does to other types of traffic. Because network traffic between different servers will be tightly synchronized as applications become more distributed, such traffic patterns in data centers are not uncommon. Disaggregated storage and large-scale data processing are data applications center which benefit out of the quick congestion control.

These applications demand high quantity and a very small latency, besides they usually include traffic of the network between a closely attached servers, with the significant quantity of flow(f) starting in the same time. In this work, we look at the novel family of rapid window-based congestion control algorithms that network workers can utilize to keep networking and

running at high rates even as link speeds reach thousands of gigabits per second. These can be called “Proactive Explicit Rate Control" Algorithm(PERC algorithms). This will allow them to simultaneously retain the buffers that are close to empty efficiently by using all the links, allowing applications to attain to a very low latency and a very high throughput in general[2]. However, this congestion is most common in African countries, particularly Nigeria.

The Internet is the worldwide computer network that communicates data using packet switching and is always based on the Transmission Protocols or Internet Protocols. It began as a small part of the NSFnet with a few nodes and has grown rapidly over the last two decades.

It now bonds billions of hosts, reaching millions of people, to serves as a worldwide information exchange structure. With the Internet's services, billions of people from all over the world will be able to easily search and access encyclopedic information on any subject, billions of people will be able to interact with one another via email and instant runners, and business will be conducted in new and more effective and effectual ways and the Internet, arguably the most important invention of the twentieth century, has essentially altered our way of life[3]. The Internet's enormous success can be attributed to bettering designs and protocols. The Internet will continue to evolve in order to keep up with improvements in the communication and the constant request for more connectivity and bandwidth.

Internet has grown into a massive network, diverse, circulated system of unprecedented difficulty. In the layers of the links, there is cable, wireless, satellite and fiber that are links with bandwidths ranging from a few Kbps to Gbps and propagation times ranging from nanoseconds to thousands of milliseconds. Despite the fact that nearly none of them existed when the original internet architecture was created or constructed, Ethernet Local Area Networks, token ring, ATM,

SONET and FDDI are all used at the same time. New applications are constantly appearing in the application layer, such as multi-player network gaming, the WWW, streaming multimedia, peer-to-peer file sharing and so on. While every ISP is motivated by profit, the Internet inter-domain and topology routing become significantly more difficult and complex to grasp. The Internet is a huge challenge for networking experts to describe and evaluate since it is an expanding difficult system with unparalleled scale[4].

Internet's invention and progress is the outcome of an engineering project cycle that relies heavily on heuristics, intuitions, experiments and simulations. Formulating theories for such a complicated empirical system subsequently appears to be impossible at first appearance, which is one of the reasons why Internet theories lag behind its implementations. However, tremendous progress has been made in recent years in developing strong mathematical foundations for the Internet in a variety of domains, including Internet topology, routing, congestion control, and so on.

Previous Internet research relied mainly on simulations and measurements, both of which have drawbacks. For example, the network measurement cannot predict impacts of new Internet protocols before they are deployed. Due to memory and processor speed limits, simulations can only be used for small networks with simple topologies. We cannot assume that the protocol that works well on a minor network would work well over the Internet. Furthermore, verifying the validity of a mathematical analysis is far easier than verifying the practicality of protocols in large-scale complicated networks[5]. A theoretic structure can substantially aid our understanding of the benefits and drawbacks of current Internet technologies, as well as guide us in the development of new protocols to address recognized issues and future networks. Intuition-based design might easily overlook the importance of some system aspects, resulting in a mediocre

solution or even disastrous implementation. The initial design of the HTTP protocol is a perfect example of a success catastrophe: "The HTTP protocol utilized by the World Wide Web is a perfect example of a success disaster." If its originators had imagined it in usage over the entire Internet and examined the resulting ramifications through analysis or simulation, they could have considerably improved its design, resulting in a more stable Internet today[6]. In some few years ago, internet has seriously grown significantly. The internet now connects millions, if not billions, of people and machines, and it has had a positive and harmful impact on many facets of our life. Residential fiber optic networks and DOCSIS 3.0 are examples of next-generation consumer networks that are pushing the boundaries of what was previously thought to be impossible. New services with ever-increasing bandwidth requirements, such as high-definition video-on-demand, online music, and cloud-based storage services, are, for example, rapidly altering how we use the Internet[7]. CERN and Fermilab, for example, are able to exchange large volumes of data with science research institutions all around the world because to high-speed transatlantic networks. By offering a connection-oriented, dependable, byte-stream service, the Transmission Control Protocol(TCP) has permitted the Internet to survive on a variety of networks, ranging from high-speed, very reliable fiber optic links to error-prone wireless links. Furthermore, Transmission Control Protocol (TCP) provides a critical service to the Internet's stability known as Congestion Control. In any network, congestion control is a critical issue. It is inextricably linked to router and network capacity. Because the Internet is made up of many different networks with different capabilities, it is particularly vulnerable to congestion. Before and after processing, routers must buffer packets in queues. If the rate at which packets arrive exceeds the network's capacity, the router must store them in a queue until it can forward them to the next hop[8]. If the arrival rate exceeds the departure rate, the queue will eventually expand to the point that the router has no

choice but to drop packets, disrupting network traffic. As a result, throughput will drop as the packet loss rate rises. The answer is to reduce router queue occupancy and, as a result, packet loss rates. Sources must reduce their sending rate in order to achieve this, sources must always reduce their transfer rate in order to achieve this. Early TCP versions, on the other hand, did not behave in this manner. Because Transmission Protocol is a dependable procedure, it is expected to retransmit missing segments. However, its initial retransmission behavior was overly aggressive, placing the network at risk of congestion collapse. When a heavily overloaded network achieves a steady state, the actual useful throughput is very low, this is known as a congestion collapse[9].

1.2 Statement of the Problem

Congestion control is the method for improving of the routine of a communication network. This optimization essentially translates to transmitting rates from several data sources being as high as possible while not overburdening the network. Packet losses are the most common indicator of network overload; when the pace of packet arrivals at the link that is more than the capacity, the conforming queue will begins to fill, and when is fully queued, the packets will be discarded and network's bottleneck links will be fully used.

Previous research on window-based congestion management has focused on the starting network, adaptive congestion, and end-to-end mechanisms, but the PERC Algorithm does not account for packet losses and network overloading. As a result, how may network overloading be avoided and packet losses reduced?

1.3 Aim and Objectives

The main purpose of this thesis is to present Proactive Explicit Rate Control (PERC) method for max-min fair rates that accounts for network overloading and packet losses.

The following goals will help to attain the goal:

i. To create a new model/algorithm to deal with Network Overloading.

ii. To use PERC(Proactive Explicit Rate Control) Algorithm to Track Packet Losses.

iii. To ensure that our proposals improve in terms of performance (Simulation).

1.4 Scope of the Study

The goal of this work is to develop a PERC algorithms that will calculates maximum and minimum fair rates prioritizes short flows as needed. ("stateless Proactive Explicit Rate Control" algorithm) is the first algorithm which converges to accurate in a recognized finite time, find the maximum and minimum rates., to the best of our knowledge. This method does not necessitate synchronization of switches or the maintenance of per-flow state.

1.5 Significance of the Study

The main significance this works is to maintain the running of the system close to its evaluated capacity, even when faced with extreme overload. The goal is to provide acceptable service to a limited number of consumers rather than providing poor service to everyone.

1.6 Organization of Report

The theory behind PERC algorithms is that if we separate rate computation from real-time traffic of data and to use clear information, for example to set the flows, the network limits) instead the signal of the congestion, and we can quickly determine optimum rates. In this study, the optimum rate allocation is assumed to be the maximum-minimum fair allocation. The following is how the rest of the work is organized: In Chapter 2, Fair is a per-flow state PERC algorithm. In **Chapter 2**, we describe PERC algorithms and equally discussed network

maximum-minimum fairness. In order to stimulate s-PERC, we look at a variety of PERC algorithms in this thesis. We begin with the Fair PERC algorithm that will congregate to the global maximum-minimum fair amount of allocation using just maximum-minimum fair rate estimates at every level of the link. The fair is quick and very straightforward, the maximum-minimum fair calculation necessitates a per-flow state. **Chapter 3**. These PERC algorithms does not require a per-flow state. n-PERC, or "naive" PERC, is our first attempt to eliminate the per-flow formal, the per-flow state of Fair is changed with a few aggregate variables. As the name implies, n-PERC has temporary difficulties with bottleneck rate computations, and its time of convergence is unbounded. These gives rise to s-PERC, or P "stateless" PERC, which solves the transient issues of n-PERC and converges in a time that can be shown. The key is to be creative. **Chapter 4**. s-PERC for data centers: a review I discuss some of the design decisions that went into making s-PERC viable and deployable, as well as our 4x10Gb/s NetFPGA prototype, we compare the simulation results for s-PERC to RCP, pFabric, and an ideal maximum-minimum rate allocator, focusing on flow completion times (FCTs). For medium to large flows, we find that PERC is extremely near to an optimal maximum-minimum rate allocator, while for the smallest flows, it delivers the shortest flow completion durations conceivable. From our 40Gb/s NetFPGA test platform, we provide genuine measurements of s-PERC, TCP, and DCTCP. **Chapter 5.** Open questions and research possibilities in PERC algorithms are described in future work. It goes over how PERC can be utilized in different networks, such as private Wide Area Networks, and how it can be modified to fulfill different needs. Other algorithms can be configured in programmable switches:

1.7 Operational Definition of Terms

Internet: Internet is a global communication in term of electronic network that links computer networks and structural computer facilities.

Network Overloading: Network Overloading is when you put too many hosts in a broadcast domain.

The Congestion Window: This is the one of the parameters that determines the number of bytes that can be outstanding at any given time is called congestion window.

Congestion Control: Congestion control refer to tactics and systems that can either prevent or alleviate traffic congestions.

Network Congestion: When we have too many communications moving through the internet at the same time, this is known as congestion network. or Congestion of Network occurs when the amount of packets being transmitted across the network exceeds the network's packet handling capability.

Window: Microsoft's Windows operating system is a graphical operating system. It lets users view and save files, run applications, play games, and watch videos, as well as connect to the internet.

Window Based Program: A Windows-based program is software that runs on a computer that runs the Microsoft Windows operating system.

Packet Losses: Packet loss occurs when data packets fail to reach their intended destination on a network.

Internet Protocol: The mechanism or procedure in which data is transmitted from a computer to the other over the internet is known as Internet Protocol. Each computer on the Internet (known as a host) has at one IP address that distinguishes it from all other computers on the network.

Algorithms: An algorithm is a procedure or collection of rules that a computer uses to do calculations or other problem-solving actions. A set of instructions for solving a problem or completing a task is known as an algorithm.

Simulation: Simulation is typically used to evaluate or forecast the existing or future performance of a business process.

Data Communication: Data communication refers to the interchange of information between a sender and a receiver across a transmission means such as a wire connection.

Throughput: Throughput is the total number of packets successfully acquired by the sink node per unit time. In order to design an efficient algorithm, the throughput obtained should be high.

Packet Delivery Ratio: This measure indicates the quantity of packets transmitted to the sink as a percentage of the number of packets created by the source nodes. The technique is considered efficient when the packet delivery ratio approaches 100%. This parameter is the most widely used metric in WSNs.

Packet Loss Rate: This is the ratio of the total number of packets sent to the sink node divided by the rate during which packets are lost or rejected due to buffer overflow.

Fairness: The degree of variety in data sending rate is referred to as fairness. The protocol would benefit from an algorithm that ensures fairness across all source nodes transmitting packets.

Hop-By-Hop Delay: Hop-by-hop delay gauges the algorithms' efficiency in terms of congestion.

Energy Consumption: This is the amount of energy expended by sensor nodes in the WSN during packet transmission, reception, and forwarding is measured by energy consumption.

Buffer Overflow: Buffer overflow occurs when the queue length or the quantity of packets in the network reaches a certain threshold.

Endnotes

1. Lavanya Jose(2019) Proactive Congestion Control a Dissertation Submitted to the Department of Computer Science Stanford University.

2. Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Pias: Practical statistics nagnostic drift scheduling for commodity information centers. IEEE/ACM Transactions on Networking, 25(4):1954–1967, August 2017.

3. Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh the Lam, Carlo Contavalli, and Amin Vahdat. Carousel: Scalable visitors shaping at quit hosts. In Proceedings of the 2017 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '17, pages 404–417, New York, NY, USA.

4. Kolawole I Oyeyinka(2011) Communications and Network, 2011, 3, 85-98 doi:10.4236/cn.2011.32011 Published Online May 2011.

5. Pandey, D.,&Kushwaha, V. (2020). An exploratory learn about of congestion control methods in Wireless Sensor Networks. Computer Communications.

6. Huang, S., Dong, D., & Bai, W. (2018). Congestion control in high-speed lossless records center networks: A survey. Future Generation Computer Systems, 89, 360-374.

7. Majumder, T., Mishra, R. K., Singh, S. S., & Sahu, P. K. (2020). Robust congestion manipulate in cognitive radio community the use of event-triggered sliding mode based on accomplishing laws. Journal of the Franklin Institute.

8. Cui, L., Yuan, Z., Ming, Z., & Yang, S. (2020). Improving the Congestion Control Performance for Mobile Networks in High-Speed Railway with the aid of Deep Reinforcement Learning. IEEE Transactions on Vehicular Technology.

9. Pakdel, H., & Fotohi, R. (2021). A firefly algorithm for energy administration in wireless sensor networks (WSNs). The Journal of Supercomputing, 1-22.

Chapter Two

Literature Review

2.1 Conceptual Review

2.1.1 Congestion of the Internet

When the request for specific or some resources, such as connection bandwidth, surpasses the available capacity, the Internet becomes congested. Long transfer delays, substantial loss of the packet, frequent transmission of the packets and even probable congestion downfall or collapse, in which network channels are completely consumed while throughput obtained by an application is near to zero, are all symptoms of congestion. It is a self-evident that some measures should be in place to protect the network from being significantly overloaded for a lengthy period of time in order to ensure current network performance.

One spontaneous alternative is to make use of the network provisioning to supply additional resources. According to some researchers, high memory, high-speed connectivity, and rapid processors are all advantages that will help solve the traffic problem in computer networks. Despite the fact that bandwidth has expanded tremendously in the previous decade, the need for more bandwidth has remained constant, and new applications, peer-to-peer file sharing, for example, have required far more bandwidth than projected. The need for effective congestion

control techniques has been exacerbated rather than alleviated by the Internet's expanding not solve capacity, despite the fact that we aim to accomplish performance, stability, and fairness in a more heterogeneous environment.

As a result, even in the future high-speed network, congestion control remains a critical topic. Congestion control is the study of developing and analyzing distributed methods for sharing network resources among competing users. The aim is to match demand to available resources in order to avoid congestion and underutilization while also allocating resources fairly. Internet Congestion Control is made up of two parts. The first is a Transmission Control Protocol source algorithm that dynamically adjusts the transmission rate based on congestion along its path. The other one is the router-based Active Queue Management approach, which updates congestion information and transmits it back to sources, either implicitly or explicitly, in the form of packet loss, delay, or marking. We'll take a quick look at a couple of them[1].

2.1.2 Internet Protocol(IP) Networking

The communication systems' layered policy is a modularized method in which each layer at a particular node must know how to connect with the other layers directly above and below it at the particular node, but only with other layers in the same layer at isolated nodes. The Internet Protocol (IP) layer (or networking layer) must, for example, understanding how to transport IP packets using the link-layer, but it does not need to know about the receiver's link-layer.

Layer one is Application
Layer two is Presentation
Layer three is Session

Layer four is Transport
Layer five is Network
Layer six is Link
Layer seven is Physical

Figure 2.1: The OSI networking stacks.

There are seven(7) layers in the Open Systems Interconnection(OSI) standard model, as shown in Figure 2.1 on the left. Physical layer, presentation layer, links layer, transport layer, session layer, networking layer, and application layer are listed in order from bottom to top. The Internet Protocol layer, which parallels to the networking layer of the OSI model, is at its core. IP or Internet Protocol, has the advantage of being able to interact across a variety of networking technologies, including Ethernet, point-to-point lines, and cellular networks[2]. The abstract concept of a connection and a link layer accommodates these many technologies, in the networking stack, the link layer sits right beneath the IP layer and transports IP packets between the nodes that share the same links, there are some distinct link types and transport layer and application layer protocols on the Internet, but they are all utilized with the same packet transport service: The transport layer is part of the Internet Protocol (IP) layer, and the Transmission Control Protocol (TCP) is the protocol of primary interest. TCP divides a data stream into packets, ensuring consistent delivery even when the IP layer loses, reorders, or duplicates packets, while also sensing the network's condition to avoid overloading it. Everything above the transport layer is commonly referred to as "application layer" in the context of IP networking, rather than the building of applications and application protocols, and there is no split into session layer, presentation layer, and so on. The IP layer lacks complex mechanisms for resource

reservation or allocation; when a router or link is overloaded, the default approach is to discard the packets it can't handle and leave it up to the communicating endpoints to sort things out as best they can.

These are crucial design decisions, the network basic is straightforward, but endpoints must be erudite so as to communicate with the network[3]. The end-to-end idea distinguishes this design from many previous network topologies. It differs from the architecture of typical wired telephony systems, in which a mobile device at an end point may be made up of only a dozen analog components[4].

2.2Theoritical Review

2.2.1 Algorithms that are proactive

The Algorithm for "Proactive Explicit Rate Control" (PERC)"determines the max-min rate allocations by exchanging messaging control packets with switches along the path over a few rounds." The control packets contain the drift allocations as an explicit rate. For the max-min fairness aim, we investigate PERC algorithms in this thesis. We begin by describing a series of well-known assumptions regarding PERC algorithm setup before examining the idea of maximum-minimum fairness that allows us to verify statements regarding when the various PERC algorithms will converge. In this chapter, we consider Fair, a simple PERC algorithm that converges to the specific world max-min allocation by doing nearby max-min honest computations at every hyperlink (depending on "demands" of flows). Fair is based on a contemporary method known as d-CPG; however, Fair requires per-flow country at the links, which makes it unfeasible for networks with many flows. The majority of reminiscence in a

switch is committed to routing and forwarding, leaving little memory available for other tasks[5]. Furthermore, manipulating a enormous amount of recollection at nanosecond time scales, which is the current standard for high-speed is difficult[6].

2.2.2 The Models

We simplify the setup for PERC algorithms by making a series of assumptions. Fair and s-PERC, two PERC algorithms, converge to specific maximum-minimum fees in 40N and 60N rounds, where the number of bottleneck links in the highest reliance chain is N. There are molds that are more demanding than the other (e.g., the set of flows is always fixed, a control packet is in no way released). For a realistic deployment (of s-PERC), loosen up the tougher assumptions (marked) as follows:

1. We can generate a network topology from an subjective network are in the form of a graph, G(V; E), where V denotes the set of switches and E signifies the set of links. The topology is fixed, and the capacity of the linkages are defined by C: E R is greater than 0.

2. Network transmits a definite set of flows, F, each of which traverses a subset Pf of the links and each of which is accepted by each link l E. Number of the flows(f) is denoted by J = |F|, and the number of linkages is denoted by K = |E|. Number of flows(f) carried by link is denoted by the shorthand N(l) = |Ql|.

3. From the source end-host to a receiver end-use, flow data packets are sent at rates X: F R0, with the rate of a flow f F limited only by the capacity of links in Pf. As a result, all links l must have fQl X(f) C. (l). There is no constraint on the flow's origin or endpoint.

4. Per flow, there is only one of the outstanding control packet. The fields of the control packet differ from one algorithm to the next. The control packet is sent and received by the sender and recipient over the same Pf connections as the data packets in the flow. As long as the flow is operational, the packet control is gotten and restructured at each of the link[7].

Furthermore, the control packets are strictly adjusted by the links in all of the algorithms described in this thesis; the end host may not certainly adjust the control packet but to inform that a flow is begin or ending. Bandwidth is one of the parameters in the control packet that all connections assign to the flow. The basis end host merely modifies the data packet rate to match the most recent packet's lowest allocation, then returns the packet to its original state.[8]

2.2.3 Fairness (Maximum-Minimum)

Consider the structure in Figure 2.1 as an example of a maximum-minimum fair rate allocation. J = 2 flows and K = 3 linkages are present. For flow fG, which is bottlenecked at link l12, the maximum-minimum fair allocation is 120Gb/s, while for flow fB, which is bottlenecked at link l30, it is 18Gb/s. Because the only flow it transports, fB, is bottlenecked at another link, link l20 has extra capacity.

Definition: Whenever we say flow(f) is bottlenecked at a link(s), what we mean is that link is entirely full and that flow(f)has the highest rate of all the flows in Ql. If and only if every flow is bottlenecked at some point, a rate distribution for flows is said to be max-min fair. There is always a single maximum-minimum fair allocation. In the literature, there are many similar definitions of maximum-minimum fairness, infact, including the following: a reasonable rate distribution If and only if there is no other viable allocation, X is maximum-minimum fair. If Y(f)

is greater than X(f) for flow(f), and some other flow(f) must exist in a way that X(f) is less than or equal to X(f) and Y(f) less than X(f)[9].

Figure 2.1: An example of a setup. M = 2 flows and N = 3 linkages are present. Each flow passes through a subset of the connections. At link l12, the green flow fG is bottlenecked to 12Gb/s, whereas the blue flow fB is bottlenecked to 18Gb/s. The link capacity and fair rate allocations for the flows are shown below.[10].

2.2.4 *A Per-Flow PERC Algorithm with a Max-Min*

Fair is a mechanism which uses per-flow status at every connection to generate maximum-minimum rates. The algorithm may congregates to global maximum-minimum fair rates after 4 numbers(N) of cycles, where N is the number of bottleneck links[11]. Fair is built on past work, as is its dependency-chain-based analysis.[12].

2.2.5 Algorithm for calculating a local maximum and minimum rate

The link then calculates the most recent flow limit rate (Line 6). The flow's limit rate e is the slowest rate it receives some of the other links (as gathered from the control packet). The bandwidth allocated to the flow is a = min(e; b), where e denotes flow bottlenecking elsewhere and b denotes flow bottlenecking at link(l)[13]. Finally, the connection adjusts the flow's local limit rate, bottleneck rate, and allocation, which are all recorded in the flow's control packet. The allocations stored in the control packet are only seen by the end hosts. Overtime, we anticipate bottleneck rates and distributions to converge to maximum-minimum fair rates[14].

2.2.6 The Algorithm's Characteristics

The Fair algorithm has two important Characteristics that do not rely on per-flow state.

1. Each link's bottleneck rate is calculated on a max-min basis. In a limit rate of 40Gb/s for flow fB and e = 1 for flow fG during update 3, allocating the bottleneck rate of 15Gb/s for both flows is max-min fair since they are both bottlenecked at the connection[15].

2. Assume that links "estimthates" a flow to be bottlenecked in B if and only if its limit rate exceeds the local maximum-minimum fair rate. Then, depending on the link's assessment, there could be multiple flows traveling from E to B during a single update. Given the new limit rates, link l30 predicts that after update 3, both flows will be bottlenecked at the link (local max-min fair rates are 15Gb/s). Given a limit rate of 20Gb/s for flow fB, the connection determined fB to be in E prior to the update (e = 20Gb/s is less than the maximum-minimum fair rate of 30Gb/s). As a result, the link not only identified the new flow fG as bottlenecked with update 3, but it is also modified its estimate of the other flow fB from the E to that of B[16].

2.2.7 The Fair Algorithm's Convergence

Theorem 2.5.1 Once the network's set of flows has stabilized, *Fair is always guaranteed to converge to the maximum-minimum fair allocation in fewer than or equal to 4N rounds, where N is the number of bottleneck connections in the maximum-minimum allocation[17].*

The evidence is based on instruction on the linkages in increasing order of their maximum-minimum rates. As a result, we begin by recursively defining what it means for a connection to converge.

2.3 Review of Empirical Studies

2.3.1 Results of the Simulation

A small-scale packet-level was simulated in the Fair algorithm to test 3 primary assertions regarding PERC algorithms' improvements over reactive algorithms and an earlier PERC algorithm we'll name Charny. We compare and contrast Fair and Charny before

presenting our assessment. Charny, like Fair, is a flow per state algorithms that have been proved to converge. At the changeover, the bottleneck rate is established differently in Charny than in F air[18]. A link l in generates a bottleneck rate that is consistent with E(l) estimates but may necessarily fair and keeps track of whether each flow is in B(l) of E(l). The Charny control packet also has a single rate field that is updated by each connection in the flow's path. When a bottleneck changes during the Charny method, the previous bottleneck connection must update (increase) the rate contained in the packet before the new bottleneck link can fill in its rate, resulting in sluggish updates. To test three main claims regarding PERC and reactive algorithms[19], we conduct the following experiments:

(1) **Time of Convergence**: Fair converges quicker than other state-of-the-art PERC algorithms, and PERC algorithms in general converge faster than reactive algorithms. Fair's Convergence Times were compared to the reactive algorithms RCP and Charny.

(2) **Slow convergence** periods result in long Flow in the Completion Times, especially at high speeds and with longer round-trip delays. FCTs of F air were compared to reactive FCTs. Graph 2.5: One TOR is connected to four hosts via 100Gb/s active and non-active links for convergence time studies. RCP and DCTCP algorithms, as well as Charny's PERC algorithm.

(3) Chains of the Dependency: In the decency chain, the algorithms take lengthier to converge as the dependence chains between flows grow longer; nevertheless, some reactive algorithms execute poorer than the PERC algorithms.

We arrived at the conclusions using OMNET++ implementations of reactive (RCP), PERC (Fair, Charny), optimal algorithms for maximum-minimum rate allocation, and ns2 simulations of DCTCP[20].

2.3.2 Scheduling in the Network

Due to potentially uneven speeds or busy connections, data is frequently queued in buffers21 when devices join. Data in a well-behaving buffer will drain between peaks, whereas data in a badly-behaving buffer would remain, wasting space required to process new packets and causing interruption to the program for no gain.[21].

Five general techniques to promoting good buffering are as follows:

1. delaying packets to increase interleaving packets from various sources or match upstream and downstream speeds;

2. rejecting packets to indicate senders to slow down;

3. asking senders to slow down;

4. choosing different load-balancing routes, and

5. separating packets into categories[22].

This section begins at the bottom layer and shows how the five broad tactics listed above have been applied to the network. In addition, the section examines how well they achieve the five network design goals indicated in the abstract.

The aims are to:

1. decrease the latency,

2. increase the bandwidth, and

3. resources are shared in accordance with agreements.

Allow for progressive implementation, and cut down on administrative costs[23].

When tasks done in hardware must be performed in software layers (e.g., ensuring end-to-end delivery over various networks), it will almost certainly lose the end-to-end debate. When two feasible solutions compete for end-points and middle-boxes, the end-point solution will be more cost-effective and durable in the face of changing technology[24].

2.3.3 The Link-layers

If some networks' Link-layer technology can limit buffer expansion while maintaining per-hop fairness. The Ethernet switches may have allowed for minimal the link flow management by delivering a command to transferring devices, instructing them to stop transferring data.

Protocols that do not differentiate between senders were implemented using multicast packets. Priorities for Ethernet were also overlooked in the early drafts of the standards[25].

The Priority Flow Control, which overcomes the flaws in the earlier Ethernet command, and Enriched the selection of the transmission, which allows numerous protocols to be used simultaneously, are features of latest Ethernet standards, sometimes known as Data Center Ethernet or Converged Boosted Ethernet. Priorities are established to share each other's unused bandwidth, with the Congestion notification functioning as a cap. Information is given to TCP and other transport layer protocols to assist them in determining the rate of their transmissions. With the advances, even Ethernet will be unable to match the performance and dependability of fiber optics. More advanced technologies include Infiniband and Omnipath. Credit-based flow control in Infiniband and Quantized Congestion Notification in Ethernet are two more advanced link-layer options (QCN).

Despite its success in supercomputing, infiniband still suffers from the blocking and congestion in the procedure of the Parking Problems when multiple flows are present. Despite its success in supercomputing, infiniband still suffers from blocking and congestion in the form of the Parking Lot Problem, which happens when several flows in a network fabric are processed as if they were one. Because infiniband enables hardware-level congestion control, decreasing or eliminating HoL is a hot topic of research[26]. However, Infiniband's congestion control has been

removed due to unpredictable performance, according to an informal survey of DOE labs. While Infiniband has been at the vanguard of high-speed commodity networks, it has not been able to compete due to its increased cost and complexity. The only QCN-enabled hardware for Ethernet's link-layer congestion made the standardization of IEEE Std 802.1Qbb Priority Flow Control, to which objections were voiced on the grounds control that the authors of this study are aware of comes from Mellanox, which is probable because they offer similar features in their Infiniband products[27].

"The promise of [QCN] congestion control that it could generate a stalemate, more bearable," said the editor of the QCN standard in a personal communication in January 2021.

The worst-case scenario of all source packets arriving at the same switch port at the same time must be taken into consideration by the SRP admission control. There can only be 8 sources if a buffer can only be 8 packets deep owing to latency constraints. Even big switches are built up of modules that limit the amount of port contention to a maximum of 64:1. It wouldn't be too horrible if there were a restriction of 64 sources. It's possible that this ratio will improve. Packet pacing is required for TSN, and the Fair Queue qdisc is currently the best pacer for Linux. However, it doesn't appear to be the same as the one described in the TSN standard. The TSN bridge traffic shaper is only required to pace all flows as a whole, not per-flow. For successful pacing, the FQ qdisc, on the other hand, is designed to interleave packets from various flows[28].

2.3.4 Traffic Shaping and Classification

The hardware layer or the software layer above it can shape and classify traffic. The traffic determining throttles bandwidth or rates communications by suspending packets, while categorization allows different tactics, such as shaping, to upset applications selectively. Le Boudec and Thiran's Network Calculus discusses traffic modeling by using Min-plus algebra to

reason about the equations that reflect arrival and service curves. The results reveal that a switch's buffer requirement for actual flows equals the sum of the burst sizes that can arrive simultaneously for a single destination port, regardless of any other parameter. They demonstrate that because traffic is serialized at the first bottleneck, peer network bursts only have to be paid for once. Of course, numerous peer links carrying flows to the same transmit port could make up a fabric's next hop. To give real-time assurances, simple cooperative traffic shaping is employed to limit the size of the transmit queues used in an Ethernet switch. The burstiness induced by the shaping interval is employed in the calculations used to calculate the amount of a reservation's buffer. The extreme delay of a packets are determined by the total of all buffers divided by the switch's speed + a switch-specific multiplexing constant. This bound assumes that the switch has a single huge First In First Out queue and do not have a separate queue for each transmit port, which is serviced round-robin. When the shaping interval is lowered, burstiness, buffer size, and maximum latency are all reduced, but CPU demand is increased. The whole buffer space and all-out packet delay increase in lockstep with the number of clients until the switch's memory limit is reached. Finally, because the worst-case situation is unusual, this form of overprovisioning fails to properly utilize a network's theoretical capacity, and it demands hosts knowing middle-box characteristics, which is impractical[29].

2.3.5 Scheduling of Queues

Typically, queue scheduling methods are considered separately from the other types of resource scheduling. The most prevalent queue schedulers are drop away from Processor Sharing on early time-shared computer systems, beyond First In First Out and its linked "Drop Tail" semantics. With the addition of weights to distinct traffic classes, queue scheduling has advanced

beyond Processor Sharing's basic Round-robin algorithm. Shortest Job First is the ancestor of a second family of queuing disciplines (SJF). Hunger is a problem for SJF and its preemptive variant, Shortest Remaining Time, yet they perform well under workloads with a heavy-tailed distribution and a large number of short flows. There seems to be slight overlap between current queue scheduling research and real-time processor scheduling research. Earliest Deadline Initially (EDF) is mentioned in a few network studies, although they are the exception rather than the rule, and much progress has been done since EDF was first published. Active Queue Management (AQM) manipulates packets based on the state of the queue, unlike a traffic shaper, which treats all packets traveling through it the same way.

The packet is discarded or flagged when the number of packets or bytes in a queue equipped with the Random Early Detection (RED) AQM approaches an upper limit. The packets are left alone if the queue length falls below a certain threshold. In relation to the thresholds between these criteria, RED drops or marks a packet at random in proportion to the length of the queue. RED is commonly supported in switches and can signal TCP to reduce its congestion window, although it is challenging to configure and rarely used in practice[30].

Controlled Delay is a prominent Active Queue Management (AQM) strategy for reducing buffer bloat by limiting the duration packets spend in a queue. By default, Controlled Delay uses a 100ms average RTT and a 5ms delay target. It was designed and tested for Internet edge routers, but it could potentially operate with lower latency networks as well. Controlled Delay main flaw is that it can't control latency unless it's in charge of the slowest link, which is exactly what happens when flows converge someplace else on the network. The Fair Queuing packet scheduler in the Linux kernel works with TCP to set a pacing rate that improves flow interleaving. Because an AQM prevents non-TCP flows from overfilling buffers when an AQM

is employed, AQMs and enhanced TCP congestion control approaches work well together in general. To deliver the largest theoretical value, AQM would need to be widely applied across a network. Even if an AQM like Controlled Delay is only utilized at the network's edge, it prevents users from overfilling their own buffers. Fair Queuing with Controlled Delay or CAKE30 adds crucial features like hashing flows to different queues and timing packets to the AQM+TC system, making it more successful. A traffic shaper must be installed if the network on which Controlled Delay is used does not provide back-pressure (that is, if the network allows new packets to be inserted even if the next hop does not have adequate buffer space), so that Controlled Delay can manage the queue of the slowest connection[31].

Software AQMs must break flows from the aggregate packet stream since a network cannot afford enough hardware queues to keep each flow separate in order to ensure fairness or QoS. This is problematic since it adds to the workload (end points higher in the stack already treat flows independently) and complicates dissection (Network Address Translation). At least under Linux, queuing rules are implemented after packets have been translated on egress but before they are translated on ingress[32].

2.3.6 Protocols for Transport

For ensuring data and information transfer dependability, the most widely used transport protocol is the Transmission Control Protocol (TCP). TCP is used by applications via a socket interface, which hides many of the connection specifics. Other protocol provides similar securities to TCP, but they are very problematic to utilize in practice since middle-boxes frequently block TCP or the Unreliable Datagram Protocol connections (UDP). Middle-boxes have impeded TCP's progress by modifying its headers and option fields in a way that hinders the adoption of new options as allowed by the TCP standard. TCP sustains amidst its insufficient

performance in a variety scenarios because it creates few assumptions about the information of the networks it traverses and can still provide satisfactory performance in the overwhelming majority of instances, as long as packet damage was caused by overcrowding rather than the connection's reliability. TCP uses the idea of using the receiver's acknowledgments (ACKs) as a clock to adhere to the 'conservation of packets' principle, despite the fact that it has no means of knowing how fast a network is or where bottlenecks are. The ACK clock starts with a slow-start phase that determines bandwidth fast by increasing the congestion window a limitation on the amount of data provided every Round-trip-time (RTT). Only if a retransmit timer depending on a strong RTT and variance prediction concludes that the data should really have arrived a long time ago is data resent. In the meantime, the Congestion Avoidance phase responds to a congestion signal once every RTT by shrinking the congestion window and progressively probing for bandwidth. The history of TCP's use of delay is dubious. Although it should be used, it cannot be the only traffic control signal. TCP Vegas was an older delay-based congestion management variant that worked well on its own but struggled with loss-based TCPs when coexisting. Other delay-based protocols followed Vegas, with varied degrees of success; for example, FAST TCP, Compound TCP, and others have not gained widespread use. Nonetheless, both Facebook's New Vegas and Google's various projects are working on delay-based procedures or enhancements[33].

The RACK algorithm is particularly relevant to this dissertation since it makes decisions based on a similar threshold—1.25RT Tmin versus 1.17RT Tmin. RACK, on the other hand, is disconnected from congestion control on purpose. It's solely used to identify when packets go missing. It also has a one-millisecond default and lower bound for its threshold. TCP Santa Cruz modeled switch queuing by adding RFDs over a time interval and dividing by the average packet

service time over that same time interval. There have been no noisy delay measurements to deal with because TCP Santa Cruz was developed on the NS-2 network simulator. PCP, a non-TCP congestion control protocol, uses probing packets to evaluate whether the channel can currently handle a certain load and then uses short, scheduled, elevated bursts to achieve the required throughput. PCP outperforms normal TCP in a number of ways, including responsiveness and packet loss rate, so it rescues from incast after a small amount of packet loss. PCP was never more than a simulated prototype, despite all of these advantages. They point out that existing interconnections among AQMs and TCP senders make traffic control difficult. Relentless TCP decreases the transmission window by the number of segments discarded rather than halves it once per RTT if one or more packets are dropped, because of this simple and direct effect, a traffic controller can easily modify the rate of a flow. Relentless TCP has not been extensively used since it is incompatible with other TCPs[34].

Many tries to generate a TCP alternative have been made, including, RCP, XCP, and many others. The XCP and RCP have a lot of good ideas, they usually run into problems. TCP has a lot of inertia, thus messing with what TCP has already figured out is trivial. Many ideas are on the wrong side of the End-to-End dispute as well. While clean slate designs can provide useful data, it appears that the most useful research is focused on improving rather than replacing TCP. When a sender receives an acknowledgement *(ACK) marked with Congestion Exists (CE), standard ECN support directs them to half their window once every RTT, while DCTCP keeps track of the ratio of CE-marked bytes to total bytes ACKed to determine the amount of congestion. When the congestion ratio is 1, DCTCP half its window, and lesser ratios cause it to back off correspondingly less. DCTCP reverts to normal TCP Reno when the recipient does not support ECN. If the receiver supports ECN but has not been modified to relay ECN with delayed

ACKs, DCTCP will underestimate the level of congestion. Kato produced a DCTCP variant that is only one-sided. The performance of DCTCP is harmed when the receiver has been modified. New TCP header settings may make it possible for a sender to detect tampering with a receiver. Unfortunately, this type of solution might generate issues when intermediate boxes manipulate headers without adequately supporting new or infrequently used options. The major technique for ensuring DCTCP senders interact with modified recipients is currently configuring per-route congestion management. While this works for homogeneous subnets, when communication is necessary between a large number of hosts managed by various companies, it becomes increasingly difficult and infeasible.

Switches must be set to correctly mark ECN in order to use DCTCP. DCTCP is easier to set up than RED, and it can take use of Ethernet devices that enable RED. In many circumstances, DCTCP, on the other hand, is difficult to implement. It's possible that a cloud provider won't be able to force all tenants to use a buffer-friendly TCP, or that activating per-route congestion control on the application side isn't fine-grained enough for the apps that run on their cloud.[35].

In these cases, it's desirable to revert to a behaviour that's as similar to the actual as feasible while imposing less constraints. We presume that existing RTT metrics are appropriate for the task based on our first findings. Congestion control strategies based on existing delay metrics have failed to achieve low bottleneck queue depth on switching or tight latencies distributions akin to DCTCP up to this point. By implementing Explicit Congestion Notice (ECN) or increased time stamping, certain recent TCP congestion management versions reduce latency, increase bandwidth, and allocate the resources more equitably than previous TCPs. It's not always possible to configure switch queues to properly indicate ECN or to isolate low-latency

protocol from aggressive legacy traffic. Changing transmission and reception, changing drivers, and adding new TCP options is a difficult operation36 because to the inertia and variety of networks. While change might be beneficial in some situations, networks are notorious for their aversion to it. Take IPV6, ECN, RED, and FQ CoDel into consideration. Even as hardware and software support becomes more widely available, network managers and application developers fail to adapt their settings quickly (or at all) to take advantage of it. It's important not to undervalue the challenges of implementing a large number of different protocol headers. Because the inconveniences of using IPV4 were insufficient to persuade every client, server, and middle-box to change, it took 20 years for IPV6 to reach 10% adoption (e.g. router, firewall, load balancer, and management system). Increased pressure from address depletion may be sufficient to force widespread change by 202036. RTT-fairness through sub-window adjustments, improved fairness, ultra-low latency with phantom queues, deadline-awareness, minimizing flow completion times, sender-side only DCTCP, application to wireless networks stability enhancements, elimination of Slow Start in conjunction with Data Center Bridging, and various ideas for deploy ability enhancements are among the many enhancements proposed to improve or leverage DCTCP.

Academia isn't the only one attempting to advance DCTCP. The Internet Engineering Task Force (IETF) is debating ways to improve congestion notification and DCTCP, as well as DCTCP's vulnerability to ACK-loss. Apple has enabled ECN in all of its apps, which could lead to more ECN marking in routers and the ability to use DCTCP across the Internet. Those routers, on the other hand, would have to be configured to designate ECN as required by DCTCP while also allowing DCTCP to coexist with other TCP variants. Such a shift should not be predicted. Other data center and Internet congestion control methods have not been supplanted by DCTCP.

To reason about congestion in wide-area networks, the CAIA Delay-Gradient (CDG) TCP uses minimum and maximum RTTs, with a focus on coexistence with loss-based congestion control. It was included in the kernel of Linux 4.2. It's straightforward to set up because it's only a sender-side change. Incast Congestion Control for TCP is one of the few receiver-side congestion control techniques (ICTCP). It indicates that in some situations, such as incast, when numerous servers transmit data to a single client, the receiver is better suited to adjust to or avoid high congestion conditions. Their technique uses TCP Vegas' congestion management, which is more effective at preventing incast than DCTCP. ICTCP was created as a Microsoft Windows driver, allowing it to modify the behavior of virtualized guests covertly. Google's Chrome Project has evaluated the Quick UDP Internet Connection (QUIC) in order to reduce connection and transit latency. It seeks to send an initial payload without the requirement for the several handshakes required by secure TCP connections. Due to Forward Error Correction, QUIC can recover from data loss without incurring retransmission time. Although the original documents said that QUIC would pace packets, the most recent documentation only references CUBIC and New Reno congestion control. The better pacing capabilities of the Fair Queuing packet scheduler are most likely to blame for this shift.[37].

On the other hand, has RTTs that are 4-6 times higher than DCTCP because to the lack of ECN and AQM. Later Remy investigations demonstrate that the Tao protocols come near to matching the performance of an omniscient scheduler, however DCTCP was left out. It will be interesting to see if machine-generated congestion control is practical or if it can lead to new and enhanced congestion knowledge. Despite the fact that Remy develops protocols, Dong et al. contend that, like most TCPs, it explores a space of hardwired reactions to packet level events, and that its performance can suffer when the real network does not meet its assumptions.

Performance-oriented Congestion Control (PCC) is a sender-side TCP modification that adjusts rate and packet pacing based on ongoing experiments conducted with rates ranging from 1% to 5%. PCC has fewer assumptions than traditional congestion management algorithms, but it does have one: it must repeatedly try higher rates, even if those attempts always reduce measured utility. Despite the fact that the PCC has a public prototype, the actual implementation of its packet pacing either uses a whole core for each flow or is fragile dependent on the operating system version or virtualization. A long line of TCP congestion control variants have attempted to keep congestion low while requiring little or no network equipment change or configuration, however most are unable to compete for bandwidth with loss-based TCPs. To some extent, loss-based TCP and some versions, such as CAIA Delay-Gradient (CDG) TCP, can coexist. Others are unable to be practical owing to a lack of maturity or a lack of performance. The implementation of packet pacing in a prototype of Performance-oriented Congestion Control (PCC) raises problems. Lee, et al. present DX, which demonstrates that reliable queue delay measurements may be accomplished even for high-speed networks by changing drivers, adding TCP options, and modifying both senders and receivers. Their congestion response is driven by the ratio of the recorded average queuing time to an estimate of the number of competing flows, resulting in higher utilization and lower latency than DCTCP. Because widespread support for new TCP header options is rare, and different networks cannot be expected to have compatible hosts, the combination of these enhancements is practically impossible to implement in practice. TIMELY uses hardware timestamps, delay gradients, and rate control to control RDMA traffic congestion. While this study does not make a direct comparison to DX or TIMELY, it is reasonable to suppose that TCP Inigo would benefit from improved timestamps as well. TCP Inigo, on the other hand, may be run on any device without requiring any additional work.

2.3.7 Application

Lower layers of the network may be more equipped to control congestion than an application. Because of a unique set of skills, the network stack was created. That is one of the reasons for the existence of the Unreliable Datagram Protocol. When a programmer has control over a large number of end points, he or she can take the most effective measures to reduce network traffic. TCP evolution is similarly tough without breaking it. Not only must new ideas be superior, but they must also be implementable. It seems that application developers will be willing to build their own network stacks, thanks to Google's experimental QUIC protocol and now Transport over UDP. This flexibility comes with a number of drawbacks: encapsulation slows down performance, allows for proprietary solutions, and can fragment networking efforts into even more efforts that must be debugged separately.

Despite the fact that one-sided paradigms are less congested than two-sided paradigms, they nonetheless have significant congestion. In comparison to simple throttling, one-sided paradigms may be ineffectual. Random Access behavior exhibits a fascinating performance inversion phenomenon: a blocking communication and congestion avoidance solution outperforms a hand-optimized for communication overlap approach. The technique that maximizes overlap while avoiding congestion produces the greatest outcomes. They recommend lowering the number of cores used and the rate at which cores send messages, which will result in a 2% improvement in shared processes, a 60% development in fine-grained application benchmarks, and a 17% improvement in the NAS Parallel Benchmarks[38].

2.3.8 Routing

The purpose of this dissertation is to find a solution to the congestion hitches that rise in simple switched networks with full bisection capacity. As a network's design becomes more

intricate, issues with how to appropriately use many routes will surely arise. To increase network utilization, admission control, for instance, will need to be paired with a routing algorithm. B4 is a worldwide software-defined Network from Google. Each site is reduced to a single node on a network, with a single edge linking it to each distant site. Using a specific version of ECMP hashing, all links from a site to a remote site are processed as if they were one. The topology was simplified for scalability considerations. B4 functions with Flow Groups source, destination, and QoS tuples rather than individual applications[39]. In order to achieve scalability, this was also done. B4 associates Flow Groups with a series of Tunnels that represent routes, prioritizing the shortest paths. A Tunnel Group is made up of that mapping and the weights that go with it. The weight/priority is the slope of the Bandwidth Function, which is used to provide QoS. Google attempted to use the LP algorithm, which is an ideal fair sharing method, but it was too sluggish. Instead, they devised a speedier algorithm that provides equal levels of fairness and utilization. It rotates through the Flow Groups, prioritizing preferred tunnels (min cost path with no contention). When many Flow Groups need more bandwidth and must use paths that shared an edge, the method iterates to obtain a fair share value that is then passed to the Flow Groups' Bandwidth Functions, resulting in their fair share ratios. Bandwidth is distributed in stages. The demand of each Flow Group is met, with slack distributed according to Bandwidth Function. After that, the ratios must be quantized to ensure that they match the hardware capabilities. B4 had control over 2700 Flow Groups and 240 Tunnel Groups as of November 12th. The majority of their Traffic Engineering Operations took less than 5 seconds to complete. The Traffic Engineering Optimization approach reduces the time to a third of what it was before. Hedera is a network flow scheduling solution for data centers that uses simulated annealing to centrally schedule long-lived flows and ECMP forwarding for short-lived flows. Natural bandwidth

demands take precedence over QoS requirements[40]. Rather than designating a core switch to each flow, they scale by assigning one to each host. Their algorithm can design 27K hosts and 250K large flows in 100-200 milliseconds with a demand measurement and reallocation frequency of 5 seconds. In most circumstances, Hedera surpasses static ECMP hashing, but it can still be improved. Because Hedera sometimes performs worse than the "ideal" non-blocking switch they employed, it's impossible to compare it to it. In their shuffling experiment, Hedera achieved 86 percent of the bisection bandwidth as the control network[41].

2.3.9 Approaches with Multiple Layers

Traditional traffic shaping may still cause a bottleneck backlog to emerge and packets to be lost, even when complemented by a global routing algorithm that assures pathways are not overloaded. The bottleneck would have to be able to handle the worst-case situation of all flows on that route bursting at the same time. Congestion trees and the Parking Lot Problem affect even lossless networks like Infiniband. Congestion control on Infiniband is disabled in practice since it needs to be adapted to the system's unique traffic patterns. If traffic patterns change, it can have a significant impact on overall throughput [2]introduced pFabric, in which flows are treated as first-class citizens, flow completion time is lowered, and rate control and flow scheduling are separated. End-hosts use a small range of TCP methods to set packet priority and alter rate separately. Switches implement priority scheduling and dropping, and they can handle the earliest deadline first (EDF). pFabric assumes 10Gbps host rates, a fixed retransmit timeout based on 3RTT, and a fixed chronic congestion threshold. Those advantages are unlikely to hold with heterogeneous and changing host rates, which can be expected with progressive upgrades and energy-proportional networks. In simulations, Fabric achieves near-optimal flow completion

speeds, although its architecture allows for starvation, and coexisting priority schemes demand reconfiguration[42].

By automatically setting priority and rates, you may attain end-to-end storage tail latency QoS. Users must supply a Service Level Objective and a trace of access patterns to the global controller, which greedily explores across a reduced search space. While Priority Meister's auto setup makes it theoretically adaptable to a variety of applications, it was only demonstrated for a few simultaneous tasks. Its worst-case latency evaluation is limited to a single load and ignores the interaction of workload arrival and service curves. The controller must adjust rates and priorities as workloads or service curves change. With the exception of highly busy applications, it meets 99.99 percent of latency requirements because it sees the network as a black box. It manages DCTCP, distributed arbitration, and priority scheduling with minimal end-host changes and prioritization scheduling. In PASE arbitrators, pruning and delegation optimizations have an overhead-accuracy trade-off. They assume that the tree topology and traffic in both halves of the tree are the same, and they modify accordingly if this is not the case[43] focuses on minimizing task completion time (i.e. sets of flows linked to waiting customers) with Baraat, recognizing that Shortest Flow is the most efficient. First scheduling isn't significantly better than Fair Sharing when there are a lot of flows per task. They argue that when a task is small, FIFO order is the best option, and when a task is huge, restricted multiplexing is the best option. To make consistent task-level scheduling decisions, switches use a unique task priority assigned by a common entry point. The historical task size distribution of a data center is utilized to determine the threshold governing when a task is considered heavy. QJUMP is a traffic control system that combines traffic shaping, classification, and priority queues[44].

This strategy can guarantee performance, but it isn't ideal. Flows in the same class may cause problems by filling up shared buffers, even though the number of queues is substantially lower than the number of possible types of traffic. Furthermore, there is no guarantee that once a flow reaches a new administrative domain, the owner will use the same traffic classification scheme or priorities.

2.4 Theoretical Framework

2.4.1 The Congestion of Window

Before the World Wide Web, the biggest study of congestion happened when some Scholars prevented the Internet from collapse. It's a terrific example of performing the most basic thing in the most effective way possible. It established a firm foundation for the Internet and established the Transmission Control Protocol (TCP) as one of the most successful protocols ever invented. TCP, despite its success, has a lot of flaws. The statistical multiplexing notion, which is a fancy way of saying, is used by most current networks. "Let's all just go for it when we're ready because we're probably not going to strike each other." There are smarter network solutions out there, but simple hardware has a lot of advantages in terms of cost. On college campuses, a bicyclist flies through a crosswalk at rapid speed, which is a real-life example of statistical multiplexing. In many circumstances, it appears that the rider would be unable to accurately gauge traffic speed or prevent a collision if a vehicle came into their path. The cyclist appears to be staking their life on the fact that the crosswalk is generally unoccupied. When an Ethernet packet is transmitted through a switch with overflowing queues, the story is similar.

Consider a second analogy for how a TCP packet flow behaves. TCP flows are similar to a delivery train with a deaf and blind engineer on board. It is entirely the engineer's responsibility. Collisions are utilized to evaluate whether or not they should slow down or come to a complete

stop. Because the train has a long journey ahead of it and a deadline to meet, the engineer boosts his speed as a general rule. When the train shudders as a result of colliding with anything, the engineer reduces the train's speed to half of what it was before. Buffers are being used in our networks to temporarily hold packets, which is a good thing[45].

2.4.2 Congestion control

The congestion control cam be realized in an internetwork's network layer and transport layer, i.e. a network made up of multiple types of networks, such as internet. The congestion problem was readily established in the late 1980s, and academics working on solutions chose a solution on the transport layer. Because of its self-clocking nature, the transport layer is already capable of dealing with heterogeneous networks, congestion control on this layer makes sense because this layer guarantees that data is delivered reliably, and avoiding congestion is a factor in this. This section explains how congestion control was added to TCP (and SCTP, which uses a congestion control technique that is very similar to TCP's congestion control), as well as how this mechanism could be enhanced in the future.

TCP relies on Multiplicative Decrease and Additive Increase (AIMD). A TCP host must be able to control its transmission rate in order to use AIMD. A basic approach would be to utilize timers with their expiration times adjusted according to the AIMD rate. Unfortunately, keeping timers for a large number of TCP connections is tricky. Instead, according to Van Jacobson, TCP congestion can be intentionally reduced by limiting the transmission window. A TCP connection can only transfer data at the rate of window, where window is the difference between the sending window of the hosts and the window reported by the receiver. A congestion window is used in TCP's congestion control technique. Each TCP connection's TCB stores the current value of the congestion window(cwnd), and the sender's window is bound by min, which is the current

sending window and the latest received receive window. The TCP congestion control's Additive Increase feature increases the congestion window by MSS bytes every round-trip time. This phase is known as the congestion avoidance phase in TCP literature. Once congestion is detected, the Multiplicative Decrease element of the TCP congestion control divides the current value of the congestion window[46].

When a Transmission Control Protocol connection is recognized, the sender host has no idea whether or not the network segment it will utilize to reach the destination is crowded. It will begin with a small congestion window to avoid producing too much congestion. He suggests starting with the bytes window. Because the TCP congestion control scheme's additive rise component increases the congestion window by MSS bytes per round-trip time, the TCP connection may have to wait many round-trip times before being able to utilise the available bandwidth efficiently. This is particularly essential in situations where the bandwidth rttbandwidth rtt product is high. The slow-start method is included in the TCP congestion management strategy to avoid waiting too many round-trip times before reaching a congestion window wide enough to optimally utilize the network. The goal of the TCP slow-start phase is to get the cwnd to an acceptable value as rapidly as possible. Every round-trip time during slow-start, the congestion window will be doubled. A new variable in the TCB is used by the slow-start algorithm: ssthresh (slow-start threshold). The thresh is a guess about the last value of the cwnd that was not congested. It's set up at the sending window and refreshed after every congestion incident.

How congestion is detected is a critical point that any congestion control solution must address. The TCP congestion control scheme's earliest implementations had a straightforward

and practical approach: packet losses indicate congestion. When a network is overburdened, router buffers fill up and packets are dropped. Packet losses in wired networks are mostly driven by congestion. Packets can be lost in wireless networks for a variety of reasons, including transmission failures and other factors unrelated to congestion. To provide a dependable supply, TTCP already detects segment losses. There are two forms of congestion in the TCP congestion management scheme:

• a slight case of congestion If TCP receives three duplicate acknowledgements and conducts a quick retransmit, it deems the network to be minimally congested. If the quick retransmission is successful, it means that only one segment was lost. TCP conducts a multiplicative decrease in this scenario, and the congestion window is divided by two. The updated value of the congestion window is used to set the slow-start threshold.

• a lot of congestion When TCP's retransmission timer expires, it deems the network to be significantly congested. TCP retransmits the initial segment in this situation, and the slow-start threshold is set to 50% of the congestion window. TCP performs a slow-start once the congestion window is reset to its initial value[47].

When there is severe congestion, the congestion window evolves as shown in the diagram below. Slow-start is used by the sender until the initial sections are dropped and the transmission timeout runs out at the beginning of the link. At this point, the ssthresh has been set to half of the current congestion window, and the congestion window has been reset to one segment. Slow-start is used by the sender until the congestion window exceeds the sshtresh, after which the missing segments are resent. When sections are lost and the retransmission timeout lapses, the congestion window widens linearly until congestion avoidance is used.

In data centers and on the Internet, overburdened networks are a constant source of anxiety. The long tail of changes in latency can be costly for businesses, and congestion exacerbated by bufferbloat47 causes suffering for everyday users. In the worse scenario, such as servers in a storage network broadcasting data to the same customer (i.e. incast), network congestion might prevent flows from passing through a busy port on a switch, resulting in performance collapse. In addition, as the number of consumers and storage speeds increase, the network would be put under increased strain. Highly Efficient Computing (HPC) systems commonly execute closely connected simulations that are highly sensitive to delay variability, because global progress is only as fast as the slowest work. Switching to a fine-grained asynchronous programming approach, according to some experts, will address their problems. Independent jobs, on the other hand, have a limited ability to continue working on their own, and regardless of programming style, congestion can influence performance.

This is due to insufficient scheduling technology being used on loads that are greater than it was designed for. Despite the fact that HPC programmers expect fine-grained asynchronous communication to aid48, the X-Stack Program, which investigates ways to make scientific computing achieve scale performance, notes that "congestion control and flow control mechanisms are of significant concern at very large scale..." At the application level, congestion management is insufficient.

2.4.3 Networks of Different Types

Current network technology, regardless of how fast or expensive it is, fails to provide a convincing solution to congestion and guarantees end-to-end functionality (i.e. QoS). Of course, network capabilities range from low-cost and unreliable to high-cost and reliable. One of the most important contributions of the research is TCP Inigo, which is presented in Chapter 4 and

can be deployed on nearly any network because its methods do not require any additional support. Even the most complex and expensive networks, such as computing systems and clusters used for scientific simulation, must use TCP to a certain extent, even if they use faster protocols internally. It should be noted that this dissertation is mostly concerned with wired networks

Although the concepts of prolongation remain valid, wireless science introduces a new set of issues that are outside the scope of this study. The three basic forms of storage networks are network attached storage (NAS), storage area networks (SAN), and distributed file systems. NAS, which uses one or more servers to provide a file system interface across a modern Ethernet network, is the most common and least expensive storage network. Storage arrays connected to a high-performance network like Fibre Channel make more expensive SANs, which appear to a host as a neighborhood system. Wide Area Network (WAN) and Local Area Network (LAN) are the two forms of distributed file systems (LAN). Huge-scale location systems service a large number of consumers and use a wide range of technologies. Local region systems, on the other hand, are intended to provide a high-performance parallel file system for a limited number of users. In general, even if they were designed from the start, all networks evolve over time.

Carriers are under pressure from the market to combine stored products with converging technology that provide reliable, or near-lossless, transportation (e.g. Fibre Channel over Ethernet and Converged Enhanced Ethernet). Adoption of new regulations, on the other hand, can take a decade or more, and it is a slow and uneven process. Priority Flow Control (PFC) was formerly commonplace in business, but it can now cause impasse. The Quantized Congestion Notification was often used to assuage fears of PFC deadlock, however it was rarely used. Even

while it is supported[49], the tons older Explicit Congestion Notification (ECN) standard is generally underutilized. As a result, most Ethernet networks experience packet drops[48].

2.4.4 Mechanisms based on windows

One of the most difficult challenges posed by the Internet's massive growth in terms of both size and number of users was how to provide a simple and efficient distribution of usable network resources. TCP50 is the Internet's most frequently used transport layer protocol today. TCP is mostly utilized by applications that require dependable, in-order packet delivery from a source to a destination. The dynamic window glide manipulate introduced by Van Jacobson is a key component in TCP. TCP, which uses window-based flow control, is currently used by the majority of Internet connections. In TCP, flow control is accomplished by imposing a sliding-window method. The sliding window's size determines the number of bytes (segments) in transit, or those that have been transmitted but not yet acknowledged. The edges of TCP's sliding-window mechanism can be multiplied from both sides, i.e., when a byte is delivered, the window slides from the right-hand side, and when an ACK is received, it slides from the left-hand side[48].

As a result, the window size determines the maximum number of bytes awaiting an ACK. The window size is dynamically modified based on the amount of buffer space available in the receiving TCP buffer. The sender TCP keeps track of the current window by continuing an announced window (awnd) for the purpose of glide control. According to the available buffer space, the awnd prevents buffer overflow at the receiver. However, in the event of network congestion, this no longer addresses buffer overflow in intermediate routers. As a result, TCP's CC mechanism uses a congestion window (cwnd) to implement its CC mechanism, which follows an Additive Increase Multiplicative Decrease (AIMD) coverage[49].

The idea is that if a sender needs to assess the available buffer space in the bottleneck router along the TCP path, it should be able to alter its cwnd without trouble, eliminating buffer overflows both in the community and at the receiver. The issue, on the other hand, is that routers no longer work at the TCP layer, therefore they can't change the window using TCP ACK segments. Only if TCP expects community congestion each time a retransmission timer expires and reacts to network congestion in this way by modifying the cwnd to the changing network conditions can the problem be avoided[50].

As a result, the cwnd adaption is based on the AIMD design, which comprises three excellent phases:

(i) Begin slowly and gradually escalate.

(ii) Additive (linear) increase to minimize congestion.

(iii) Recovering congestion using a multiplicative reduction.

The AIMD coverage limits the number of packets (or bytes) that can be sent at once. The AIMD diagram looks like a seen tooth pattern, with the range of packets increasing (additive amplify phase) until congestion arises, then dropping off as packets are rejected (multiplicative decrease phase)[51].

2.4.5: Begin slowly (Exponential Increase)

Slow-start is one of the algorithms used in TCP's CC. The slow-start approach is based on the idea that the size of cwnd begins with one Maximum Segment Size (MSS) and gradually expands when fresh ACKs arrive. This has the effect of exploring the network's available buffer space. Slow-start increases the size of the cwnd by one MSS each time a TCP segment is ACK-ed, as shown in Figure 2.1[52]. (a). TCP sends one section first (cwnd is one MSS). It sends two segments after getting the ACK for this segment and after a Round Trip Time (RTT), i.e., cwnd

is incremented to two MSSscwnd is incremented to four when the two transmitted segments are ACK-ed, and TCP sends four new segments, and so on. This method starts slowly but grows exponentially, as the name implies. Slowstart, on the other hand, does not last endlessly. As a result, the sender uses a variable called the slowstart threshold (ssthresh), and when the size of cwnd hits this threshold, slow-start is turned off, and the TCP's CC mechanism moves on to the next phase. The initialized size of thresh is bytes. It's also worth noting that the slow-start algorithm is critical for avoiding congestion collapse[53].

2.4.6: Avoiding Congestion (Additive Increase)

TCP incorporates the congestion avoidance algorithm, which limits the expansion of cwnd to follow a linear pattern, in order to slow down the exponential rise of the size of cwnd and so avoid congestion before it occurs. The slow-start phase ends when the size of cwnd exceeds ssthresh, and the additive phase begins. When the entire window of segments is ACK-ed, increment cwnd by one MSS to achieve the linear rise. Each time an ACK is received, the value of cwnd is increased by 1/cwnd. As a result, for each RTT, the cwnd is increased by one MSS.

2.10.3: Recovery from Congestion (Multiplicative Decrease)

When network congestion occurs, cwnd must be reduced to avoid more network congestion and, eventually, congestion collapse. If a sending TCP needs to retransmit a segment, it can only estimate that congestion has occurred. This circumstance can occur in one of two ways:

The Retransmission Time Out (RTO) timer has elapsed, or three duplicate ACKs have been received, and the size of the threshold variable ssthresh is set to half of the current cwnd in both situations. The multiplicative decline algorithm is used to regulate the ssthresh variable. As a

result, if there are multiple RTOs in a row, this technique exponentially decreases the TCP's transmitting rate.

2.4.7. *What is the Cause of Congestion?*

Congestion can be defined in a variety of ways, but in simple words, if the entire sum of demands on a supply exceeds its available capacity for any time interval (Equation 2-1), the source is said to be congested for that interval. Demand > Available Resources (mathematically). Equation. Congestion Issues There are numerous sources in computer networks, including buffers, link bandwidths, processing times, servers, and so on. Packet loss occurs when the buffer house reachable at the destination is considerably less than the buffer house necessary for the arriving traffic for a brief time. Similarly, if the total number of visitors attempting to access the hyperlink exceeds the bandwidth available, the connection will be disabled. When the number of packets dropped into the subnet by the hosts surpasses the carrying capacity of the subnet, all of them are provided, and the number of packets supplied is proportional to the range communicated.

However, as traffic grows above the network's capacity, the routers become overwhelmed and begin dropping packets.

Due to dropped packets and retransmissions, overall performance deteriorates to the point where almost no packets are delivered. Mixed links with particular bandwidths exist for wired networks such as the INTERNET. The bottleneck is the node with the lowest bandwidth along a path from the source to the vacation destination. Typically, congestion occurs in the bottleneck because it receives more information than it is capable of sending out. Packets will be queued and occasionally dropped in this case. As a result, response time will increase, and throughput will suffer as well.

2.4.8. Costs of Congestion

• As the packet arrival price approaches the hyperlink capacity, there are significant queue delays.

• Retransmissions should be performed by the sender to compensate for packets lost due to buffer overflow.

• In the face of significant delays, an unnecessary retransmission by the sender may cause a router to utilize its connection bandwidth to forward unwanted copies of a packet.

• When a packet is dropped along the path to its destination, the transmission capacity utilized at each of the upstream links to advance that packet to the point where it is dropped is squandered.

2.4.9. What is the definition of congestion control?

When the load is low, the number of packets delivered is proportional to the number of packets transmitted, and the latency is essentially constant. When the load hits the community capacity (the knee point), the number of packets transmitted does not grow. Instead, packets will be queued, and the lengthening period will be extended.

2.4.10 Policies to Reduce Congestion:

The development of a congestion manipulate scheme is affected by any computer network architectural or operational parameter that impacts both sides of Equation 2-1. As a result, any sketch choice that has an impact on load (request) or resource allocation can be regarded part of the network's overall congestion management plan. The retransmissions policy addresses how quickly a sender times out and what it sends when it does. A sender who times out quickly and uses Go-Back-N to retransmit all the packets will create a larger burden on the device than a

slower sender who uses Selective-Repeat. The buffering policy is closely related to this. If all out-of-order packets are discarded by receivers, these packets will have to be broadcast again later, increasing the load. Selective-Repeat outperforms Go-Back-N when it comes to congestion control.

• Congestion is also influenced by acknowledgement coverage.

The acknowledgment packets incur extra traffic if every packet is recounted right now. More timeouts and retransmissions might be decreased if acknowledgments are saved and piggybacked onto reverse traffic, but at the cost of a slower response in case of congestion. A tight flow management scheme (for example, a narrow window) minimizes the facts rate and so helps to alleviate congestion, but at the expense of throughput. As can be seen from the examples above, there are numerous compromises to consider when deciding on the range of policy parameters.

• A discard coverage at the community layer is a rule that specifies which packets should be dropped when there is no house in the queue. A good policy can help with traffic control, while a bad policy can make things worse. By dispersing traffic across all lines, a correctly designed routing algorithm can assist avoid congestion. Finally, packet lifetime management refers to the length of time a packet can survive before being deleted. If it is too long, misplaced packets may travel for a long period through the network, and if it is too short, packets may arrive at their destination earlier than expected, resulting in retransmissions.

• The transport layer has the same challenges as the information hyperlink layer, but determining the timeout period is more challenging since the transit time over the network is less predictable than the transit time over a wire between the two routers. Extra packets will be resent unnecessarily if the timeout interval is too short.

2.4.11: Algorithms for congestion control

Algorithm for a Leaky Bucket

Consider a bucket with a small gap in the bottom as an example of what we're talking about. The discharge is constant regardless of what charge water enters the bucket at. When the bucket is full of water, any remaining water pours over the facets and is lost.

Similarly, each network interface has a leaky bucket, and the leaky bucket method has the following steps:

1. When the host wishes to send a packet, it is placed in the bucket.

2. The bucket leaks at a steady pace, indicating that the network interface sends packets at a consistent rate.

3. The leaky bucket transforms sporadic traffic into regular traffic.

4. In practice, the bucket is a finite queue with a finite rate of output.[53]

Algorithm for token buckets

1. Token bucket algorithm is required:

2. Regardless of how bursty the traffic is, the leaky bucket technique enforces output sampling at the average rate.

3. In order to deal with bursty traffic, we'll need a flexible algorithm that won't lose data.

Token bucket algorithm is one such algorithm.

The following are the steps of this algorithm:

1. Tokens are thrown into the bucket at regular intervals.

2. The bucket holds the maximum amount of material.

3. If a packet is equipped, a token from the bucket is removed, and the packet is dispatched.

4. The packet cannot be sent if there is no token in the bucket.[54].

2.4.12 Transport Layer

1. The fourth layer from the top is the transport layer.

2. The transport layer's principal function is to provide verbal data.

Immediately interchange offerings to utility strategies running on outstanding hosts.

3. The transport layer allows software processes to communicate logically running on exceptional hosts

4. Despite the fact that software processes on distinct hosts are no longer physically connected, the application methods layer uses the logical connectivity offered by the transport to communicate with one another.

5. In the stop structures, the transport layer protocols are respected, but not in the start structures.

6. On a laptop network, network applications can employ many protocols.

TCP and UDP, for example, are two transport layer protocols that provide the community layer with a wide range of services.

8. Multiplexing/demultiplexing is supported by all transport layer protocols.

9. It also provides a variety of services, including reliable statistics transmission, bandwidth assurances, and delay guarantees.

10. Every function in the utility layer has the ability to send a message via TCP or UDP.

11. Both of these protocols are used by the software for communication.

12. TCP and UDP will then communicate with the internet layer's net protocol.

13. The transport layer can be studied and written to by the applications.

As a result, we can argue that verbal communication is a two-way mechanism. [55].

2.4.13: he Transport Layer is a service provider.

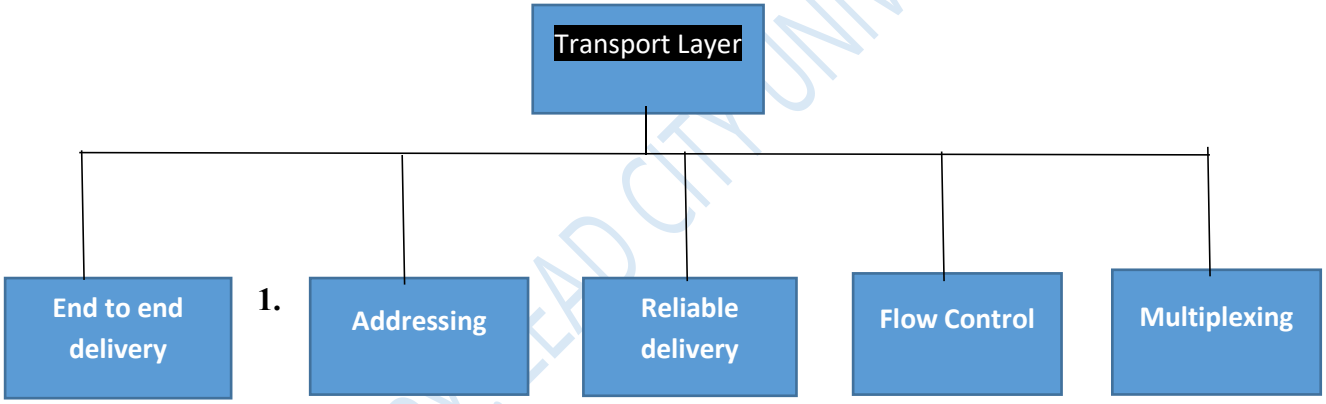
1. Services identical to those provided by data connection layer are provided by the transport layer.

2. In an internetwork, the network layer provides services within a single network, whereas the transport layer provides services across many networks.

3. The data link layer is in charge of the physical layer, while the transport layer is in charge of the lower layers.

The five categories of services supplied by transport layer protocols are as follows::

1. End-to-end delivery
2. Addressing
3. Reliable delivery
4. Flow control
5. Multiplexing



End-to-end delivery: The transport layer sends the full message to its intended recipient. As a result, it assures that a whole message is sent from source to destination.

Reliable delivery: The transport layer ensures that packets are not lost or damaged by retransmitting them.

There are four components to dependable delivery:

a. Error control

b. Sequence control

c. Loss control

d. Duplication control

Error Control (2.12.5)

1. Error Control is the basic function of reliability. In truth, no transmission can be guaranteed to be error-free.

2. Transport layer protocols are intended to ensure that data is transmitted without errors.

3. The data link layer also includes an error-handling system, but it only ensures error-free transport from node to node.

4. End-to-end dependability is not guaranteed by node-to-node reliability.

5. The data link layer examines each network for errors.

If an issue occurs within one of the routers, the data link layer will not be able to detect it. It only detects mistakes that occurred between the beginning and the finish of the link. The transport layer performs end-to-end error checking to verify that the packet is error-free. [56].

2.4.14: Sequence Control

1. Sequence control, which is done at the transport layer, is the second part of reliability.

2. The transport layer is in charge of guaranteeing that the packets received from the upper layers may be utilized by the lower layers on the sending end.

3. It ensures that the various segments of a communication may be decoded on the receiving end.

2.4.15 Loss Control

1. The third aspect of reliability is loss control.

2. The transport layer ensures that all of a transmission's pieces, not just portions of them, arrive at their destination.

3. A transport layer assigns sequence numbers to all transmission pieces on the transmitting end.

4. The receiver's transport layer can use these sequence numbers to identify the missing section. [57].

2.4.16: Duplication Control

1. The fourth aspect of reliability is duplication control.

2. The transport layer ensures that no duplicate data reaches the target.

3. Sequence numbers are used to identify missing packets, as well as to identify and delete duplicate segments by the receiver. [58].

2.4.17: Flow Control

1. Flow control prevents the transmitter from sending too much information to the recipient.

2. If the receiver becomes overloaded, it discards packets and requests retransmission.

3. As a result, network congestion occurs, lowering system performance.

4. Flow control is handled by the transport layer.

5. It employs the sliding window protocol, which improves data transmission efficiency while also regulating data flow to avoid overburdening the recipient.

6. Rather than being frame orientated, the sliding window protocol is byte based.[59].

2.4.18: Multiplexing

1. To boost transmission efficiency, the transport layer employs multiplexing.

One of two methods for multiplexing can be used:

3. Upward multiplexing: Upward multiplexing refers to the usage of the same network connection by numerous transport layer connections.

4. Downward multiplexing: In downward multiplexing, a single transport layer link is used to handle several network connections. The transport layer can use downward multiplexing to divide a connection into numerous channels in order to increase throughput. When networks have limited or slow capacity, this sort of multiplexing is used[60]

2.4.19 Problems of Congestion Control

When the demand exceeds the available resources, congestion arises. As a result, it is anticipated that as resources become more economical, the congestion problem will fix itself. As a result of the congestion, the following concerns will arise:

i. Congestion is linked to a lack of buffer space, which will be addressed once memory is large enough to support large memories.

ii. Slow links are the source of congestion. When high-speed links become available or unoccupied for usage, the problem will be solved.

iii. Slow processors are the source of congestion. When the processors' speed is increased, the problem will be fixed.

Contrary to these principles, without proper protocol redesign, the aforementioned advancements may result in increased congestion and, as a result, reduced window performance. A huge buffer space will not solve the congestion problem. Memory that is less expensive hasn't addressed the congestion situation. Networks with infinite-memory switches have been proven to be just as prone to congestion as networks with low-memory switches[61].

2.4.20 Congestion Problems and Solutions:

In simple terms, if the entire amount of requests on a resource exceeds its available capacity for any time interval, the resource will be congested for that interval.

If the sum of the demand is larger than or equal to the available resources (Demand > Available Resources), then:

There are many resources in computer networks, such as buffers, link bandwidths, processing times, servers, and so on. Packet loss happens when the buffer space available at the destination is less than that necessary for the arriving traffic for a brief period of time. Similarly, if the total

amount of traffic attempting to enter a link exceeds the bandwidth available, the link will become congested. The above definition of congestion is important in distinguishing congestion problems and solutions, despite its simplicity. Depending on the number of resources available, a congestion problem can be described as a single resource problem or a distributed resource problem. Users must provide all of the intelligence required to address the congestion problem if the only resource involved is a dumb resource, such as a LAN medium. CSMA/CD (Carrier Sense Multiple Access with Collision Detection), token access, register insertion, and other LAN access methods are examples of solutions for single, dumb resource congestion. If a resource, such as a name server, is intelligent, it can allocate itself correctly. If the resources are spread, as in a store and forward network, the problem becomes more complicated. When using links as resources, for example, user demands must be regulated so that overall demand at each link is less than its capacity. In this work, we're interested in a group of challenges dealing with distributed resource congestion.

The above simple definition of congestion allows us to divide all congestion schemes into two categories: those that dynamically enhance available resource and those that dynamically decrease demand.

2.4.21 The Different Types of Congestion Issues

(i). ***Schemes for Resource Creation***: Resource Creation Schemes restructure resources in real time to increase their capacity. • One example is dial-up connections that can only be introduced during periods of high usage.

• The bandwidth of satellite links is boosted by increasing their power.

• Path splitting, which sends increased traffic along paths that may not be appropriate in low-stress situations. Users of the resource do not need to be rewarded under any of the above

approaches because they may not even be aware of the network congestion. Congestion problems are solely the responsibility of the network.

(ii). ***Schemes to Reduce Demand***: Demand Reduction Schemes attempt to bring the request down to the amount of resources available. Most of these methods require that the user (or other control points) be informed of the network's load situation so that traffic can be adjusted. There are three types of schemes in this category:

• Service Denial Methodologies: During congestion, these methods prevent new sessions from being started. A typical example of such a strategy is the telephone company's busy tone. Similar approaches are used in connection-oriented computer networks, where congestion at any intermediary node prevents new sessions from starting up.

Service Degradation Schemes: These schemes require all users (both existing and new) to minimize their load. Dynamic window methods demonstrate this strategy, in which users adjust the number of outstanding packets in the network based on traffic.

• Scheduling Schemes: These schemes require users to arrange their demands in such a way that the overall demand is less than the capacity. This technique is exemplified by several contention schemes such as polling, priority, and reserve. All scheduling systems are a subset of the service degradation strategy, it should be noted.

Because beginning a new session on a connectionless network does not necessitate informing all intermediary resources, the service denial strategy cannot be employed successfully. Service degradation and scheduling strategies are commonly used in such networks.

All congestion control, resource generation, and demand reduction strategies may necessitate the network measuring the entire load on the network before taking corrective action. The first element is always referred to as feedback, while the second is referred to as control. A feedback

signal is provided from the congested resource to one or more control points, which subsequently take remedial action, depending on the load. The control point in demand reduction systems is the traffic source node, however in resource creation schemes, the control points could be different network intermediary nodes (or sources). Several reaction mechanisms have been postulated, including:

- Explicit communications are transmitted from the congested resource to the control point in the form of feedback messages. Choke packets, source quench messages, and permits are all terms used to describe these types of messages. When choke packets62 or source quench messages are received, the sources reduce their load, but if they are not received, they raise it.
- • Routing Message Feedback: Each intermediary resource communicates its load level (usually in terms of queue length or delay) to all nearby nodes, which then alter the amount of traffic routed to that resource. This concept is exemplified by the delay adaptive routing that was utilized in ARPAnet at one time. Because the rate of change of delay across a node was significantly faster than the pace at which control could be modified, this method was discovered to generate an excessive number of routing messages.
- • Rejecting Additional Traffic: No explicit messages are issued in this manner. Incoming packets, on the other hand, are either lost or not acknowledged, resulting in a backpressure. As a result, queues form at other nodes, putting backpressure on their neighbors. Backpressure builds up as it moves closer to the source. This strategy is only effective if the congestion is just temporary. Otherwise, the backpressure propagating

throughout the network unfairly affects traffic that isn't even using the overloaded resources.

- Probe Packets: Probe Packets require sources to send probe packets over the network and alter their loads in response to the probe packets' delay.
- Feedback Fields in Packets: By putting feedback in a special in all packets, this solution reduces the overhead generated by feedback messages.
- • Feedback could be included in packets traveling backwards (towards the source of congested traffic) or forwards (towards the destination), with the destination relaying the information to the source. There have also been a few other suggestions for control location:
- Transport Layer: End systems are in the perfect situation to efficiently change the load because they produce it. Dynamic window systems are an example of such controls at the transport layer. Control may be implemented between the burst and the final intermediary systems (entry-to-exit) rather than between the end systems if the network and end systems are under distinct governmental authorities, as in public networks. Access to the Internet: The source node's network layer access controls, similar to traffic lights at highway entrance ramps, allow new traffic to enter the network only if it is not congested. The input limit technique, for example, accomplishes this by imposing suitable restrictions on buffers dedicated to traffic originating at the node and transit traffic.
- Network Layer: If the routers become overburdened, they can take rapid action by limiting the number of packages sent by sources that send more than their fair share. This method is exemplified by the fair queuing system, several buffer class schemes, and the leaky bucket algorithm. These approaches are especially beneficial for public networks

that may not be able to guarantee that end systems will lessen the load on a congestion feedback signal.

- Data Link Layer: Data link level flow control procedures can be used to exert control at the data link level at each hop.[62].

2.4.22 Why Is It So Difficult to Solve the Congestion Problem?

Despite the fact that several congestion-control techniques have been proposed, the search for new ones continues. This area has been the subject of study for at least two decades. This is due to two factors. To begin with, there are prerequisites for congestion control techniques that make finding a good solution challenging. Second, the design of a congestion scheme can be influenced by a variety of network regulations. As a result, a method designed for one network might not operate on a network with a different architecture. This section delves deeper into the topic of prerequisites. Some researchers believe that feedback should only be delivered when there is a low load, and that the absence of feedback indicates a high load. Even such techniques are undesirable due to the fact that network resources are also employed for non-networking purposes. As a result, the resources utilized to handle these extra messages may have been better allocated to these other programs. When the load is minimal and everyone's needs can be met, fairness may not be crucial; nevertheless, when the available resources are insufficient to meet the demand, it is critical that the available resources be allocated equally. By definition, a scheme is fair if all users receive a nonzero share of the resources. Some studies suggested that even if there is no starving, a scheme might be unfair if resources are distributed unequally. Some people wish to prioritize long-distance traffic, while others want to provide equal throughput to all customers.

The plan must be adaptable. The amount of capacity available on a network is continually changing. The available capacity increases or decreases as the number of nodes and links increases or decreases. Demand rises and falls in lockstep with the number of users. It is necessary for the congestion control strategy to dynamically match demand to available capacity. As a result, consumers should be asked to increase consumption when additional capacity becomes available, and to reduce consumption when capacity is reached. The demand curve and the capacity curve should be nearly identical. When there is congestion, transmission failures, out of sequence packets, deadlocks, and lost packets all increase drastically.

2.4.23 The Policies that disturb the Congestion Control

The connecting method is the most important network policy. There are two different types of networks:

(a). Dedicated to making connections

(i). Layer of the Network:

• Queuing and service strategy for packets

• Packet drop strategy for packets

• Packet routing strategy for packets

• Lifetime control strategy for packets

(ii). • Retransmission strategy

• Out-of-order packet caching strategy

• Acknowledgement strategy

• Flow control strategy

• Buffer management approach

• Transport Layer:

(iii). Data Link Layer:

Retransmission strategy at the data connection level

• Service strategy and data connection level queuing

• Packet drop method at data connection level

• Acknowledgement approach for data links at the link level

• Data link level control approach requires an intermediate node in the path to be used to reserve certain resources for a specific time period. The session will not begin if the required resources are unavailable.

In connectionless networks, new sessions can start without preserving resources at intermediate nodes. This allows you to modify the paths of your existing connections on the fly. It is obvious that service denial tactics will work in connection-oriented networks but not in connectionless networks. In connection-oriented networks, path splitting should be set up at session start up time, while in connectionless networks, it can be dynamically launched and canceled throughout a session. Packet queuing and service restrictions in intermediary nodes effect resource distribution among users. Each output connection, each input link, or a combination of the two may have its own queue in an intermediate node. In some networks, each source has its own queue, which ensures that all sources are treated equally. However, this does not assure that consumers from the same source traveling to different destinations are treated equally.

• Maintaining a separate queue for each source-destination combination ensures fairness among users from the source to many destinations. Several methods for efficiently maintaining and servicing such queues have been proposed. One strategy is to serve customers in a round-robin fashion. As a result, queues with large packets will receive more bandwidth than queues

with little packets. There have also been plans presented to address this imbalance. When there is inadequate buffer space in a queue, the packet drop policy determines which packets are dropped. The first packet in the queue, the final packet in the queue (the incoming packet), or a randomly selected packet are some of the options. • The option you choose is determined by the type of application. The older the message is in real-time communications, the less important it is. As a result, it is preferable to place packets towards the front of the line. This traffic is referred to as'milk,' in contrast to file and terminal traffic, which is referred to as 'wine,' because older messages are more valuable than fresh ones. Some have recommended random dropping to ensure justice, but others have questioned its effectiveness. Route selection policies, in general, and path splitting policies, in particular, have an impact on resource allocation and, as a result, network congestion. • In today's networks, even if a parallel high-speed path is clogged, a low-speed path will be completely unused. Path splitting is only done between parallel links connecting the same nodes or between paths of the same speed (one hop).

The length of time a packet spends in the network before being dropped is affected by lifetime control policies. If the lifespan is too short or too lengthy, there may be too many useless retransmissions. The transport protocol's round-trip delay estimate and timeout interval computation techniques have a considerable impact. Indeed, developing a solid method for predicting round-trip time in the presence of packet loss was the first step toward developing a congestion-control solution. Using the mean and variance of the round-trip delay to reduce the probability of false timeout alarms enhances the efficiency of congestion control techniques that use timeouts. Timeout-based congestion methods are affected by the amount of packets retransmitted after a packet loss. The appropriate number may be determined by the destination's out-of-order packet caching policy. If the receiving transport does not cache out-of-order packets,

a single packet loss could result in the entire window being retransmitted. However, a comparison of numerous options revealed that, regardless of the destination's caching policy, it is better to retransmit just one packet if the packet loss is due to congestion. The packet acknowledgment policy has an impact on the time it takes for congestion information to reach the source. There may be too much traffic if every packet is recognized, but the congestion feedback is quick. When some acknowledgments are withheld, the burden caused by acknowledgments is reduced, but congestion feedback is delayed. The design of the congestion control scheme is influenced by the control policy applied at the transport layer. Window-based and rate-based flow control methods are the two most common types. The destination determines the maximum number of packets that a source can deliver in a window-based scheme. This aids in the resolution of the destination's buffer shortage problem. In response to a congestion feedback signal from the network, the source can narrow the window even more. The destination defines a maximum rate in terms of packets per second or bits per second that the source is authorized to deliver in the rate-based system.

The choice between window-based and rate-based follow control systems is influenced by the destination's bottleneck resource. If the destination, on the other hand, has a limited amount of memory, it may want to employ a window-based flow control method to limit the number of packets it may receive at once. Choosing the measure for representing the rate involves similar concerns. Packets per second or bits per second are the options. The rate restriction should be stated in bits per second if the bottleneck or another device in the link has a capacity indicated in bits per second.

2.4.24 What is the difference between web and window applications?

Windows application:

A Windows application is the one that runs on the windows operating system. This can be used to generate graphical user interface forms. Using the Microsoft Visual Studio IDE, we can construct web applications. J#, C#, Visual Basic C++, and other programming languages can be used to accomplish this. The windows applications are on the computer system. Windows-based program is software that runs on a computer that runs the Microsoft Windows operating system. These apps are designed to be installed and run locally on a Windows computer, and they are usually cross-platform compatible. Many Windows-based programs are incompatible with Apple's macOS operating system, while the majority are incompatible with Linux-based systems.

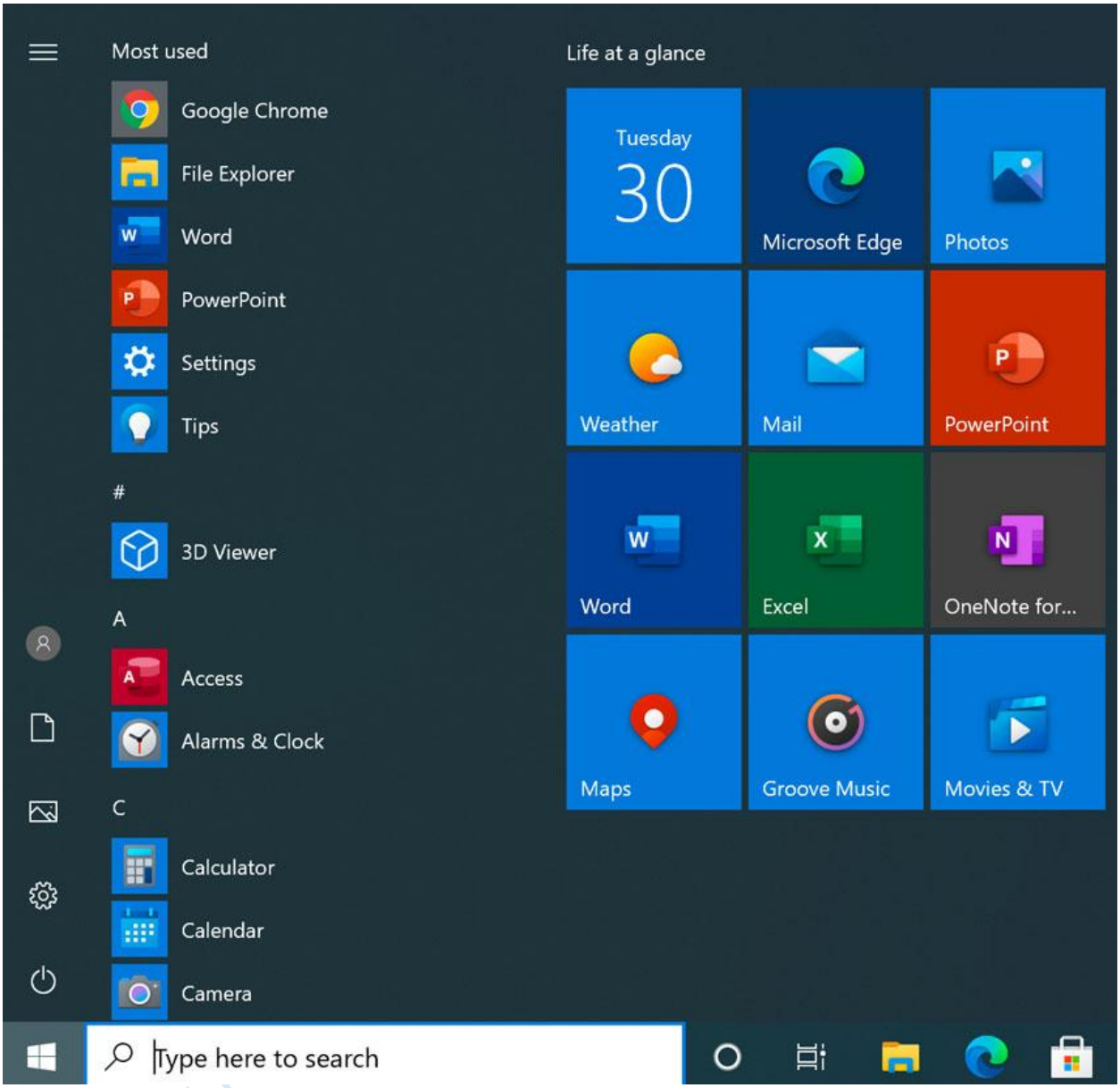


Figure 4: Computer Window

Web application:

A web application is a program that runs on a web browser and utilizes a web server. It takes use of the Internet Information Services (IIS) setup from Microsoft (in developing web applications). .net may be used to create a wide range of web applications. There are a variety of these, ranging from simple HTML pages to complex corporate systems.

Figure 5: Web Application

Table 4: Differences between window applications and web applications

S/N	**Windows Application**	**Web Application**
1	It runs on the Windows operating system and is installed on the Windows platform.	On the web server, the web application is installed.
2	It can only be accessed from the system on which it is installed.	It can be accessible via the internet from anywhere in the world.
3	This application can be run straight from the system's operating system.	To operate web applications, an internet information services server is required.
4	It is only compatible with the Windows operating system.	It runs on a wide range of systems, including Mac, Linux, Solaris, and

		Android.
5	It is unique to bit. It will not work on a 64-bit operating system if it was developed for a 32-bit OS.	The web application is unaffected by the system type.
6	Adobe Photoshop, Adobe ImageReady, Adobe Photoshop, Microsoft Excel, Microsoft Word, and Microsoft Powerpoint are some examples.	Chrome, Internet Explorer, and Firefox are among examples.

Endnotes

1. Niels Moller{2018}Window-Based Congestion Control A Dissertation Submitted To Stockholm, Sweden Isbn-978-91-7178-831-3.

2. I. Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In International Workshop on Protocols for Fast Long-Distance Networks, Lyon, February 2015.

3. N. M¨oller, K. H. Johansson, and H. Hjalmarsson. Making retransmission delays in wireless hyperlinks friendlier to TCP. In IEEE Conference on Decision and Control. IEEE, 2014.

4. Niels Moller{2018}Window-Based Congestion Control A Dissertation Submitted To Stockholm, Sweden Isbn-978-91-7178-831-3.

5. Kanagarathinam, M. R., Singh, S., Sandeep, I., Kim, H., Maheshwari, M. K., Hwang, J., & Saxena, N. (2020). NexGen D-TCP: Next Generation Dynamic TCP Congestion Control Algorithm. IEEE Access, 8, 164482-164496.

6. Patil, J., Tokekar, V., Rajan, A., & Rawat, A. (2020, July). SMDMTS–Scalable Model to Detect and Mitigate Slow/Fast TCP-SYN Flood Attack in Software Defined Network. In 2020 International Conference on Computational Performance Evaluation (ComPE) (pp. 290-295). IEEE.

7. Guan, S., Jiang, Y., & Guan, Q. (2020). Improvement of TCP Vegas algorithm based on ahead route delay. International Journal of Web Engineering and Technology, 15(1), 81-95.

8. Ahmad, M., Ahmad, U., Ngadi, M. A., Habib, M. A., Khalid, S., & Ashraf, R. (2020). Loss Based Congestion Control Module for Health Centers Deployed via Using Advanced IoT Based SDN Communication Networks. International Journal of Parallel Programming, 48(2), 213-243.

9. Zhu, J., Jiang, X., Jin, G., & Li, P. (2020, October). CaaS: Enabling Congestion Control as a Service to Optimize WAN Data Transfer. In International Conference on Security and Privacy in Digital Economy (pp. 79-90). Springer, Singapore.

10. Li, Y., Cao, G., Wang, T., Cui, Q., & Wang, B. (2020). A novel neighborhood region-based dynamic contour mannequin for photo segmentation the use of Bayes theorem. Information Sciences, 506, 443-456.

11. Raman, C. J., & James, V. (2019). FCC: Fast congestion control scheme for wi-fi sensor networks using hybrid gold standard routing algorithm. Cluster Computing, 22(5), 12701-12711.

12. H. Haile, K.-J. Grinnemo, S. Ferlin et al., End-to-end congestion control tactics for high throughput and lowdelay in 4G/5G mobile networks, Computer Networks (2020), doi: https://doi.org/10.1016/j.comnet.2020.107692.

13. Xu, W., Zhou, Z., Pham, D. T., Ji, C., Yang, M., & Liu, Q. (2011). Hybrid congestion manage for high-speed networks. Journal of Network and Computer Applications, 34(4), 1416-1428.

14. Jose, L., Yan, L., Alizadeh, M., Varghese, G., McKeown, N., & Katti, S. (2015, November). High speed networks need proactive congestion control. In Proceedings of the 14th acm workshop on warm matters in networks (pp. 1-7).

15. Leith, D., Shorten, R., & Lee, Y. (2005, August). H-TCP: A framework for congestion control in high-speed and long-distance networks. In PFLDnet Workshop.

16. Najmuddin, S., Asim, M., Munir, K., Baker, T., Guo, Z., & Ranjan, R. (2020). A BBR-based congestion control for delay-sensitive real-time applications. Computing, 102(12), 2541-2563.

17. Huang, H., Sun, Z., & Wang, X. (2020). End-to-End TCP Congestion Control for Mobile Applications. IEEE Access, 8, 171628-171642.

18. Mohammed, Y. A. (2006). Evaluation of TCP Based Congestion Control Algorithms Over High-Speed Networks.

19. Xu, L., Harfoush, K., & Rhee, I. (2004, March). Binary increase congestion manipulate (BIC) for quickly long-distance networks. In IEEE INFOCOM 2004 (Vol. 4, pp. 2514-2524). IEEE.

20. Jamali, S., & Fotohi, R. (2017). DAWA: Defending towards wormhole attack in MANETs through the usage of fuzzy logic and synthetic immune system. The Journal of Supercomputing, 73(12), 5173-5196.

21. Singh, K., Singh, K., & Aziz, A. (2018). Congestion manipulate in wireless sensor networks by way of hybrid multi-objective optimization algorithm. Computer Networks, 138, 90-107.

22. Fotohi, R., & Bari, S. F. (2020). A novel countermeasure technique to shield WSN in opposition to denial-of-sleep assaults the use of firefly and Hopfield neural network (HNN) algorithms. The Journal of Supercomputing, 1-27.

23. Li, W., Zhou, F., Chowdhury, K. R., & Meleis, W. (2018). QTCP: Adaptive congestion manage with reinforcement learning. IEEE Transactions on Network Science and Engineering, 6(3), 445-458.

24. Pakdel, H., & Fotohi, R. (2021). A firefly algorithm for strength administration in wireless sensor networks (WSNs). The Journal of Supercomputing, 1-22. 25. Turkovic, B., Kuipers, F. A., & Uhlig, S. (2019, June). Interactions between congestion control algorithms. In 2019 Network Traffic Measurement and Analysis Conference (TMA) (pp. 161-168). IEEE.

26. Zaminkar, M., Sarkohaki, F., & Fotohi, R. (2021). A method primarily based on encryption and node ranking for securing the RPL protocol communications in the IoT ecosystem. International Journal of Communication Systems, 34(3), e4693.

27. Quwaider, M., & Shatnawi, Y. (2020). Congestion manage model for securing net of things information flow. Ad Hoc Networks, 106, 102160.

28. Jacobson, V. Congestion avoidance and control. ACM Sigcomm Comput. Commun. Rev. 1988, 18, 314–329.

29. Floyd, S.; Henderson, T. The NewReno Modification to TCP's Fast Recovery Algorithm. 1999. Available online: https://tools.ietf.org/ html/rfc2582 (accessed on 21 April 2020).

30. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. ACM Queue 2016, 14, 50:20–50:53.

31. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR Congestion Control. In Proceedings of the IETF 97th Meeting, Seoul, Korea, 13–18 November 2016; Available online: https://www.ietf.org/proceedings/97/slides/slides-97-iccrgbbr congestion-control-02.pdf (accessed on 18 February 2020).

32. Kleinrock, L. Power and deterministic rules of thumb for probabilistic problems in pc communications. In Proceedings of the International Conference on Communications, Boston, MA, USA, 10–14 June 1979; pp. 1–10. Electronics 2021, 10, 615 17 of 18

33. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR Congestion Control: An Update.

34. Mahmud, I.; Kim, G.H.; Lubna, T. BBR-ACD: BBR with superior congestion detection. Electronics 2020, 9, 136.

35. Su, B.; Jiang, X.; Jin, G. Rethinking the fee estimation of BBR congestion control. Electron. Lett. 2020, 56, 239–241.

36. Najmuddin, S.; Asim, M.; Munir, K.; Baker, T.; Guo, Z.; Ranjan, R. A BBR-based congestion control for delay-sensitive real-time applications. Computing 2020, 102, 2541–2563.

37. Do, H.; Gregory, M.A.; Li, S. SDN-based Wireless Access Networks Utilising BBR TCP Congestion Control. In Proceedings of the twenty ninth International Telecommunication Networks and Applications Conference (ITNAC), Auckland, New Zealand, 27–29 November 2019; pp. 1–8.

38. Wei, W.; Xue, K.; Han, J.; Xing, Y.; Wei, D.S.; Hong, P. BBR-based Congestion Control and Packet Scheduling for Bottleneck Fairness Considered Multipath TCP in Heterogeneous Wireless Networks. IEEE Trans. Veh. Technol. 2020, 70, 914–927.

39. Jia, M.; Sun, W.; Wang, Z.; Yan, Y.; Qin, H.; Meng, K. MFBBR: An Optimized Fairness-aware TCP-BBR Algorithm in Wired-cumwireless Network.

40. Cardwell, N. BBR v2: A model-based congestion control. In Proceedings of the ICCRG IETF 104th Meeting, March 2019.

41. Cardwell, N.; Cheng, Y.; Yeganeh, S.H.; Jha, P.; Seung, Y.; Yang, K.; Swett, I.; Vasiliev, V.; Wu, B.; Hsiao, L.; et al. BBRv2: A Model-based Congestion Control. In Proceedings of the IETF 106th Meeting, Singapore, Singapore, 16–22 November 2019.

42. Hock, M.; Bless, R.; Zitterbart, M. Experimental comparison of BBR congestion control. In Proceedings of the International Conference on Network Protocols (ICNP), Toronto, ON,

Canada, 10–13 October 2017.

43. Ma, S.; Jiang, J.; Wang, W.; Li, B. Fairness of Congestion-Based Congestion Control: Experimental Evaluation and Analysis. arXiv 2017, arXiv:1706.09115.

44. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR Congestion Control. In Proceedings of the IETF 97th Meeting, Seoul, Korea, 13–18 November 2016; Available online: https://www.ietf.org/proceedings/97/slides/slides-97-iccrgbbr congestion-control-02.pdf (accessed on 18 February 2020).

45. Van Jacobson. Congestion avoidance and control. In ACM SIGCOMM pc verbal exchange review, quantity 18, pages 314–329. ACM, 1988.

46. Jeffrey Dean and Luiz André Barroso. The tail at scale. Communications of the ACM, 56(2):74–80, 2013.

47. Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the internet. Queue, 9(11):40, 2011.

48. Sally Floyd. Tools for the comparison of simulation and testbed scenarios. https:// tools.ietf.org/html/draft-irtf-tmrg-tools-05, 2005.

49. Mirja Kühlewind, Richard Scheffenegger, and Bob Briscoe. Rfc 7560: Problem announcement and requirements for improved accuracy in explicit congestion notification (ecn) feedback. https://tools.ietf.org/html/rfc7560, 2015.

50. Mahmud, I.; Kim, G.H.; Lubna, T. BBR-ACD: BBR with advanced congestion detection. Electronics 2020, 9, 136.

51. Jia, M.; Sun, W.; Wang, Z.; Yan, Y.; Qin, H.; Meng, K. MFBBR: An Optimized Fairness-aware TCP-BBR Algorithm in Wired-cumwireless Network. In Proceedings of the IEEE

INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, ON, Canada, 6–9 July 2020; pp. 171–176.

52. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR Congestion Control. In Proceedings of the IETF 97th Meeting, Seoul, Korea, 13–18 November 2016; Available online: https://www.ietf.org/proceedings/97/slides/slides-97-iccrgbbr congestion-control-02.pdf (accessed on 18 February 2020).

53. Andrew S. Tanenbaum, David J. Wetherall, " Computer Networks", fifth Edition, Pearson Education Publ. – 2011.

54. Andrew S. Tanenbaum, David J. Wetherall, " Computer Networks", fifth Edition, Pearson Education Publ. – 2011.

55. Andrew S. Tanenbaum, David J. Wetherall, " Computer Networks", fifth Edition, Pearson Education Publ. – 2011.

56. Andrew S. Tanenbaum, David J. Wetherall, " Computer Networks", fifth Edition, Pearson Education Publ. – 2011.

57. Andrew S. Tanenbaum, David J. Wetherall, " Computer Networks", 5th Edition, Pearson Education Publ. – 2011.

58. Andrew S. Tanenbaum, David J. Wetherall, " Computer Networks", 5th Edition, Pearson Education Publ. – 2011.

59. Andrew S. Tanenbaum, David J. Wetherall, " Computer Networks", fifth Edition, Pearson Education Publ. – 2011.

60. Andrew S. Tanenbaum, David J. Wetherall, " Computer Networks", fifth Edition, Pearson Education Publ. – 2011.

61. J. Nagle, \On Packet Switches with Infinite Storage," IEEE Transactions on Communications, Vol. COM-35, No. 4, April 1987, pp. 435-438

62. Raj Jain Digital Equipment Corp. 550 King St. (LKG 1-2/A19) Littleton, MA 01460 Internet: Jain@Erlang.enet.DEC.Com Published in IEEE Network Magazine, May 1990, pp. 24-30.

Chapter 3

Research Methodology

In order to meet the thesis's goals and objectives, we introduced s-PERC, a PERC algorithm for Packet Losses and Network Overloading. We will stimulate s-strategy PERC's of spreading bottleneck rates only when they are high enough using an algorithm dubbed n-PERC. In the worst-case situation, simulation results show that n-PERC can take an indiscriminately long time to converge.

3.1 n-PERC: a Naive PERC Algorithm without Per-Flow State

At the links, the PERC approach removes the flow state. The each connection (l) includes an estimates of flows bottlenecked at the links and somewhere else, which we will call B(l) to differentiate them from true sets. For each connection, the flow's control packet contains information as to whether the flow is in B(l) or not. Per-flow state does not need to be stored by the link.

Only two aggregate values are stored in the PERC algorithm's link: Summation of E(SumE) and Summation of B(SumB) and Number of B (NumB). NumB denotes the number of flows in B(l) that are assigned their exact limit rate, and SumE denotes the total of the flow allocations in(l), where each of the flow is assigned its accurate limit rate. While these SumE and NumB numbers may not produce the correct local maximum-minimum rate, when Equation 2.2 is used to substitute the real SumE and NumB values with Number of B and Summation E:

$$R = \frac{C - \text{Summation of E}}{\text{Number of B}} \quad \text{..... 3.1 Equation}$$

The rate(R) may be used to classify the flow into B and, but only if the flow's most recent limit rate is used. The question is whether the rates will eventually converge to the optimal maximum-minimum rate. Control packet for each link l comprises three components. (Table 3.1): the bottleneck state(s), that indicates whether the flow is in B or not, the allocation a[l], this rate(R) can then be used to classify a flow into B and, but only if the flow's most recent limit rate is used. The question is whether rates will eventually converge into the optimal maximum-minimum rates. The bottleneck status(s), which shows whether the flow is in B or not, the allocation a, and the bottleneck rate b are all included in the packet control for each link l (Table 3.1) and the bottleneck rate b^l.

Table 3.1: n-PERC Flow fG's first control packet, The first and second rows, respectively, provide the information for each connection.

Links	The State of a Bottleneck (s) Allocation (a)	Level of bottlenecking
l_{30}	E @ 0	∞
l_{12}	E @ 0	∞

The packet control is only adjusted by the end host to signify when the flow is complete; the packet control is only adjusted by the links to inform when the flow is complete. It merely adjusts the data packet transmission speed to match the control packet's lowest allocations in this situation, then returns the control packet unchanged.

The link's algorithm (Algorithm 4) is as follows. SumE = 0 and NumB = 0 are the beginning values for a link l. When it observes a flow, it utilizes Equation 3.2 to calculate the flow's bottleneck rate, assuming the flow would be bottlenecked here. This would be for a new flow f.

$$b = \frac{(C - Summation\ of\ E)}{(Number\ of\ B)} \quad \text{…… Equation 3.2}$$

after *NumB* has been incremented.

The bound rate e of the flow is the slowest rate it can get from any other link. If the connection estimates a bottleneck rate that is strictly higher than the flow's limit rate, the flow will be classified as A; otherwise, the flow will be categorised as B. SumE and NumB are then changed based on the new classification. The limit rate e is included in SumE for a flow; otherwise, NumB is used to continue the flow. (5th line) The connection uses the modified SumE and NumB values in the subsequent flow, leading to a new bottleneck rate. The link l adjusts the control packet's bottleneck rate b, allocation a, and bottleneck status field s at the end of the update. Other links (i.e., links other than l) just look at the b[l] field of link l's bottleneck rate, to

appropriately update running estimations NumB and SumE, only link l utilizes the allocation a[l] and bottleneck status s[l] variables. When a flow moves from E to B, the link looks up the flow's previous allocation, for example.

Table 3.2: Algorithm 4: Algorithm for Control Packet Processing

```
# Initialization
cwnd = MSS  # window based congestion in bytes
ssthresh= swin # in bytes


# Ack arrival
```

```
if tcp.ack > snd.una :  # new ack, no congestion
  if  cwnd < ssthresh :
    # networkoverloading : increase very quickly cwnd
    # double cwnd  every rtt
    cwnd = cwnd + MSS
  else:
    # networkoverloading links : increase slowly cwnd
    # increase cwnd by one mss every rtt
    cwnd = cwnd+ mss*(mss/cwnd)
else: # duplicate or old ack
  if tcp.ack==snd.una:    # duplicate acknowledgement
    dupacks++
    if dupacks==3:
      retransmitsegment(snd.una)
      ssthresh=max(cwnd/2,2*MSS)
      cwnd=ssthresh
  else:    # ack for old segment, ignored
    dupacks=0
Expiration of the retransmission timer:
 send(snd.una)     # retransmit first lost packets
 sshtresh=max(cwnd/2,2*MSS)
 cwnd=MSS
```

We believe that over time, the E and B sets at each link converge to true E and B sets, and that the connection's bottleneck rate b converges to the correct maximum-minimum fair allocation for the flows of B. This algorithm is known as n-PERC (nave Proactive Explicit Rate Control).

3.1.1 The PERC Algorithm is a method for calculating the probability of a certain event occurring

Let us look at an example to observe how the n-PERC algorithm's rates change over time. The topology and workload were the same as in the algorithm example, and we have replicated graphic below for your convenience. We'll look at the first four updates because they are sufficient for the debate that follows.

The Flow fB appears for the first time at link l20. The link thinks the flow is unrestricted everywhere, calculates a bottleneck rate of 20Gb/s, and assigns it to the flow. At connection l30, flow fB is visible. The link calculates a bottleneck rate of 30 gigabits per second, determines that the flow is limited to 20 gigabits per second, and assigns 20 gigabits per second. So far, the distributions have been identical to those of the F air algorithm. The link understands that the entire allocation of E flows is 20Gb/s when Flow fG is seen at link l30 during update 3. It uses Equation 2.2 where Number of B = 1 and Summation of E = 20 to determine a bottleneck rate of b = 10Gb/s for flow fG, assuming flow fG is not constrained (e =).

Figure 3.1: An example of PERC in action. J = 2 flows and K = 3 linkages are present. Each flow passes through a subsection of the connections. At link l12, the green flow fG is bottlenecked to 12Gb/s, whereas the blue flow fB is bottlenecked to 18Gb/s. The link capacity and fair rate allocations for the flows are shown below[2].

Round	**Period**	**Occurrence**	**e**	**b**	**a**	**State of the run**
1	1	flow f_B @ Link l_{20}	∞	20	20	B
1	2	flow f_B @ link l_{30}	20	30	20	E
1	3	flow f_w @ link l_{30}	∞	10	10	B
1	4	flow f_w @ link l_{12}	10	12	10	E

l_{20}: Number of B=1, Summation of E=0 f_B: $b[l_{20}]=20$, $b[l_{30}]=30$

l_{30}: Number of B=0, Summation of E =20 f_W: $b[l_{30}]=10$, $b[l_{12}]=12$

l_{12}: Number of B =1, Summation of E =0

2	5	flow f_w @ Link l_{30}	12	10	10	B
2	6	flow f_w @ Link l_{12}	10	12	10	E
2	7	flow f_b @ Link l_{20}	30	20	20	B
2	8	flow f_b @ Link l_{30}	20	15	15	B

l_{20}: Number of B =1, Summation of E =0 f_w: $b[l_{30}]=10$, $b[l_{12}]=12$

l_{30}: Number of B =2, Summation of E =0 f_B: $b[l_{20}]=20$, $b[l_{30}]=15$

l_{12}: Number of B =1, Summation of E =0

3	9	flow f_w @ link l_{12}	15	15	12	E
3	10	flow f_w @ link l_{15}	12	12	12	B
3	11	flow f_b @ link l_{20}	20	18	18	B
3	12	flow f_b @ link l_{18}	18	20	18	E

l_{20}: Number of B =0, Summation of E =18 f_w: $b[l_{30}]=15$, $b[l_{12}]=12$

l_{30}: Number of B =1, Summation of E =20 f_B: $b[l_{30}]=18$, $b[l_{20}]=20$

l_{12}: Number of B =1, Summation of E =0

Figure 3.2: For the packet control updates in the first three sessions of the PERC algorithm.

The flow's computed limit rate e, as well as the control packet state, are displayed (the bottleneck rate b, allocation a, and the bottleneck states after the update). When the connection judges that the flow is bottlenecked and assigns a = b = 10Gb/s while also refreshing the control

packet, we report the link state (NumB; SumE) at the end of each round. This differs from Fair, which calculated a bottleneck rate of b = 15Gb/s and assigned a flow rate of a = b = 15Gb/s. Flow fG's limit rate is e = 10Gb/s when it is next spotted at its true bottleneck link l12 during update 4, which is the lowest bottleneck rate at any other connection. This is less than the bottleneck rate of the link, which is b = 12Gb/s, and the link fails to recognize its bottleneck flow.

Even with challenges, detailed numerical simulations imply that the technique will eventually stabilize, however there is no known time limit.

3.2 The s-PERC algorithm is a stateless PERC with a fixed convergence speed

We sustain an additional variable named MaxE at each link in PERC, which is updated every time l identifies a flow as. At all times, the allocation of the highest (l) flow is at least equal to MaxE. Only when b MaxE is reached do we propagate a bottleneck rate b. What makes you think this will work? The bottleneck is essentially the leftover capacity shared evenly across the B flows after eliminating all allocations C - SumE. The flow in may need to be classed as a B flow if one of the allocations exceeds the bottleneck rate (as in the n-PERC example).

To put it another way, a bottleneck link rate of b MaxE is incompatible with the set of flows we've assumed to be in; as a result, the bottleneck rate is:

$\frac{C - Number\ of\ B}{Summation\ of\ E}$ might be too low to propagate[4].

3.2.1 PERC Algorithm's Variables

Each link l's control packet contains four fields (Table 3.2). The bottleneck state s[l], that indicates whether the flow is in B, the allocations a[l], and the bottleneck rate b[l] in n-PERC are similar. In order for the bottleneck rate on connection l to spread, the packet additionally carries a "ignore" bit i[l], which must be cleared. The notation for s-PERC is as follows:

Table 3.2: Initial Control Packet for Flow fG in s-PERC. The information for each link is in the first and second rows, respectively. fG crosses two links, l30 and l12, and the information for each link is in the first and second rows, respectively.

Link	**Allocation of bottleneck state**	**Rate of bottlenecking**	**Ignore this part**
1_{30}	E @ 0	0	1
1_{12}	E @ 0	0	1

1. The value of a[l] in the f's control packet at time t is called alf, that originally stood for allocation of a flow f at link l at time t. (t). We call the bottleneck state of a flow f at link l at time t the value of s[l] sent in f's control packet at time t slf (t). We'll adopt the convention that t- and t+ refer to the time a little before and just after an update, respectively, assuming the control packet is changed at the link precisely at time t.

2. The set of flows that link l deems to be in or "bottlenecked else" at time t will be referred to as (t). This information is carried in s[l] flow control packets in Ql, in which Ql is the collection of all flows that pass across link l:

$$\hat{E}(t) = \{f : f \in Q_l;\ s^l_f(t) = E\}:$$

3. The collection of flows that link l considers to be in B or "bottlenecked here" at the link l at time t will be referred to as B(t). This data is carried in s in the flow control packets in Ql:

$$B(t) = \{f : f \in Q_l;\ s^l_f(t) = B\}:$$

There are four variables that make up the state at the link. The first two are identical as those in PERC: Summation of E, the total of portions of flows measured bottlenecked elsewhere by link l, and Number of B, the number of flows considered bottlenecked at the link by link l.

MaxE and MaxE0 are two additional variables that are used to evaluation the maximum allocation of a flow that is considered bottlenecked elsewhere by connection[5].

We express the link state of a link l at any time t in terms of the information carried in the control packets of the connection l's flows (these relations are derived from Algorithm 6):

1. SumE(t) is the sum of flow allocations that are considered in or "bottlenecked elsewhere" by link l at time t. This data is saved in the SumE variable at the following link:

$$\text{SumE}(t) = \sum_{F \in \hat{E}(t)} a^{l}_{f}(t)$$

2. The number of flows(NumB(t)) at time t that link l considers to be in B or "bottlenecked here" at the link l. At the link, NumB(t) = |B(t)|, this information is saved in the NumB variable

$$: NumB(t) = |B(t)|.$$

3. M refers to the maximum allotment of an E flow at link l at time t. (t). We don't track this directly at the link as it would require per-flow state. Instead, two variables, m MaxE and MaxE1, are kept at the connection, with MaxE representing an upper constraint on M.

$$M(t) = \max_{F \in \hat{E}(t)} a^{l}_{f}(t)$$

3.2.2 The PERC Algorithm (s-PERC)

The link's algorithm is called every time a control packet is received. The link algorithm additionally repeats the procedure on a regular basis to rearrange the variables MaxE and MaxE1. With the exception of the following changes, the per-packet technique at the link is similar to n-PERC. The rate at which the flow obtains the least propagated bottleneck rate from any other

link is the flow's limit rate e. The limit rate is reached when all other links' ignore bits are set to 1 (i[l] = 1).After the upgrade, the link checks to determine if the bottleneck rate is still too low. The ignore bit is set to True if b MaxE is True, indicating that the rate is too little to broadcast.

The connection requires MaxE and an extra variable, MaxE1, in order to determine a flow's maximum allocation. When a flow is classified as, both MaxE1 and MaxE are updated, so they both climb each time a flow is classified as and assigned a higher value. After each round, they're also reset. Each round begins with the previous round's maximum allocation, whereas MaxE is reset to the old value of MaxE1 to begin with the previous round's maximum allotment. MaxE is an upper restriction to M because we described round as being long sufficient for every flow to be viewed at every connection at least once[6]

3.3 In Action with s-PERC

Let us have a look at sample to observe how the rates of the s-PERC algorithm change over time (Figure 3.4). We will use the same topology and workload as n-PERC and Fair, which we've duplicated below for your convenience.

An s-PERC arrangement in action is shown in Figure 3.3. There are J = 2 flows and K = 3 links. A subset of the links is traversed by each flow. The green flow fG is limited to 12 Gb/s at connection l12, while the blue flow fB is limited to 18 Gb/s. Below are the link capacity and fair rate allocations for the flows.

Round	**Period**	**Event**	**MaxE**	**e**	**b**	**a**	**state**	Ignore
1	1	flow f_B @ link l_{20}	0	∞	20	20	B	

1	2	flow f_B @ link l_{30}	0	20	30	20	E	
1	3	flow f_w @ link l_{30}	20	∞	10	10	B	T
1	4	flow f_w @ link l_{12}	0	∞	12	12	E	

l_{20}: Number B=1,Sum E=0,Max E=0,MaxE'=0 f_B: b[l_{20}]=20, b[l_{30}]=30

l_{30}: Number B =1,SumE=20,MaxE=20,MaxE'=0 f_W: b[l_{30}]=20(**T**), b[l_{12}]= 12

l_{12}: Number B =1,SumE=0,MaxE=0,MaxE'=0

2	5	Flow f_w @ Link l_{30}	20	12	10	10	B	T
2	6	Flow f_w @ Link l_{12}	0	∞	12	12	B	
2	7	Flow f_b @ Link l_{20}	0	30	20	20	B	
2	8	Flow f_b @ Link l_{30}	20	20	15	15	B	T

l_{20} Number B =1,SumE=0,MaxE=0,MaxE'=0 f_W: b[l_{30}]=10(**T**), b[l_{12}]=12

l_{30} Number B =2,SumE=0,MaxE=0,MaxE'=0 f_B: b[l_{20}]=20, b[l_{30}]=15(**T**)

l_{12}: Number B 1,SumE=0,MaxeE=0,MaxE'=0

3	**9**	**Flow f_w @ Link l_{12}**	**0**	**15**	**15**	**12**	**E**
3	**10**	**Flow f_w @ Link l_{15}**	**0**	**12**	**12**	**12**	**B**
3	**11**	**Flow f_b @ Link l_{20}**	**12**	**20**	**18**	**18**	**B**
3	**12**	**Flow f_b @ Link l_{18}**	**0**	**18**	**20**	**18**	**E**

l_{20}: Number B =0,SumE=18,MaxE=18,MaxE'=0 f_w: b[l_{30}]=15, b[l_{12}]=12

l_{30}: Number B =1,SumE=20,MaxE=12,MaxE'=0 fB: b[l20]=20, b[l30]=18

l_{12}: Number B =1,SumE=0,MaxE=0,MaxE'=0

The packet control changes for the first three(3) rounds of the PERC algorithm are shown in Figure 3.4. The link state (MaxE before the update), the flow's maximum rate e, and the control packet status are all shown (the bottleneck rate b, allocation a, bottleneck state s, and

ignore bit after the update). We show the link state at the end of each round (SumE, NumB, MaxE). At link l20, flow fB appears for the first time. The link makes the assumption that the flow is unconstrained elsewhere and applies a bottleneck rate of 20 to the flow. At connection l30, flow fB is visible. The link calculates a bottleneck rate of 30 gigabits per second, determines that the flow is limited to 20 gigabits per second, and allocates 20 gigabits per second. So far, the updates have been identical to those of the F air algorithm[9].

During update 3, when flow fG is detected at link l30, the link recognizes that the complete authorization of E flows is 20Gb/s. It uses Equation 2.2 (where NumB = 1 and SumE = 20) to determine a bottleneck rate of b = 10Gb/s for flow fG, assuming flow fG is not constrained (e =). The link detects that the flow is bottlenecked at the link and allocates a = b = 10Gb/s because the limit rate e = 1 is higher. The connection considers b is too low to transmit to the next link since MaxE = 20Gb/s is greater than b and sets the ignore bit in the control packet to True. When flow fG is next detected at its actual bottleneck link l12 during update 4, the link ignores the 10 Gb/s bottleneck rate from link l30 and assumes flow fG is not constrained (e =). The link calculates a b = 12Gb/s bottleneck rate. The link properly anticipates that the flow is bottlenecked at the link since the limit rate is higher, and allocates b = 12Gb/s. As a result, link l30 prevents link l12 from propagating its problematic bottleneck rate of 10Gb/s, allowing link l12 to quickly detect its bottleneck flow fG. However, the allotment for flow fG on link l30 is still 10Gb/s. The bottleneck rate at link l30 for flow fG does not increase from b = 10Gb/s to b = 15Gb/s until after update 8 (when both flows are B). b = 15Gb/s exceeds e = 12Gb/s at update 9, and flow fG is correctly labelled E at link l30. Because we can see that the flow fG is always tagged B at its bottleneck link l12, E at link l30, and allocated exactly its max-min fair rate 12Gb/s at both links, we say it has converged at this point. Next, we'll look at flow fB. Note that

at update 8, flow fB is assigned a locally maximum fair rate based on the limit rates at link l30, however the connection does not propagate the bottleneck rate to link l20 since MaxE > b = 15Gb/s. Because no flows at link l30 were designated E, MaxE is reset to 0 at the end of the round. Once flow fG is tagged E after update 9, it stabilizes at MaxE = 12Gb/s. MaxE = 12Gb/s is less than b = 18Gb/s on the following update of flow fB at link l30 (update 11), link l30 not only correctly marks flow fB as B at b = 18Gb/s, but it also propagates the bottleneck rate to link l20. As a result, link l20 accurately records the flow fB E at its max-min fair rate during update 12. Because we can see that flow fB is always tagged B at its bottleneck link l30, E at link l20, and allotted exactly its max-min fair rate 18 Gb/s at both links, we declare it has converged at this point[10].

3.4 Simulations with PERCs Algorithms

We employed n-PERC and s-PERC numerical simulations (in Python) in a fully linked network of K links, where each link has a standard throughput of 10Gb/s and a uniform delay of 10 us (with some random jitter on the order of a nanosecond to allow reordering of packets). For each workload, we generate a set of K long-lived flows at random, with each flow crossing the same number of links P. (For the sake of simplicity). This is not a packet-level simulation, and we merely use an event-queue to perform the computations required for control packet processing at intervals indicated by network delays and timeouts (e.g., every round). We also employ a constant value rather than dynamically altering round based on the amount of flows. For multiple J; K; P combinations, such as J = K = 1000 and J = K = 100, and P 10; 40; 50; 80, as well as different random seeds for packet sorting and flow paths, we examined thousands of alternative network and flow configurations. We confirmed that per bottleneck link rate, s-PERC converges in no more than six rounds. While no simulation failed to converge in n-PERC, we did

find that the convergence time for some setups can take significantly longer than six rounds for each bottleneck link.

The worst case convergence time for four bottleneck links (two unique bottleneck rates) was 60 rounds (Figure 3.5). Figure 3.5a shows a time series of bottleneck rates at the connections for the n-worst PERC situation. In this setup, K = 100 links convey J = 100 flows. We chose a consistent capacity of 10Gb/s and a uniform delay of 10s for all links. P = 80 random links must be traversed by each flow. 880s has been chosen as the round figure. It turns out that all flows are bottlenecked at one of four sites. The bottleneck rates are the same in three of the four links. The n-PERC method takes over 60 rounds to converge all four connections (two unique bottleneck rates), whereas s-PERC converges in just six rounds for the same scenario (Figure 3.5b).

(a) Time series of the fair sharing rates of n-PERC bottleneck links (o) and MaxE (x).

(b) Time series of fair share rates (o) and MaxE (x) at bottleneck connections for the s-PERC[11].

Figure 3.5 shows an example of n-PERC convergence that takes a long time. The fair share rate is defined as (C - SumE=NumB) (-1 when NumB = 0). The faded disks represent the optimal max-min fair sharing rates (o).

3.5 s-PERC Convergence Proof

3.5.1 Algorithms for Centralized Water Filling

The centralised water-filling algorithm and the CPG algorithm are WFk algorithms, which are centralised algorithms that calculate max-min fair rates. A WFk algorithm (see Process 7) is an iterative algorithm in which connections compute a fair share rate (which we call WF rate to distinguish it from the max-min rate) in each iteration, and then the algorithm selects a set

of bottlenecked links to remove from the network. A link is deleted in each iteration if it has the lowest WF rate across all neighbors up to k degrees, where a (first-degree) neighbor of a link l is defined as a link that shares a flow with l in that iteration. As a consequence, k = 1 corresponds to the centralized water-filling algorithm, which removes links with the lowest WF rate among all remaining links in the iteration, and k = 0 corresponds to the CPG algorithm, which removes links with the lowest rate among their first-degree neighbors. When a link is removed from the network, the flows carried by that link in that iteration are given the connection's WF rate, and the link is also removed. The algorithm stops when there are no more flows. As we'll see, all WFk algorithms compute max-min fair rates for flows, and the s-PERC convergence behavior may be characterized in terms of WF2.

Notation: Table 3.3 summarizes the notation used in WFk algorithms. The fair share rate determined by link l in iteration n of some WFk algorithm is denoted as Rnk[l]. To distinguish Rnk[l] from the max-min fair rate, we refer to it as a WF rate rather than a "fair share rate."

Table 3.3: k = 1 for the centralized water-filling algorithm, k = 1 for the CPG algorithm, and k = 2 for the algorithm we introduce to analyze s-PERC is a commonly used notation in the context of a WFk algorithm run for a given set of flows and links, where k = 1 for the centralized water-filling algorithm, k = 1 for the CPG algorithm, and k = 2 for the algorithm we introduce to analyze s-PERC. We also clarify how these new terms relate to those established in 2.1, when we specified the PERC algorithm setup, in brackets. The first set of rows corresponds to the variables of the WFk algorithm[12].

$Rkn[l]$	Fair share rate derived by link l in iteration n (WF rate)
$N^k_n[l]$	With iteration n, the number of flows transported by connection l
$C^k_n[l]$	at iteration n, link l's remaining capacity

$A[f]$	*Flow f has*
$X1^k_n(l)$	*been allotted a certain amount of bandwidth.*
$X2^k_n(l)$	(first-degree neighbors) A group of connections in iteration n that share a flow with l.
	(second-degree neighbors) A group of links that share a flow with X1kn (l)
W^k	(Note that W = W and W1 = D) total number of iterations
L^k_i	set of links removed in iteration i, by convention L^k_0 is the empty set and $L^k_{W\,k+1}$ is the set of links that are never removed (Note that $L\infty_i = L_i$)
LL^k_n	set of links removed in iterations up to and including n, that is, $L^k_{0;\,:::;}\, L^k\, n$
FL^k_n	set of flows carried by links in LL^k_n
$FL^k_n(l)$	subset of flows in FL^k_n carried by link l
LG^k_n	
FG^k_n	Lk n+1;:::; LkW k+1 is the collection of links that survive in iteration n + 1.
$FG^k_{n(l)}$	subset of flows in FGkn carried by link l that do not cross LLkn set of flows carried by links in LGkn that do not cross LLkn

Endnotes

1. Miyazawa, K.; Sasaki, K.; Oda, N.; Yamaguchi, S. Cycle and Divergence of Performance on TCP BBR. In Proceedings of the IEEE International Conference on Cloud Networking (CloudNet), Tokyo, Japan, 22–24 October 2018.

2. Zhang, Y.; Cui, L.; Tso, F.P. Modest BBR: Enabling Better Fairness for BBR Congestion Control. In Proceedings of the IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018.
3. Claypool, S.M. Sharing but not caring-Performance of TCP BBR and TCP CUBIC at the network bottleneck. In Proceedings of the Fifteenth Advanced International Conference on Telecommunications, Nice, France, 28 July—1 August 2019.
4. Song, Y.J.; Kim, G.H.; Cho, Y.Z. BBR-CWS: Improving the Inter-Protocol Fairness of BBR. *Electronics* **2020**, *9*, 862.
5. Tao, Y.; Jiang, J.; Ma, S.; Wang, L.; Wang, W.; Li, B. Unraveling the RTT-fairness Problem for BBR: A queueing model. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.
6. Kim, G.H.; Cho, Y.Z. Delay-Aware BBR Congestion Control Algorithm for RTT Fairness Improvement. *IEEE Access* **2019**, *8*, 4099–4109.
7. Zhang, S. An Evaluation of BBR and its variants. *arXiv* **2019**, arXiv:1909.03673.
8. Yang, M.; Yang, P.; Wen, C.; Liu, Q.; Luo, J.; Yu, L. Adaptive-BBR: Fine-grained congestion control with improved fairness and low latency. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakech, Morocco,15–18 April 2019; pp. 1–6.
9. Sun, W.; Jia, M.; Zhang, G.; Wang, Z. RFBBR: A Rtt Faireness Awared Algorithm Based on BBR. In Proceedings of the 2020 IEEE International Conference on Smart Internet of Things (SmartIoT), Beijing, China, 14–16 August 2020; pp. 124–131.

10. Kim, G.H.; Mahmud, I.; Cho, Y.Z. Fairness improvement of BBR congestion control algorithm for different RTT flows. In Proceedings of the 2019 International Conference on Electronics, Information, and Communication (ICEIC), Auckland, New Zealand, 22 25 January 2019; pp. 1–2. *Electronics* **2021**, *10*, 615 18 of 18

11. Kim, G.H.; Song, Y.J.; Mahmud, I.; Cho, Y.Z. Enhanced BBR congestion control algorithm for improving RTT fairness. In Proceedings of the 2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN), Zagreb, Croatia, 2–5 July 2019; pp. 358–360.

12. Jain, V.; Mittal, V.; Tahiliani, M.P. Design and implementation of TCP BBR in ns-3. In Proceedings of the 10th Workshop on Ns-3, Surathkal, India, 13–14 June 2018; pp. 16 22.

Chapter Four

Analysis of s-PERC

4.1 Algorithm for Packet Controls

This chapter will go through the s-PERC and PERC algorithms and how they regulate Network Congestion and Packet Control. The s-PERC algorithm is versatile enough to be used in a variety of networks. According to s-simulations PERC's for network congestion, it can converge up to 10–15 times quicker than reactive solutions.

Arrivals and departures of the Dynamic Flow: Despite the fact that the flows number in real networks varies, and the s-PERC approach is explained as though the number of flows remains constant. Let's have a look at how s-PERC handles flows and networks as they come and depart. S[l] = E and a[l] = 0, i.e. (bottlenecked somewhere and assigned 0 Gb/s) for every link l in the control packet for a new flow, forcing the link to increase NumB throughout the update. A terminating flow's last control packet will be identified with a F IN so that the flow's contribution to the link's aggregate state can be eliminated (NumB or SumE).

1: Summation of E = 0: the sum of flow sharing out in this sphere. NumB = 0:

2.:number of flows in B in this round in the initial kingdom at hyperlink l

3: MaxE = 0: the majority of flow allocations were moved to in the last round.

4: MaxE1 = 0: in this round, the maximum allocation of flows was moved to

5: SumE1 = 0: sum of flow allocations in view of the previous round. When considering that ultimate round, shadow state

6: NumB1 = 0: range of flows in B.

7: RoundNumber = 1 Reset interval for mixture variables

8: If s[l] is equal to E, then Previously, the flow is bottlenecked somewhere else.

9: a[f] - SumE Update the link kingdom to account for the fact that the flow will be slowed.

10: NumB NumB Plus 1

Algorithm for the Processing of Control Packet

4.2 Hardware s-PERC Prototype

Using 4x10 Gb/s NetFPGA SUME platform, we constructed the s-PERC switch data plane in hardware. The MoonGen DPDK packet processing library is used to implement the end host. The prototype shows that s-PERC can be used with high-speed links. It is used to assess s-performance. PERC'

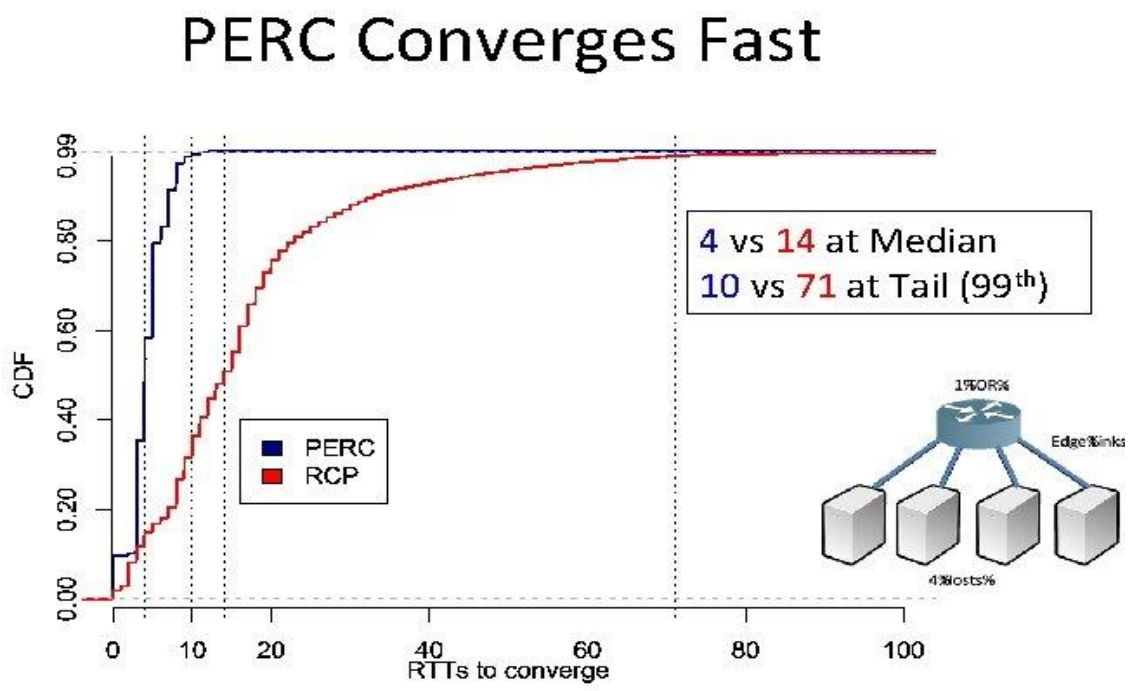


Figure 4.1: Convergence of RTT

4.2.1 PERC Switch on NetFPGA

The packet processing logic for the switch was written in P4 and compiled to the NetFPGA platform by using the P4 NetFPGA workflow, which is based on the Xilinx SDNet compiler. The NetFPGA switch can manage more than 40 Gb/s thanks to its 200 MHz core clock and 256-bit datapath. The P4 pipeline looks like this: The switch process control and data packets in a variety of ways, following standard L2 forwarding logic. Control packets may go

through a number of processing stages in order to conduct the computations as listed in Algorithm. Data packets are classed as low priority, whereas control packets proceed through a sequence of processing steps to complete the computations stated in Algorithm.

Atomic operations are conducted on the state variables deposited in the switches in two phases (Sum E; Num B, and Max E). Rather than being expressed in P4, they are applied in Verilog and used in the processing pipeline via the P416's extern interface.

Implementing 32-bit division at line rate is the most difficult component of the switch prototype design. We employ the lookup method mentioned in Section 4.1. We only needed roughly 32 Kb of exact-match memory and also TCAM entries. These requirements are easily accommodated by an FPGA and are insignificant in comparison to the memory requirements of line-rate switch ASICs. Our NetFPGA switch has two 128-KB queues on each output port: a high-priority queue for control packets and a low-priority queue for data traffic.

4.2.2 End Host in MoonGen PERC

The s-PERC gives up two main tasks:

(i) it accepts and resend s-PERC modify packets for the each flow

(ii) it implements per-flow rate limitations depending on the bandwidth requirements specified in acquired manage packets. Our prototype, which is based on Carousel, uses a timing wheel to limit glide rates. We set allocated 1 GB/s for traffic manipulation and 9 GB/s for traffic recording in our prototype. Data Centers: 4.3 s-PERC Evaluation

To begin, we compare the timings of s-convergence PERCs to (the reactive) RCPs in 4.3.1. Second, we compare the timings for s-PERC flow completion (FCTs). In section 4.3.2, there are three reactive schemes: p-Fabric, and an ideal maximum-minimum fair allocator. The

ns-2 simulator is being used to calculate convergence time and accuracy in this section. Finally, we run micro-benchmarks to ensure that s-PERC is resilient to lost control packets and incorrect switch hardware calculations. The numerical simulations of s-PERC in Python are used in these micro benchmarks, which are detailed in 4.3.3. Finally, in section 4.4, we estimate our 4 x 10 Gb/s NetFPGA hardware prototype by relating s-PERC Convergence times to DCTCP and Transport Control Protocol(TCP).

4.3.1 Time to Converge

The algorithms converges once a segment of the flows are exchanged with new ones, a process known as "churn," in our earliest experiments.

Workload: At random, we start flows between pairs of hosts, averaging 20 flows per server. When the rates have converged, we replace 40% of the flows. To evaluate the scheme's convergence times and resistance to unexpected changes in the traffic matrix, we replace all flows at once. Our graphics illustrate the results of 1000 convergence tests with ten changes per run over 100 runs. The number of flows per server is comparable to the number of concurrently active flows we saw on congested lines.

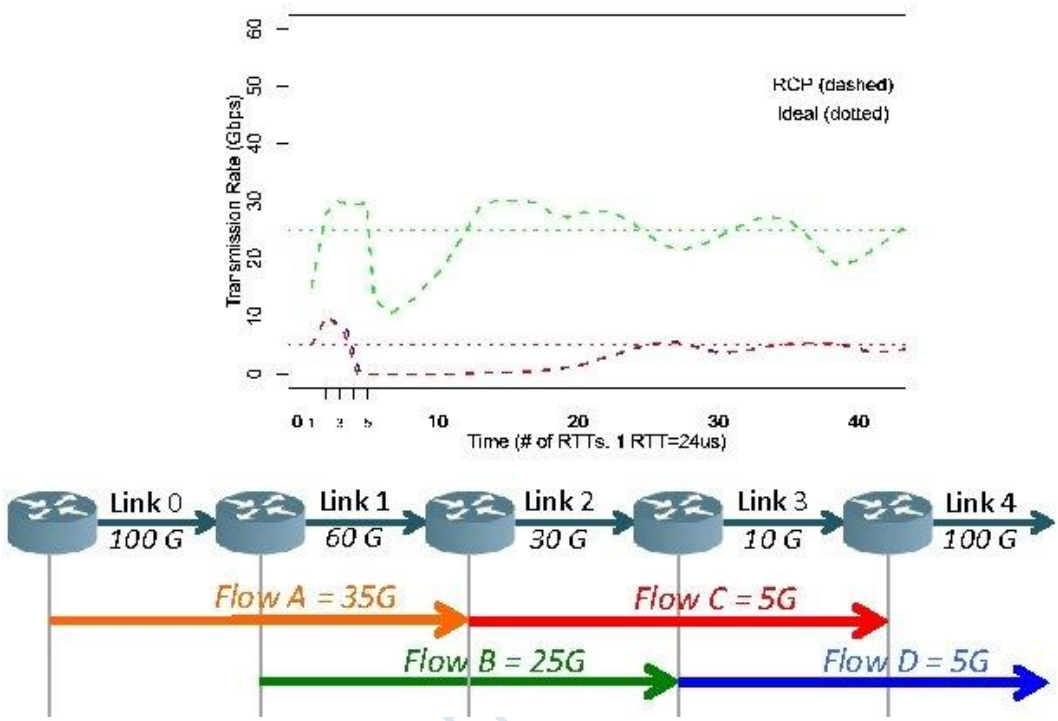


Figure 4.2: Flow of Links

4.3.2 Time to Complete a Flow

Workload: We assume Poisson flow arriving timings based on flow size distributions from two genuine workloads from data centers running search and data mining applications with loads of 70% and 85%. The normalized flow completion times (FCTs) of various flow groups are investigated. To normalize the actual flow completion times, the time it would take to broadcast the flow on an unloaded network is used.

We categorize flows by their size and the percentage of total bytes they transfer. The flows are arranged in ascending order of size. The smallest flows, accounting for 1% of the total

bytes, are stored in the first bin. The remaining bins are filled with equal amounts of the remaining flows (by bytes supplied). Both workloads we're looking at have a high tail, with a few large flows carrying the majority of the data. The vast majority of flows are little, and they all go into the first bin.

While the tests in the previous chapter absorbed on the facts middle situation, the s-PERC technique is widely utilized and may be employed in any network. In a wide-area network, preliminary simulations show that s-PERC can converge 10–15 times faster than reactive alternatives (WAN). The convergence periods for s-PERC and RCP in a topology identical to Google's B4 inter–data-center WAN are shown in Table 5.1. There are 18 various data-center locations, and the linkages that connect outstanding websites all have the same potential but differ in propagation times. RTTs of distinct flows in a WAN, unlike in a data center, can range from one to thousands of milliseconds depending on the flow's path.

The first version, s-PERC basic, does not optimize latency for short flows since all flows must wait for a rate from the control packets before sending any packets, and all flows are given the same priority across all links. In the second edition of s-PERC short, brief flows are begun at line rate and prioritized at all connections. Flows having a BDP5 of less than one are considered short.

Results: Assume that the search burden is 70% full (Figure 4.2a). We begin by contrasting the max-min fair systems. Because of the increased RTT required to obtain a beginning rate, basic s-PERC and RCP both start at x = 2 for the 1 **Control Packet** owest flows (bin 1). Because of lower delays, s-PERC basic is 25-50 percent better than RCP for the lowest

flows. Both are close to the ideal max-min allocator and to each other for the remaining flows that carry 99 percent of the bytes.

4.4 Hardware Evaluation of NetFPGA

In a test setting with three NetFPGA switches and four servers linked via 10 Gb/s lines, we tried TCP Reno, DCTCP, and s-PERC. We executed two experiments for each congestion control algorithm: (1) an incast trial with three senders transmitting to a single receivers, and (2) a dependency chain experiment with multiple bottleneck links relying on one another. The goal of the evaluation is to see how much quicker a practical implementation of s-PERC converges than existing reactive algorithms at 40 Gb/s using the NetFPGA SUME platform (4 x 10 Gb/s) and to verify that s-PERC can be implemented in hardware at 40 Gb/s using the NetFPGA SUME platform (4 x 10 Gb/s).

The architecture utilized for both studies with 10 Gb/s links is shown in Figure 4.6. In the incast experiment, hosts H1, H2, and H3 deliver a set of flows to host H4, all of which begin at around the same time. We establish a two-length dependency chain for the experiment: The green and blue flows are inserted initially, followed by the red flows. The bottleneck before the red flows begins is link B1, which switches to link B2 when the red flows are added, allowing the green flow to raise its rate.

Table 4.2 on a NetFPGA testbed, illustrates the results of incast experiments. Based on a total of ten runs, the average and standard deviation of convergence times were calculated.

Convergence Time (ms) TCP DCTCP s-PERC	# flows		
2	137 ± 165	45 ± 12	0.27 ± 0.36
3	483 ± 496	44 ± 14	0.23 ± 0.55

10	N/A	N/A	0.21 ± 0.29

Endnotes

[1.] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, C. Stephen Gunn, For wan distributed computing, Bwe is a flexible, hierarchical bandwidth distribution system. SIGCOMM '15, New York, NY, USA, pages 1–14, in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication,

Chapter Five

Conclusion, Recommendations and Future Work

5.1 Conclusion

The common of mainstream of the control of congestion algorithms are the reactive control systems that calculate costs solely by reacting to congestion indicators on Round Trip Time scales and then take tiny steps toward convergence over a long period of time. Buffers can quickly fill up before there is time to react as the networks become faster and more records can fit into each RTT. As a result, we concentrated on PERC algorithms in this thesis, which do not rely on congestion signals and instead compute rates for all flows proactively utilizing express world data (such as the number of flows crossing a link). Congestion control, we believe, should converge in a time-limited manner, principally through the utilization of essential dependency chains, which are the properties of the visitor's matrix and network design. The previous efforts to estimate the network's share quote proactively failed due to the need for per-flow data and the algorithms' inability to converge.

We developed PERC algorithm that will be sensible (this does not comprise per-flow state, and calculations can be completed at line rate in some very simple hardware) and assured to converge; simulations and a hardware prototype show that PERC is churn-resistant and converges several times faster than other algorithms. In a data center, we put PERC to the test to verify that, thanks to its faster convergence, it attains flow conclusion times that are very close to an ideal maximum-minimum scheme than a reactive algorithm. In real-world applications, PERC outperforms systems that prioritize flows, such as p-Fabric, by delivering low latencies for brief flows and high throughput for enormous flows. According to the simulations, fast-converging PERC algorithms like s-PERC would be ideal for long-haul networks where precise, predictable bandwidth distribution between flows is necessary to avoid congestion. As we have seen in this

chapter, there are still a lot of work to do in the field of PERC algorithms in high-speed networks, so we hope this isn't the final word.

5.2 Recommendations

Additional approaches to s-PERC optimization are discussed.

1. In networks with long links, the links state in the s-PERC that modify the packet might become a source of contention. For each hyperlink in the flow, the s-PERC update packet (Figure 4.1) has five(5) pieces of kingdom (approximately eight(8) bytes).It's unclear whether the control packet state can be refactored (like SLBN did) to simply lift the bottleneck nation (one bit) per-link.

2. Instead of considering software program bottlenecks, the s-PERC technique assumes that servers would adopt network-based rates. However, due to technical constraints, it is conceivable that the server will not be able to send at the allocated rate. There is one option is to model, the bottleneck as an additional (unique to each flow) hyperlink on the flow's path, with a capability that varies over time to reflect the software program bottleneck rate. This optimization is no longer done in our simulations or prototypes, and it will be left to future research.

3. When RTTs are not uniform, as they are in the Wide Area Network, some flows may be asked to increase their rates long before other flows (with longer RTTs) using the same links have done so. As a result, queuing at shared links may occur. As soon as the manipulation packets come, the quit hosts in s PERC exchange their rates.

The end host replies instantly when asked to cut the transmitting charge, but any price increase is delayed by a positive interval, according to a study. To completely appreciate the benefits of this s-PERC enhancement, more research is required.

4. Separate control and facts packets, each with its own bandwidth allocation, are included in the PERC algorithms proposed in this thesis. Other systems, such as these, in which control statistics are piggybacked on statistics packets, would be fascinating to investigate to see how link use varies as convergence speed improves. Advances in networking can assist PERC algorithms in a variety of ways. Programmable switches, as previously said, allow for a more straightforward deployment process. The addition of new abstractions to the software stack could also be beneficial, but in a situation where many congestion management strategies are based on window, recent work on novel abstractions for rate-limiting character flows at high speeds, for example, makes it easier to apply rate-based PERC algorithms.

5.3 Suggestion Area for Further Research

The s-PERC technique is widely used and may be used in any network, whereas the tests in the earlier section concentrated on the factual middle case. Preliminary simulations demonstrate that s-PERC can converge 10 to15 times quicker than reactive alternatives in a wide-area network (WAN). Table 5.1 shows the convergence durations for s-PERC and RCP in a topology equivalent to Google's B4 inter–data-center Wide Area Network.

There are various data-center locations, and the linkages that connect outstanding websites all have the same potential but differ in propagation times. Unlike with a data center, the RTTs of different flows across a WAN might range from one to millions of milliseconds based on the flow's course. In addition, the BDP for a 100 ms long route on a 10Gb/s Wide Area Network network is three orders of magnitude more than the BDP for a 10s long route in a 100 Gb/s data-center network. The following are the outcomes of our experiment. Every millisecond, we launch a hundred long-lived flows from each online domain (toward a randomly chosen

holiday destination site) and measure the expenses connected with them (as configured at the source end-host).

Bibliography

Journal Papers

Niels Moller{2018}Window-Based Congestion Control A Dissertation Submitted To Stockholm, Sweden Isbn-978-91-7178-831-3.

Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In International Workshop on Protocols for Fast Long-Distance Networks, Lyon, February 2015.

N. M¨oller, K. H. Johansson, and H. Hjalmarsson. Making retransmission delays in wireless links friendlier to TCP. In IEEE Conference on Decision and Control. IEEE, 2014.

Niels Moller{2018}Window-Based Congestion Control A Dissertation Submitted To Stockholm, Sweden Isbn-978-91-7178-831-3.

Kanagarathinam, M. R., Singh, S., Sandeep, I., Kim, H., Maheshwari, M. K., Hwang, J., ... & Saxena, N. (2020). NexGen D-TCP: Next Generation Dynamic TCP Congestion Control Algorithm. IEEE Access, 8, 164482-164496.

Patil, J., Tokekar, V., Rajan, A., & Rawat, A. (2020, July). SMDMTS–Scalable Model to Detect and Mitigate Slow/Fast TCP-SYN Flood Attack in Software Defined Network. In 2020 International Conference on Computational Performance Evaluation (ComPE) (pp. 290-295). IEEE.

Guan, S., Jiang, Y., & Guan, Q. (2020). Improvement of TCP Vegas algorithm based on forward direction delay. International Journal of Web Engineering and Technology, 15(1), 81-95.

Ahmad, M., Ahmad, U., Ngadi, M. A., Habib, M. A., Khalid, S., & Ashraf, R. (2020). Loss Based Congestion Control Module for Health Centers Deployed by Using Advanced IoT Based SDN Communication Networks. International Journal of Parallel Programming, 48(2), 213-243.

Zhu, J., Jiang, X., Jin, G., & Li, P. (2020, October). CaaS: Enabling Congestion Control as a Service to Optimize WAN Data Transfer. In International Conference on Security and Privacy in Digital Economy (pp. 79-90). Springer, Singapore.

Li, Y., Cao, G., Wang, T., Cui, Q., & Wang, B. (2020). A novel local region-based active contour model for image segmentation using Bayes theorem. Information Sciences, 506, 443-456.

Raman, C. J., & James, V. (2019). FCC: Fast congestion control scheme for wireless sensor networks using hybrid optimal routing algorithm. Cluster Computing, 22(5), 12701-12711.

H. Haile, K.-J. Grinnemo, S. Ferlin et al., End-to-end congestion control approaches for high throughput and lowdelay in 4G/5G cellular networks, Computer Networks (2020).

Xu, W., Zhou, Z., Pham, D. T., Ji, C., Yang, M., & Liu, Q. (2011). Hybrid congestion control for high-speed networks. Journal of Network and Computer Applications, 34(4), 1416-1428.

Jose, L., Yan, L., Alizadeh, M., Varghese, G., McKeown, N., & Katti, S. (2015, November). High speed networks need proactive congestion control. In Proceedings of the 14th acm workshop on hot topics in networks (pp. 1-7).

Leith, D., Shorten, R., & Lee, Y. (2005, August). H-TCP: A framework for congestion control in high-speed and long-distance networks. In PFLDnet Workshop.

Najmuddin, S., Asim, M., Munir, K., Baker, T., Guo, Z., & Ranjan, R. (2020). A BBR-based congestion control for delay-sensitive real-time applications. Computing, 102(12), 2541-2563.

Huang, H., Sun, Z., & Wang, X. (2020). End-to-End TCP Congestion Control for Mobile Applications. IEEE Access, 8, 171628-171642.

Mohammed, Y. A. (2006). Evaluation of TCP Based Congestion Control Algorithms Over High-Speed Networks.

Xu, L., Harfoush, K., & Rhee, I. (2004, March). Binary increase congestion control (BIC) for fast long-distance networks. In IEEE INFOCOM 2004 (Vol. 4, pp. 2514-2524). IEEE.

Jamali, S., & Fotohi, R. (2017). DAWA: Defending against wormhole attack in MANETs by using fuzzy logic and artificial immune system. The Journal of Supercomputing, 73(12), 5173-5196.

Singh, K., Singh, K., & Aziz, A. (2018). Congestion control in wireless sensor networks by hybrid multi-objective optimization algorithm. Computer Networks, 138, 90-107.

Li, W., Zhou, F., Chowdhury, K. R., & Meleis, W. (2018). QTCP: Adaptive congestion control with reinforcement learning. IEEE Transactions on Network Science and Engineering, 6(3), 445-458.

Pakdel, H., & Fotohi, R. (2021). A firefly algorithm for power management in wireless sensor networks (WSNs). The Journal of Supercomputing, 1-22.

Turkovic, B., Kuipers, F. A., & Uhlig, S. (2019, June). Interactions between congestion control algorithms. In 2019 Network Traffic Measurement and Analysis Conference (TMA) (pp. 161-168). IEEE.

Zaminkar, M., Sarkohaki, F., & Fotohi, R. (2021). A method based on encryption and node rating for securing the RPL protocol communications in the IoT ecosystem. International Journal of Communication Systems, 34(3), e4693.

Quwaider, M., & Shatnawi, Y. (2020). Congestion control model for securing internet of things data flow. Ad Hoc Networks, 106, 102160.

Jacobson, V. Congestion avoidance and control. *ACM Sigcomm Comput. Commun. Rev.* 1988, *18*, 314–329.

Floyd, S.; Henderson, T. *The NewReno Modification to TCP's Fast Recovery Algorithm*. 1999. Available online: https://tools.ietf.org/ html/rfc2582 (accessed on 21 April 2020).

Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. *ACM Queue* 2016, *14*, 50:20–50:53.

Kleinrock, L. Power and deterministic rules of thumb for probabilistic problems in computer communications. In Proceedings of the International Conference on Communications, Boston, MA, USA, 10–14 June 1979; pp. 1–10. *Electronics* 2021, *10*, 615 17 of 18

Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR Congestion Control: An Update.

Mahmud, I.; Kim, G.H.; Lubna, T. BBR-ACD: BBR with advanced congestion detection. *Electronics* 2020, *9*, 136.

Biodata

Personal Information

Name: Ayemowa Matthew Ojo

Address: Ok/KS No 4, Okesasa Street, Omuooke-Ekiti, Ekiti State, Nigeria.

Sex: Male

Marital Status: Married

Phone Number: 08030675302

Date of Birth; 13th December, 1978

Local Government Area: Ekiti East Local Government

State: Ekiti State

Nationality: Nigerian

Name of Next of Kin: Ayemowa Christiana Odunayo

Phone Number of Next of Kin: 08065713642

Number of Children and their ages: Three (10-8-5)

Address of Next of Kin: Ok/KS No 4, Okesasa Street, Omuooke-Ekiti, Ekiti State, Nigeria.

E-Mail: ayemowaodunayo@gmail.com

Institutions Attended With Dates

Lead City University, Ibadan, Oyo State	2020-till date
Lead City University, Ibadan, Oyo State	2018-2020
Federal University of Technology, Akure, Ondo State	2010-2012
University of Ilorin, Ilorin, Kwara State	2002-2006
Bishop Philips Academic, Iwo Road, Ibadan, Oyo State	June, 2002
C.A.C Primary School, Omuooke Ekiti, Ekiti State	1984-1990

Academic Qualifications and Certificate Obtained with Dates

M.Sc. in Computer Science	2020-till date
PGD in Computer Science	2019-2020
PDG in Statistics	2010-2012
B.Sc.(Ed) Mathematics	2002-2006
West Africa School Certificate	June, 2002
First School Leaving Certificate	1984-1990

Working Experience

College of Education, Ila-Orangun 2008-2011
Part-time Mathematics/Computer Science Lecturer

St. Louis School, Isikan, Akure, **Ondo State** 2009-2015
Mathematics/Computer/Music Teacher

College of Education, Ikere, Ekiti State 2013-2015
Part-time Mathematics/Computer Science Lecturer

Gateway (ICT) Polytechnic, Saapade, Ogun State 2015-Till date
PA to the Rector/Lecturer

Professional Qualifications

University of Port Harcourt 2021
CCNAv7

University of Pretoria, South Africa 2018
Total Quality Management

Professional Membership

Institute of Strategic Management, Nigeria(**ISMN**)	2021
Mathematics Association of Nigeria(**MAN**)	2019
Teachers Registration Council of Nigeria(**TRCN**)	2009

Conferences/Seminars/Workshops attended with Dates

Leadership Skills for a VUCA Environment, **Cape Town, South Africa**	2022
Strategic Management Workshop, Institute of Strategic Management, **Abuja**	2021
Renewable Energy, **Gateway Polytechnic, Saapade**	2020
Effective Research Finding Communication, **Gateway Polytechnic, Saapade**	2020
Functional Security, NSCDC, **Ogun State**	2020

Employee Well-Being in Tertiary Institutions, **Osun State** 2020

Modern Service Delivery, **Dubai**, **United Arab Emirate** 2019

Computer Appreciation for Managers, **Gateway Polytechnic, Saapade** 2019

Enhancing Staff Alignment to Tertiary Institutions, **OAU, Osun State** 2019

Management and Administration of Polytechnic, **Gateway Polytechnic, Saapade** 2019

Entrepreneurship Education, **Gateway Polytechnic, Saapade** 2018

Mathematics as Equipment in Solving Nigeria Issues, **Ogun State** 2017

Advance Digital Appreciation Programme, **Gateway Polytechnic, Saapade** 2017

Ethical Values in Administration, **Knutsford University**, **Accra**, **Ghana** 2017

Quality Assurance in Polytechnic Education, **Gateway Polytechnic, Saapade** 2017

Capacity Building Training, **Gateway Polytechnic, Saapade** 2017

Science, Technology and Entrepreneurship, **Gateway Polytechnic, Saapade** 2015

Team Building Management, **Akure** 2011

List of Journal Publications

(i). **M.O Ayemowa** & O.O Oni, a Quality of Service Based Performance Analysis of Routing in Mobile Ad Hoc Networks, Saapade Journal of Management, Science and Technology, Vol. 3, No. 1 2022.

(ii). [1]**Ayemowa Matthew**. [2]A.A Waheed, [3]O.L Abraham, Window-Based Congestion Control. International Journal of Advanced Multidiscipline Research and Studies. Vol. 2, Number 2. Page 318-321, 2022

(iii) **Ayemowa Matthew**, Dr. Ajayi W., Emmanuel Adediran, Iyanuoluwa Fatoki & Alonge Opeyemi "Testing of embedded System with a slight look at mobile devices" American Journal of Computer Science and Information Technology. Volume 9, Number 9. 9885, October, 2021

(iv) Ogunyinka T.K, Dada I.D, Oni O.O and **Ayemowa M.O**, A Robust Prediction Model for Candidate's Admission using Fletcher-Reeves Conjugate Gradient Method. Federal Polytechnic, Ilaro Journal of Pure & Applied Sciences Volume 3, Issue 1.

(v) Sotonwa O.E, Eleyewo I., **Ayemowa M.O** and Ibikunle O. Mathematical Analysis of Controlling Hepatitis B Virus, Saapade Journal of Management, Science and Technology, ISSN2536-6467(Paper), Volume 2, December, 2018.

List of Conference Papers

(i). Ibikunle Olajide, Oyefusi Adebayo and **Ayemowa Mathew O**. "Accessing of Overcrowded Mathematics Classroom, Academic Performance, Teaching and Learning, *ASUP National Conference, Yaba College of Technology, Lagos, 27th - 9th October, 2020".*

(ii). O.O Oni, T.K Ogunyinka, A.T Aderanti and **M.O Ayemowa.** A Comparative Study of Video Conferencing Systems, *3rd National Conference, Gateway Polytechnic Saapade, 13th - 15th July, 2021.*

(iii). **Ayemowa M.O** & O.O Oni(2021), Candidate's Admission using Fletcher Reeves(FR) Conjugate Gradient Method. *National Conference of management, science & Technology, Gateway(ICT) Polytechnic, Saapade.*

Book

M.O Ayemowa, Basic General Mathematics for Colleges of Educational**,** 2012.

Administrative Experience and Appointments:

Administrative experiences are as follow:

i. **Secretary**, Promotion Committee, 2016, Gateway Polytechnic, Saapade.
ii. **Secretary**, Academic Paper Review Committee on Promotion, 2017, Gateway Polytechnic, Saapade.
iii. **Secretary**, Academic Paper Review Committee on Promotion, 2018, Gateway Polytechnic, Saapade.

iv. **Secretary**, Academic Paper Review Committee on Promotion, 2019, Gateway Polytechnic, Saapade.

v. **Member**, Tetfund Staff Development Committee, 2020-2021, Gateway Polytechnic, Saapade.

vi. **Personal Assistant to the Rector**, 2015-2022, Gateway Polytechnic, Saapade.

Interests and Activities

Making Research, Playing Musical Instruments, Reading and Sports

…… ……

Signature Date

University Compliance Form

This is to certify that this thesis by Matthew Ojo AYEMOWA with Matriculation Number LCU/PG/001006 in the Department of Computer Science, Faculty of Engineering and Technology, Lead City University, Ibadan is in full compliance with the approved University's Format and Style.

_____ _____

Signature Date