데이터센터를 위한 지연 시간 기반
네트워크 혼잡 탐지 및 제어

Latency-based Congestion Detection and Control for Datacenters

이 창 헌 (李 昌 鉉, Lee, Changhyun)
전산학과
Department of Computer Science

KAIST

2015
데이터센터를 위한 지연 시간 기반 네트워크 혼잡 탐지 및 제어

Latency-based Congestion Detection and Control for Datacenters
Latency-based Congestion Detection and Control for Datacenters

Major Advisor : Professor Moon, Sue Bok
Co-Advisor : Professor Han, Dongsu

by

Lee, Changhyun
Department of Computer Science
KAIST

A thesis submitted to the faculty of KAIST in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science. The study was conducted in accordance with Code of Research Ethics¹.

2015. 5. 27.
Approved by
Professor Moon, Sue Bok

[Major Advisor]

¹Declaration of Ethical Conduct in Research: I, as a graduate student of KAIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.
데이터센터를 위한 지연 시간 기반 네트워크 혼잡 탐지 및 제어

이 창 현

위 논문은 한국과학기술원 박사학위논문으로 학위논문심사위원회에서 심사 통과하였음.

2015년 5월 27일

심사위원장  문 수 복  (인)
심사위원  한 동 수  (인)
심사위원  김 동 준  (인)
심사위원  신 인 식  (인)
심사위원  Jitendra Padhye  (인)
ABSTRACT

Large-scale datacenters have become an essential part of today's online services, and performance issues within datacenter networks have received much attention in recent years. In datacenter workloads, latency-sensitive flows are prevalent, so achieving low latency is important in improving end-users' experience. Since queueing delay dominates total network delay in datacenter networks, lowering queueing delay has been a major research challenge. To solve the queueing delay problem, congestion control algorithms should be re-designed to target datacenter networks' own needs.

The nature of congestion feedback largely governs the behavior of congestion control. In datacenter networks, where RTTs are in hundreds of microseconds, accurate feedback is crucial to achieve both high utilization and low queueing delay. Previous proposals for datacenter congestion control predominantly leverage ECN or even explicit in-network feedback to minimize the queuing delay.

In this dissertation, we explore latency as accurate congestion feedback and develop a new congestion control algorithm that is specifically designed to achieve low queueing delay in datacenter networks. Before developing the algorithm, we first analyze the required properties for congestion control mechanism in datacenters and derive our design choices. Our first design choice is the use of latency as a congestion feedback instead of packet loss or ECN-based feedback. Previous TCP algorithms such as TCP Vegas have already proposed to use latency feedback, but it has been known that end-to-end latency measurements are noisy and cannot represent accurate congestion level in the networks. To overcome such limitations in our algorithm, we measure end-to-end latency in driver-level and hardware-level. We implement driver-level measurement with Intel DPDK platform and hardware-level measurement with Mellanox timestamping-enabled network interface card. We come up with additional calibration techniques on packet bursts and hardware DMA. After removing all the sources of errors, we demonstrate that latency-based feedback is up to an order of magnitude more accurate and fine-grained than ECN-based feedback; the number of packets in the network switch can be accurately inferred by our latency measurements. Our second
design choice is accurate window size calculation. DX, our congestion control algorithm, calculates the number of packets to saturate the network link without queueing every RTT. Such fine-grained calculation is only possible due to our accurate latency measurement. One of the advantages of using DX is that there is no parameter setting whatsoever. Finally, we implement DX in Linux kernel and evaluate the performance with testbed experiments and large-scale simulations. In our dumbbell topology experiments, DX achieves 5.33 times smaller queueing delay at 1Gbps link than DCTCP, the most popular ECN-based solution, and 1.57 times smaller queueing delay at 10Gbps. We believe that improvement in hardware timestamping technology will enable better performance at 10Gbps and even higher network link rates in future datacenter networks.
Contents

Abstract ............................................................................. i
Contents ................................................................................ iii
List of Tables .......................................................................... v
List of Figures ........................................................................ vi

Chapter 1. Introduction ......................................................... 1
  1.1 Thesis Contributions ...................................................... 4
  1.2 Thesis Overview ............................................................ 5

Chapter 2. Congestion Control for Datacenters ......................... 6
  2.1 Required Properties ....................................................... 6
    2.1.1 Low queuing delay .................................................... 7
    2.1.2 High utilization ......................................................... 8
    2.1.3 Easy deployment and configuration ............................ 8
    2.1.4 High scalability ....................................................... 9
    2.1.5 Good fairness ........................................................ 9
  2.2 Pros and Cons of Existing Solutions ............................... 9
  2.3 Challenges in End-to-end Approach .............................. 10
    2.3.1 Loss-based Congestion Detection ............................. 11
    2.3.2 Small Bandwidth-delay Product ................................ 11
    2.3.3 Aggressive Window Size Adjustment ...................... 13
  2.4 Our Approach ............................................................. 15
    2.4.1 Completely end-to-end operation .............................. 15
    2.4.2 Latency for congestion signal ................................. 15
    2.4.3 Conservative and subtle window size adjustment ........ 16

Chapter 3. Congestion Feedbacks for Datacenters ..................... 18
  3.1 Overview of Congestion Feedbacks ............................... 18
    3.1.1 Explicit congestion notification ............................... 18
    3.1.2 Explicit in-network feedback .................................. 19
    3.1.3 Latency feedback .................................................. 19
  3.2 Evaluation of Congestion Feedbacks ............................... 19
    3.2.1 Explicit congestion notification ............................... 19
    3.2.2 Latency feedback .................................................. 22
  3.3 Impact of Feedbacks’ Quality ........................................ 23
Chapter 4. Accurate Latency Feedback Measurement

4.1 Removing Sources of Errors ........................................... 27
4.2 One-way Delay Measurement ........................................ 27
  4.2.1 One-way queuing delay ........................................ 31
  4.2.2 Handling clock drift ........................................... 32
4.3 Implementation ......................................................... 33
  4.3.1 Software timestamping ........................................ 33
  4.3.2 Hardware timestamping ....................................... 33
  4.3.3 LRO handling ................................................ 33
  4.3.4 Burst mitigation ............................................... 34
4.4 Evaluation of Latency Measurement ................................. 34
  4.4.1 Accuracy of queuing delay in Testbed .......................... 34

Chapter 5. DX Congestion Control .................................. 39

5.1 Limitations of TCP Vegas ........................................ 39
5.2 DX Core Algorithm .................................................. 40
5.3 Window Convergence and Steady-state in DX .................... 43
5.4 Implementation ....................................................... 44
5.5 Evaluation of DX Performance .................................... 46
  5.5.1 DX congestion control in testbed ............................ 46
  5.5.2 Impact of measurement error in simulation ................. 48
  5.5.3 Microscopic behavior in simulation .......................... 48
  5.5.4 Common datacenter traffic patterns in simulation .......... 51
  5.5.5 Multi-bottleneck scenario in simulation .................... 53
  5.5.6 Real-world datacenter workload in simulation ............ 54
5.6 Related Work .......................................................... 56
  5.6.1 Latency-based Feedback in Wide Area Network ........... 56
  5.6.2 ECN-based Feedback in Datacenter Networks ............. 57
  5.6.3 In-network Feedback in Datacenter Networks .............. 57
  5.6.4 Flow Scheduling in Datacenter Networks ................... 57

Chapter 6. Conclusion .................................................. 59

6.1 Summary of contributions ......................................... 59
6.2 Concluding remarks and future directions ........................ 60

References ................................................................. 62
List of Tables

2.1 Existing transport solutions for datacenters and their functionality .......................... 10

2.2 BDP converted into window size of a 1500 byte packet ........................................... 12

4.1 Source of errors in latency measurement and our technique for mitigation ................. 27

5.1 Base RTT measured by ten TCP Vegas senders ......................................................... 40
## List of Figures

2.1 Queue buildup with minimum sending rate ........................................ 12  
2.2 Congestion window size evolution for a sample TCP flow ....................... 13  
3.1 Congestion level vs. ECN fraction and its moving average ...................... 20  
3.2 Congestion level vs. ECN fraction and its moving average when CWND=5 ...... 21  
3.3 Granularity level at each window size ............................................. 22  
3.4 Congestion level vs. measured latency ............................................. 23  
3.5 Tradeoff from the ECN marking threshold (K) with different RTT ............ 24  
3.6 Input load variation controlled by ECN-based and latency-based congestion control .................................................. 25  
3.7 Queue length distribution .............................................................. 25  
4.1 Round-trip time measured in kernel .................................................. 28  
4.2 Timeline of timestamp measurement points ....................................... 28  
4.3 H/W timestamped inter-packet gap at 10 Gbps .................................. 30  
4.4 Example timestamp calibration with bursty packets .............................. 30  
4.5 One-way queuing delay without time synchronization ........................... 30  
4.6 Improvements with noise reduction techniques .................................... 35  
4.7 Effect of calibration in H/W timestamped inter-packet gap at 10 Gbps ........ 35  
4.8 Improvement on RTT measurement error compared to kernel’s ............... 36  
4.9 Accurate congestion level reflected in RTT measurements ..................... 36  
4.10 Accuracy of queuing delay measurement ......................................... 37  
5.1 Queue length with the increasing number of TCP Vegas flows ............... 40  
5.2 Steady-state window size of DX ...................................................... 43  
5.3 CWND convergence of two flows .................................................... 44
5.4 Queue length comparison of DX against DCTCP in Testbed .......................... 47
5.5 Impact of latency noise to headroom and queue length ............................... 48
5.6 Queueing delay and utilization of HULL, DCTCP, and DX .......................... 50
5.7 Trade-off between utilization and queueing delay ...................................... 51
5.8 Convergence of a new flow in DCTCP and DX ......................................... 51
5.9 Fairness of five flows in DCTCP and DX .................................................. 52
5.10 Reaction to incast traffic ................................................................. 52
5.11 99th-percentile FCT of small flows under large background flows .............. 53
5.12 Multi-bottleneck topology and traffic scenario ......................................... 54
5.13 Flow completion time of search workload ............................................... 55
5.14 Flow completion time of data mining workload ....................................... 56
Chapter 1. Introduction

The history of network congestion control is closely related to the evolution of computer networks; different types of networks demand different goals from congestion control mechanisms. In the beginning of the Internet, the main focus of congestion control was delivering intact data and preventing congestion collapse. A large variety of congestion control algorithms have been developed in this era including TCP Tahoe [1], TCP Reno [2], and TCP NewReno [3]. In early 2000s, high speed and long distance networks became more common in the Internet, and the primary interest in congestion control moved to how to fill the large network links as fast as possible. The corresponding congestion control algorithms are HSTCP [4], BIC [5], CUBIC [6], FAST TCP [7], etc. The extensive deployment of wireless networks also introduced a brand-new challenge: how to handle packet losses induced by lossy link condition, not congestion. TCP Westwood [8], TCP Veno [9], ATCP [10], JTCP [11], and Freeze-TCP [12] are the examples of algorithms in wireless environment. In fact, it is a never-ending game between the evolution of networks and the development of new congestion control mechanism. Our best courses of action are to quickly identify the requirements of the new type of networks and develop an appropriate algorithm in a timely manner.

Recent explosive growth of datacenter networks has raised a new research question in congestion control. Modern online services process end-users’ interactive queries through their datacenter networks [13], and the response time has a huge impact on user experience. Therefore a large portion of datacenter workload is highly latency-sensitive, and achieving low latency has become a major challenge in congestion control mechanism for datacenters. To solve this problem, many new algorithms have been proposed in recent years. The proposed solutions, however, either require in-network architectural support that is costly for real world deployment or achieve unsatisfying latency reduction. The latter type of solutions, which are end-to-end approaches such as DCTCP [14], mostly focus on developing efficient rate control algorithms, but the actual performance limitations of such approaches come from the low quality of congestion feedback (e.g., ECN-based feedback).

The quality of network congestion control fundamentally depends on the accuracy and granularity of congestion feedback. In fact, the entire history of congestion control has largely been about identifying the “right”
form of congestion feedback. From packet loss and explicit congestion notification (ECN) to explicit in-network feedback [15, 16], the pursuit for accurate and fine-grained feedback has been central tenet in designing better congestion control algorithms. As the requirements for congestion control diversified [17], novel congestion feedback often enabled innovative congestion control behaviors that formed the basis of a number of flexible and efficient congestion control algorithms [18, 19].

With the advent of datacenter networking, identifying and leveraging more accurate and fine-grained feedback mechanisms became even more crucial [14]. Round trip times (RTTs), which represent the length of the control loop, have decreased from hundreds of milliseconds in the wide area network (WAN) to hundreds of microseconds. Queuing delay now dominates the end-to-end latency in datacenters [14]. As a result, proposals for datacenter congestion control predominantly leverage ECN (e.g., DCTCP [14] and HULL [20]) or explicit in-network feedback (e.g., RCP-type feedback [16]), to minimize the queuing delay and the flow completion times.

This dissertation takes a relatively unexplored path of identifying a better form of feedback for datacenter networks. In particular, this dissertation explores the prospect of using the network latency as congestion feedback in the datacenter environment. Although the idea has been explored previously in WAN [7], this work examines the prospect for the first time in datacenter networks where round trip times may be less than 100 microseconds (3-4 orders of magnitude smaller than that of WAN). We believe identifying a new form of feedback should open up the design space for congestion control algorithms—congestion controls have evolved over time, but the nature of a particular form of feedback remains the same. For example, many transport proposals including DCTCP [14], D2TCP [21], L2DCT [22], PASE [23], and HULL [20] commonly rely on the same ECN feedback.

In this dissertation, we explore latency as accurate congestion feedback and develop a new congestion control algorithm that is specifically designed to achieve low queueing delay in datacenter networks. Before developing the algorithm, we first analyze the required properties for congestion control mechanism in datacenters and derive our design choices. Among the introduced properties, our main focus is on low queueing delay and easy deployment. In fact, no previous proposals satisfy these two properties at the same time, and there exists a clear trade-off between two groups of datacenter congestion control solutions: end-to-end and in-network solutions. To break this trade-off, we design our algorithm to work in a completely end-to-end manner and operate with a better form of congestion feedback.
Our first design choice is the use of latency as a congestion feedback instead of packet loss or ECN-based feedback. Previous TCP algorithms such as TCP Vegas have already proposed to use latency feedback, but it has been known that end-to-end latency measurements are noisy and cannot represent accurate congestion level in the networks [24]. To overcome such limitations in our algorithm, we measure end-to-end latency in driver-level and hardware-level instead of TCP stack in kernel. We implement driver-level measurement with Intel DPDK [24,25] and hardware-level measurement with Mellanox timestamping-enabled network interface card [26]. We come up with additional calibration techniques on packet bursts and hardware DMA. After removing all the sources of errors, we demonstrate that latency-based feedback is up to an order of magnitude more accurate and fine-grained than ECN-based feedback; the number of packets in the network switch can be accurately inferred by our latency measurements. We claim that this latency-based implicit feedback can be as accurate as in-network explicit feedback, such as that of RCP [16], in signaling congestion.

Our second design choice is accurate window size calculation. DX, our congestion control algorithm, calculates the number of packets to saturate the network link without queueing every RTT. Such fine-grained calculation is only possible due to our accurate latency measurement. We take the increase in one way delay as a measure of congestion as increased latency signals that packets are accumulating in the switch buffers. One of the advantages of using DX is that there is no parameter setting whatsoever. Finally, we implement DX in Linux kernel and evaluate the performance with testbed experiments and large-scale simulations. In our dumbbell topology experiments, DX achieves 5.33 times smaller queueing delay at 1Gbps link than DCTCP, the most popular ECN-based solution, and 1.57 times smaller queueing delay at 10Gbps. In addition, we conduct extensive evaluation in ns-2 simulator. We show that DX achieves comparable queueing delay with HULL [20], an in-network solution, while fully utilizing the link just the same as DCTCP [14]. DX flows converge faster than DCTCP flows because they reduce unnecessary window adjustment during the convergence period. We also test DX with both synthetic and empirical datacenter workload and observe impressive queueing delay improvement over DCTCP. We believe that improvement in hardware timestamping technology will enable better performance at 10Gbps and even higher network link rates in future datacenter networks.
1.1 Thesis Contributions

We summarize the contributions of this thesis below.

1. We study the required properties for congestion control mechanism in datacenters and analyze the pros and cons of existing solutions. We highlight the key requirements as low queueing delay, high utilization, easy deployment, high scalability, and good fairness. Our analysis show that none of the existing solutions meet all the requirements.

2. We quantitatively compare ECN-based congestion feedback and latency-based feedback. Although many proposed congestion control for datacenters use ECN-based feedback, there has not been a thorough study on the quality of ECN-based feedback. Our results show that ECN-based feedback does not capture the level of congestion in the network.

3. We design and implement a novel queueing delay measurement methodology and show that it is accurate enough to infer how many packets are in the switch buffers. To do so, we identify the sources of errors in end-to-end network latency measurement and systematically remove each source in software-level and hardware-level.

4. We propose a technique for one-way queueing delay measurement without any clock synchronization among nodes. One-way delay measurement usually requires accurate clock synchronization with hardware support, so it is impractical to be used in the network with a number of servers. Our technique is completely software-based and easy to be deployed.

5. We develop DX, a new latency-based congestion control algorithm that achieves low queueing delay in datacenter networks. DX utilizes accurate queueing delay measurements and calculates the appropriate window size to saturate network links without queueing. Our performance evaluation shows that DX achieves up to five times lower queueing delay than DCTCP, the well-known ECN-based congestion control.

6. Based on our design, we implement a prototype of DX in Intel DPDK platform and Linux kernel. We run the prototype on the servers with Intel 1Gbps NIC, Intel 10Gbps NIC, and Mellanox 10Gbps NIC and show the feasibility of DX in high speed datacenter networks.
1.2 Thesis Overview

This thesis is structured as follows. In Chapter 2, we discuss the ideal properties of congestion control in datacenters and analyze whether the existing solutions meet such requirements. We further study the hurdles in designing a congestion control mechanism for datacenters focusing on end-to-end approach and then provide our vision to tackle the challenges. Chapter 3 reviews the commonly used congestion feedbacks in wide-area and datacenter networks and evaluate the accuracy and granularity of them. Chapter 4 describes our methodology to measure accurate queueing delay in software-based and hardware-based approaches. We also show that our latency measurement is accurate enough to infer the queue length at network switches. Chapter 5 proposes a new congestion control algorithm DX and provide extensive performance evaluation in testbed experiments and simulations. Finally, Chapter 6 concludes this thesis by summarizing our contributions and giving future directions.
Chapter 2. Congestion Control for Datacenters

Congestion control algorithms for datacenter networks have received much attention in recent years. The pioneering work is DCTCP [14], which has introduced the use of ECN feedback in sizing congestion window; DCTCP successfully reduces network queueing delay when compared to conventional TCP such as TCP Reno [2]. Since DCTCP, there have been numerous proposals to decrease network queueing delay and flow completion time. While these solutions have satisfied their performance goals to some extent, there does not exist a single perfect solution that is readily usable in current data center networks.

To understand why such is the case, we take a step back and study what the requirements of congestion control are in datacenter networks. We bring up the following five major requirements: low queueing delay, high utilization, easy deployment and configuration, high scalability, and good fairness.

With respect to the introduced requirements, we then give an overview of the existing congestion control solutions in datacenters and analyze the advantages and disadvantages of each solution. After comparing the solutions, we narrow our scope into end-to-end approach as we believe in-network approach inherently suffers from deployment overhead. Focusing on end-to-end approach, we explain the challenges of existing end-to-end congestion control algorithms when used in datacenters and provide our plans to tackle such challenges.

2.1 Required Properties

Data center networks are built for massive data and computing intensive jobs, so they are formed in a different way from wide area networks, aka the Internet. The main differences come from the topology of the network and traffic workload, and they introduce new challenges and requirements to the existing transport protocol used in the Internet.

The key topological characteristic of data center networks is the short physical distance between servers. While two communicating nodes in the Internet can be placed in different continents and can be as far as ten thousand miles from each other, data center nodes are placed in a single geographical location and two communicating nodes are normally within the distance less than ten miles. Such short distance leads to short propagation delay for
a packet, so other delays such as queueing delay become more visible in the total network delay when compared to the Internet.

On the other hand, the workload in datacenter networks also exhibits unique characteristics. The traffic in data center networks can be categorized into two groups: query traffic and background traffic [14]. Query traffic follows the partition/aggregate communication pattern as upper-layer aggregator servers send out requests to lower-layer servers and collect the responses. These requests and responses are short-sized and latency-critical. Different from the query traffic, background traffic is a mixture of small flows and large flows that are not much sensitive to delivery time; they require high throughput than small latency.

Under the aforementioned environment, it has been reported that conventional transport protocols do not perform efficiently in data center networks, and it has motivated a few courses of work such as DCTCP [14], HULL [20], PDQ [27], etc. The primary goals of these solutions are achieving low queueing delay and short flow completion time. Although such objectives are considered as a top priority, there still exist other requirements that a well designed transport protocol is supposed to meet when we take a step back and think over. We list up the necessary features below and explain why each feature must be considered.

2.1.1 Low queueing delay

Low queueing delay becomes essential in datacenter networks due to the unique characteristics in both network topology and traffic workload. As mentioned earlier, a large portion of the end-to-end network delay comes from packet queueing in the middle, so the flow performance highly depends on the queueing delay. For example, if two nodes are connected with a 200 meter network cable, the propagation delay is only about 1 $\mu$s. At 10 Gbps links, a single maximum transmission unit (MTU) packet experiences transmission delay of 1.2 $\mu$s. If we assume 100 packets are queued in the network switch, which is more than a normal scenario with conventional TCP, we see 120 $\mu$s of queueing delay, and it takes up 99% of the total delay.

Queueing delay directly affects end-to-end latency, and a large portion of datacenter workload is latency-sensitive. One example of the latency-sensitive workload is partition/aggregate workload. Previous work [14] show that large end-to-end latency harms not only the response time perceived by end-users but also the quality of served contents. Today’s web services’ process users’ queries in their datacenters. To answer a user’s search query, multiple responses from different data center nodes need to be gathered at the upper-layer aggregator. In
this process, any responses that arrive later than pre-defined deadline are left behind, damaging the quality of the search response.

The importance of low queueing delay has been pointed out in many research papers [14, 27], and most datacenter congestion control proposals have paid the best attention to how to reduce queueing delay. In fact, this is a unique requirement different from wide-area networks.

2.1.2 High utilization

Link utilization is closely related to the overall throughput of the flows in the network. Although a large portion of traffic in datacenter network is latency-sensitive, there still are throughput-oriented traffic such as map reduce jobs. As long as we have both type of traffic in the same network, we cannot sacrifice utilization in datacenters.

Achieving high utilization and low queueing delay simultaneously, however, is not a simple thing to do, and there is always a trade-off between the two goals. For example, if a network link remains under-utilized, the queueing delay of such link can be as low as zero whenever network is idle, but the flow throughput in this case will be severely decreased. Likewise, more packets in the queue helps increasing link utilization. Therefore it is important that we find the right balance and keep the sacrifices to the minimum.

2.1.3 Easy deployment and configuration

When we design a new transport protocol, it is necessary to think through how to deploy such protocol in the real-world. The best course of action would be to keep the existing hardware and modify only the software part of the network system. More changes the deployment of the protocol requires, more costly the transition from the old one to the new one is. In this case, the administrator will need to carefully consider whether the performance benefit is large enough to cover the deployment expense.

Many datacenter congestion control proposals follow in-network approach [14, 18, 27–29], which requires complex modification of in-network switches. Datacenters are in a better place to apply such changes than wide-area networks as they are usually controlled by a single authority. Replacing all the switches in the network, however, is still not an easy decision and comes with costs. Some solutions utilize network functions supported only by high-end network switches [30]. Ideally, the new transport protocol should make only software-level
changes and work in end-to-end. In fact, we have not seen many protocols deployed in the real-world production
datacenters except for DCTCP, an end-to-end solution [31].

In addition, the new protocol should be easy to use without much configuration overhead. If the perfor-
mance of a certain protocol is sensitive to configuration parameters, it is impossible to use it in general datacenter
networks. Such parameter setting is also extra cost to network administrators.

2.1.4 High scalability

Although data center networks are much smaller than wide-area networks, the transport protocol should still
be scalable. The number of servers in a datacenter goes up to tens of thousands as of now, and it may increase even
more in the future. If a new protocol works only with a small number of nodes, it cannot be a long-term solution.
We can provide some examples where the protocol has limited scalability. The first example is storing states in
the switch or server. Such state keeping requires memory space, and the memory is scarce resource especially
for network switches. The second example is using a centralized controller. Proposals with a centralized rate
controller [32] may suffer scalability problem as the overhead of controller increases with the number of nodes in
the network.

2.1.5 Good fairness

Throughput fairness among flows is still important in datacenter networks. Some previous solutions have
proposed priority-based rate allocation [20], and the flows with low priority get less bandwidth. Unless intention-
ally discriminated, all the flows should be treated fairly in the network. For instance, if two flows have the same
priority, they should be able to acquire the same bandwidth when other network settings are identical.

2.2 Pros and Cons of Existing Solutions

So far we have presented the requirements of the ideal transport protocol for datacenter networks. We now
analyze the existing solutions to see how close they are to the ideal. Table 2.1 shows which requirement is being
satisfied or not by each data center transport protocol. We observe that none of the existing solutions passes all
the criteria, and each one has its own advantages and disadvantages.

Conventional TCP algorithms in wide-area networks satisfy all the requirements but low queueing delay.
Since TCP uses packet loss for congestion feedback, it can learn congestion event only when queue is full, hence large queueing delay. Unless TCP uses a different type of congestion feedback, low queueing delay requirement cannot be satisfied. DCTCP [14] is developed in this line of thinking. It uses ECN-based congestion feedback and achieves lower queueing delay than conventional TCP algorithms. The weaknesses of DCTCP are in two parts. First, its queueing delay reduction is not very impressive when compared to switch-based solutions like HULL [20] and QCN [33]. Second, it runs with multiple parameter values that need to be customized to each network environment. HULL is a sequel solution of DCTCP, and it modifies network switches to implement phantom queues. HULL achieves lower queueing delay than DCTCP, but the deployment becomes more difficult and costly as all the network switches need to be modified or replaced. In addition, HULL adds even more parameter values than DCTCP so configuration also becomes more complicated. Finally, QCN leverages a new type of congestion feedback which can only be generated by QCN supporting switches, so it also suffers from deployment cost in real-world networks.

### 2.3 Challenges in End-to-end Approach

In this section, we describe the challenges in designing a perfect congestion control algorithm for datacenter networks. We limit the scope of our analysis to end-to-end approach because easy deployment, one of the key requirements mentions above, can be realized best with algorithms that run in an end-to-end manner.

We first demonstrate why conventional end-to-end congestion control algorithms such as TCP Reno [2] fail to achieve low queueing delay in datacenter networks. We pinpoint the key properties in TCP and datacenter networks that are prone to increase queue length at switches. Since we aim at extremely low queueing delay, things that did not matter in wide area network can now be a serious concern in datacenter network. We describe three most important challenges in this section: loss-based congestion detection, small bandwidth-delay product,
and aggressive window size adjustment.

2.3.1 Loss-based Congestion Detection

The first step to alleviating congestion in the network is to know exactly when congestion occurs. The most popular TCP variants used in wide area networks (WAN) are NewReno [3] and CUBIC [6] in today’s Internet, and these protocols all rely on packet loss events to notice congestion. Although they perform well in WAN where high throughput is the most required property, using packet loss as a congestion signal is not helpful in lowering queueing delay in datacenter networks for the following reasons. First, a queue needs to grow to the maximum length to cause a packet loss, so the queue repeatedly becomes full during TCP transfers, putting huge queueing delay to the network. Second, a packet loss can only inform a TCP sender of the existence of congestion, not the amount of congestion. Such binary knowledge gives limited opportunity to determine the optimal sending rate for TCP flows. To mitigate these known challenges, several alternative ways for congestion detection have been proposed. In the next chapter, we describe the exiting congestion feedbacks and discuss whether they are appropriate solutions for datacenter networks.

2.3.2 Small Bandwidth-delay Product

One of the unique characteristics of datacenter networks is small round-trip time (RTT) between nodes; it has been reported that order of hundreds of microseconds RTT is commonly observed in today’s datacenter networks [14]. Such low RTT results in small bandwidth-delay product (BDP), which is calculated as \( \text{Link Capacity} \times RTT \). BDP indicates the maximum amount of data that can be carried in the network links, or the maximum amount of data that is not yet acknowledged by a receiver in TCP’s point of view. Therefore the BDP value is closely related to the proper congestion window size that data flows can use.

To give a clear idea, we present sample BDP values for different types of network in Table 2.2. The values are shown in window size unit for a typical full-sized 1500-byte TCP packet. We can see the BDP value is as low as only eight packet window in the 1Gbps datacenter network. It means that eight is the maximum number of packets that can be sent without acknowledgements, and beyond that number will be queued in the network switches.

The bandwidth-delay product has been an important factor when designing a congestion control for a given type of network. For example, a group of TCP flavors such as FAST [7], BIC [5], and CUBIC [6] has aimed
Table 2.2: BDP converted into window size of a 1500 byte packet

<table>
<thead>
<tr>
<th>Capacity</th>
<th>1Gbps</th>
<th>10Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>100µs (DCN)</td>
<td>8.3</td>
<td>83</td>
</tr>
<tr>
<td>100ms (WAN)</td>
<td>8333</td>
<td>83333</td>
</tr>
</tbody>
</table>

Figure 2.1: Queue buildup with minimum sending rate

at networks with very large bandwidth-delay product as wide are networks have become much faster since the
notion of TCP was first proposed. The most significant issue in such environment is how to fill up the network
link as quickly as possible and minimize the link idle time. On the other hand, there has not been much focus on
small BDP networks because not many networks had been known to have small BDP until the time datacenter
networks became a major part of Internet service. In small BDP regime of datacenters, since filling up the link
does not take much time anyway, the important issue arises after the network link becomes full; we need to handle
the flows’ sending rate just enough to utilize the link, and not too intrusive to other flows sharing the same link.
Previous TCP variants used in the Internet have never been developed for meeting this goal, so they fail at being
not intrusive and using small queue and only succeed in achieving high utilization for most cases.

With small BDP, we can potentially suffer from queue buildup in the network where current congestion
control algorithms cannot come in and help at all. In fact, the size of congestion window can be reduced to even
one, when a large number of flows are sharing the same bottleneck link. To give an idea for this scenario, we
conduct a simple ns-2 simulation on a dumbbell topology where datacenter nodes are connected through a single
10 Gbps switch, and end-to-end propagation delay is 10 µs. We gradually increase the number of TCP flows from
0 to 50 until 2.1 second in simulation time. The congestion window size of each flow is fixed to one during the simulation, so each flow repeatedly sends one packet, wait for the corresponding ACK, and then send another packet; we call this pingpong transfer. This pingpong transfer is the slowest way of sending data in the current TCP, but we find that as soon as the number of flows exceeds 19, the queue is starting to grow and experience increasing delay. The result is shown in Figure 2.1. In this scenario, there is no way of doing throughput control on these flows other than stopping them for a retransmission timeout after a loss occurs. The Internet does not have to worry about this case because of large BDP from large RTT, but it is a practical concern in datacenters.

### 2.3.3 Aggressive Window Size Adjustment

The small BDP value has different impacts on each phase of TCP’s congestion control, and it can cause trouble to the phase that needs to react to congestion. First, in the slow start phase, a TCP flow performs multiplicative...
increase in congestion window size until it saturates the network link. In a small BDP network, the slow start phase is finished within only few number of round trips because the required congestion window size to fill up the link capacity is very small. Therefore small BDP value is advantageous to the slow start when the concern is filling up the link. In terms of delay, however, the last window increase right before escaping the slow start may bring instant high queueing delay to the network, and such high delay is a huge concern in datacenters.

On the other hand, in the congestion avoidance phase where a TCP flow increases window size by one every round trip until it sees congestion, the flow faces congestion more frequently in a small BDP network than in a large BDP network. When a flow in a small BDP network exits from slow start, it halves its window size and enters congestion avoidance phase. Then it takes only few round trips for the flow to reach the maximum available window size again, and repeat the same behavior. In fact, congestion avoidance phase in small BDP networks does not effectively avoid congestion although its name suggests otherwise.

To observe the above behavior in the real packet transfers, we conduct a simulation using ns-2 simulator in a simple dumbbell topology. We use 1 Gbps wide-area network (WAN) and 1 Gbps datacenter networks (DCN) environment for comparison, and the queue size at the middle switch is set to the BDP value following the common rule-of-thumb [34]. In each environment, we start five identical TCP Newreno flows and measure the congestion window size for the first flow every round trip. Figure 2.2 shows the evolution of congestion window size during the period of 500 round trips; we note that direct comparison is meaningful since we use the number of round trips instead of time. As we expected, the DCN flow quits slow start phase earlier than the WAN flow, and Newreno seems to work fine for small BDP networks to this point. Entering the congestion avoidance phase, however, the DCN flow almost immediately hits the maximum congestion window again and decreases the window size, and it goes on and on; the flow experiences more than 50 times of congestion in 500 round trips. Frequent congestion indicates frequent queuing event that is not favored in datacenters. Meanwhile, we observe the WAN flow takes hundreds of round trips to reach the maximum available window size in congestion avoidance phase, effectively avoiding congestion as the protocol was originally designed. The take-home message here is that the amount of window size increase/decrease of TCP is too aggressive for small BDP networks. This may not be a problem if our goal was to keep the link as busy as possible, but our goal in datacenters is to shorten the congestion period to minimize the queueing delay.
2.4 Our Approach

Motivated by the requirements for transport protocol in datacenter networks, we describe the design choices for our new protocol and explain why we make such choices in detail.

1. Completely end-to-end operation

2. Latency for congestion signal

3. Conservative window size adjustment

2.4.1 Completely end-to-end operation

As we have seen in the previous analysis, the existing solutions that require in-network support or switch modification can never achieve the easy deployment property. Therefore we strictly stay in an end-to-end manner so that the protocol is scalable and easy to deploy. It means that we do not require any form of modification from network switches and routers, so the current infrastructure in existing datacenters do not need to be replaced or specially configured to use our solution.

2.4.2 Latency for congestion signal

In the previous section, we have pointed out loss-based congestion detection is ineffective in reducing queueing delay in datacenter networks, so we need another way of learning congestion in the network. The most popular choice of previous solutions was using ECN marking in the TCP header; when queue size grows beyond a threshold, a switch starts setting a Congestion Encountered (CE) bit in incoming packets to notify the sender of the congestion. ECN works more effectively than packet loss does, as setting a threshold smaller than the physical queue size allows a switch to notify senders before the queue actually overflows. In addition, ECN can provide multi-bit information on congestion like the way it was used in DCTCP [14].

Using ECN, however, still has several drawbacks. First, determining the appropriate threshold value for every switch is never an easy task. Too large threshold close to the physical queue size does not make much difference from loss-based congestion detection, and too small threshold will cause under-utilization of network link capacity. Various type of traffic in the network and complex topology makes the decision even harder. Second, even with the optimal threshold value, the switch allows at least the threshold number of packets to occupy its queue, adding
the corresponding amount of queueing delay to every round trip of TCP flows. The queueing delay in this case is of course smaller than what we see in loss-based detection, but it still can be an issue in datacenter networks where a microsecond increase is visible in total round trip time. Third, packets that is distributed into multiple switches queues may not be detected by ECN. Each switch decides whether to set CE bits or not based on only its own queue length, so the queueing delay spanning the whole path is not taken care of by anyone. For example, each switch can be legitimately queueing 10 packets with a threshold value 20, but when a TCP flow is passing five such switches, the total number of queued packets, queueing delay in other words, can be up to 100.

Knowing that ECN cannot solve all the problems, our choice as a congestion signal is the measured delay between a sender and a receiver. Delay between two hosts directly shows how congested the network path is for them. Increase in delay means more congestion, and even one more packet queueing in the network is reflected in the measured delay. Delay itself is multi-bit information without further treatment, and it is more fine-grained than the ECN-based information as delay is a continuous value. To summarize, using delay provides early detection and exact level of congestion.

Using delay to measure congestion in the network is not a completely new idea, as there have been quite many delay-based TCP variants originating from the popular Vegas [35]. The delay-based TCP have not gotten popular in the Internet, and one critical reason for that is the compatibility issue with other TCP flavors. When delay-based TCP flows are mixed with loss-based TCP flows in a network, delay-based ones acquire less throughput than loss-based ones, and it discourages people to use delay-based TCP in the Internet [36]. In datacenters, on the other hand, it is possible that a single administrative authority decides which TCP flavor to run in the whole network. In other words, there is no need for competition among different flavors, so using delay-based TCP in datacenters is fairly practical.

2.4.3 Conservative and subtle window size adjustment

Window size adjustment in small BDP networks should receive extra attention. Failure in doing so results in frequent oscillation of window size along with recurring congestion as we observed in the previous section. The major cause of the failure in the NewReno simulation is that, when the window size is in increasing phase, it lasts so long that it ends up with a serious congestion. At that point, because the congestion has become too severe, the window size needs to be decreased by a large amount, which brings the window size back in the increasing phase.
In our approach, we try to find the optimal window size just enough to fully utilize the link and maintain the queue size at near zero. To do so, we take a conservative position on increasing window size, as we increase the window size only when the measured queueing delay is zero. We can also think about reducing the amount of increase to less than one each round trip to make it even less aggressive which is better for queue, but it will take more time for a flow to reach the available bandwidth although there is no congestion. Therefore we keep the one window increase at a time policy in our algorithm.

Since we minimize the time for increasing phase, we also need to modify the decreasing phase as well to avoid under-utilization of link capacity. In the spirit of maintaining the optimal window size, we drop the window size when the measured queueing delay is larger than zero, and the amount of drop is proportional to the measured queueing delay. Decreasing only the necessary amount helps reduce the unnecessary increasing phase in the future. The above measures to window size adjustment can be realized due to the use of delay as a congestion signal.
Chapter 3. Congestion Feedbacks for Datacenters

As low queueing delay in data centers becomes more important in providing quality service to end-users, many recent proposals for congestion control have deviated from the traditional loss-based approach used in most wide-area TCP variations; waiting for the queue growth to generate packet loss feedback is directly against the low queueing delay requirement in data centers. In search for more reactive congestion feedback than packet loss, most proposals for datacenter congestion control leverage ECN or explicit in-network feedback [14, 20–22, 33]. In this chapter, we describe each type of feedback in detail and compare them with latency-based feedback.

3.1 Overview of Congestion Feedbacks

3.1.1 Explicit congestion notification

DCTCP [14] and many other proposals [20–22] use ECN to detect congestion before the queue actually overflows. Using ECN allows DCTCP and other variants to measure the level of congestion and adjust the window size appropriately. Typically, congestion level is measured by calculating the fraction of ECN-marked ACK packets out of the total ACK packets in each window. For example, two ECN-marked packets in a ten packet long window give a congestion level 0.2 on a scale from 0 to 1, and more marked packets will result in a higher congestion level. To absorb instant fluctuations in queue occupancy, DCTCP takes the moving average of the sample fractions over multiple windows and estimate the probability with which the queue length is higher than the marking threshold. After detecting congestion, it decreases the window size in proportion to the congestion level. This allows DCTCP to maintain the average queuing delay small, near the marking threshold, $K$. There are many variations of DCTCP such as D2TCP [21] and L2DCT [22], but the way of using ECN as congestion feedback is exactly the same as DCTCP.

QCN provides another form of explicit congestion notification in datacenters [33, 37]. It provides multi-bit congestion notification for finer-grained feedback for link-layer congestion management. This feedback can only be generated from QCN-supporting switches, and such switches have not become mainstream in the real world yet.
3.1.2 Explicit in-network feedback

Explicit in-network feedback provides multi-bit congestion indicator that is much more accurate and fine-grained than ECN and has been used in several proposals for datacenter networking [18,38,39]. The key difference is that it also notifies how much the network is under-utilized, in addition to signaling congestion. Such feedback enables multiplicative-increase and multiplicative-decrease (MIMD), which results in high utilization and fast convergence [15–17]. However, this “idealized” feedback requires in-network support. Currently, we are unaware of any commodity switches that are capable of generating explicit in-network feedback [14].

3.1.3 Latency feedback

TCP Vegas [35] has introduced latency congestion feedback in wide-area network. If latency can be accurately measured to reflect the network congestion, it has more benefits than other types of feedback. First, it is implicit feedback that does not require any in-network support. Second, it can take on a much larger range of values than ECN or QCN [33, 37], offering a finer-grained form of feedback. The difference from in-network feedback of RCP [16] or XCP [15] is that latency feedback cannot signal the remaining network capacity when the network is not being fully utilized, but only notifies when and how congested the network is. Our proposal in this paper is using latency feedback in datacenter networks. In Chapter 4, we show that our measurement methodology effectively captures the network queueing delay in datacenter environment.

The rest of this chapter provides a quantitative comparison of the feedback. Note that we only try to evaluate the feedback itself, not the congestion control algorithm using the feedback.

3.2 Evaluation of Congestion Feedbacks

3.2.1 Explicit congestion notification

We quantify the accuracy of DCTCP’s ECN feedback with respect to an ideal form of explicit in-network feedback that accurately reflects the congestion level, such as that of RCP or XCP. To do this, we take the queue size as the ground truth congestion level and plot the measured feedback using ns-2 simulations.

We use a simple dumbbell topology of 40 DCTCP senders and receivers with 10 Gbps link capacity. The RTT between nodes is 200 µs, the ECN marking threshold is set to $K = 35$ as proposed in the original paper [14], and the queue capacity on the bottleneck link is set to 100 packets. We note that the queue does not overflow
during the simulation. Each sender starts a DCTCP flow and records the congestion level given by the fraction of ECN marked packets for each congestion window. During the simulation, we collect both the sample fraction and its moving-average value used by DCTCP. For comparison, we take the average switch queue occupancy during the window as the ground-truth congestion level.

Now we show the correlation between the congestion level given by ECN-based feedback and the corresponding switch queue length (the ground truth). Positive correlation in this analysis would indicate that ECN feedback well reflects the level of congestion in the network.

Figure 3.1 shows the percentage of ECN marked packets and its moving average as used by DCTCP. The $x$-axis represents the ground-truth, and the $y$-axis indicates the measured level of congestion (percentage of ECN marked packets). Along with the ECN congestion feedback, we plot a line for the ideal congestion feedback that informs the exact number of packets in the queue. The ideal congestion feedback models a form of explicit in-network feedback similar to that of RCP [16]. For example, the ideal feedback at 100% congestion, with respect to the maximum queue size, should be 100 queued packets, which is the amount to reduce in the next round to achieve zero queuing delay.

From this simple experiment, we make the following three key observations:

*Accuracy is low.* The fraction of ECN-marked packets is not a good indicator of the congestion level. About 50% of feedback is either 0 or 100; 16% (33%) of the times, the measured congestion level was 0 (100). Values other than 0 and 100 do not reflect the level of congestion accurately either. A wide range of switch queue
occupancy shares the same feedback. For example, both the queue lengths, 28 and 64 can result in the feedback of 80%. As a result, the Pearson correlation coefficient between the actual congestion and measured feedback was only 0.6924 (compared to 0.9998 for latency feedback); 1.0 is the highest correlation and 0.0 means no correlation. The RMSE (root mean square error) with respect to the ideal feedback was 33.79 (compared to 1.05 of latency feedback).

Granularity is too coarse. The congestion feedback in Figure 3.1 is very coarse grained. The fundamental reason is its dependency on the window size. For example, five is the most frequently appearing window size in our simulation; the average congestion window size is 5.2 packets. In this case, the feedback (i.e., ECN-marked fraction) can only take on six values from 0%, 20%, 40%, 60%, 80%, to 100%, while the actual congestion level is between 9 and 69 packets (61 different levels) in Figure 3.2.

While the dependency on the window size puts upper bound on the granularity of ECN-based feedback, such upper bound is still not easily achieved in the real transfer. We show the theoretical granularity level (upper bound) and the achieved granularity level by each window size in Figure 3.3; the maximum granularity is not always achieved. Note that the ideal granularity level is 101 at all window size in our simulation.

Taking the moving average does not help and even degrades the accuracy as the measured congestion level stays the same for a wide range of queue lengths (Figure 3.1). These results contradict the general belief that the modified use of ECN as proposed in DCTCP is effective in quantifying the level of congestion. We do observe in Figure 3.1 that the moving average smooths out the extreme congestion level values of 0s and 1s. However,
very little correlation exists between the ECN-based congestion feedback and the actual queue lengths. The measured congestion level (i.e., the moving average) always resides between 0.475 and 0.755, while the actual queue occupancy (the ground-truth congestion level) varies between 7 and 70 packets. As a result, the correlation coefficient drops to 0.1790, and the RMSE with respect to the ideal feedback is relatively high at 24.63. It is hard to say that the feedback accurately reflects the network congestion especially when there is such low correlation.

3.2.2 Latency feedback

In the face of the above disparity between the actual queueing and ECN-based feedback, we have turned to latency feedback as an alternative. As both senders and receivers are under the same administrative domain in datacenter networks, we assume that we could instrument both ends, and high-precision latency measurements are feasible. Later in Chapter 4, we introduce our detailed techniques for accurate latency measurement. Assuming that latency can be measured accurately for now, we verify that latency measurements accurately reflect the ground-truth congestion level, using ns-2 simulation.

The congestion level is measured once for every congestion window in the following way. The sender measures the RTT for every data packet and sets its minimum as the base RTT without queueing delay. The difference between the base RTT and a sample RTT represents the queueing delay, which is the congestion level.

Figure 3.4 shows the actual congestion level versus latency based congestion-level measurement. For ease of comparison, we consider the maximum possible queueing delay as the congestion level 100% and translate the
measured latency into congestion level accordingly. Latency feedback (i.e., queuing delay) naturally reflects the average queue lengths. The correlation coefficient is as high as 0.9998, and the RMSE against the ideal feedback is only 1.05, which is 32 times smaller than the raw ECN fraction feedback, and 23 times smaller than the moving average.

Through the simulation results, we claim that delay measured at the sender is much more accurate and reliable congestion feedback than ECN-based feedback (whether raw or moving average). We are fully aware that measuring accurate RTT and calculating accurate queuing delay face more challenges in the real-world network than in the simulation, and we show how to deal with such challenges in the later chapters.

### 3.3 Impact of Feedbacks’ Quality

Inaccurate feedback results in an undesirable tradeoff between queuing delay and utilization. We demonstrate the adverse effect and show how accurate latency-based feedback can solve this problem.

#### 3.3.1 Trade-off between queuing delay and utilization

The ECN marking threshold represents a fundamental trade-off. If the threshold is set too high, queuing delay increases, which adversely impacts the flow completion times of short flows [14]. If it is set too low, the switch buffer occupancy falls short in maintaining high utilization. In fact, TCP-like congestion control algorithms require the buffer size to be proportional to the bottleneck link capacity and RTT (or bandwidth delay product).

In DCTCP, the lower bound of ECN threshold $K$ for maintaining high utilization is $K > (C \times RTT)/7$ [14].
This further complicates the tradeoff. In real-world datacenters, the capacity can differ across links (1 to 40 Gbps), and the RTT can vary significantly from 10s of $\mu$s to 100s of $\mu$s. Communication happens within a rack (small RTTs) and across multiple switches (large RTTs). To fully utilize the network with a range of RTTs, the marking threshold, $K$, needs to be set conservatively by taking the maximum RTT in the network. Thus, small-RTT flows do not fully benefit from the ECN-based congestion control as they could have filled the link with a much lower threshold and with a much lower queueing delay.

We vary the previous ns-2 simulation with different values of $K$ (10, 40, and 80) and RTTs (200 $\mu$s to 800 $\mu$s). Figure 3.5 illustrates the outcome. We plot the resulting utilization against the average queuing delay. To achieve 100% utilization with 800 $\mu$s RTT flows, the marking threshold should be at least 80, but in this case, 200 $\mu$s RTT flows experience 43.2 $\mu$s of extra queueing delay.

Note the queuing delay-utilization trade-off is not specific to DCTCP. Any TCP-like congestion avoidance behaviors require buffering packets proportional to the bandwidth-delay product to achieve high utilization [20]. Our accurate feedback, however, enables us to break this tradeoff.

### 3.3.2 Benefits of latency-based feedback

Accurate congestion feedback often enable innovation in congestion and flow scheduling behaviors. For example, RCP has enabled many different transport solutions, such as D3 [18], FCP [17], Baraat [39], Oktopus [38], and Hedrian [40], because RCP generates accurate rate feedback. We believe accurate latency feedback
can be beneficial in many regards, including sub-packet-level congestion control [41] and multipath congestion control [42]. In this work we focus on one of the more fundamental benefits: we show that accurate latency feedback enables us to break the utilization-latency trade-offs inherent in the bandwidth-delay product.

To highlight the benefit, we compare DCTCP (ECN-based) and DX, a latency-based congestion control algorithm we develop in this dissertation. Because latency feedback signals the amount of excessive packets in the network, it allows senders to calculate the maximum number of packets it can send to target full utilization and zero queueing delay. We show the aggregate sending rate (or input load) of 40 senders sharing a 10 Gbps bottleneck link, and the queue size distribution measured at the bottleneck queue using simulations. When the
senders transmit packets faster than 10 Gbps in total, the excess goes to the bottleneck queue. Figure 3.6 shows the aggregate sending rate, and Figure 3.7 shows the distribution of queue lengths.

Due to the low accuracy of ECN feedback, senders repeatedly overshoot or undershoot the link capacity, and the sending rate fluctuates between 9.26 Gbps to 10.52 Gbps. Latency feedback, on the other hand, shows much less fluctuation. We see that latency-based feedback significantly reduces the queue length (by more than 9x); the median delay for latency-based feedback is only 3 packets compared to ECN’s 28 packets.
Chapter 4. Accurate Latency Feedback Measurement

In Chapter 3, we have learned that latency feedback has advantages in accuracy and granularity. To benefit from such quality feedback, the next step is to measure accurate latency in real-world networks. This chapter describes our methodology for accurate end-to-end delay measurement and queueing delay calculation. We begin by explaining the sources of errors in delay measurements and present our solutions to remove each type of delay noise. Based on our design, we implement the delay measurement methodology and evaluate the accuracy in comparison to the ground truth queue length retrieved by the network switch.

4.1 Removing Sources of Errors

<table>
<thead>
<tr>
<th>Source of error</th>
<th>Elimination technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-host network stack</td>
<td>Exclude host stack delay</td>
</tr>
<tr>
<td>(≈ 100µs)</td>
<td></td>
</tr>
<tr>
<td>I/O batching &amp; DMA bursts</td>
<td>Burst reduction &amp; error calibration</td>
</tr>
<tr>
<td>(tens of µs)</td>
<td></td>
</tr>
<tr>
<td>Reverse path queuing</td>
<td>Use difference in one-way latency</td>
</tr>
<tr>
<td>(≈ 100µs)</td>
<td></td>
</tr>
<tr>
<td>Clock drift</td>
<td>Frequent base delay update</td>
</tr>
<tr>
<td>(long term effect)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Source of errors in latency measurement and our technique for mitigation

Latency measurement can be inaccurate for many reasons including variability in end-host stack latency, NIC queuing delay, and I/O batching. In this section, we describe several techniques to eliminate such sources of errors. Our goal is to achieve a level of accuracy that can distinguish even a single MSS packet queuing at 10 Gbps, which is 1.2 µs. This is necessary to target near zero queuing as congestion control should be able to back off even when a single packet is queued.
Before we introduce our solutions to each source of error, we first show how noisy the latency measurement is without any care. Figure 4.1 shows the round trip time measured by the sender’s kernel when saturating a 10 Gbps link; we generate TCP traffic using iperf [43] on Linux kernel. the sender and the receiver are connected back to back, so no queueing is expected in the network. Our measurement shows that the round-trip time varies from 23 µs to 733 µs, which potentially gives up to 591 packets of error. The middle 50% of RTT samples still exhibit wide range of errors of 111 µs that corresponds to 93 packets. These errors are an order of magnitude larger than our target latency error, 1.2 µs.

Table 4.1 shows four sources of measurement errors and their magnitude. We eliminate each of them to achieve our target accuracy (∼1µsec).

**Removing host stack delay:** End-host network stack latency variation is over an order of magnitude larger than our target accuracy. Our measurement shows about 80µs standard deviation, when the RTT is measured in the
Linux kernel’s TCP stack. Thus, it is crucial to eliminate the host processing delay in both a sender and a receiver.

For software timestamping, our implementation choice eliminates the end host stack delay at the sender as we timestamp packets right before the TX, and right after the RX on top of DPDK [25]. Hardware timestamping innately removes such delay.

Now, we need to deal with the end-host stack delay at the receiver. Figure 4.2 shows how DX timestamps packets when a host sends one data packet and receives back an ACK packet. To remove the end host stack delay from the receiver, we simply subtract the \( t_3 - t_2 \) from \( t_4 - t_1 \). The timestamp values are stored and delivered in the option fields of the TCP header.

**Burst reduction:** TCP stack is known to transmit packets in a burst. The amount of burst is affected by the window size and TCP Segmentation Offloading (TSO), and ranges up to 64 KB. Burst packets affect timestamping because all packets in a TX burst get the almost the same timestamp, and yet they are received by one by one at the receiver. This results in an error as large as 50\( \mu s \).

To eliminate packet bursts, we use a software token bucket to pace the traffic at the link capacity. The token bucket is a packet queue and drained by polling in SoftNIC [44].

At each poll, the number of packets drained is calculated based on the link rate and the elapsed time from the last poll. The upper bound is 10 packets, which is enough to saturate 99.99% of the link capacity even in 10 Gbps networks. We note that our token bucket is different from TCP pacing or the pacer in HULL [20] where each and every packet is paced at the target rate; our token bucket is simply implemented with very small overhead. In addition, we keep a separate queue for each flow to prevent the latency increase from other flows’ queue build-ups.

**Error calibration:** Even after the burst reduction, packets can be still batched for TX as well as RX. Interestingly, we find that even hardware timestamping is subject to the noise introduced by packet bursts due to its implementation. We run a simple experiment where sending a traffic near line rate 9.5 Gbps from a sender to a receiver connected back to back. We measure the inter packet gap using hardware timestamps, and plot the results in Figure 4.3. Ideally, all packets should be spaced at 1.23\( \mu s \). As shown in the figure, a large portion of the packet gaps of TX and RX falls below 1.23\( \mu s \). The packet gaps of TX are more variable than that of RX, as it is directly affected by I/O batching, while RX DMA is triggered when a packet is received by the NIC. The noise in the H/W is caused by the fact that the NIC timestamps packets when it completes the DMA, rather than timestamping
them when the packets are sent or received on the wire. We believe this is not a fundamental problem, and H/W timestamping accuracy can be further improved by minor changes in implementation.

In this paper, we employ simple heuristics to reduce the noise by accounting for burst transmission in software. Suppose two packets are received or transmitted in the same batch as in Figure 4.4. If the packets are spaced with timestamps whose interval is smaller than what the link capacity allows, we correct the timestamp of the latter packet to be at least transmission delay away from the former packet’s timestamp. In our measurement at 10Gbps, 68% of the TX timestamp gaps need such calibration.
4.2 One-way Delay Measurement

4.2.1 One-way queuing delay

So far, we have described techniques to accurately measure RTT. However, RTT includes the delay on the reverse path, which is another source of noise for determining queuing on the forward path. A simple solution to this is measuring one-way delay which requires clock synchronization between two hosts. PTP (Precision Time Protocol) enables clock synchronization with sub-microseconds [45]. However, it requires hardware support and possibly switch support to remove errors from queuing delay. It also requires periodic synchronization to compensate clock drifts. Since we are targeting a microsecond level of accuracy, even a short term drift could affect the queuing delay measurement. For these reasons, we choose not to rely on clock synchronization.

Our intuition is that unlike one-way delay, queuing delay can be measured simply by subtracting the baseline delay (skewed one-way delay with zero queuing) from the sample one-way delay even if the clocks are not synchronized. For example, suppose a clock difference of 5 seconds, as depicted in Figure 4.5. When we measure one-way delay from A to B, which takes one second propagation delay (no queuing), the one-way delay measured would be -4 seconds instead of one second. When we measure another sample where it takes 2 seconds due to queuing delay, it would result in -3 seconds. By subtracting -4 from -3, we get one second queuing delay.

Now, there are two remaining issues. First is obtaining accurate baseline delay, and second is dealing with clock drifts. When there is no clock drift, the baseline can be obtained by picking the minimum one-way delay amongst many samples. The frequency of zero queuing being measured depends on the congestion control algorithm behavior. Since we target near zero-queuing, we observe this every few RTTs.

We describe the delay measurement components in more detail below.

At the sender side, we measure and update three types of delay measurements: sample one-way delay, base one-way delay, and base RTT. The queuing delay is calculated by subtracting the base one-way delay from the sample one-way delay. Base RTT is used to calibrate the base one-way delay. We explain how each delay is measured and updated in detail below.

Sample one-way delay (OWD): First, the sender writes its local time in the Timestamp Value (TSval) field as a TCP option in the packet header. Once the packet arrives at the receiver, the receiver reads the timestamp, computes the difference in timestamps by subtracting the sender’s timestamp from the receiver’s local time. The
difference is written in the TSval field of the ACK packet and is returned to the sender. When the sender reads the difference from the received ACK packet, the sender treats this difference as a sample one-way delay.

**Base one-way delay (OWD):** Base one-way delay represents the one-way delay measured when queueing is zero. The best practice for obtaining the one-way base delay is picking the minimum delay among the observed samples; smaller delay means less queueing, and thus the minimum of past measurements should be the closest obtainable to zero queueing. TCP Vegas uses this approach in obtaining base RTT [35].

**RTT measurement:** When a sender transmits a packet with the TSval option, the receiver echoes back the sender’s timestamp with the Timestamp Echo Reply (TSecr) option in the ACK packet. When the ACK packet arrives at the sender, the sender can calculate the sample RTT from its current clock time and the echoed back timestamp. We set the base RTT as the minimum of past sample RTT measurements.

We compute the sample queuing delay by subtracting the based OWD from the sample OWD. We then use the average queueing delay in a window as the latency feedback in DX. Note even if the two clocks are not synchronized, the queueing delay measurements are accurate without clock drifts.

### 4.2.2 Handling clock drift

A standard clock drifts only 40 nsecs per msec [46]. This means that the relative error between two measurements (e.g., base one-way delay and sample one-way delay) taken from two clocks during a millisecond can only contain tens of nanoseconds of error. Thus, we make sure that base one-way delay is updated frequently (every few round trip times). One last caveat with updating base one-way delay is that clock drift differences can cause one-way delay measurements to continuously increase or decrease. If we simply take minimum base one-way delay, it causes one side to update its base one-way delay continuously, while the other side never updates the base delay because its measurement continuously increases. As a workaround, we update the base one-way delay when the RTT measurement hits the new minimum or re-observe the current minimum; RTT measurements are not affected by clock drift, and minimum RTT implies no queueing in the network. This event happens frequently enough in DX, and it ensures that clock drifts do not cause problems.
4.3 Implementation

We measure four timestamp values: $t_1$ and $t_2$ are the transmission and reception time of a data packet, and $t_3$ and $t_4$ are the transmission and reception time of a corresponding ACK packet.

4.3.1 Software timestamping

To eliminate host processing delay, we perform TX timestamping right before pushing packets to the NIC, and RX timestamping right after the packets are received, at the DPDK-based device driver. As explained in We use `rdtsc` to get CPU cycles and transform this into nanoseconds timescales. We correct timestamps using techniques described in the previous section. All four timestamps must be delivered to the sender to calculate the one-way delay and the base RTT. We use TCP’s option fields to relay $t_1$, $t_2$, and $t_3$.

To calculate one-way delay, the DX receiver stores a mapping from expected ACK number to $t_1$ and $t_2$ when it receives a data packet. It then puts them in the corresponding ACK along with the ACK’s transmission time ($t_3$). The memory overhead is proportional to the arrived data of which the corresponding ACK has not been sent yet. The memory overhead is negligible as it requires store 8 bytes per in-flight packet. In the presence of delayed ACK, not all timestamps are delivered back to the sender, and some of them are discarded.

4.3.2 Hardware timestamping

We implemented hardware-based timestamping on Mellanox ConnectX-3. We use a DPDK-ported Mellanox driver. Although the hardware supports RX/TX timestamping for all packets, its driver did not support TX timestamping. We modified the driver to allow RX/TX timestamp all packets.

The NIC hardware delivers timestamps to the driver by putting the timestamp in the ring descriptor when it completes the DMA. This causes an issue with the previous logic to carry $t_1$ in the original data packet. To resolve this, we store mapping of expected ACK number to the $t_1$ at the sender, and retrieves this when ACK is received.

4.3.3 LRO handling

Large Receive Offload (LRO) is a widely used technique for reducing CPU overhead on the receiver side. It aggregates received TCP data packets into a large single TCP packet and pass to the kernel. It is crucial to achieve 10 Gbps or beyond in today’s Linux TCP stack. This affects DX in two ways. First, it makes the TCP receiver
generate fewer number of ACKs, which in turn reduce the number of \( t_3 \) and \( t_4 \) samples. Second, even though \( t_1 \) and \( t_2 \) are acquired before LRO bundling at the driver, we cannot deliver all of them back to the kernel TCP stack due to limited space in the TCP option header. To work around the problem, for each ACK that is processed, we scan through the previous \( t_1 \) and \( t_2 \) samples, and deliver the aggregate information which is average with sample count. In fact, instead of passing all timestamps to the TCP layer, we only passes one-way delay \( t_2 - t_1 \) and rtt \(((t_4 - t_1) - (t_3 - t_2))\)

4.3.4 Burst mitigation

As shown in the previous section, burstiness from I/O batching incurs timestamping errors. To control burstiness, we implement a simple token bucket with burst size of \( MTU \) and rate set to link capacity. SoftNIC does polling on token bucket to draw packets and pass it to the timestamping module or the NIC. If polling loop takes longer than the transmission time of a packet, the token bucket emits more than one packet, but limits number of packets to keep up with link capacity.

4.4 Evaluation of Latency Measurement

4.4.1 Accuracy of queuing delay in Testbed

In our latency measurement evaluation, we ask the following question and answer with testbed experiments.

- Can DX obtain the accuracy of a single packet’s queuing delay in high-speed networks?

For our testbed experiments, we use Intel 1 GbE/10 GbE NICs for software timestamping and Mellanox ConnectX-3 40 GbE NIC for hardware timestamping; the Mellanox NIC is used in 10 Gbps mode due to the lack of 40 GbE switches.

Effectiveness of noise reduction techniques: To quantify the benefit of each technique, we apply the techniques one by one and measure RTT using both software and hardware. Two machines are connected back to back, and we conduct RTT measurement at 10 Gbps link. We plot the standard deviation in Figure 4.6. Ideally, the RTT should remain unchanged since there is no network queueing delay. In software-based solution, we reduce the measurement error (presented as standard deviation) down to 1.98 \( \mu s \) by timestamping at DPDK and applying burst control and calibration. Among the techniques, burst control is the most effective, cutting down the error
Techniques

Figure 4.6: Improvements with noise reduction techniques

Figure 4.7: Effect of calibration in H/W timestamped inter-packet gap at 10 Gbps

by 23.8 times. In hardware solution, simply timestamping at NIC achieves comparable noise with all techniques applied in the software solution. After inter-packet interval calibration, the noise drops further down to 0.53 µs, less than half of a single packet’s queueing delay at 10 Gbps, which is within our target accuracy.

Calibration of H/W timestamping: We look further into how calibration affects the accuracy of hardware timestamping. Figure 4.7 shows the CDF of inter packet gap measurements before and after calibration for both RX and TX. The calibration effectively removes the inter packet gap samples smaller than link transmission delay which originally took up 68% for TX and 32% for RX.

Overall RTT measurement accuracy improvement: Now, we look at how much overall improvements we made on the accuracy of RTT measurement. We plot the CDF of RTT measurement for our technique using hardware
Figure 4.8: Improvement on RTT measurement error compared to kernel’s

and RTT measured in the Kernel in Figure 4.8. The total range of RTT has decreased by 62 times, from $710 \mu s$ to $11.38 \mu s$. The standard deviation is improved from $80.7 \mu s$ to $0.53 \mu s$ by two orders of magnitude, and falls below a single packet queuing at 10 Gbps.

**Verification of queuing delay:** Now that we can measure RTT accurately, the remaining question is whether it leads to accurate queuing delay estimation. We conduct a controlled experiment where we have a full control over the queuing level. To create such scenario, we saturate a port in a switch by generating full throttle traffic from one host, and inject a MTU-sized ICMP packet to the same port at fixed interval from another host. This way, we increase the queuing by a packet at fixed interval, and we measure the queuing statistics from the switch to verify our queuing delay measurement.
We first test our simulation methodology by observing RTT measurements with different ICMP packet injection rate. Figure 4.9 shows the RTT measurements taken from the TCP flow. We vary the ICMP ping injection rates from 10 to 40 packets per second in this simulation. The slope in Figure 9 accurately represents the packet accumulation rate at the switch buffer; twice the rate results in twice the slope in our RTT measurements.

Figure 4.10 shows the time series of queuing delay measured by DX along with the ground truth queue occupancy measured at the switch (marked as red squares). We use software and hardware timestamping for 1 Gbps and 10 Gbps, respectively. Every time a new ping packet enters the network, the queueing delay increases by one MTU packet transmission delay: 12 $\mu$s at 1 Gbps and 1.2 $\mu$s at 10 Gbps. The queue length retrieved from the switch also matches our measurement result. The result at 10 Gbps seems noisier than at 1 Gbps due to the
smaller transmission delay; note that the scale of Y-axis is different in two graphs. Note that the fastest interval of queue length retrieval at the switch is 20ms, so the ground truth may not align exactly with the queueing delay measurements.

Overall, we observe that our noise reduction techniques can effectively eliminate the sources of errors and result in accurate queuing delay measurement.
Chapter 5. DX Congestion Control

In this chapter, we develop a new latency-based congestion control algorithm called DX, which utilizes the accurate queueing delay measurement in the previous chapter. We begin by pointing out the limitations of TCP Vegas, the most popular latency-based congestion control algorithm, when used in datacenters. We then explain the core algorithm of DX congestion control and give details on our implementation. Finally, we evaluate DX algorithm against DCTCP and HULL and show improvements in lowering queueing delay.

5.1 Limitations of TCP Vegas

Our latency measurement serves as much more accurate congestion feedback than previous kernel-based latency measurement, so existing latency-based congestion control algorithms can also benefit from it. In this subsection, we study whether existing latency-based algorithms can be used in datacenter networks to meet the low queueing delay requirement. The first latency-based algorithm proposed in wide-area networks is TCP Vegas [35], and other later proposed algorithms share the same core idea with TCP Vegas. Therefore we focus on TCP Vegas and analyze its performance in datacenter networks. If TCP Vegas turns out to work well, then we will not need to develop another algorithm and just re-use TCP Vegas. If not, we need to figure out why TCP Vegas does not work and use the lessons learned to design a new algorithm.

We conduct ns-2 simulation in a dumbbell topology to test if TCP Vegas achieves low queueing delay. In this simulation, we have ten idle senders in the beginning, and we activate each sender with 0.5 second interval so that we have ten active flows in the end. Figure 5.1 shows the queueing delay evolution as the number of flows is increased. We notice that the queueing delay increases with the number of flows in the bottleneck link; each flow adds up its own share of queueing to existing queueing. The queue length is consistently as high as 42 packets from 4.5s to 5.0s where ten flows are sharing the link. As datacenter workloads are very dynamic and the number of flows is not bounded, TCP Vegas cannot always guarantee low queueing delay.

We observe another drawback of TCP Vegas in the fairness among flows. According to the TCP Vegas algorithm, a sender determines the congestion level from (measured RTT - base RTT). This approach can provide...
fairness only when all the senders maintain the same base RTT value. The simulation result, however, tells us otherwise. Table 5.1 shows the base RTT of each sender retrieved at the end of simulation. The first flow has 205us for base RTT and the last flow has 238us; later flows in the network get larger base RTT. In this case, the last flow under-estimates the congestion level and tries to send faster than it is supposed to.

From the above simulation, we learn two lessons to be used in designing a new congestion control algorithm for datacenters: i) the algorithm should be able to drop the queue length down to zero quickly as soon as it observes congestion; ii) the algorithm should take into account the number of flows in the network when decreasing window size. We explain how we reflect these lessons in our algorithm in the next subsection.

<table>
<thead>
<tr>
<th>Flow ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base RTT (us)</td>
<td>205</td>
<td>205</td>
<td>209</td>
<td>213</td>
<td>215</td>
<td>221</td>
<td>224</td>
<td>228</td>
<td>233</td>
<td>238</td>
</tr>
</tbody>
</table>

Table 5.1: Base RTT measured by ten TCP Vegas senders

5.2 DX Core Algorithm

We present the first congestion control algorithm to target zero queuing delay based on implicit feedback, without in-network support. Because latency feedback signals the amount of excessive packets in the network, it allows senders to calculate the maximum number of packets to drain from the network while achieving full utilization. This section presents the basic mechanisms and design of our new congestion control algorithm, DX.

Our target deployment environment for DX is datacenter networks and we limit our evaluations to datacenter
networks. Also we assume that DX is the only congestion control used in the environment.

Using the queueing delay, DX makes a decision on whether to increase or decrease congestion window in the next round at every RTT. Zero queueing delay indicates that there is still more room for packets in the network, so window size is increased by one at a time (i.e., additive increase). On the other hand, any positive queueing delay means that a sender must decrease the window. We update the window size according to the formula below:

$$new\, CWND = \begin{cases} \text{CWND} + 1, & \text{if } Q = 0 \\ \text{CWND} \times (1 - \frac{Q}{V}), & \text{if } Q > 0, \end{cases} \quad (5.1)$$

where $Q$ represents the latency feedback, that is, the average queueing delay in the current window, and $V$ is a self-updated coefficient of which role is critical in our congestion control.

When $Q > 0$, the window should decrease proportional to the current queueing. By how much should we decrease window to drain the queue completely? This depends on the number of flows sharing the bottleneck. The aggregate sending rate of these flows should decrease to drain the flow. $V$ is the coefficient that accounts for the number of competing flows. We drive the value of $V$ using analysis below.

We denote the link capacity (packets / sec) as $C$, the base RTT as $R$, single-packet transmission delay as $D$, the number of flows as $N$, and the window size and the queueing delay of flow $k$ at time $t$ as $W_{k}(t)$ and $Q_{k}(t)$, respectively. Without loss of generality, we assume at time $t$ the bottleneck link fully utilized and the queue size is zero. This corresponds the case where the sum of the window size equals to the bandwidth delay product $C \cdot R$:

$$\sum_{k=1}^{N} W_{k}(t) = C \cdot R \quad (5.2)$$

Since none out of $N$ flows experiences congestion, they all increase their window sizes by one at time $t + 1$:

$$\sum_{k=1}^{N} W_{k}(t+1) = C \cdot R + N \quad (5.3)$$

Now all the senders observe a positive queueing delay and they respond by decreasing the window size using the multiplicative factor, $1 - Q/V$, as in (5.1). As a result, at time $t + 2$, we expect fewer packets in the network.
However, we want just enough packets to fully saturate the link and achieve zero queuing delay in the next round.

We calculate the total number of packets in the network (in both the link and the queues) at time $t + 2$ from the sum of window size of all the flows.

$$\sum_{k=1}^{N} W_k(t+2) = \sum_{k=1}^{N} W_k(t+1) \left(1 - \frac{Q_k(t+1)}{V}\right)$$  \hspace{1cm} (5.4)

Assuming every flow experiences maximum queueing delay $N \cdot D$ in the worst case, we get:

$$\sum_{k=1}^{N} W_k(t+2) = \sum_{k=1}^{N} W_k(t+1) \left(1 - \frac{N \cdot D}{V}\right)$$

$$= (C \cdot R + N)(1 - \frac{N \cdot D}{V})$$  \hspace{1cm} (5.5)

We want total number of in-flight packets at time $t + 2$ to equal to the bandwidth delay product:

$$(C \cdot R + N)(1 - \frac{N \cdot D}{V}) = C \cdot R$$  \hspace{1cm} (5.6)

Solving for $V$ results in:

$$V = \frac{N \cdot D}{1 - \frac{C \cdot R}{C \cdot R + N}}$$  \hspace{1cm} (5.7)

Among the variables required to calculate $V$, the only unknown is $N$, which is the number of concurrent flows. The number of flows can be estimated from the sender’s own window size because $d_x$ achieves fair-share throughput at steady state. For notational convenience, we denote $W_k(t+1)$ as $W^*$ and rewrite (5.3) as:

$$\sum_{k=1}^{N} W_k(t+1) = N \times W^* = C \cdot R + N \Leftrightarrow N = \frac{C \cdot R}{W^* - 1}$$

Using (5) and replacing $D$, single-packet transmission delay, with $(1/C)$, we get:

$$V = \frac{R \cdot W^*}{W^* - 1}$$  \hspace{1cm} (5.8)

In calculating $V$, the sender only needs to know the based RTT, $R$, and the previous window size $W^*$. No additional measurement is required. We also do not need to rely on external configuration or parameter settings, unlike the ECN-based approaches. Even if the link capacity in the network varies across links, it does not affect our calculation of $V$. 

– 42 –
Note that even though we target queuing delay of zero after detecting the congestion. Queuing is inevitable because of the additive increase. Our maximum queueing buildup is $N$ packets when flows are synchronized, where $N$ is the number of flows. Note this does not depend on the bandwidth delay product. In contrast, DCTCP’s the maximum queue buildup is $K + N$, where is a function of delay bandwidth product [14].

5.3 Window Convergence and Steady-state in DX

Here we provide a simple analysis on steady-state behavior of DX. When a certain number of flows are sharing a bottleneck link, we can calculate the ideal window size for each flow that makes no queueing at all. We call this lower bound window size. Now we compare the theoretical window size provided by DX formula to the lower bound value to evaluate how close DX is from ideal. Then we also conduct a simulation in the same network scenario and compare the window size in simulation to theoretical window size. We present the result in Figure 5.2. To test multiple cases, we increase the number of flows from two to ten and plot all of them in one figure. From the figure, we can say that both theoretical and simulational window size values are very close to the ideal lower bound.

Next we observe the convergence behavior of DX in comparison to DCTCP. For this observation, we use the same convergence analysis methodology from an AIMD analysis paper [47]. In the beginning, one flow is occupying the total link bandwidth. Then the second flow comes into the link, and after a certain amount of time, both flows converge to the fair share throughput. We plot the change in both flows’ window size in Figure 5.3.
X-axis is the window size of the first flow, and Y-axis is the window size of the second flow. By comparing two graphs, we see that DX takes a more efficient path to the convergence point. After the flows converge to the steady-state, DX also shows more stable behavior than DCTCP.

5.4 Implementation

We have implemented DX in two parts: time-stamping and delay calculation as a module inside a DPDK-based NIC driver called SoftNIC and latency based congestion control in the Linux’s TCP stack. It allows packets to be relayed between DPDK and Linux’s networking stack. This separation provides few advantages: (i) it measure latency more accurate than doing so in the Linux Kernel; (ii) legacy applications can take advantage of DX without modification; and (iii) it separates the latency measurement from the TCP stack, and hides the differences in hardware implementations, such as timestamp clock frequencies or timestamping mechanisms. We
present the implementation of software- and hardware-based latency measurements and modifications to the kernel TCP stack to support latency feedback.

We implement DX congestion control algorithm in the Linux 3.13.11 kernel. We add DX as a new TCP option that consumes 14 bytes of additional TCP header. The first 2 bytes are for the option number and the option length required by TCP option parser. The remaining 12 bytes are divided into three 4 byte spaces and used for storing timestamps and/or an ACK number.

Most of modifications are made in the tcp_ack() function in TCP stack. This is triggered when an ACK packet is received. An ACK packet carries one-way delay and RTT in the header that are pre-calculated by the DPDK-based device driver. For each round trip time, the received delays are averaged and used for new CWND calculation. The current implementation takes the average one-way delay observed during the last round trip.

Practical considerations: In real-world networks, a transient increase in queueing delay $Q$ does not always mean that the network is congestion. Reacting to wrong congestion signals results in low link utilization. There are two sources of error: measurement noise and instant queueing due to packet bursts. Although we have shown that our latency measurement has a low standard deviation up to about a microsecond, it can still trigger a window reduction as DX reacts to a positive queueing delay whether large or small. On the other hand, instant queueing can happen with even very small number of packets. For example, if two packets arrive at the switch at the exactly same moment, one of them will be served after the first packet’s transmission delay.

To tackle such practical issues, we come up with two simple techniques. First, we use headroom when determining congestion; DX does not decrease window size when $Q < \text{headroom}$.

Second, to be robust against transient increase in delay measurements, we use the average queueing delay during an RTT period. In an ideal network without packet bursts, the maximum queueing delay is a good indication of excess packets. In real networks, however, the maximum is affected easiest by instant queueing. Taking the minimum removes the burstiness best, but it also removes congestion experienced by part of the window. Hence we chose the average to balance them out.

Note that DCTCP, a previous ECN-based solution, also suffers from bursty instant queueing and requires higher ECN threshold in practice than theoretic calculation [14].
5.5 Evaluation of DX Performance

We evaluate the performance of DX using both testbed experiments and ns-2 simulations. Throughout the evaluation, we answer the following questions:

- Can DX achieve minimal queuing delay while achieving high utilization?
- How does DX perform in large scale networks with realistic workloads?

By using testbed experiment, we evaluate DX against the DCTCP and verify that it reduces queuing in the switch up to five times. Next, we use ns-2 packet level simulation to conduct more detailed analysis and and evaluate DX in large-scale with realistic workload. First, we verify the DX’s effectiveness by looking at queuing delay, utilization and fairness. We then quantify the impact of measurement errors on DX to evaluate its robustness. Finally, we perform large scale evaluation to compare DX’s overall performance against the state of the art: DCTCP [14] and HULL [20].

5.5.1 DX congestion control in testbed

Using the accurate queueing delay measurements, we run our DX prototype with three servers in our testbed; two nodes are senders and the other is a receiver. We use iperf [43] to generate TCP flows for 15 seconds. For comparison, we run DCTCP in the same topology. The ECN marking threshold for DCTCP is set to the recommended value of 20 at 1 Gbps and 65 at 10 Gbps [14]. During the experiment, the switch queue length is measured every 20 ms by reading the register values from the switch chipset. We first present the result at 1 Gbps bottleneck link in Figure 5.4a. In both protocols, two senders saturate the bottleneck link with fair-share throughput. The queue length is measured in bytes and converted into time.

We observe that DX consistently reduces the switch queue length compared to that of DCTCP. The average queueing delay of DX, 37.8 µs, is 4.85 times smaller than that of DCTCP, 183.4 µs. DX shows 5.33x improvement in median queue length over DCTCP (3 packets for DX and 16 packets for DCTCP). DCTCP’s maximum queue length goes up to 24 packets, while DX peaks at 8 packets.

We run the same experiment with 10 Gbps bottleneck. For 10 Gbps, we additionally run DX with hardware timestamp using Mellanox ConnectX-3 NIC. Figure 5.4b shows the result. DX (HW) denotes hardware timestamping, and DX (SW) denotes software timestamping. DX (HW) decreases the average queue length by 1.67
times compared to DCTCP, from 43.4 $\mu s$ to 26.0 $\mu s$. DX (SW) achieves 31.8 $\mu s$ of average queuing delay. The result also shows that DX effectively reduces the 99th-percentile queue length by a factor of 2 with hardware timestamping; DX (HW) and DX (SW) achieve 52 packets and 38 packets respectively while DCTCP achieves 78 packets.

To summarize, latency feedback is effective in maintaining low queue occupancy than ECN feedback, while saturating the link. DX achieves 4.85 times smaller average queue size at 1 Gbps and 1.67 times at 10 Gbps compared to DCTCP. DX reacts to congestion much earlier than DCTCP and reduces the congestion window to the right amount to minimize the queue length while achieving full utilization. DX achieves the lowest queueing delay among existing end-to-end congestion controls with implicit feedback that do not require any switch.
In the next section, we also show that DX is even comparable to HULL, a solution that requires in-network support and switch modification.

### 5.5.2 Impact of measurement error in simulation

We run ns-2 simulation using a dumbbell topology with 10 Gbps link capacity. Before the main simulation, we evaluate the impact of latency noise to the headroom size and average queue length. We generate latency noise using normal distribution with varying standard deviation. The noise level is multiples of 1.2 μs, single packet’s transmission delay. As the simulated noise level increases, we need more headroom for full link utilization. Figure 5.5 shows the required headroom for full utilization and the resulting queue length in average. We observe that even if the noise becomes as large as 6 μs, DX can sustain noise error by simply increasing headroom size followed by the same amount of increase in queue length. Note that the standard deviation of our hardware timestamping is only 0.53 μs.

### 5.5.3 Microscopic behavior in simulation

In this section, we look at the commonly used metrics for evaluating congestion control algorithms, and show the potential benefits of using latency based feedback for congestion control.

First, we run ns-2 simulation using a dumbbell topology with 10Gbps link capacity. We vary the number of simultaneous flows from 10 to 30 as queuing delay and utilizations are correlated with it; the number of senders has a direct impact on queueing delay as shown in DCTCP [14]. We measure the queuing delay and utilization,
and plot them in Figure 5.6.

**Queueing delay:** Many distributed applications with short flows are sensitive to the tail latency as the slowest flow that belongs to a task determines the completion time of the task [48]. Hence, we look at the 99th percentile queuing delay as well as the average queueing delay. On average, DX achieves 6.6x smaller queueing delay than DCTCP with ten senders, and slightly higher queuing delay than HULL. At 99th percentile, DX even outperforms HULL by 1.6x to 2.2x. The reason that DX achieves such low queuing is because of the immediate reaction to the queuing whereas both DCTCP and HULL uses weighted averaging for reducing congestion window size that takes multiple round trip times.

**Utilization:** DX achieves 99.9% of utilization which is comparable to DCTCP, but with much smaller queuing. HULL sacrifices utilization for reduction in the queuing delay achieving about 90% of the bottleneck link capacity. The reason that DX can achieve such high utilization with low queuing is because of the accuracy of feedback. Since DX reacts to the queuing early, it requires to target the accurate window to prevent under-utilization.

In Figure 5.7, we plot the tradeoff between queuing and utilization for three congestion control algorithms. It is better the closer to the left most corner which means lower queuing and high utilization. We see that DX is much closer to the ideal in comparison to both DCTCP and HULL. It clearly shows the benefit of fine grained and accurate feedback.

**Convergence time:** Convergence time is an important performance metric for new incoming flows. When no other prioritization or scheduling policies is defined, the network is expected to offer fair-share throughput to a new flow as soon as possible. We measure the throughput convergence time of two DX flows by generating a new flow after the existing flow occupies the full link capacity. Figure 5.8 compares the result from DCTCP and DX; we omit the result for HULL here as it has very similar behavior with DCTCP. We define convergence as reaching 90% of expected fair share throughput.

DX takes 45msec to converge while DCTCP takes 51msec. The main source of improvement is DX’s efficient window control. As shown in the Figure 5.8a, a DCTCP flow frequently changes its rate, and most of the time the large fraction of increment is later offset by decrement. A DX flow, on the other hand, minimizes such inefficient rate change. Note that DX converges slower than TCP reno (31msec of convergence time [49]), which aggressively relinquish the bandwidth by halving windows.
Figure 5.6: Queueing delay and utilization of HULL, DCTCP, and DX
Figure 5.7: Trade-off between utilization and queueing delay

Figure 5.8: Convergence of a new flow in DCTCP and DX

Fairness and throughput stability: To evaluate the throughput fairness, we generate 5 identical flows in the 10 Gbps link one by one with 1 second interval and stop each flow after 5 seconds of transfer. In Figure 5.9, we see that both protocols offer fair throughput to exiting flows at each moment. One interesting observation is that DX flows have more stable throughput than DCTCP flows. This implies that DX provides higher fairness than DCTCP in small time scale. We compute the standard deviation of throughput to quantify the stability; 268 Mbps for DCTCP and 122 Mbps for DX.

5.5.4 Common datacenter traffic patterns in simulation

Here, we focus on specific traffic patterns that are common in data center networks and show the performance benefit of DX.

Incast traffic: Although DX does not originally aim at solving the incast problem [50] by design, maintaining small queue and quick responsiveness to the congestion give advantages to certain type of incast traffic.
When a node in datacenters sends queries to multiple different nodes, the responses to the query can arrive back at the sender simultaneously within a short period of time. To simulate such response traffic in data centers, we generate 40 flows to a single receiver at varying flow generation rate and observe the queue length at the bottleneck switch. Each flow is 2MB size in our simulation.

Figure 5.10a shows the peak queue length at the time of flow arrival. DCTCP suffers from instant queue buildup of hundred packets throughout all the flow generation rates, even at 10 flows per microsecond. On the other hand, DX has lower queue buildup. To examine the difference in handling incast traffic, we show the behavior of congestion window size. Figure 5.10b shows the congestion window size when the incast happens at 40 flows per 100 µs. The result shows that DX reacts to the congestion much earlier than DCTCP does reducing excess packets in the network.

**Small flows with large background flows:** Small flows often co-exist with large bulk transfer in datacenters. In this case, the flow completion time (FCT) of small flows suffers due to the increased queuing delay. To simulate
DX’s behavior in such scenario, we produce small flows with large background flows in the network. The large flow has infinite size starting at the beginning of the simulation, and we increase the number of large flow senders from 10 to 30. We then generate a thousand 1 KB flows according to a Poisson arrival process. Figure 5.11 shows 99th percentile FCT for DCTCP, HULL, and DX. The base RTT between a sender and a receiver is 200 µs. Note that the y-axis begins from the base RTT.

DX achieves the minimum 99th percentile FCT among the three. DX performs 18% and 2% better than DCTCP and HULL respectively with 10 background flows. As the number of background flows increases, the 99%- FCT of DX does not increase as much compared to DCTCP and HULL; 10 additional senders increase only 3-4 µs in FCT. This behavior originates from DX’s zero queue targeting. Even when many large flows are present at the bottleneck switch, the queue length frequently falls down to a small value close to zero.

5.5.5 Multi-bottleneck scenario in simulation

We evaluate the performance of DX in multi-bottleneck scenario. We use the same network topology used in DCTCP evaluation [14] as shown in Figure 5.12. In this topology, there are three sender groups (e.g., SG1, SG2, and SG3) and two receiver groups (e.g., RG1 and RG2). When the simulation begins, SG1 sends best-effort traffic to RG1, and SG2 sends traffic to RG2. At the same time, RG2 also receives traffic from SG3. When all the senders and receivers are active, the 10Gbps network link between Switch 1 and Switch 2 becomes a bottleneck link for SG1 and SG2, and the 1Gbps link between Switch 3 and RG1 becomes a bottleneck link for SG1 and SG3. Therefore the traffic from SG1 to RG1 passes through two bottleneck links. In this scenario, the ideal fair throughput is 50Mbps for SG1 and SG3, and 475Mbps for SG2. When we run DX, SG1 gets 51.5Mbps, SG2 gets 477.7Mbps, and SG3 gets 48.5Mbps. For comparison, we also run DCTCP with the same setting, and we get
46Mbps for SG1, 475Mbps for SG2, 54Mbps for SG3. We conclude that DX provides fair enough and comparable throughput to DCTCP even with multi-bottleneck network scenario.

5.5.6 Real-world datacenter workload in simulation

To understand the performance of DX in a large-scale data center environment, we perform simulations with realistic topology and traffic workload. The network consists of 192 servers and 56 switches that are connected as a 3-tier fat tree; there are 8 core switches, 16 aggregation switches, and 32 top-of-rack switches. All network links have 10Gbps bandwidth, and the path selection is done by ECMP routing. The network topology we use is similar to that of HULL [20]. Once the simulation starts, the flow generator module selects a sender and a receiver randomly and starts a new flow. Each new flow is generated following Poisson process to produce 15% load at the edge. We run simulation until we have 100,000 flows started and finished.

To test realistic workload, we choose flow size according to empirical workload reported from real-world data centers. We use two workload data: web search [14] and data mining [51].

Web search workload: The web search workload mostly contains small and medium-sized flows from a few KB to tens of MB; more than 95% of total bytes come from the flow smaller than 20MB, and the average flow size is 654KB [29]. In Figure 5.13, we present the flow completion times in four flow-size groups: (0KB,10KB),
For the flows smaller than 10KB, DX significantly reduces the 99th percentile FCT; it is 4.9x smaller than DCTCP and 1.9x smaller than HULL. DX also achieves minimal flow completion time in the 10KB-100KB group.

In larger flow size group, the performance of DX falls between DCTCP and HULL. DX achieves 7.7% lower average flow completion time compared to HULL and 20.9% higher than DCTCP for flows of size 10 MB and greater. This is because when ACK packets from other flows share the same bottleneck link, the queuing delay increases slightly. As a result, DX senders respond to the increased queuing delay. This is a side effect of targeting zero queueing. Because ACK packets are small and often piggy-backed on data packets we believe this is not a serious problem, but leave this as future work.

Data mining workload: The data mining workload is comprised of tiny and large-sized flows from hundreds of bytes to 1GB. The flow size is highly skewed that 80% of flows are smaller than 10KB [29] so 95% of bytes come from flows larger than 30MB; the average flow size is 7,452KB. The flow completion time of data mining workload is presented in Figure 5.14.

The performance improvement of DX is more outstanding than search workload. In the three flow groups up to 10MB, DX flows finish early in every case. The biggest benefit comes from the smallest flow group as tail
FCT is 6.0x smaller than DCTCP and 1.9x than HULL. We believe that DX works better in the traffic workload that has skewed flow size distribution. For the largest flow group, DX suffers the same problem from the search workload but still shows shorter completion time than HULL.

5.6 Related Work

5.6.1 Latency-based Feedback in Wide Area Network

There have been numerous proposals for network congestion control since the advent of the Internet. Although the majority of proposals use packet loss to detect network congestion, a large body of work has studied latency feedback. Latency-based TCP all agree on latency being more informative source of measuring congestion level, but the purpose and control mechanism is different in each protocol. TCP Vegas [35] is one of the earliest work and aims at achieving high throughput by avoiding loss. FAST TCP [7] is also designed to quickly reach the fair-share throughput and uses latency for an equation parameter. TCP Nice [52] and TCP-LP [53] operate in low priority minimizing interference with other flows. So far, latency-based approach has only been used in wide area network, and no protocol is known to target zero queueing delay.
5.6.2 ECN-based Feedback in Datacenter Networks

Monitoring congestion level at the switch can help controlling the rate of TCP to minimize queuing. To control the rate of TCP flows so that the switch queue length remains small, monitoring the level of congestion in the network links is prerequisite. As one of the ways to get such information, using the ECN marking in the TCP header has received much attention recently. DCTCP [14] uses the predefined threshold, and end-nodes then count the number of ECN marked packets to determine the degree of congestion and decrease the window size accordingly. HULL [20] is a similar to DCTCP, but sacrifices a small portion of the link capacity with phantom queue implemented at switches to detect congestion early and to achieve lower queueing delay than DCTCP. D²TCP [21] also follows the same line of idea as DCTCP, and it uses gamma correction function to take into account each flow’s deadline when adjusting the congestion window size. As another variant of DCTCP, L²DCT [22] considers flows’ priority when reducing window size, and the priority is determined by the scheduling policy used in the network. ECN* [54] proposed dequeue marking for ECN to work effectively in data centers. The aforementioned ECN marking approaches require modification of the TCP stack in end-node OS as well as minor parameter tunings at switches.

5.6.3 In-network Feedback in Datacenter Networks

A few approaches have proposed to modify network switches in a way that TCP senders or middle switches can learn congestion status more quickly and accurately. D³TCP [18] similar mechanism to RCP so that it can control rate of flows to implement deadline based scheduling. DeTail [28] has implemented a new cross-layer network stack so that flows can avoid congested paths in the network, and PDQ [27] suggests distributed scheduling of flows that posses different priority. On the other hand, another form of in-network feedback is generated by a centralized controller in the network. Fastpass [32] employs this concept and orchestrates the rate of all the nodes in the network. These solutions are much harder to deploy than end-to-end solutions.

5.6.4 Flow Scheduling in Datacenter Networks

Finally, we note that flow scheduling approaches, such as pFabric [29], PDQ [27], Varys [55], and PASE [23], also offer low flow completion times using prioritization and multiple queues. While some solutions intermix the congestion control and flow scheduling [23], we believe that congestion control and flow scheduling are largely
orthogonal. For example, PASE adopts a DCTCP-like rate control scheme for lower priority queues [23] to ensure fairsharing and low queuing delay. Thus, in general, our latency-based feedback is orthogonal to flow scheduling approaches.
Chapter 6. Conclusion

In this dissertation, we design and develop DX, a new congestion control algorithm for datacenters, which operates with accurately measured latency feedback. Before designing DX, we identify the requirements for congestion control in datacenters and the challenges in achieving low queueing delay. We then compare the existing congestion feedback, mainly ECN-based and latency-based, and claim latency feedback has much potential in datacenter networks. As a replacement of ECN-based feedback, we explore latency feedback for congestion control in data center networks. We show that using latency feedback has huge benefit in both accuracy and granularity when compared to ECN. To acquire reliable latency measurements, we develop both software and hardware-based solutions to measure the network-side latency only. Our noise reduction techniques include removing kernel latency, mitigating I/O bursts, and calibrating bursty timestamps. Our measurement results show that we can achieve sub-microseconds of accuracy with the hardware-based solution. Based on the accurate latency feedback, we develop DX that achieves high utilization and low queueing delay in data center networks. DX reduces queueing delay by up to 5.33x times in 1Gbps experiments when compared to DCTCP, a previous ECN-based solution. In large-scale simulations, the queueing delay reduction is comparable or better than HULL. We conduct extensive evaluation in simulation including utilization, queueing delay, convergence time, multi-bottleneck scenario, synthetic workload, and real datacenter workload. Our prototype implementation shows that DX can be a practical solution in the real-world datacenters.

6.1 Summary of contributions

Among our contributions, we would like to emphasize the following two perspectives.

- We evaluate the capability of ECN-based feedback in measuring the network congestion level. Although ECN-based feedback has been widely used in congestion control algorithms for datacenters since DCTCP [14], no previous work has questioned whether such feedback correctly reflects the congestion level. We come up with two metrics, accuracy and granularity, and show that ECN-based feedback is neither accurate nor fine-grained. We also show that the moving average of ECN feedback cannot distinguish heavy congestion
from light congestion.

- We are the first to use end-to-end latency measurement as accurate congestion feedback. It has been almost twenty years since the development of the first latency-based TCP [35], and the common belief that latency cannot provide reliable information due to multiple sources of errors has predominated in the network research area. We have finally overturned this belief using low latency packet processing and hardware timestamping technology.

- We have developed the first end-to-end congestion control algorithm that targets near zero queueing delay. Knowing the exact number of packets in the switch queue from latency measurement, we are able to calculate the optimal window size that fully utilizes the link bandwidth without any queueing. We realized this idea in our algorithm DX and verified the performance with simulation and testbed experiments.

6.2 Concluding remarks and future directions

Explosive growth of online services lead to building large scale datacenters. Such large datacenters require high bandwidth and low latency network to run large-scale processing that spans thousands of servers like map reduce and to process user facing request in very short period at the same time. Many datacenters, however, use commodity switches to reduce the cost whereas HPC cluster use Inifiband or even specialized interconnect for better performance. As a results, data centers typically rely on TCP to control congestions on best effort network.

Minimizing flow completion time and queueing delay has been an active research topic in datacenter networking. Various types of solutions have been proposed in the literature, and we can categorize them in largely two groups. One to add more functions to network switches and get performance benefits from the new features. The other line of work is end-to-end approaches that make changes in only server-side and run with commodity switches. Among the two design choices, we claim that end-to-end approaches are more suitable in datacenter networks for the sake of deployment, and DX is also designed in this spirit.

Although congestion control algorithms have diversified over many decades, one unchanging ground is that congestion control is an interplay between congestion feedback and control algorithm. There have been numerous research efforts in congestion control since TCP [1], but most work has been done in the latter part; the quality of congestion feedback has not been improved much from simple packet loss, roughly measured latency, and ECN
marking. This thesis expands the design space of congestion control by contributing to the congestion feedback part. DX is the first algorithm to utilize this new type of congestion feedback, and we expect that many future algorithms can benefit from our latency feedback to serve their own design goals.

The current noise level of our latency measurement is good enough for 1Gbps as we can confidently differentiate queueing delay from measurement noise. At 10Gbps, we still can infer the number of packets in the switch queue but not as accurate as at 1Gbps. We expect that future improvement in hardware timestamping technology will enable better accuracy at link rates larger than 10Gbps. Nowadays 40Gbps links are being deployed in cutting-edge datacenter networks where transmission delay is only 300ns. To support 40Gbps, the precision of timestamping needs to be in at least hundreds of nanoseconds level, and modern timestamping NICs have already reached such level. Therefore further optimization on DMA operation and timestamping module for measurement noise reduction can allow DX to work in even 40Gbps networks.
References


Summary

Latency-based Congestion Detection and Control for Datacenters

인터넷 서비스들이 처리해야 하는 데이터의 규모가 늘어나면서, 데이터 센터의 성능이 급속도로 향상해야 한다. 이에 따라 데이터 센터의 내부 네트워크에서 필요한 성능 요구 사항들이 새롭게 생겨났는데, 서비스 사용자들이 경험하는 응답 시간은 줄이기 위해 노드와 노드 사이의 delay를 낮추는 것이 가장 큰 이슈 중 하나로 떠올랐다. 기존의 광역 네트워크에서도 낮은 delay가 요구되는 것은 당연한 일이었지만, 데이터 센터 네트워크만의 구조적 특징이 delay 문제의 중요성을 더욱 부각시키게 되었다.

데이터 센터 네트워크가 기존의 광역 네트워크로부터 구별되는 가장 큰 차이점은 노드들 사이의 거리가 상대적으로 매우 짧아서 propagation delay가 전체 delay에서 차지하는 비중이 거의 없어진다는 부분이다. 따라서 스위치에서 발생하는 queueing delay가 전체 delay의 대부분을 차지하게 되고, 이 queueing delay를 줄이는 것이 사용자 경험을 크게 향상시키는 방법이 될 수 있다.

데이터 센터의 queueing delay를 줄이기 위해서 다양한 방법들이 이전 연구에서 제안되었는데, 그 중 가장 대표적인 연구인 DCTCP는 과거의 TCP에서 네트워크 혼잡의 신호로 사용하던 패킷 손실을 대신하여 스위치가 패킷을 마킹할 때 사용하는 ECN 필드를 사용하였다. 이를 통해 네트워크 혼잡을 한 단계씩 증가시켜, 실제로 스위치 큐에서 끝나있는 패킷의 수를 추적하여 그에 따른 타당도 크기를 줄이게 되어 네트워크 링크를 100%로 사용하면서도 스위치 큐를 비교적 낮게 유지할 수 있게 된다. 하지만 이 방식에서는 ECN threshold를 관리자가 설정해주어야 하는 어려움과, 추적한 스위치의 패킷 개수가 확장할 수 있다는 단점이 존재한다. DCTCP 이외에도 HULL과 같은 스위치 기반의 솔루션이 존재하지만, 이들은 성능을 얻기 위해 스위치 내부를 수정해 야하기 때문에, 실제 네트워크에 구현해서 사용하기에 어려움이 따른다.

본 연구에서는 end-to-end delay 측정을 통하여 데이터 센터 네트워크에서 스위치의 쿼 길이를 더욱 정확하게 알아내고, 이 정보를 이용하여 데이터 센터 네트워크의 queueing delay를 효과적으로 줄일 수 있는 새로운 알고리즘인 DX를 제안한다.

먼저 end-to-end delay 측정값으로부터 스위치 큐 길이를 알아내기 위해서, 우리는 스위치 큐가 아닌 다른
곳으로부터 나오는 delay, 즉 네트워크가 아닌 호스트에서 발생되는 delay를 최대한 제거하는 측정 방식을 택하였다. 호스트에서 발생되는 delay를 네트워크 delay로 잘못 받아들일 경우, 스위치의 큐를 실제보다 크게 측정하게 되는 오류가 생길 수 있다. 호스트 delay를 제거하기 위하여 우리는 우선 커널 수준이 아닌 드라이버 수준에서 패킷 타임 스텝핑을 하였고, 더 나은 성능을 위해 패킷이 네트워크 인터페이스 카드(NIC)를 빠져나 가기 직전인 하드웨어 수준에서의 타임 스텝핑도 구현해 테스트 하였다. 이에 추가로 burstiness로부터 오는 순간적인 delay와 하드웨어 DMA로부터 발생하는 delay 보정 기술을 개발하였고, 그 결과 10Gbps 링크 실험의 경우 드라이버 수준 측정에서 2us의 오차, 하드웨어 수준 측정에서 500ns의 오차로 네트워크 delay를 측정할 수 있게 되었다.

이처럼 정확하게 측정된 queueing delay를 이용하여 우리는 DX라는 새로운 네트워크 혼잡 제어 알고리즘을 개발하였다. Reno와 같은 과거의 TCP에서 네트워크 혼잡이 발생했을 때, 원도우 크기를 반으로 줄이던 것에 비해, DX는 전송 노드가 현재 알고 있는 스위치 큐의 패킷 개수를 이용하여 다음 라운드 전송에서 네트워크 링크 bandwidth를 빠르게 제어하기 위해 필요한 원도우 크기를 수학적으로 계산하게 된다. 이 계산 과정에서 필요한 정보들은 전송하는 노드에서 모두 알고 있는 값으로, 네트워크 관리자가 따로 파라미터 설정을 하지 않아도 된다는 장점을 가지고 있다.

우리는 DX 알고리즘을 드라이버와 리눅스 커널에 각각 구현하였다. delay 측정 구현은 Intel DPDK 기반으로 만들어진 SoftNIC 프레임워크 안에서 이루어졌고, 측정된 delay를 받아서 실제 혼잡 제어하는 부분은 리눅스 커널의 TCP 스택에 구현하였다. DX의 성능을 측정하기 위해서 시뮬레이션과 실제 실험을 같이 활용하였는데, 시뮬레이션의 경우 ns-2 시뮬레이터를 사용하였고, 실험은 구현 프로토콜을 실제 머신과 스위치 환경에서 수행하였다. 보유한 머신의 수많은 환경으로 소규모의 성능 측정은 실험을 이용하였고, 보다 대규모의 성능 측정을 위해서는 시뮬레이션을 이용하였다. 먼저, 실험 측정에서는 1Gbps, 10Gbps의 네트워크 링크 하나를 두 개의 노드가 경쟁적으로 사용하는 시나리오를 만들었고, 측정 결과 1Gbps에서 DX가 DCTCP보다 스위치 큐를 5.33배 적게 사용하면서도 링크 utilization을 100%로 유지할 수 있음을 확인하였다. 같은 시나리오의 10Gbps 환경에서는 1.57배의 성능 향상이 있었다. 대규모 시뮬레이션에서는 192개 노드로 이루어진 fat-tree 구조의 데이터센터를 생성하였고, 공개되어 있는 실제 데이터 센터의 워크로드를 사용하여 각 flow의 completion time을 측정하였다. 그 결과, 10KB 이하의 작은 flow의 경우 DX를 이용했을 때 평균적으로 1.9 배 가량 빨라 끝나는 것을 알 수 있었다.

본 연구의 파급 효과는 크게 두 가지로 나누어 볼 수 있다. 먼저 네트워크 혼잡 제어를 위한 신호로 기존의
패킷 손실, ECN을 대체하는 정확한 delay 측정 방법의 제안이다. Delay를 혼잡 제어 신호로 사용하는 개념 자체는 TCP Vegas 등 이전 연구에도 존재했으나, 높은 bandwidth와 낮은 latency를 갖는 데이터 센터 환경에서도 유용하게 쓰일 수 있도록 나노초 수준의 오차로 정확하게 측정을 하는 방법은 없었다. 이러한 새로운 네트워크 혼잡 신호는 미래에 다양한 목적을 가진 제어 알고리즘을 개발의 기반이 될 수 있다. 여기서 더 나아가 본 연구에서는 이러한 신호를 적절 활용하는 알고리즘인 DX를 만들었고, 데이터 센터의 내부 전송에 사용되었을 때 스위치 큐를 효과적으로 줄일 수 있음을 실제 구현 및 실험으로 검증하였다.