Active Queue Management Based on Control-Theoretic Approaches for Diversified Communication Services

February 2021

Ryosuke Alexander Hotchi

A Thesis for the Degree of Ph.D. in Engineering

Active Queue Management Based on Control-Theoretic Approaches for Diversified Communication Services

February 2021

Graduate School of Science and Technology
Keio University

Ryosuke Alexander Hotchi

Acknowledgements

I joined Associate Professor Kubo's laboratory in 2015. This doctoral thesis is a compilation of the six years of work at Keio University. I would like to express my gratitude to all those who allowed me to accomplish this thesis.

First of all, I would like to express my sincere gratitude to my supervisor Associate Professor Dr. Ryogo Kubo. He has been supporting my study throughout my entire research life since I have entered his laboratory. He has spared his precious time to improve my study and paper submission. I could not have attended multiple conferences nor received a Student Paper Award at the international conference NOLTA 2017 without his guidance.

Apart from my supervisor, I would like to express the gratitude to Professor Dr. Masaaki Ikehara, Professor Dr. Yukitoshi Sanada, and Associate Professor Dr. Kunitake Kaneko, for sharing insightful suggestions and giving helpful discussions. They all have played a major role in polishing up this thesis.

I owe my warm gratitude to Mr. Takanori Iwai, the System Platform Research Laboratories, NEC Corporation, for supporting my research and journal publication. His advice has helped me greatly improve the quality of the journal paper.

I am also pleased to say thank you to Mr. Hosho Chibana for giving me multiple instructions at the beginning of my research life in the laboratory. My studies could not have been achieved without his guidance.

I am warmly grateful to KLL Ph.D. Program Research Grant and Research Encouragement Scholarship for Graduate Students for supporting my research life pecuniarily. My research

environment was satisfied with their financial supports.

In the end, I am grateful to my family and everyone who supported me to complete this thesis. My parents have always supported me mentally and financially throughout my entire university life so that I only pay attention to the studies and achieving my objective without any obstacles on the way.

February 2021

Ryosuke Alexander Hotchi

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 Communications Over the Internet

Many things we see these days make use of the Internet. Numerous electronic devices, including personal computers and smartphones, can send and receive various information by accessing the Internet. With the advent of the Internet, it became possible for people worldwide to communicate with each other, control devices remotely, and collect information. In the modern world, it is safe to say that the Internet is essential infrastructure.

Figure 1-1 shows a conceptual diagram showing the transition of communication contents. Each picture shown in Fig. 1-1 denotes a type of communication content, along with the data size example under standard usage. At the dawn of the Internet technology, people used to exchange only text via the Internet, since both the bandwidth of the Internet and the specifications of the electronic devices were not capable of handling a large data. As the technology evolves, the data size of communication contents has become much larger, just like Fig. 1-1 shows. It is not rare to see people downloading movie data with their smartphones in recent years. Even now, the number of Internet users is continuously increasing. In September 2019, the penetration rate of

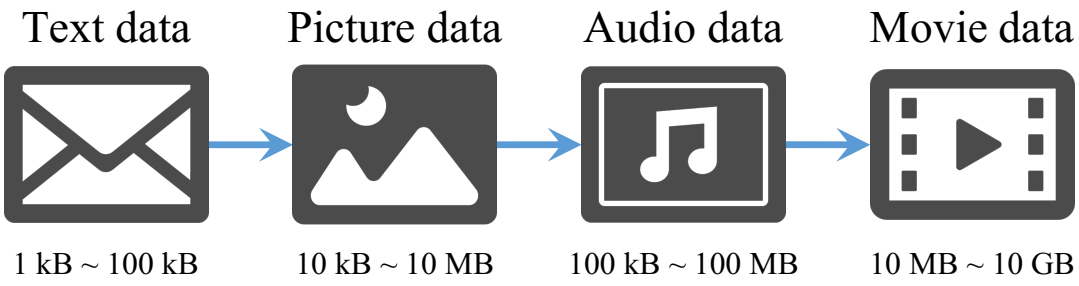


Fig. 1-1 The transition of communication contents

the Internet in Japan reached 89.8% [1], which can be roughly estimated to be over 113 million people, based on the population research result back then [2]. Considering this growth of the Internet user population and the data volume enlargement of communication contents, it is easy to foresee that the total amount of Internet traffics is rising.

In addition, the interests in the Internet-of-things (IoT) technology currently arising may further increase the total amount of Internet traffics [3, 4]. As the IoT becomes more popular, even more devices may be connected to the Internet, creating even more complicated sensor and actuator networks [5, 6]. The number of devices communicating over the Internet is expected to keep on increasing beyond the magnitude of the total human population. The recent transition from Internet Protocol version 4 (IPv4) [7] to Internet Protocol version 6 (IPv6) [8] due to the Internet Protocol (IP) address exhaustion vividly reflects this phenomenon [9].

Figure 1-2 shows the total amount of download traffic over the Internet in Japan [10]. The plots denoted as “5 ISP companies” are the estimation values calculated from the data provided by five Internet service provider (ISP) companies, which are Internet Initiative Japan Inc., OPTAGE Inc., KDDI Corporation, SoftBank Corp., and NTT Communications Corporation. The plots denoted as “9 ISP companies” are the estimation values calculated from the data provided by nine ISP companies, which are the five aforementioned companies, NTT Plala Inc., Jupiter Telecommunications Co., Ltd., BIGLOBE Inc., and NIFTY Corporation. The Ministry of Internal Affairs and Communications of Japan added the latter mentioned four ISP companies in May 2017, making the data discontinuous from November 2016 to May 2017. This is the

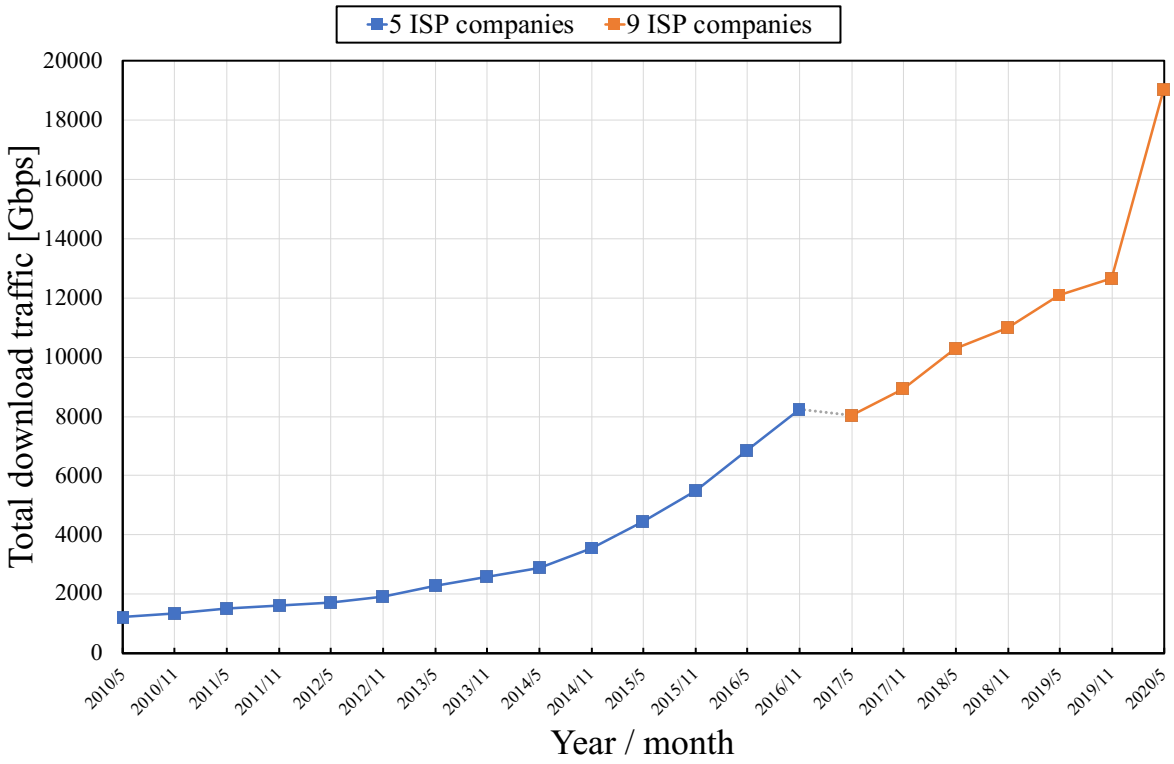


Fig. 1-2 The total amount of download traffic over the Internet in Japan.

reason why the plots switch from "5 ISP companies" to "9 ISP companies" in May 2017. The connection line from the plot of November 2016 to May 2017 is drawn with a dotted line, showing that the data were discontinuous. From Fig. 1-2, we can see that the amount of total download traffic is still increasing, and its increment rate may accelerate even more in the future.

1.1.2 Congestion and TCP

Because of these facts, recently, routers and switches are more likely to suffer from network connection failure due to excessive communication requests. This phenomenon of routers and switches being crowded with excessive communication requests is called the "congestion" of the network flows. The occurrence of network congestions would enlarge the number of packets stored in the router's buffer, and the packets newly arriving at the router would be discarded if there is not enough vacant space in the buffer. This congestion of the network flows

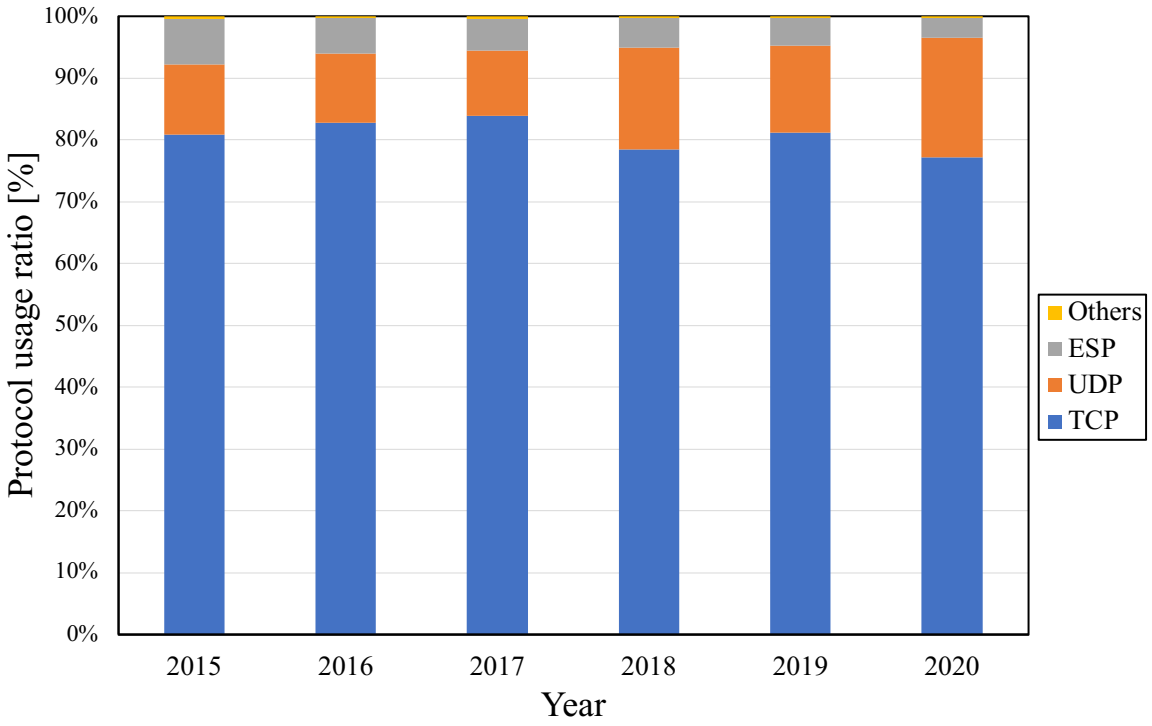


Fig. 1-3 The ratio of transport layer protocols used in Japan.

induces multiple negative impacts on the communication flows, and the communication using transmission control protocol (TCP) [11] is especially affected by the congestion.

TCP is one of the main protocols of the IP suite, commonly being treated in a set with IP and denoted as "TCP/IP". TCP is often utilized for communications through the Internet, including the communications between sensors and actuators, because of its high reliability. Figure 1-3 shows the ratio of transport layer protocols used in the last six years in Japan [12, 13]. There are multiple types of transport layer protocols other than TCP, such as user datagram protocol (UDP) [14] and encapsulating security payload (ESP) [15]. However, as Fig. 1-3 shows, while the ratio of UDP is gradually increasing due to the increasing demand for real-time communications, TCP still keeps the highest ratio.

When the network communication is done using loss-based TCP network flows, the receiver host detects congestion by acknowledging the occurrence of packet disposal and notifies the

sender host of the congestion. Upon receiving this notification, the sender host scales down the sending window size to control the congestion, resulting in reduced communication speed. Owing to this mechanism, if a mass packet disposal occurs in the router, many TCP network flows would be the target to shrink the congestion window size, which would result in multiple senders transmitting packets at a low rate, sharing the network with low throughput [16, 17]. This phenomenon is known as "global synchronization", and this greatly reduces the communication efficiency, sometimes halting the application services provided over the network [18, 19]. In addition, the likeliness of some flows being the victim of packet disposal may be vastly different compared to the others. The bursty traffics especially tend to be more vulnerable against the effect of congestion, resulting in unfairness between multiple TCP sessions sharing the same network. Due to these facts, the congestion of the traffic flows is a serious problem for communications using TCP, and many studies have been conducted on the efficient method for avoiding congestion at the routers [20, 21].

1.1.3 AQM

In order to avoid this serious congestion from occurring, a method called active queue management (AQM) has been proposed [22, 23]. AQM is a mechanism that discards the packets buffered in the bottleneck router before its buffer becomes full and serious congestion occurs. When AQM is utilized, the queue length in the buffer of the router would be always observed, and when the queue length gets larger than the predetermined threshold, the system acknowledges this as an indication of congestion and actively discards the packets in the buffer. By introducing AQM to the router, packet disposal could be done before serious congestion take place in the router, resulting in a more efficient and fair communication compared to a router using the default DropTail queue [24, 25]. Due to this fact, there have been many studies of AQM done [26, 27, 28].

As the most basic and representative AQM method, random early detection (RED) is well

known [29]. RED calculates the average queue length in the buffer and derives the packet drop probability corresponding to the average queue length. RED starts discarding packets according to the packet drop probability, which decisively differs from DropTail which is triggered by a buffer overflow in terms of fairness. In addition, the fact that RED does not wait until the buffer overflow occurs in order to start discarding packets makes global synchronization less likely to happen, resulting in better communication efficiency. Because of its simple calculation procedure, a variety of RED algorithms have been researched and analyzed [30, 31, 32, 33]. There are numbers of extent research based on RED, such as adaptive RED (ARED) [34], fair RED (FRED) [35], upper threshold RED (URED) [36], Loss ratio based RED (LRED) [37], etc. [38, 39]. In addition, there are AQM that marks explicit congestion notification (ECN) bits of the packets instead of dropping the packets [40, 41, 42], such as BLUE method [43]. However, an AQM method that discards packets does not need any alteration of existing routers, since they can notify receiver hosts of the congestion by packet drop and sender hosts can control congestion by receiving acknowledgement (ACK) message from the receiver host or timeout. Due to this fact, the AQM using packet disposal is widely utilized recently.

RED has numerous parameters and its parameterization for obtaining satisfactory performance under different circumstances is very difficult [44]. These parameters need to be selected very carefully; otherwise, RED does not perform well, resulting in decreasing throughput and increasing packet loss rate [45]. Therefore, as a scalable application of RED, the design of the AQM controller on the basis of the control theory was proposed and multiple studies have been done [46, 47, 48, 49]. In particular, it is well known that AQM based on the proportional-integral-derivative (PID) control scheme is effective [50], and its controllers were improved by various methods [51, 52, 53]. Other than that, a control theory based AQM using proportional-integral (PI) controller [54, 55] and proportional-derivative (PD) controller [56] were also proposed for TCP/AQM networks to stabilize the queue length around its target value. Compared with a primitive algorithm such as RED, AQM based on the control theory tends to have higher throughput [57]. Thus, studies on AQM based on control theory have been actively conducted

in recent years [58, 59, 60, 61, 62, 63].

1.1.4 Diversified Communication Services

Along with the development of communication technologies, the definition of communication services itself has been diversifying. In the past when the exchange of text data was the state-of-the-art technology, the throughput was the major concern of the communication services. As technology evolves, different factors became to be considered. For example, the functions of TCP exist to maintain its reliability, which is another factor of better communication service. As another example, Quality-of-Experience (QoE) was not an aspect of communication services in the past but is a well-considered factor in modern days.

Along with its diversifying definition, a variety of communication services are desired to be accomplished. The implementation of virtualized infrastructure via the network is one of the popular considerations in current days. The communication under a high-latency network such as interstellar communication is another example of communication service. A user may require better data transmission efficiency or smaller transmission latency under certain communication networks. The development of IoT technology is a comprehensible example of the diversity of communication services. The technologies to improve communication quality under such diversifying communication services are desired.

1.2 Orientation of the Research

This thesis aims to improve the quality of the communication services by utilizing AQM based on control-theoretic approaches. Figure 1-4 is a diagram that shows the classification of TCP congestion avoidance methods.

The characteristic of AQM techniques is that the communication quality could be improved by implementing software-based approaches without making any physical alteration to the system. The congestion avoidance may also be accomplished by improving the communication

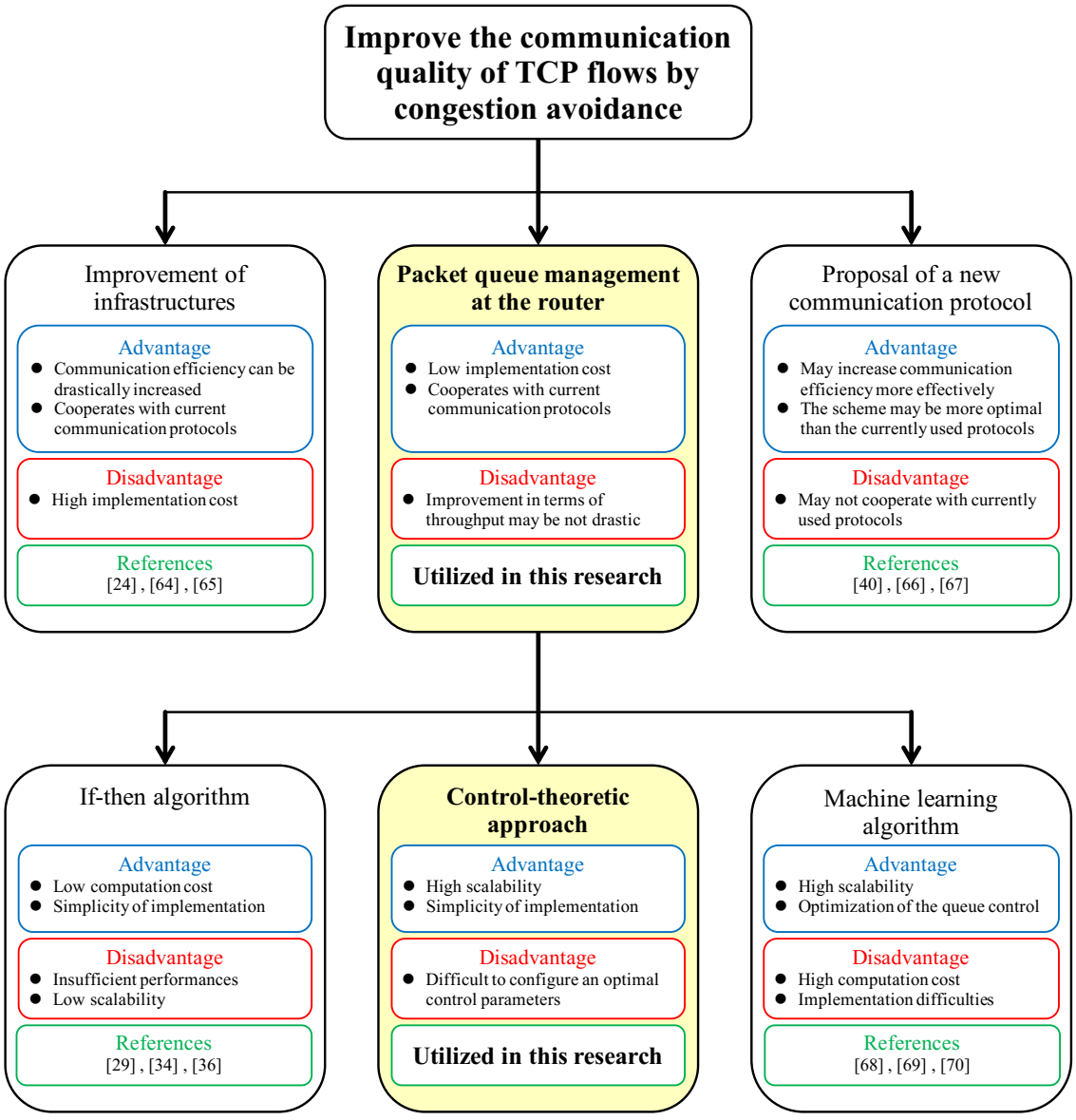


Fig. 1-4 Classifications of TCP congestion avoidance methods.

infrastructure itself [24, 64, 65] or proposing a new communication protocol [40, 66, 67], as shown in Fig. 1-4. However, the former example requires a very high implementation cost, and the latter example may not cooperate with the currently utilized communication protocols. The attempt to improve the quality of the communication services through the utilization of AQM is cost-effective and does not need an establishment of a new protocol. The downside of this approach is that this would generally not be able to drastically, in the magnitude of doubly or more, improve the communication efficiency such as throughput.

This thesis focuses on proposing novel AQM control schemes based on the control-theoretic approaches. Other than that, simple if-then algorithms [29, 34, 36] or machine learning algorithms [68, 69, 70] exist, as shown in Fig. 1-4. Compared to the simple if-then algorithms such as RED, AQM based on control-theoretic approaches are known to have higher scalability and overall better performances. Compared to machine learning algorithms, AQM based on control-theoretic approaches tend to have lower calculation costs and are relatively simpler to implement. Despite the existence of multiple studies about AQM based on control theory been done, there is ample scope of improvement remaining in this field of study.

This thesis aims to achieve communication quality control which could adapt to the diversified communication services flexibly by utilizing AQM based on control-theoretic approaches. The considered communication services in this thesis are remote control, high latency network control, and loss/delay aware control. Multiple novel AQM techniques based on control-theoretic approaches were proposed to achieve the above-mentioned controls. Chapter 3, 4, and 5 present these proposed AQM techniques. The positioning and novelty of the studies are clarified in each chapter.

In chapter 3, a remote TCP/AQM congestion control system [71] is proposed. A remote AQM control system can realize a cooperative control of multiple routers, which may become useful for a remote centralized multiple router control scheme.

Chapter 4 describes a TCP/AQM network system that can compensate for the effect of a large round-trip-time (RTT) delay of over 100 ms while being robust against modeling errors such

as fluctuation of the number of TCP sessions and coexistence of UDP flows [72]. The control system functioning properly under high-latency networks would be beneficial for dealing with large-delay communication services such as marine and stellar communications.

A TCP/AQM network control system that can dynamically generate the target queue length with consideration for the buffer size of the router is presented in chapter 5. An adjusting method of the target queue length to reduce the latency or packet loss ratio by switching delay-aware and loss-aware modes was proposed. The user can decide which to prioritize, low latency or low loss ratio, and the algorithm attempts to improve the corresponding communication quality.

This study attempts to improve the communication quality from various aspects, adapting to the diversified communication services flexibly.

1.3 Chapter Organization

The overall organization of the chapters is shown in Fig. 1-5. The topic discussed in chapter 3 is based on remote AQM control, while chapters 4 and 5 are based on local AQM controls.

The following chapter gives descriptions of TCP/AQM network followed by the presentation of control-theory based AQM. The characteristics of TCP, equations of AQM, and the analytical model of the congestion control system are shown.

Chapter 3 proposes a remote router control system using butterfly-shaped perfect delay compensator as a network delay compensator. This study is basic research aiming to realize a remote centralized multiple router control scheme. The novelty of this chapter is the implementation of butterfly-shaped perfect delay compensator to the non-linear TCP/AQM network congestion control system. Butterfly-shaped perfect delay compensator was originally proposed to be utilized in a linear control system like motion control, and the application of it to the non-linear TCP/AQM control system is the major novelty.

Chapter 4 proposes a robust congestion control system that would function even if there are large latency in the network. The proposed method is capable of dealing with a large latency

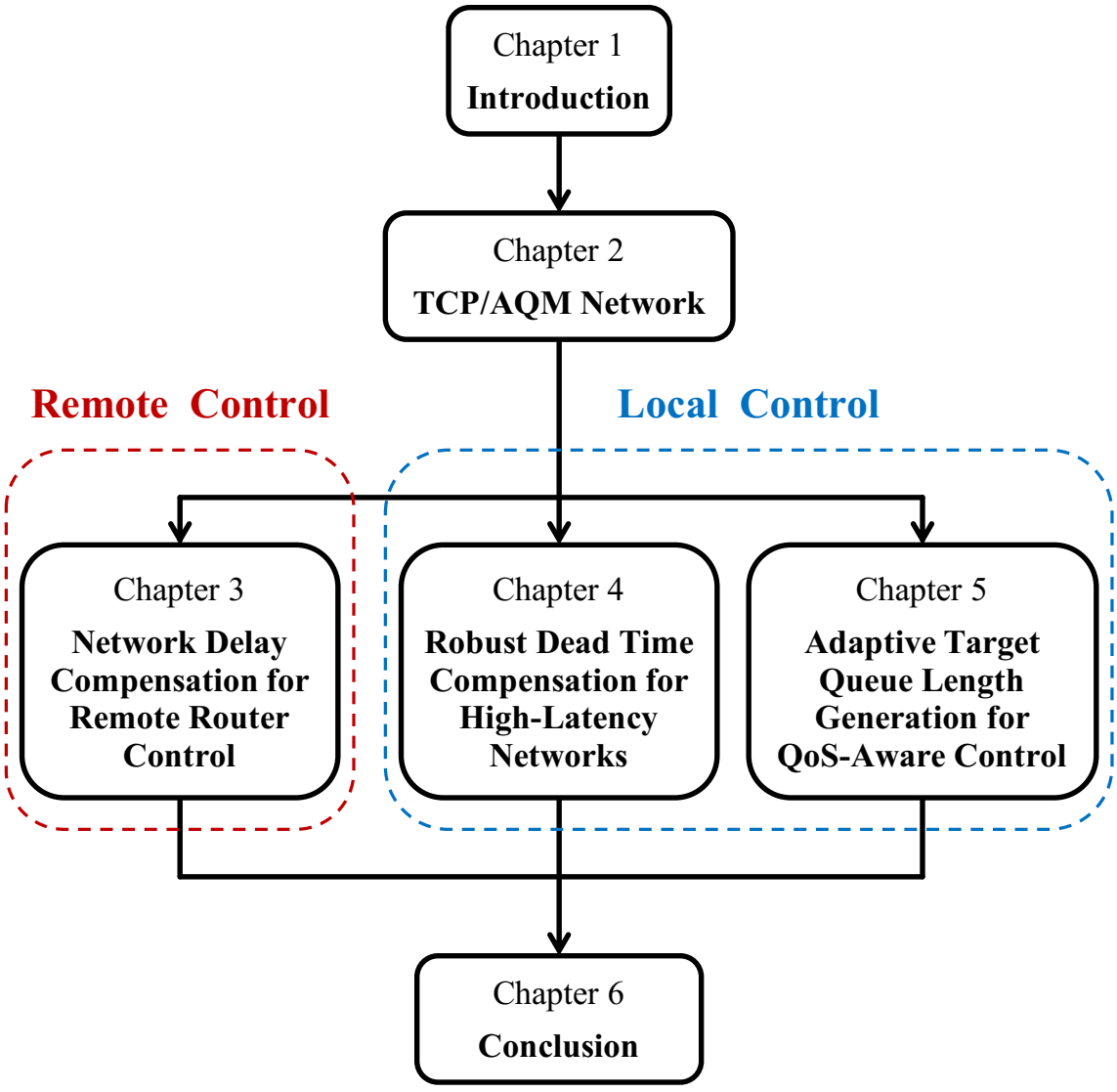


Fig. 1-5 Chapter organization.

and disturbances such as modeling errors and parameter fluctuations. The major novelty of this chapter is the integrated implementation of two compensators; disturbance observer and Smith predictor. The saturation function included in the TCP/AQM congestion control system induces a problem when implementing the aforementioned two compensators, and the proposed method overcomes the issue and achieves the integrated implementation of the compensators.

Chapter 5 proposes a novel method of controlling the target queue length of the system dynamically in order to improve the Quality-of-Service (QoS) of the system [73, 74]. An algorithm that dynamically generates target queue length without needing any information of the TCP/AQM network is proposed. The novelty of this chapter is the proposal of the algorithm that controls the target value of the system in order to increase the QoS under limited communication resources. The proposed method can increase communication efficiency or decrease communication latency with very low computation costs.

Finally, in chapter 6, the summaries and the conclusion of the researches are given.

Chapter 2

TCP/AQM Network

This chapter describes detailed information of TCP/AQM network, followed by the presentation of control-theory based AQM. The proposed methods shown in the following chapters are all based on the TCP/AQM network congestion control system based on control theory. In the first section, the concept and functions of TCP are described in detail. In the second section, a mechanism called DropTail, the basic packet queueing function utilized in the router as default, is explained. In the third section, the functions of AQM are explained, followed by the presentation of the RED algorithm. In the final section, the control-theory based AQM is presented with control block diagrams.

2.1 TCP

2.1.1 Background of TCP

When the Internet first emerged, the requirements for the Internet were free, quick, and high-speed communications. However, in the modern world where the Internet is widely utilized amongst numerous users, “reliability of communications” became the essential requirement for the Internet communications. In order to realize such a communication, TCP is utilized

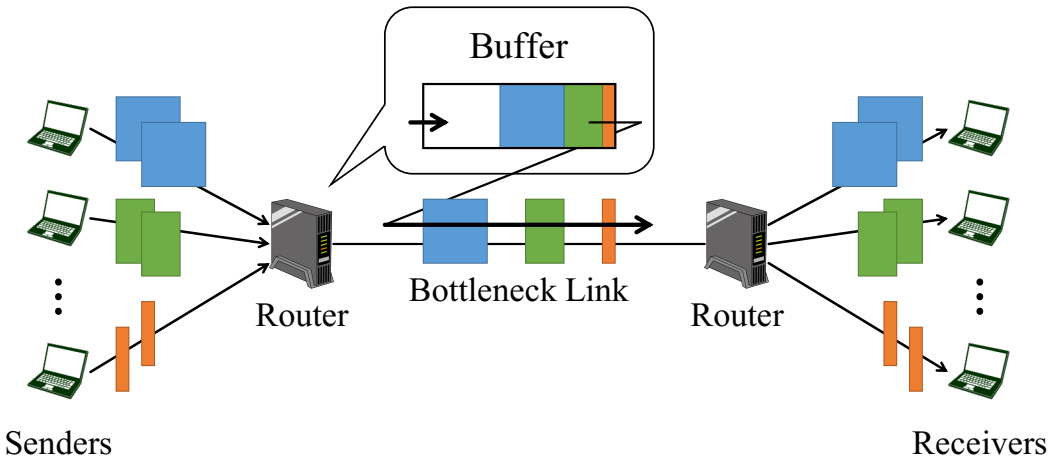


Fig. 2-1 The scheme of packet-switched network.

worldwide, keeping the highest usage ratio amongst all the transport layer protocols as shown in Fig. 1-3.

In the communication network system, there are two major communication systems utilized, which are "circuit-switched" network and "packet-switched" network. Circuit-switched network is a communication network which first establishes the connection between the sender host and the receiver host, and lets this connection be monopolized by these hosts until the connection is finished. The original telephony network is an example of this communication system. On the other hand, packet-switched network is a communication network that divides large data into small data pieces and delivers them to the receiver, just like sending a postal package to the receiver's shipping address. The connection route can be shared by multiple users, making the wastage of resources such as bandwidth smaller compared to the circuit-switched network. TCP utilizes the packet-switched network as its communication system.

Figure 2-1 shows the scheme of packet-switched network in a simple dumbbell-shaped network topology. As Fig. 2-1 shows, the packets are delivered to the receiver hosts by passing through multiple routers. The packets sent from senders are once buffered in the router creating a queue, and the router reads out the header data of the packets in order to extract the packets to the corresponding output line. By utilizing this scheme, multiple users can share the same

communication line simultaneously, thus resulting in higher bandwidth usage. If the output link is not congested and has available bandwidth for the buffered packets to be sent, the packet which arrived at the router the first would be sent, and the same procedures are repeated for packets arriving later on. This is the standard first-in-first-out (FIFO) queue at the router.

In this thesis, all the simulations were done using IPv4 packets.

2.1.2 Functions of TCP

In order to ensure the reachability of the transmitted data, TCP holds multiple functions to deal with problems such as packet loss, packet duplication, packet disorder, etc. The followings are the functions that TCP has.

2.1.2.1 Connection Establishment

TCP offers connection-oriented communication while using a packet-switched network. In connection-oriented communication, the connection is prepared before the actual data communication. In TCP, a connection establishment request packet which is consisted of a TCP header only is sent before transmitting data and waits for a reply. If the ACK comes back, then data communication is possible. Otherwise, data communication will never start. In addition, a connection break procedure is done when the data communication is finished.

2.1.2.2 Acknowledgement

The data unit utilized in TCP is called “segment”. When the receiver host successfully receives the sent segment, it transmits a confirmation reply to inform of the reception. This confirmation reply is called “ACK (Acknowledgement)”. ACK is used as an accumulation confirmation reply, which indicates how much of the continuous data were received. If no ACKs are returned to the sender host for a preset length of time, the sender host determines that the segment has been

lost and retransmits the same segment. By this function, TCP can guarantee the transmission of data.

2.1.2.3 Retransmission Control

The sender host retransmits the segment if it decides that the segment is lost. That decision is done by detecting a timeout by waiting for a certain duration. In addition, receiving ACKs of the same segment three times in a row also denotes the segment loss, which is called ACK duplication. If the window size is large, retransmission triggered by ACK duplication is much faster than that of the timeout, so it is also called as fast retransmit.

2.1.2.4 Retransmission Timeout

The duration that the sender host waits for ACK without retransmitting is called retransmission timeout (RTO). If this time passes and still no ACK messages arrive, the segment is determined to be lost and retransmission starts. To realize highly efficient communication, TCP records the round-trip-time (RTT) and its jitter and derive the RTO from them.

2.1.2.5 Window Control

The maximum amount of data packets that a sender host can send at once before receiving any ACK messages from the receiver host is called window size. This window size is determined by two values, receive window size and congestion window size, where its value is set to be equal to the smaller one. There would be a limit to how large the window size can be for each receiver, called maximum window size, which is preset for each receiver.

2.1.2.6 Flow Control

In TCP, the receiver host informs the sender host of the data size that it can receive. This data size is called receive window size, and this is defined by the receiver host. When the buffer of

the receiver host becomes nearly overflowing, the receive window size is reduced and the data transmission rate of the sender host is lowered. In other words, the required data transmission rate is determined depending on the instruction of the receiver host. This procedure of tuning the receiver window size is called the flow control.

2.1.2.7 Congestion Control

Normally, a network is shared with other hosts. This means there is a possibility that the network is already congesting because of the traffics between other hosts. If a massive amount of data were additively sent in that kind of situation, the network might get severely congested. In TCP, there is an algorithm called congestion control which controls the congestion window size and limits the amount of data packets being sent at once. There are multiple congestion control methods, and they could be divided into three types; loss-based method, delay-based method, and hybrid method.

The loss-based methods such as Reno [75] and NewReno [76] detect congestion by observing the packet loss and limit the transmission rate. The delay-based methods such as Vegas [66] detects congestion by observing the RTT of the communication. The hybrid methods such as DCTCP [77] utilizes both loss-based and delay-based approaches to control the congestion window size. In this thesis, NewReno is utilized as the TCP version of congestion window size control.

Figure 2-2 shows the conceptual diagram of the congestion window control routine of loss-based methods. As long as the communication is established without any packet drops, the congestion window size continues to increase. If a packet drop occurs, the sender host determines that there is congestion occurring, and it shrinks its window size.

This basic mechanism of enlarging and shrinking the congestion window size is the same amongst all loss-based methods. However, the actual calculation procedures of deriving the congestion window size differ by methods. Figure 2-3 shows the graph of congestion window

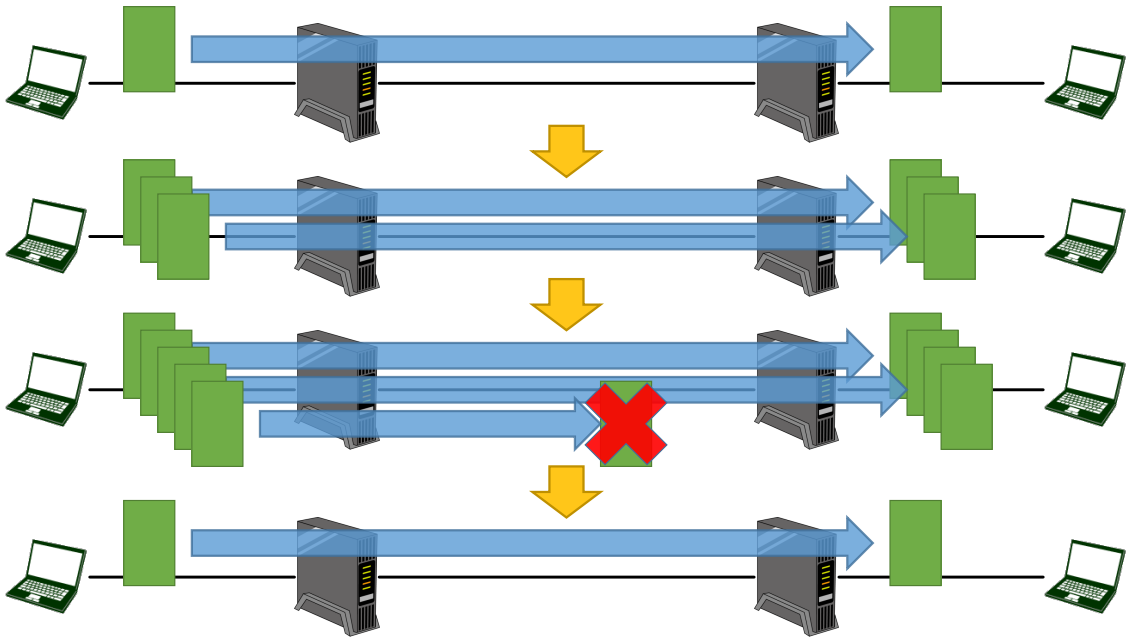


Fig. 2-2 The conceptual diagram of loss-based congestion window control.

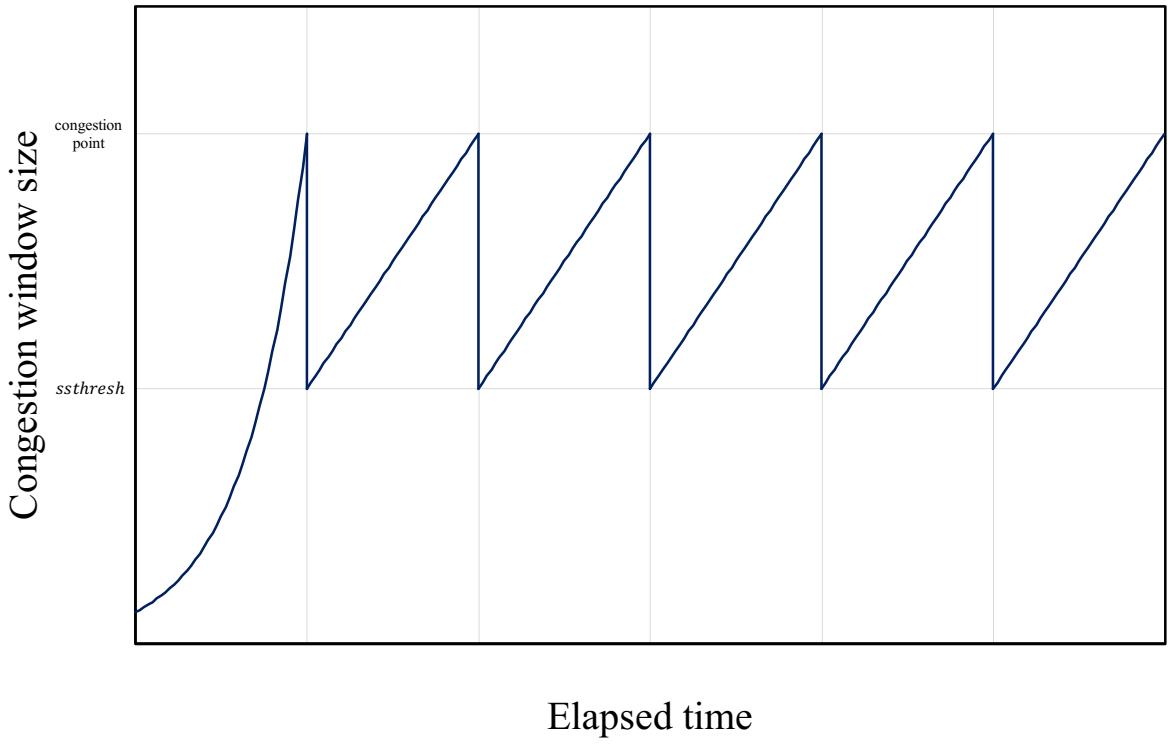


Fig. 2-3 The congestion window control of NewReno.

size fluctuation controlled by TCP NewReno. In Fig. 2-3, the timing of congestion window size drop indicates a packet loss due to the congestion. The procedure of TCP NewReno can be divided into two phases; slow start and congestion avoidance. The specific procedures of the slow start phase are as follows:

1. The congestion window size $cwnd$ is set to 1 segment and a packet is sent.
2. The congestion window size $cwnd$ is incremented by 1 segment every time the ACK returns. This procedure increases the congestion window size exponentially.
3. Continue enlarging $cwnd$ until a packet loss or timeout is detected.

After a packet loss or timeout detection, NewReno enters the congestion avoidance procedures. If a packet loss was detected, the procedures would be as follows:

1. Shrink $cwnd$ to $ssthresh$, which is equal to the half of the current $cwnd$.
2. Increase $cwnd$ by $\frac{1}{cwnd}$ every time the ACK returns. This is equivalent to increase $cwnd$ by 1 per RTT, which would be increasing $cwnd$ linearly to the elapsed time.
3. Continue enlarging $cwnd$ until a packet loss or timeout is detected.

If a timeout was detected, the procedures would be as follows:

1. Shrink $cwnd$ to 1.
2. Increase $cwnd$ in the same manner with slow start until $cwnd$ reaches $ssthresh$, which is equal to the half of the $cwnd$ when the timeout was detected.
3. After $cwnd$ reached $ssthresh$, increase $cwnd$ by $\frac{1}{cwnd}$ every time the ACK returns.
4. Continue enlarging $cwnd$ until a packet loss or timeout is detected.

By this scheme, the TCP NewReno attempts to avoid congestions.

2.2 DropTail Queue

As Fig. 2-1 shows, the packet-switched network that supporting TCP flows would let all senders send their packets to their corresponding receiver hosts, even if they share the same

route. However, if multiple TCP sessions share the same link, which is denoted as bottleneck link in Fig. 2-1. As described in 2.1.1, the router treats the packets buffered to be sent out in a FIFO manner. Thus, all packets attempting to be sent through the bottleneck link will first be buffered at the router connecting the sender hosts and the bottleneck link. The left side router in Fig. 2-1 corresponds to this, and this router is called as "bottleneck link router".

In communication using TCP, the sender hosts enlarge their window size regardless of the actual condition of the network and shrink its size only when the packet loss or timeout is detected. If there are no packet losses or timeouts, all the TCP sessions will increase their window size, and sooner or later send packets faster than the bottleneck link capacity. In such a congested situation, the queue in the bottleneck link router would keep growing, meaning that more packets would flow into the buffer than flowing out to the bottleneck link. Eventually, the queue will be too long for more packets to come in, and they would be dropped before joining the buffer queue. This phenomenon of dropping the packets arriving after the buffer is full is called "buffer overflow", and this whole mechanism of letting the queue grow until a buffer overflow occurs is called "DropTail".

Figure 2-4 shows the scheme of DropTail queue. The number of packets in the buffer queue of the bottleneck link router is denoted as "queue length". DropTail is a default algorithm utilized in the router buffer mechanism since there are no additional procedures required for implementing it. The DropTail queue would drop packets only if the buffer overflow occurs. This means TCP senders can detect the congestion only by buffer overflow, and the congestion window size shrinkage happens only after the buffer overflow has occurred.

These characteristics of detecting the congestion only by buffer overflow are known to have multiple issues. The followings are examples of the issues.

1. The dropped packets are the packets that coincidentally arrived at the router when the buffer overflowed. This means that the flow whose packet was coincidentally dropped get their flow limited.
2. The bursty traffic are likely to trigger a buffer overflow even when the transmission data

rate is low. Due to this, the bursty traffics are more likely to have their transmission rate limited, thus resulting in an unfairness of network. The bursty traffic sends a massive number of packets at once, which can induce a mass packet disposal when the buffer overflows.

3. If the congestion keeps growing, all sender hosts sharing the same router in the network would be the victim of a speed limitation due to the congestion window size control, resulting in a global synchronization (a phenomenon where the network is shared with a low throughput). If the global synchronization occurs, the network load drastically decreases, and the communication efficiency will experience a major degradation.
4. The queue length would rapidly fluctuate, making the system unstable in the sense of inconsistent queueing delay.

The decrease in communication efficiency, unfairness amongst multiple TCP senders, and instability of the system are the major issues that DropTail has.

2.3 AQM

2.3.1 Functions of AQM

In order to deal with the problems that DropTail had, AQM has been proposed. AQM actively drops the packet in the queue to maintain the queue length stable. The packets to be dropped are selected randomly. This will be performed before the queue length reaches the buffering capacity. By these procedures, AQM attempts to maintain the queue length to be stable, maintaining fairness amongst multiple TCP sessions, avoiding global synchronization, and raising the link utilization efficiency. Attempting to raise the efficiency of the communication through the procedure of proactive disposal of packets is counter-intuitive. However, this procedure is effective because the proactive packet disposal triggers the congestion window size shrinkage of TCP sessions of the randomly selected packets, making the window size shrinkage timing

diverged. On the other hand, DropTail tends to target multiple TCP sessions to shrink their congestion window size at once, making the window size shrinkage timing more concentrated, resulting in global synchronization.

AQM is a technique that can be implemented to the router additionally, which means AQM coexists with DropTail instead of overriding it. This is since the DropTail is the basic function of the FIFO queue system in the router, not the additional procedure. However, if the AQM functions ideally enough to avoid buffer overflow, the DropTail function would not be triggered at all.

Figure 2-5 shows the conceptual diagram of AQM. As shown in Fig. 2-5, AQM observes the queue length in the router and drops packet actively to avoid serious congestion. The packet drop probability is calculated based on the value of queue length observed. This actual calculation method differs depending on the AQM method utilized.

2.3.2 RED

In this section, the most basic and representative method of AQM, i.e., RED, is introduced. RED calculates the average queue length and attempts to keep this average queue length between the two preset values. The calculation of average queue length is generally performed by using exponential moving average (EMA). Additionally, the dropped packets are selected randomly, which differs from DropTail that always dropped the last arriving packets. By this mechanism, RED avoids the busty traffic being treated unfairly.

The specific algorithm of RED is as follows.

1. Define the minimum threshold $th_{\min}$ and the maximum threshold $th_{\max}$.
2. Calculate the average queue length q_{ave} by using EMA.

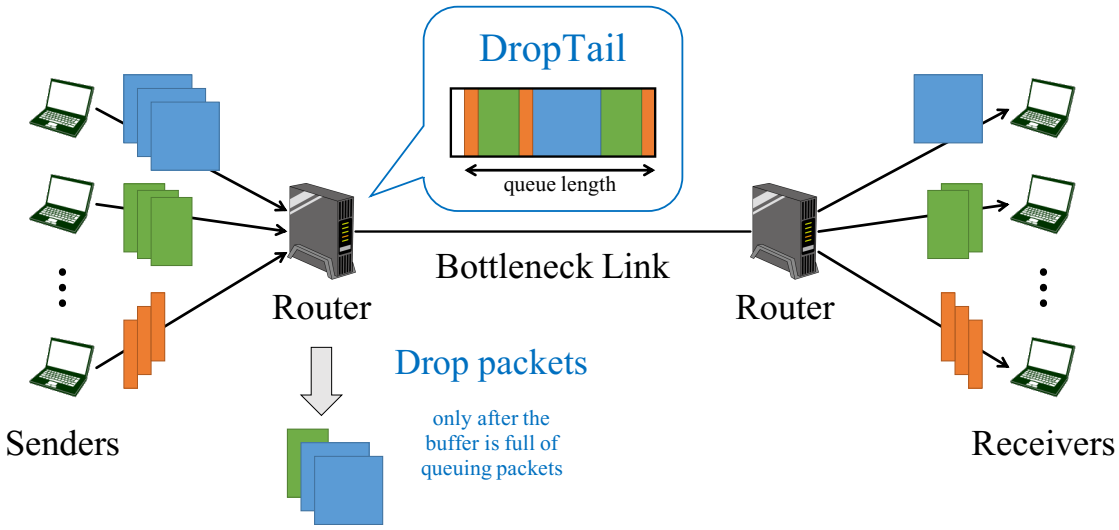


Fig. 2-4 The scheme of DropTail queue.

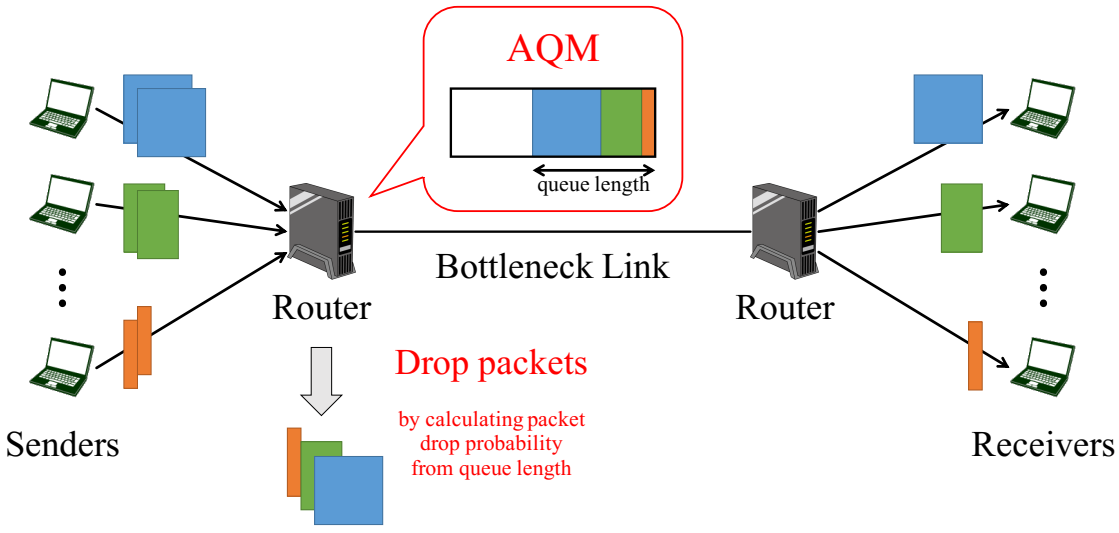


Fig. 2-5 The conceptual diagram of AQM.

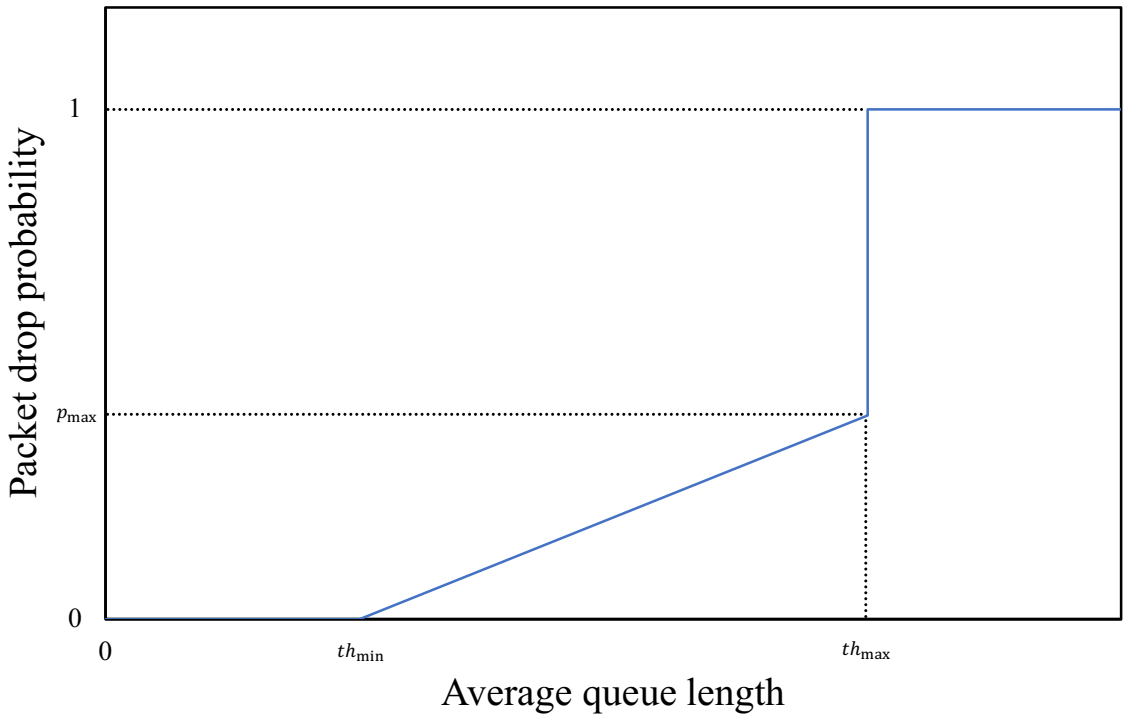


Fig. 2-6 The relationship between the packet drop probability and the average queue length in RED

3.

$$p = \begin{cases} 0 & (q_{\text{ave}} \leq th_{\min}) \\ \frac{p_{\max}}{th_{\max} - th_{\min}} * (q_{\text{ave}} - th_{\min}) & (th_{\min} < q_{\text{ave}} < th_{\max}) \\ 1 & (th_{\max} \leq q_{\text{ave}}) \end{cases}$$

4. Drop the packets in the buffer according to the value of p.
5. Repeat from procedure 2.

Figure 2-6 shows the relationship between the packet drop probability p and q_{ave} in RED. The value of p changes proportionally while the q_{ave} is in the range from $th_{\min}$ to $th_{\max}$. If the value of q_{ave} is lower than $th_{\min}$, then there will be no packet dropped. If it is greater than $th_{\max}$, then all the packets are dropped.

2.4 Control-Theory Based AQM

This section presents the control-theory based TCP/AQM network models. In this thesis, a nominal TCP/AQM network model presented in [78] is utilized. This nominal model is designed based on the linear TCP/AQM network model proposed by Hollot *et al.* [47]. The linear TCP/AQM network model was designed by linearizing a nonlinear TCP/AQM network model proposed by Misra *et al.* [46]. All of these TCP/AQM network models are presented in this section. Finally, a TCP/AQM network congestion control system using a PID controller is presented.

2.4.1 Nonlinear TCP/AQM Network Model

Misra *et al.* [46] proposed a nonlinear TCP/AQM network model by formulating the TCP window size and queue length dynamics. The nonlinear TCP/AQM network model is shown in (2.1) and (2.2),

$$\dot{W}(t) = \frac{1}{R(t)} - \frac{W(t)W(t-R(t))}{2R(t-R(t))}p(t-R(t)), \tag{2.1}$$

$$\dot{q}(t) = -C(t) + \frac{N(t)}{R(t)}W(t). \tag{2.2}$$

The variables in (2.1) and (2.2) are defined as follows:

$$
\begin{aligned}
W(t) &\doteq \text{TCP window size [packets]},\\
q(t) &\doteq \text{queue length [packets]},\\
R(t) &\doteq \text{RTT [s]} \left(\fallingdotseq \frac{q(t)}{C(t)} + T_{\mathrm{p}}\right),\\
C(t) &\doteq \text{bottleneck link capacity [packet/s]},\\
T_{\mathrm{p}} &\doteq \text{propagation delay [s]},\\
N(t) &\doteq \text{number of TCP sessions},\\
p(t) &\doteq \text{packet drop probability, where } p(t) \in [0, 1].
\end{aligned}
$$

2.4.2 Linear TCP/AQM Network Model

Hollot *et al.* [47] proposed a linear TCP/AQM network model by linearizing the nonlinear model shown in (2.1) and (2.2). In order to linearize (2.1) and (2.2), the number of TCP sessions $N(t)$ and bottleneck link capacity $C(t)$ were both assumed to be constant, i.e., $N(t) \equiv N$ and $C(t) \equiv C$. In addition, the operating point where $\dot{W} = 0$ and $\dot{q} = 0$ was defined as (W_0, p_0, q_0, R_0). From these assumptions, the following equations can be derived:

$$
\begin{aligned}
W_0^2 p_0 &= 2, && (2.3)\\
W_0 &= \frac{CR_0}{N}, && (2.4)\\
R_0 &= \frac{q_0}{C} + T_{\mathrm{p}}. && (2.5)
\end{aligned}
$$

To proceed with linearization, the dependence of the time delay argument $t - R(t)$ on queue length $q(t)$ is ignored and assumed to be fixed to $t - R_0$. On the other hand, the dependence of RTT $R(t)$ on queue length $q(t)$ in the dynamic parameters is retained. As a result, the simplified dynamics are obtained as follows:

$$\dot{W}(t) = \frac{1}{\frac{q(t)}{C} + T_\mathrm{p}} - \frac{W(t)}{2}\frac{W(t-R_0)}{\frac{q(t-R_0)}{C} + T_\mathrm{p}} p(t-R_0), \tag{2.6}$$

$$\dot{q}(t) = -C + \frac{N}{R(t)} W(t). \tag{2.7}$$

Next, the right-hand sides of (2.6) and (2.7) are defined as (2.8) and (2.9)

$$f\left(W(t), W_R(t), q(t), q_R(t), p_R(t)\right) = \frac{1}{\frac{q(t)}{C} + T_\mathrm{p}} - \frac{W(t)W_R(t)}{2\left(\frac{q_R(t)}{C} + T_\mathrm{p}\right)} p_R(t), \tag{2.8}$$

$$g\left(W(t), q(t)\right) = -C + \frac{N}{\frac{q(t)}{C} + T_\mathrm{p}} W(t), \tag{2.9}$$

where $W_R(t) \doteq W(t-R_0)$, $q_R(t) \doteq q(t-R_0)$, and $p_R(t) \doteq p(t-R_0)$.

The partial derivatives of f and g at this operating point (W_0, p_0, q_0) can be derived as follows by recalling the operating point relationships shown in (2.4) and (2.5). For simplification, (t) are treated as a constant and omitted for deriving partial differential equations in (2.10)–(2.16).

$$\begin{aligned} \frac{\partial f}{\partial W} &= -\frac{W_0}{2R_0} p_0 \\ &= \frac{W_0}{2R_0}\frac{2}{W_0^2} \\ &= -\frac{1}{R_0 W_0} \\ &= -\frac{N}{R_0^2 C} \end{aligned} \tag{2.10}$$

$$\frac{\partial f}{\partial W_R} = \frac{\partial f}{\partial W} \tag{2.11}$$

$$\begin{aligned}\frac{\partial f}{\partial q} &= \frac{\partial}{\partial q}\left(\frac{1}{\frac{q}{C}+T_{\mathrm{p}}} - \frac{WW_R}{2(\frac{q_R}{C}+T_{\mathrm{p}})}p_R\right) \\ &= -\frac{1}{R_0^2 C}\end{aligned} \tag{2.12}$$

$$\begin{aligned}\frac{\partial f}{\partial q_R} &= \frac{\partial}{\partial q}\left(\frac{1}{\frac{q}{C}+T_{\mathrm{p}}} - \frac{WW_R}{2(\frac{q_R}{C}+T_{\mathrm{p}})}p_R\right) \\ &= \frac{W_0^2 p_0}{2R_0^2 C} \\ &= \frac{1}{R_0^2 C}\end{aligned} \tag{2.13}$$

$$\begin{aligned}\frac{\partial f}{\partial p_R} &= -\frac{W_0^2}{2R_0} \\ &= -\frac{\frac{R_0^2 C^2}{N^2}}{2R_0} \\ &= -\frac{R_0 C^2}{2N^2}\end{aligned} \tag{2.14}$$

$$\begin{aligned}\frac{\partial g}{\partial q} &= \frac{\partial}{\partial q}\frac{NW}{\left(\frac{q}{C}+T_{\mathrm{p}}\right)} \\ &= -\frac{NW_0}{C\left(\frac{q_0}{C}+T_{\mathrm{p}}\right)^2} \\ &= -\frac{1}{R_0}\end{aligned} \tag{2.15}$$

$$\frac{\partial g}{\partial W} = \frac{N}{R_0} \tag{2.16}$$

Thus, by linearizing (2.6) and (2.7), the linear model could be obtained as follows:

$$\begin{aligned}\delta\dot{W}(t) = &-\frac{N}{R_0^2 C}\left(\delta W(t) + \delta W\left(t - R_0\right)\right) \\ &-\frac{1}{R_0^2 C}\left(\delta q(t) - \delta q\left(t - R_0\right)\right) \\ &-\frac{R_0 C^2}{2N^2}\delta p\left(t - R_0\right),\end{aligned} \tag{2.17}$$

$$\delta\dot{q}(t) = \frac{N}{R_0}\delta W(t) - \frac{1}{R_0}\delta q(t), \tag{2.18}$$

where $\delta W = W - W_0$, $\delta q = q - q_0$, and $\delta p = p - p_0$. A block diagram of the linearized dynamics is shown in Fig. 2-7.

Hollot *et al.* continued to simplify these dynamics by dividing the dynamics into a nominal model and modeling error. A simplified block diagram is shown in Fig. 2-8. The modeling error $\Delta(s)$ is defined as (2.19)

$$\Delta(s) \doteq \frac{2N^2 s}{R_0^2 C^3}\left(1 - e^{-sR_0}\right). \tag{2.19}$$

Finally, from the fact that the modeling error $\Delta(s)$ has extremely small gain, by excluding $\Delta(s)$ from Fig. 2-8, the TCP/AQM network model for the controller design can be derived as shown in Fig. 2-9. The transfer function $C(s)$ denotes the AQM controller for the TCP/AQM network. The AQM controller uses the queue length information in order to calculate the packet drop probability. The transfer function $P(s)$ is a combination of the nominal window dynamics, queue dynamics, the block element between these two $\left(\frac{N}{R_0}\right)$, and the minus sign before the window dynamics, as shown in Fig. 2-8. The transfer function $P(s)$ can be written as (2.20)

$$P(s) = -\frac{\frac{C^2}{2N}}{\left(s + \frac{2N}{R^2 C}\right)\left(s + \frac{1}{R}\right)}. \tag{2.20}$$

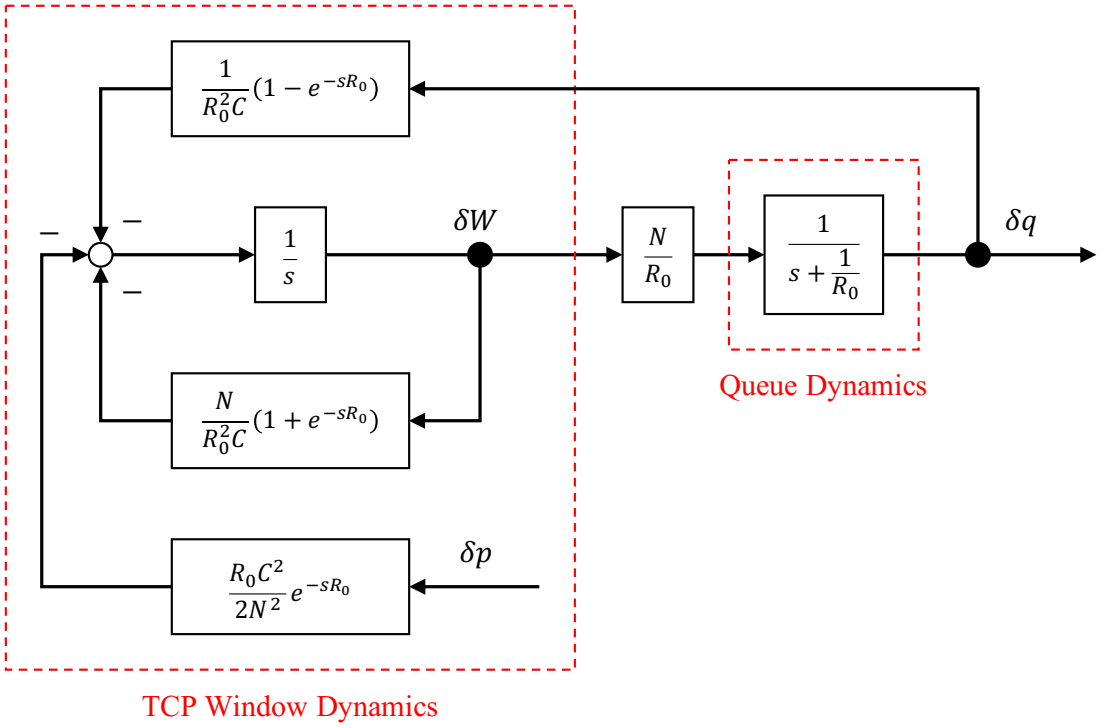


Fig. 2-7 Linearized TCP/AQM network model.

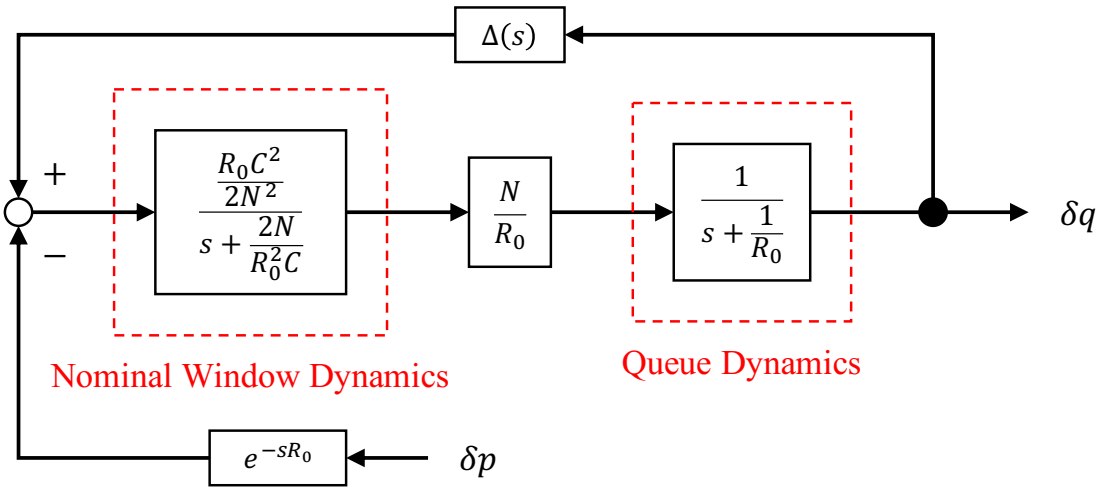


Fig. 2-8 Simplified linear TCP/AQM network model.

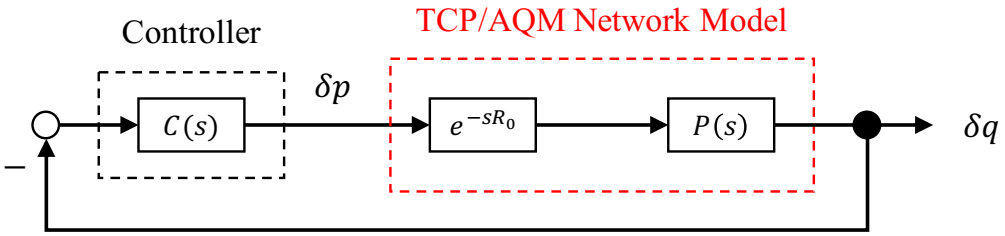


Fig. 2-9 The linear TCP/AQM control system.

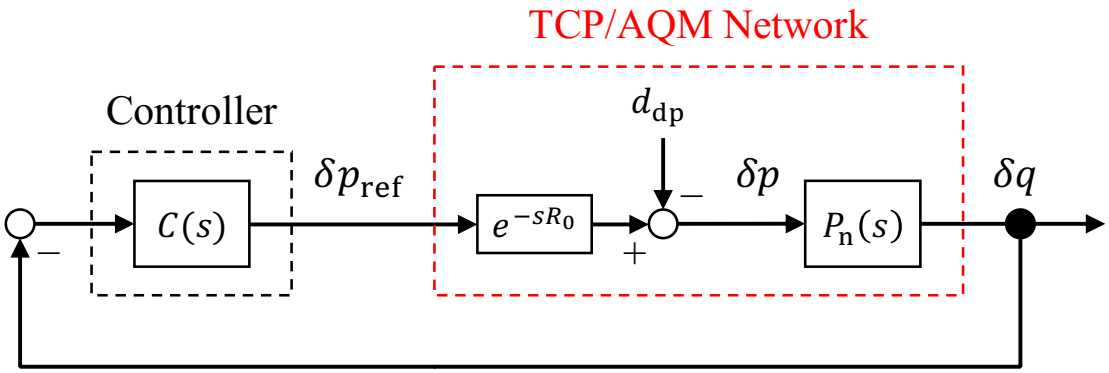


Fig. 2-10 Control system with the nominal TCP/AQM network model.

2.4.3 Nominal TCP/AQM Network Model

Based on the linear model shown in Fig. 2-9, a nominal TCP/AQM network model was proposed in [78]. This nominal TCP/AQM network model was utilized to simplify the design of the AQM controller.

Figure 2-10 shows the entire control system with a nominal TCP/AQM network model, where $P_{\mathrm{n}}(s)$, δp_{ref}, and d_{dp} denote the nominal TCP/AQM network model, reference packet drop probability, and disturbance in the packet drop probability dimension, respectively. The nominal TCP/AQM network plant model $P_{\mathrm{n}}(s)$ is defined as (2.21)

$$P_{\mathrm{n}}(s) = -\frac{C_{\mathrm{n}}^2}{2N_{\mathrm{n}}}\frac{1}{s^2}, \tag{2.21}$$

where N_{n} and C_{n} denote the nominal number of TCP sessions and the nominal bottleneck link capacity, respectively. The disturbance d_{dp} includes the modeling errors due to linearization,

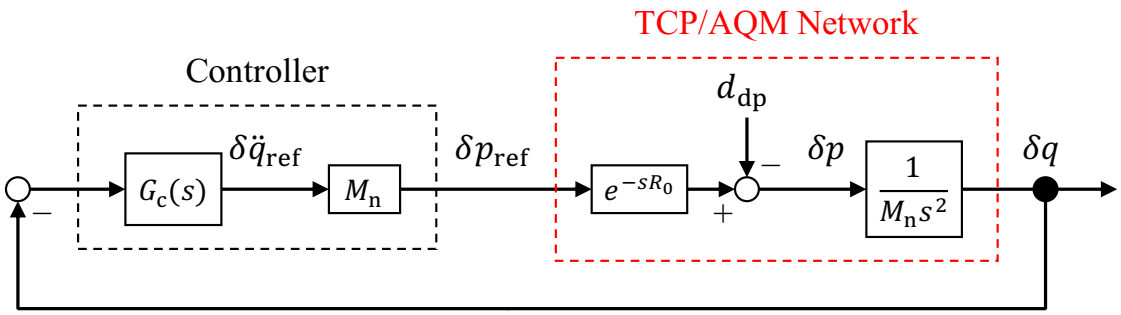


Fig. 2-11 Control system with the nominal inertia model.

nominalization, and coexistence of non-TCP flows such as UDP flows. The notation for input to the TCP/AQM network is changed from δp to δp_{ref} along with the nominalization.

In addition, the inertia model in the TCP/AQM network dynamics M_{n} is defined as (2.22)

$$M_{\mathrm{n}} = -\frac{2N_{\mathrm{n}}}{C_{\mathrm{n}}^2}. \tag{2.22}$$

From (2.21) and (2.22), the following relationship between $P_{\mathrm{n}}(s)$ and M_{n} can be derived:

$$P_{\mathrm{n}}(s) = \frac{1}{M_{\mathrm{n}}s^2}. \tag{2.23}$$

Using this inertia model M_{n}, the block diagram shown in Fig. 2-10 can be redesigned as shown in Fig. 2-11, where G_{c} denotes a feedback controller such as PID or PD, and $\delta\ddot{q}_{\mathrm{ref}}$ denotes the reference queue acceleration. The reference packet drop probability δp_{ref} is calculated as shown in (2.24).

$$\delta p_{\mathrm{ref}} = M_{\mathrm{n}}\delta\ddot{q}_{\mathrm{ref}} \tag{2.24}$$

The feedback controller $G_{\mathrm{c}}(s)$ calculates $\delta\ddot{q}_{\mathrm{ref}}$ from δq; the design of the controller is described in the following section.

2.4.4 AQM Using PID Controller

As shown in Fig. 2-11, the feedback controller $G_c(s)$ calculates the reference queue acceleration $\delta\ddot{q}_{ref}$ from δq. When the PID controller is utilized, $\delta\ddot{q}_{ref}$ is calculated as (2.25)

$$\begin{aligned}\delta\ddot{q}_{ref} &= G_c(s)\delta q \\ &= \left(K_p + K_i\frac{1}{s} + K_d s\right)(q_0 - q)\,,\end{aligned} \tag{2.25}$$

where K_p, K_d, and K_i denote the proportional gain, derivative gain, and integral gain, respectively. Thus, using (2.24) and (2.25), the reference packet drop probability δp_{ref} calculated by the PID controller can be derived as (2.26).

$$\begin{aligned}\delta p_{ref} &= M_n G_c(s)\delta q \\ &= M_n\left(K_p + K_i\frac{1}{s} + K_d s\right)(q_0 - q)\,.\end{aligned} \tag{2.26}$$

Chapter 3

Network Delay Compensation for Remote Router Control

3.1 Background

Routers that utilize an AQM controller are generally connected to one another, and congestion control in one router may also affect congestion control in the other. More efficient congestion control could be expected if information flow between routers is collected in one place and controlled cooperatively. Chibana *et al.* [79] proposed a remote congestion controller to enable cooperative AQM of multiple routers and flexible AQM taking traffic conditions in the entire network into account. However, each router must be remotely controlled via the network simultaneously in order to realize such a system.

Such a control system, i.e., a system that forms a control loop via the network, is called a networked control system (NCS) [80]. Network-induced delay is one of the major factors greatly affecting the performance and stability of the NCS [81]. If a feedback loop is formed via the Internet, its network-induced delay varies randomly depending on the number of hardware units and end-users connected to the Internet. This random network delay is unpredictable, and

remote control systems easily become unstable due to random network delay [82].

Many studies have attempted to address the effect of network-induced delay in the NCSs [83, 84]. To compensate for the network-induced delay in the TCP/AQM network, the Smith predictor (SP) [85] and adaptive SP (ASP) [86] have been proposed. The existing works aim at compensating for the round-trip delay between a server and a client. Li *et al.* [87] proposed an AQM scheme using a PI controller with the SP. The ASP can compensate for the effect of fluctuating delay as long as it can be measured, while the SP compensates only constant delay [88]. Ohsaki *et al.* [89] has proposed an AQM scheme using the RED with the ASP. However, the existing works have not discussed the time-varying network delay between the remote congestion controller and router, as shown in [79]. In addition, the SP and ASP need the time-delay model or time-delay measurement, which leads to system instability or implementation complexity.

This chapter proposes a remote TCP/AQM congestion control system using a model free time-delay compensator. This research was focused on the AQM based on control theory and used a PID controller for the AQM congestion controller. A butterfly-shaped perfect delay compensator (PDC) [90] is adopted as a time-delay compensator. The butterfly-shaped PDC was originally proposed for time-delay compensation in networked motion control systems and can sweep out time-delay elements from a feedback loop without any time-delay model. In order to apply the butterfly-shaped PDC to the TCP/AQM network, a controller model on a plant side is defined and the model mismatch between the controller model and an original controller on a remote controller side is considered. The effectiveness of the proposed controller is validated from simulations using time-varying network delays.

3.1.1 NCS

The NCS is a control system that has a feedback loop going through the network. The implementation cost of the control system would be greatly reduced if a commercial network is integrated in order to construct the NCS. The proposed remote AQM control system is an

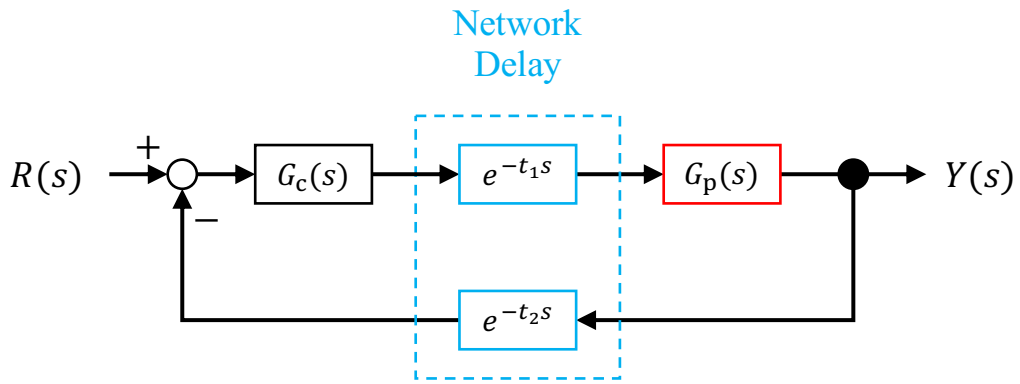


Fig. 3-1 Block diagram of a general NCS.

NCS. Network-induced delay is unavoidable since the NCS sends the control signal through the network. In addition, when considering the usage of a commercial network, network delay would not be constant and may fluctuate randomly. This unpredictable network delay is known to greatly affect the performance of the NCS.

Figure 3-1 shows the block diagram of the general NCS, which is only constructed from the controller $G_c(s)$, the plant $G_p(s)$, forward network delay t_1, and feedback network delay t_2. $R(s)$ and $Y(s)$ denote the input and output signals, respectively.

If the network has no delay, i.e., $t_1 = t_2 = 0$, the transfer function for the entire block diagram is denoted as (3.1)

$$G_{\text{woNET}}(s) = \frac{Y(s)}{R(s)} = \frac{G_c(s)G_p(s)}{1 + G_c(s)G_p(s)}. \tag{3.1}$$

The transfer function G_{woNET} is an ideal transfer function for an NCS.

The NCS transfer function that includes network delay, i.e., $t_1 \neq 0$ and $t_2 \neq 0$, is defined as (3.2)

$$G_{\text{wNET}}(s) = \frac{Y(s)}{R(s)} = \frac{G_c(s)G_p(s)e^{-t_1 s}}{1 + G_c(s)G_p(s)e^{-(t_1+t_2)s}}. \tag{3.2}$$

The transfer function of G_{wNET} shown in (3.2) is clearly more complicated compared to that of

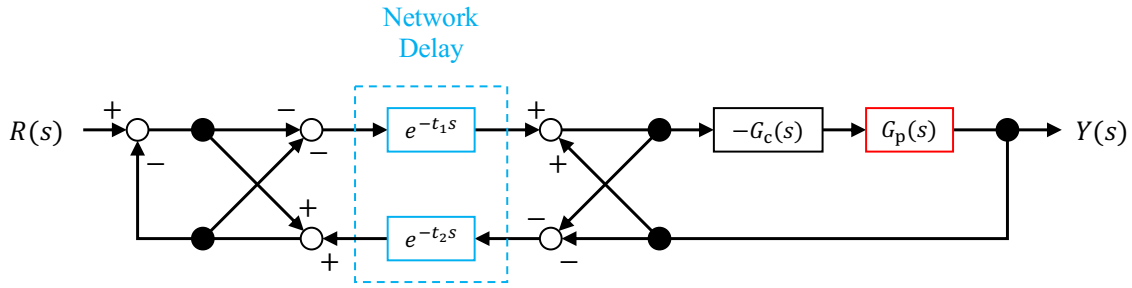


Fig. 3-2 Butterfly-shaped PDC originally proposed by Lai *et al.* [90].

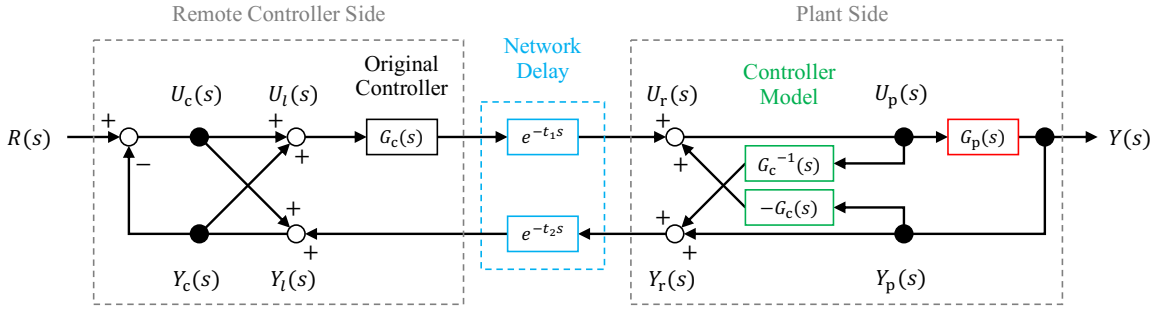


Fig. 3-3 Controller implementation of the butterfly-shaped PDC.

G_{woNET} shown in (3.1). In addition, G_{wNET} includes a time-delay element in the denominator. It is known that if the denominator of the transfer function includes time-delay elements, the design of a robust controller would become difficult, and the robustness of the entire control system degrades. Due to this fact, many recent studies have focused on the time-delay compensation method.

3.1.2 Butterfly-Shaped PDC

The butterfly-shaped PDC is a model-free time-delay compensator. The term model-free means that this compensator does not require any information regarding the time delays. Figure 3-2 shows the block diagram of the butterfly-shaped PDC originally proposed by Lai *et al.* [90]. Figure 3-3 shows the block diagram equivalent to Fig. 3-2 with the controller placed on the remote controller side of the network delay. As shown in Fig. 3-3, in the PDC-based networked control systems, the controller $G_{\mathrm{c}}(s)$ has to be implemented on both the remote controller and

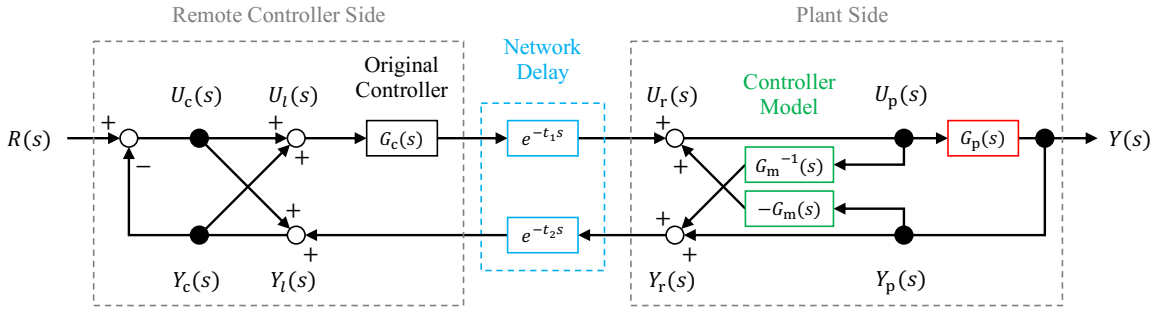


Fig. 3-4 Proposed butterfly-shaped PDC scheme considering controller model mismatch.

plant sides.

In the field of motion control, the nominal plant model is generally time-invariant as long as the plant system is not changed dynamically in operation. On the other hand, the nominal plant model of TCP/AQM network used in controller design should be frequently changed because the plant system, i.e., the amount of network traffic through routers, may fluctuate in operation. However, the controller on the plant side cannot be updated so frequently in operation because the controller is implemented to the router's firmware, whereas the software-based original controller is implemented to a remote server.

Figure 3-4 shows the proposed butterfly-shaped PDC scheme with the controller $G_c(s)$ on the plant side replaced with the controller model $G_m(s)$. In our proposed PDC-based system shown in Fig. 3-4, the original controller on the remote controller side $G_c(s)$ and the controller model on the plant side $G_m(s)$ are defined as different transfer functions to discuss their model mismatch, which would not have occurred in motion control, as assumed in [90].

In this section, it is confirmed that the block diagram shown in Fig. 3-4 compensates for network delay. First, the input and output sides of the plant butterfly element, each denotes as $U_p(s)$ and $Y_r(s)$, respectively, can be rewritten as follows:

$$U_p(s) = U_r(s) - G_m(s)Y_p(s), \tag{3.3}$$

$$Y_{\mathrm{r}}(s) = Y_{\mathrm{p}}(s) + {G_{\mathrm{m}}}^{-1}(s)U_{\mathrm{p}}(s). \tag{3.4}$$

Since the transfer function from $U_{\mathrm{p}}(s)$ to $Y_{\mathrm{p}}(s)$ can be written as (3.5), the transfer function from $U_{\mathrm{r}}(s)$ to $Y_{\mathrm{r}}(s)$ can be derived as (3.6):

$$\frac{Y_{\mathrm{p}}(s)}{U_{\mathrm{p}}(s)} = G_{\mathrm{p}}(s), \tag{3.5}$$

$$\begin{aligned} \frac{Y_{\mathrm{r}}(s)}{U_{\mathrm{r}}(s)} &= \frac{Y_{\mathrm{p}}(s) + {G_{\mathrm{m}}}^{-1}(s)U_{\mathrm{p}}(s)}{U_{\mathrm{p}}(s) + G_{\mathrm{m}}(s)Y_{\mathrm{p}}(s)} \\ &= \frac{G_{\mathrm{p}}(s) + {G_{\mathrm{m}}}^{-1}(s)}{1 + G_{\mathrm{m}}(s)G_{\mathrm{p}}(s)}. \end{aligned} \tag{3.6}$$

The forward and feedback signals right after the network, each denotes as $U_{\mathrm{r}}(s)$ and $Y_l(s)$, can be written as shown in (3.7) and (3.8), respectively.

$$U_{\mathrm{r}}(s) = U_l(s)G_{\mathrm{c}}(s)e^{-t_1 s} \tag{3.7}$$

$$Y_l(s) = Y_{\mathrm{r}}(s)e^{-t_2 s} \tag{3.8}$$

From (3.6)–(3.8), the transfer function from $U_l(s)$ to $Y_l(s)$ can be derived as (3.9)

$$\begin{aligned} \frac{Y_l(s)}{U_l(s)} &= \frac{Y_{\mathrm{r}}(s)e^{-t_2 s}G_{\mathrm{c}}(s)e^{-t_1 s}}{U_{\mathrm{r}}(s)} \\ &= \frac{\left(G_{\mathrm{p}}(s) + {G_{\mathrm{m}}}^{-1}(s)\right)G_{\mathrm{c}}(s)e^{-(t_1+t_2)s}}{1 + G_{\mathrm{m}}(s)G_{\mathrm{p}}(s)}. \end{aligned} \tag{3.9}$$

The forward and feedback output signals at the left hand side butterfly element, each denotes as $U_l(s)$ and $Y_{\mathrm{c}}(s)$, can be written as (3.10) and (3.11), respectively.

$$U_l(s) = U_{\mathrm{c}}(s) + Y_{\mathrm{c}}(s) \tag{3.10}$$

$$Y_{\mathrm{c}}(s) = Y_l(s) + U_{\mathrm{c}}(s) \tag{3.11}$$

By combining these equations, (3.12) and (3.13) can be obtained.

$$2U_{\mathrm{c}}(s) = U_l(s) - Y_l(s) \tag{3.12}$$

$$2Y_{\mathrm{c}}(s) = Y_l(s) + U_l(s) \tag{3.13}$$

From these equations and (3.9), the transfer function from $U_{\mathrm{c}}(s)$ to $Y_{\mathrm{c}}(s)$ can be derived as (3.14)

$$\begin{aligned}\frac{Y_c(s)}{U_c(s)} &= \frac{U_l(s) + Y_l(s)}{U_l(s) - Y_l(s)} \\ &= \frac{\left(1 + G_{\mathrm{m}}(s)G_{\mathrm{p}}(s)\right) + \left(G_{\mathrm{p}}(s) + {G_{\mathrm{m}}}^{-1}(s)\right) G_{\mathrm{c}}(s)e^{-(t_1+t_2)s}}{\left(1 + G_{\mathrm{m}}(s)G_{\mathrm{p}}(s)\right) - \left(G_{\mathrm{p}}(s) + {G_{\mathrm{m}}}^{-1}(s)\right) G_{\mathrm{c}}(s)e^{-(t_1+t_2)s}}.\end{aligned} \tag{3.14}$$

The forward input signal at the left hand side butterfly element $U_{\mathrm{c}}(s)$ can be expressed as (3.15)

$$U_{\mathrm{c}}(s) = R(s) - Y_{\mathrm{c}}(s). \tag{3.15}$$

By combining (3.14) and (3.15), the transfer function from the input of the control system $R(s)$ to $Y_{\mathrm{c}}(s)$ can be derived as (3.16)

$$\begin{aligned}\frac{Y_{\mathrm{c}}(s)}{R(s)} &= \frac{U_l(s) + Y_l(s)}{2U_{\mathrm{c}}(s)} \\ &= \frac{\left(1 + G_{\mathrm{m}}(s)G_{\mathrm{p}}(s)\right) + \left(G_{\mathrm{p}}(s) + {G_{\mathrm{m}}}^{-1}(s)\right) G_{\mathrm{c}}(s)e^{-(t_1+t_2)s}}{2\left(1 + G_{\mathrm{m}}(s)G_{\mathrm{p}}(s)\right)}.\end{aligned} \tag{3.16}$$

Then, by utilizing the relationship between $Y_{\mathrm{c}}(s)$ and $Y_{\mathrm{p}}(s)$ shown in (3.17) and the relationship between $U_{\mathrm{p}}(s)$ and $Y_{\mathrm{p}}(s)$ shown in (3.18), $Y_{\mathrm{p}}(s)$ can be expressed as (3.19).

$$Y_{\mathrm{c}}(s) = U_{\mathrm{c}}(s) + \left(U_{\mathrm{p}}(s)G_{\mathrm{m}}(s) + Y_{\mathrm{p}}(s)\right) e^{-t_2 s} \tag{3.17}$$

$$U_{\mathrm{p}}(s) = {G_{\mathrm{p}}}^{-1}(s) Y_{\mathrm{p}}(s) \tag{3.18}$$

$$Y_{\mathrm{p}}(s) = \frac{(Y_{\mathrm{c}}(s) - U_{\mathrm{c}}(s))}{\left({G_{\mathrm{p}}}^{-1}(s) G_{\mathrm{m}}(s) - 1\right) e^{-t_2 s}} \tag{3.19}$$

Using (3.19), the transfer function from $R(s)$ to $Y_{\mathrm{p}}(s)$ can be derived as (3.20)

$$\begin{aligned}
\frac{Y_{\mathrm{p}}(s)}{R(s)} &= \frac{G_{\mathrm{c}}(s){G_{\mathrm{m}}}^{-1}(s) - G_{\mathrm{c}}(s) G_{\mathrm{p}}(s)}{1 + G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)} \cdot \frac{1}{{G_{\mathrm{p}}}^{-1}(s){G_{\mathrm{m}}}^{-1}(s) - 1} \cdot e^{-t_1 s} \\
&= \frac{G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)}{1 + G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)} \cdot \frac{1}{G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)} \cdot \frac{G_{\mathrm{c}}(s){G_{\mathrm{m}}}^{-1}(s) - G_{\mathrm{c}}(s) G_{\mathrm{p}}(s)}{{G_{\mathrm{p}}}^{-1}(s){G_{\mathrm{m}}}^{-1}(s) - 1} \cdot e^{-t_1 s} \\
&= \frac{G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)}{1 + G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)} \cdot \frac{G_{\mathrm{c}}(s){G_{\mathrm{m}}}^{-1}(s) - G_{\mathrm{c}}(s) G_{\mathrm{p}}(s)}{G_{\mathrm{m}}(s){G_{\mathrm{m}}}^{-1}(s) - G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)} \cdot e^{-t_1 s} \\
&= \frac{G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)}{1 + G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)} \cdot \frac{G_{\mathrm{c}}(s)}{G_{\mathrm{m}}(s)} \cdot e^{-t_1 s}.
\end{aligned} \tag{3.20}$$

It is clear that $Y_{\mathrm{p}}(s) = Y(s)$, thus the transfer function for the total networked control system using PDC $G_{\mathrm{PDC}}(s)$ can be expressed as (3.21)

$$G_{\mathrm{PDC}}(s) = \frac{Y(s)}{R(s)} = \frac{G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)}{1 + G_{\mathrm{m}}(s) G_{\mathrm{p}}(s)} \cdot \frac{G_{\mathrm{c}}(s)}{G_{\mathrm{m}}(s)} \cdot e^{-t_1 s}. \tag{3.21}$$

The final transfer function shown in (3.21) consists of an ideal transfer function using $G_{\mathrm{m}}(s)$, model mismatch between $G_{\mathrm{m}}(s)$ and $G_{\mathrm{c}}(s)$, and pure forward delay t_1.

The controller model $G_{\mathrm{m}}(s)$ is generally designed identical to the original controller $G_{\mathrm{c}}(s)$. By assuming that $G_{\mathrm{m}}(s) = G_{\mathrm{c}}(s)$, (3.21) can be rewritten as (3.22)

$$G_{\mathrm{PDCmatch}}(s) = \frac{G_{\mathrm{c}}(s) G_{\mathrm{p}}(s)}{1 + G_{\mathrm{c}}(s) G_{\mathrm{p}}(s)} \cdot e^{-t_1 s}. \tag{3.22}$$

As (3.22) shows, the control system using a butterfly-shaped PDC successfully compensates the effect of the network delay.

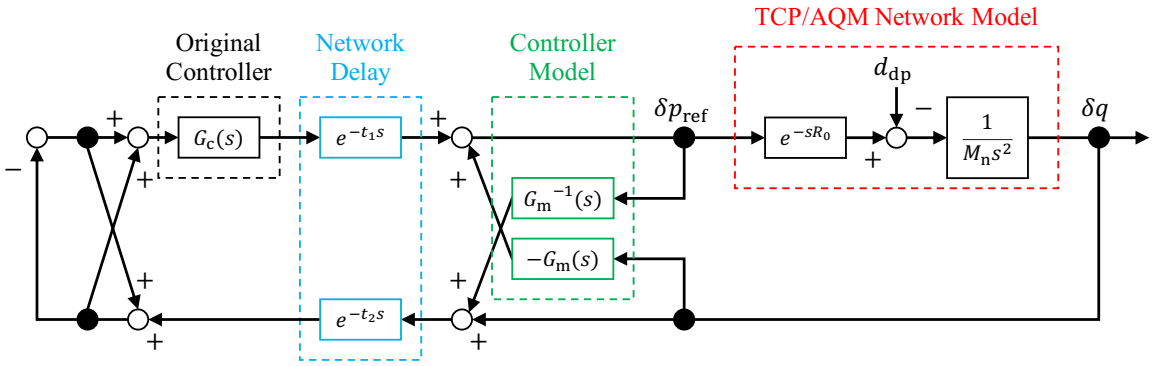


Fig. 3-5 Proposed remote AQM control system using butterfly-shaped PDC.

3.2 Control System Design

As mentioned in the previous section, the proposed control system includes an original controller and controller model, and their mismatch may occur. This section presents the block diagram of the proposed remote AQM control system with a butterfly-shaped PDC and discuss the model mismatch.

3.2.1 Proposed Remote AQM Control System Using Butterfly-Shaped PDC

Figure 3-5 shows the proposed remote AQM control system using a butterfly-shaped PDC. The specific equations describing the transfer function of the original controller $G_{\mathrm{c}}(s)$ and the controller model $G_{\mathrm{m}}(s)$ are as shown in (3.23) and (3.24), respectively.

$$G_{\mathrm{c}}(s) = M_{\mathrm{n_c}} \left(K_{\mathrm{p}} + K_{\mathrm{i}} \frac{1}{s} + K_{\mathrm{d}} s \right) \tag{3.23}$$

$$G_{\mathrm{m}}(s) = M_{\mathrm{n_m}} \left(K_{\mathrm{p}} + K_{\mathrm{i}} \frac{1}{s} + K_{\mathrm{d}} s \right) \tag{3.24}$$

In (3.23) and (3.24), $M_{\mathrm{n_c}}$ and $M_{\mathrm{n_m}}$ denote the inertia models used for designing the original controller and the controller model, respectively.

3.2.2 Model Mismatch

As shown in (3.22), if the original controller and controller model are identical, the transfer function of the total control system can be constructed only from the ideal transfer function and pure forward delay. However, it is possible that the original controller and the controller plant may differ in reality. it is supposed that the controller model is implemented in the bottleneck router's firmware in actual implementation of this system. Thus, the implemented controller model may be updated periodically, but not in real time.

As shown in (2.22), the inertia model M_{n} is defined by the bottleneck link capacity C and the nominal number of TCP sessions N_{n}. In addition, both the original controller and controller model include their own individual inertia models. Therefore, the equations defining $M_{\mathrm{n_c}}$ and $M_{\mathrm{n_m}}$ can be rewritten as (3.25) and (3.26), respectively.

$$M_{\mathrm{n_c}} = -\frac{2N_{\mathrm{n_c}}}{C^2} \tag{3.25}$$

$$M_{\mathrm{n_m}} = -\frac{2N_{\mathrm{n_m}}}{C^2} \tag{3.26}$$

In (3.25) and (3.26), $N_{\mathrm{n_c}}$ and $N_{\mathrm{n_m}}$ denote the nominal number of TCP sessions used to design original controller and controller model, respectively.

The original controller and controller model both utilize the same PID controller gain parameters, and it is not likely that the bottleneck link capacity changes over time. Therefore, the mismatch between the original controller and controller model may occur when the values of $N_{\mathrm{n_c}}$ and $N_{\mathrm{n_m}}$ are not equal. As shown in (3.21), the model mismatch ratio directly affects the transfer function of the whole system proportionally. Therefore, the design of the original controller must be adjusted to lower the proportional effect due to model mismatch while still maintaining the overall performance of the TCP/AQM congestion control system.

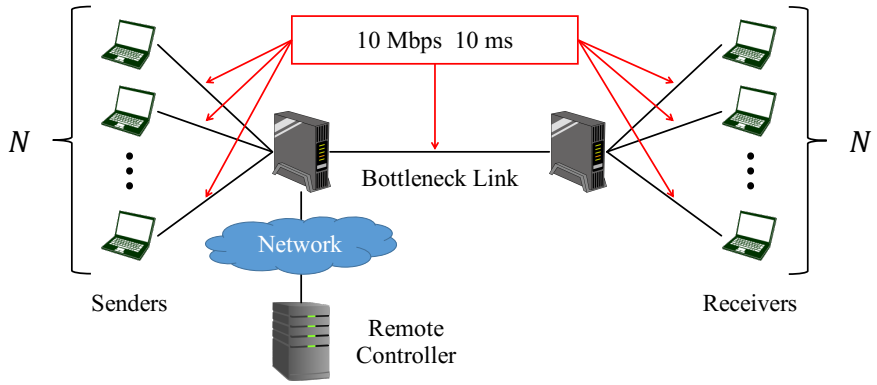


Fig. 3-6 Simulation topology.

3.3 Performance Evaluation

This section shows simulation results that confirm the validity of the proposed butterfly-shaped PDC and the effect of model mismatch.

3.3.1 Simulation Setup

Simulations were performed using the network simulator ns-2 in order to validate the proposed congestion controller. The dumbbell shaped network topology shown in Fig. 3-6 was utilized in the simulations. The value of the number of TCP sessions N varies depending on the simulation purpose, which was set to 100 if not specified. All links except one connecting the bottleneck router and the remote controller have 10 ms of latency, which makes the round-trip propagation delay of TCP sessions equal to 60 ms. The network delays between the bottleneck router and the remote controller (which are t_1 and t_2) are specified for each simulation. The parameters used in the simulations are shown in Table 3-1. The parameters used to design the PID controller are shown in Table 3-2. Control parameters were set by referring to [91]. The values of nominal number of TCP sessions used to design the original controller $N_{\mathrm{n_c}}$ and the controller model $N_{\mathrm{n_m}}$ vary depending on the simulation purpose, which were set equal to N if not specified. As the TCP protocol version, TCP NewReno was utilized.

Table 3-1 Simulation parameters

Sender side link capacity	10 Mbps
Receiver side link capacity	10 Mbps
Bottleneck link capacity C	10 Mbps
Sender side link latency	10 ms
Receiver side link latency	10 ms
Bottleneck link latency	10 ms
Packet size	1000 Bytes
Simulation duration	180 s
Packet drop probability operating point p_0	0
Router buffer size	200 packets
Target queue length q_0	100 packets
Control period	0.001 s

Table 3-2 Control parameters

K_p	Proportional gain	900
K_i	Integral gain	700
K_d	Derivative gain	55
g_dif	Cut-off frequency of pseudo-differential	50 rad/s

3.3.2 Compensation of Identical Forward and Feedback Time Delays

Simulation results using the matching controller model with forward network delay t_1 equal to feedback network delay t_2 are shown in this subsection. Figure 3-7 shows the simulation results when $t_1 = t_2 = 50$ ms. The figure shows the simulation results of the control system without the butterfly-shaped PDC, referred as "without PDC", and with the butterfly-shaped PDC, referred as "with PDC". From the simulation results, it can be seen that the queue length oscillation of "without PDC" is larger than that of "with PDC". This indicates that the network delay directly affects the stability of the system, and the PDC effectively compensates this network delay.

Figure 3-8 shows the standard deviation (SD) of the queue length for various forward and feedback delays. The SD values were calculated using all simulation results after 10 s out of the total simulation duration of 180 s, in order to avoid the effect of overshoot occurring at the start of the simulations. A larger SD value indicates that a larger queue length oscillation

is occurring. It can be seen from Fig. 3-8 that the SD is larger when network delay is larger without the PDC. However, this effect can be compensated and the SD values can be kept at a relatively lower value by implementing PDC to the system, even with a large network delay. The effectiveness of the PDC in compensating for the identical forward and feedback network delays can be verified from these simulation results.

3.3.3 Compensation of Different Forward and Feedback Time Delays

This subsection shows the simulation results using the matching controller model when t_1 and t_2 are different. Figure 3-9 shows the simulation results when t_1 = 20 ms and t_2 = 60 ms. Figure 3-10 shows the simulation results when t_1 = 60 ms and t_2 = 20 ms. Figures 3-11 and 3-12 each shows the SD value for various t_2 values when t_1 = 20 and 60 ms, respectively. From these simulation results, it can be stated that the proposed control system utilizing the PDC can compensate network delays, even if the forward and feedback delays are not identical. The relationship between forward and feedback delays also did not affect the PDC efficiency.

3.3.4 Compensation of Fluctuating Time Delays

This subsection shows the simulation results using the matching controller model while t_1 and t_2 are simultaneously fluctuating. Figure 3-13 shows two types of delay fluctuations used in the simulation. Figures 3-14 and 3-15 shows the simulation results while t_1 and t_2 are fluctuating with type A and type B. From these simulation results, it can be seen that the PDC successfully compensates for network delays, even if the network delays fluctuate.

3.3.5 Discussion About Model Mismatch

In this subsection, all simulations were performed when t_1 and t_2 were set to 50 ms. First, the value of $N_{\mathrm{n_m}}$ was set to 100, as was done in the previous simulations. Then, by assuming a situation where the actual number of TCP sessions N was changed from the original number of

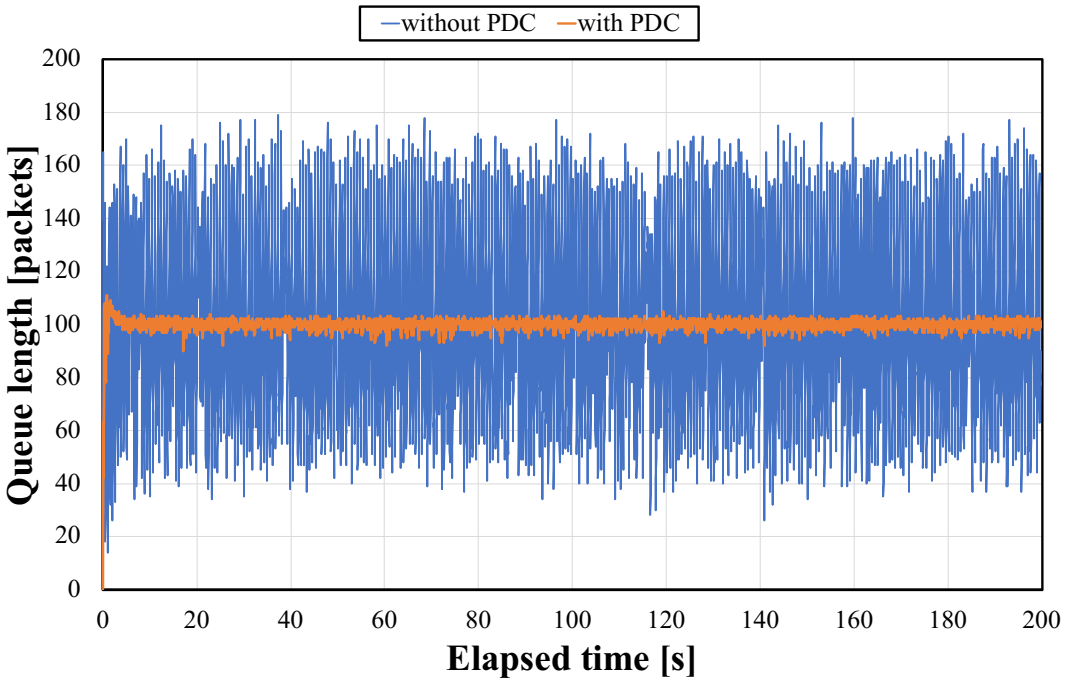


Fig. 3-7 Simulation results ($t_1 = t_2 = 50$ ms).

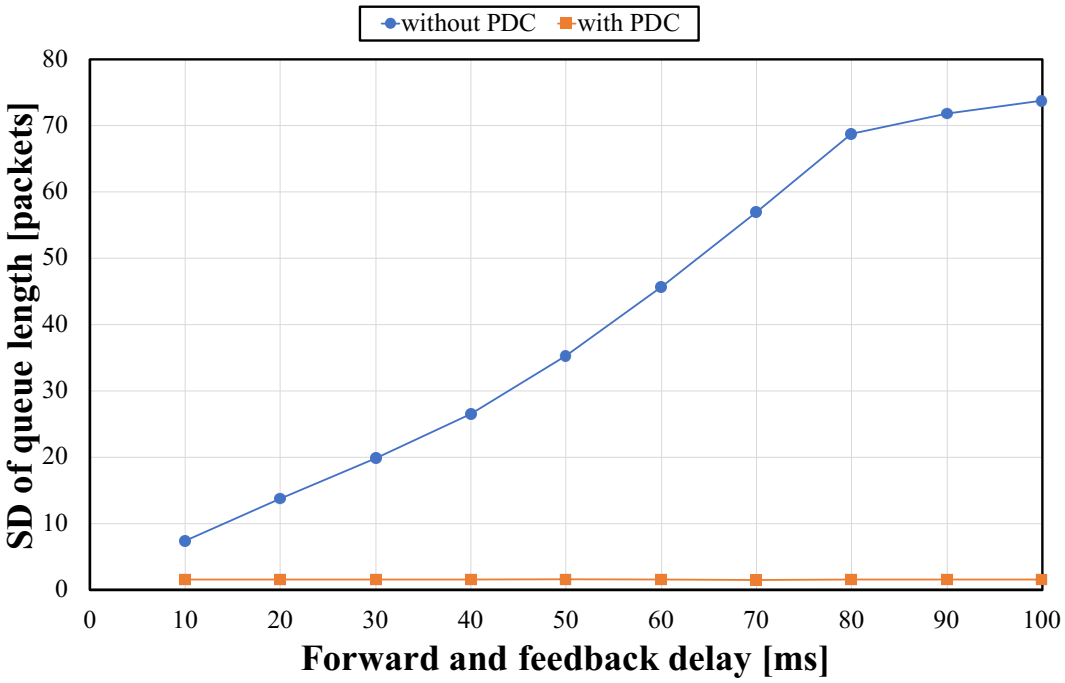


Fig. 3-8 Comparison of SD values ($t_1 = t_2$).

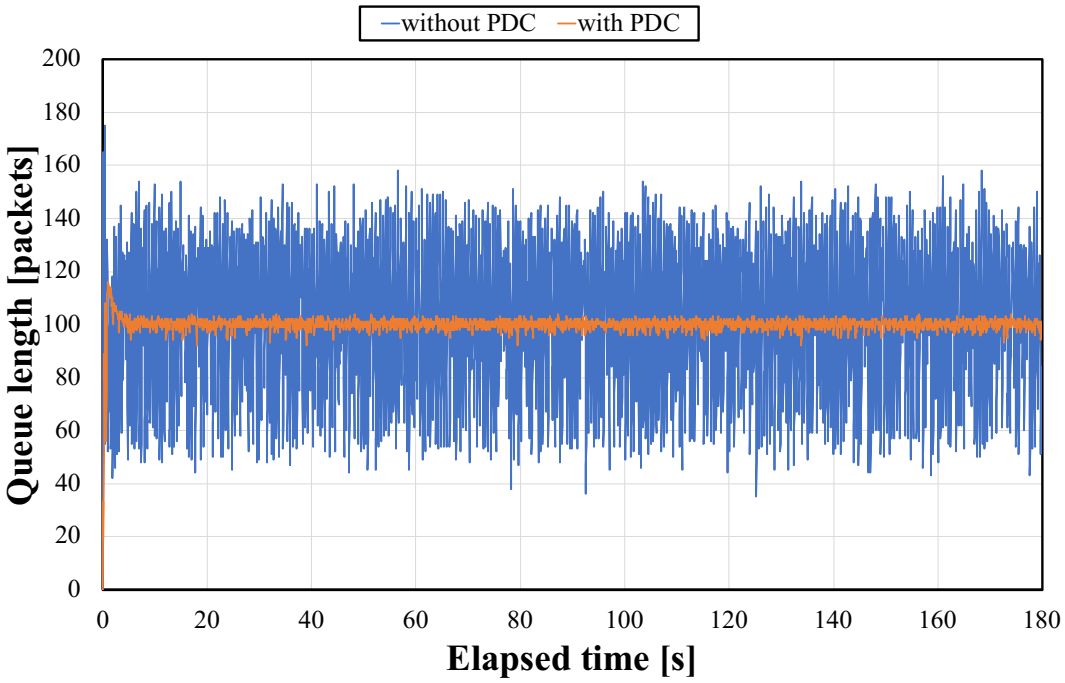


Fig. 3-9 Simulation results (t_1 = 20 ms, t_2 = 60 ms).

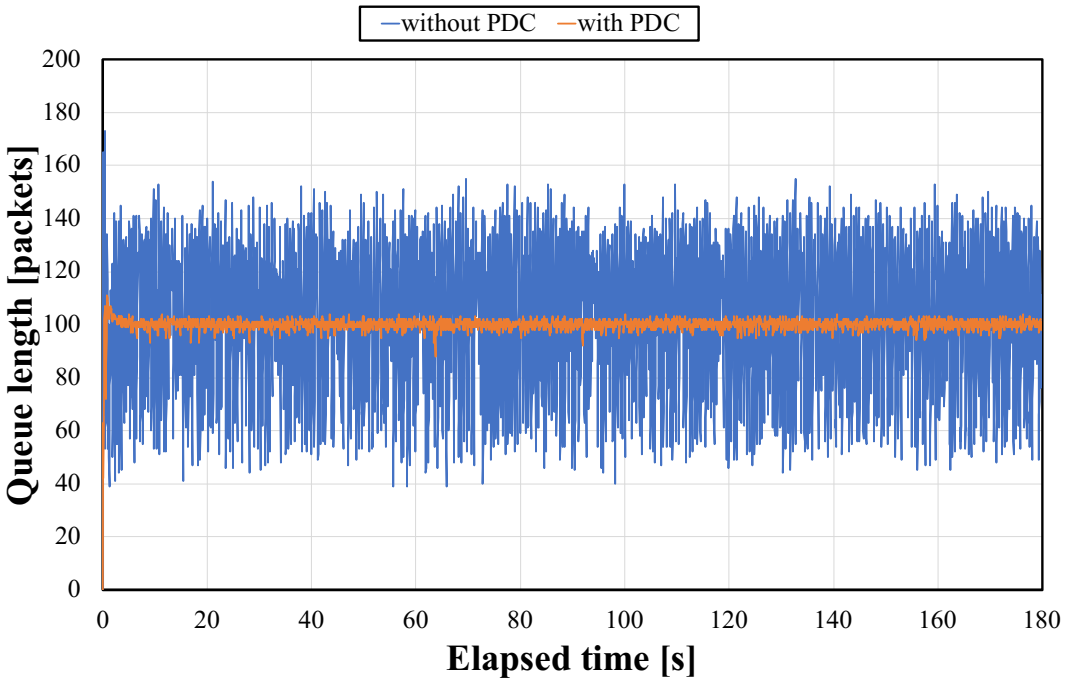


Fig. 3-10 Simulation results (t_1 = 60 ms, t_2 = 20 ms).

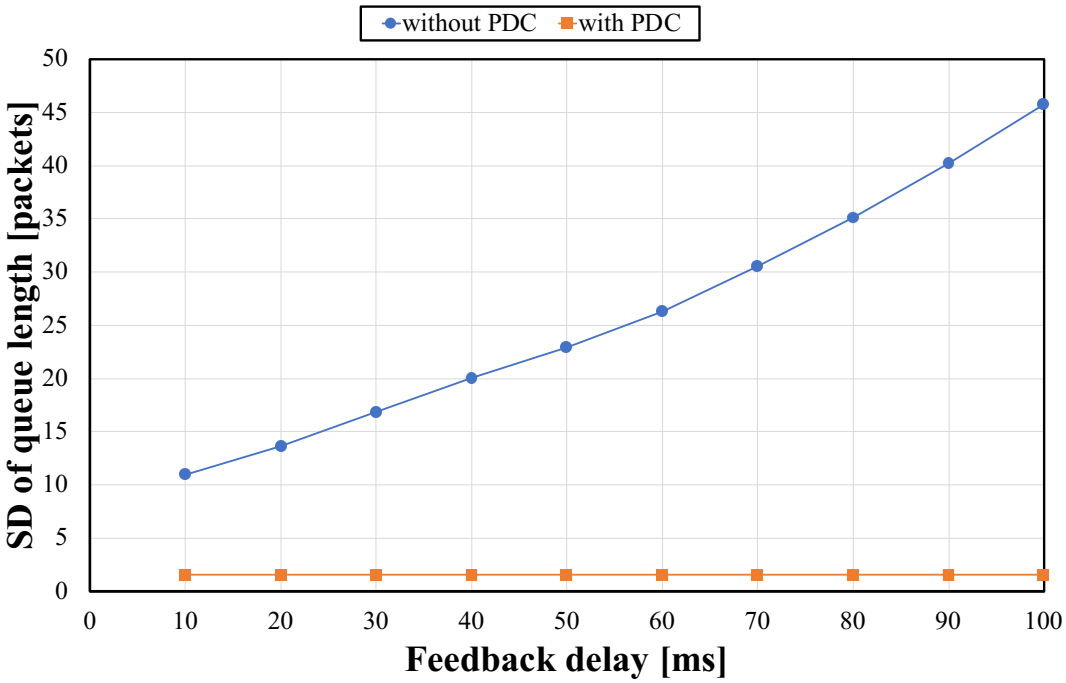


Fig. 3-11 Comparison of SD values (t_1 = 20 ms).

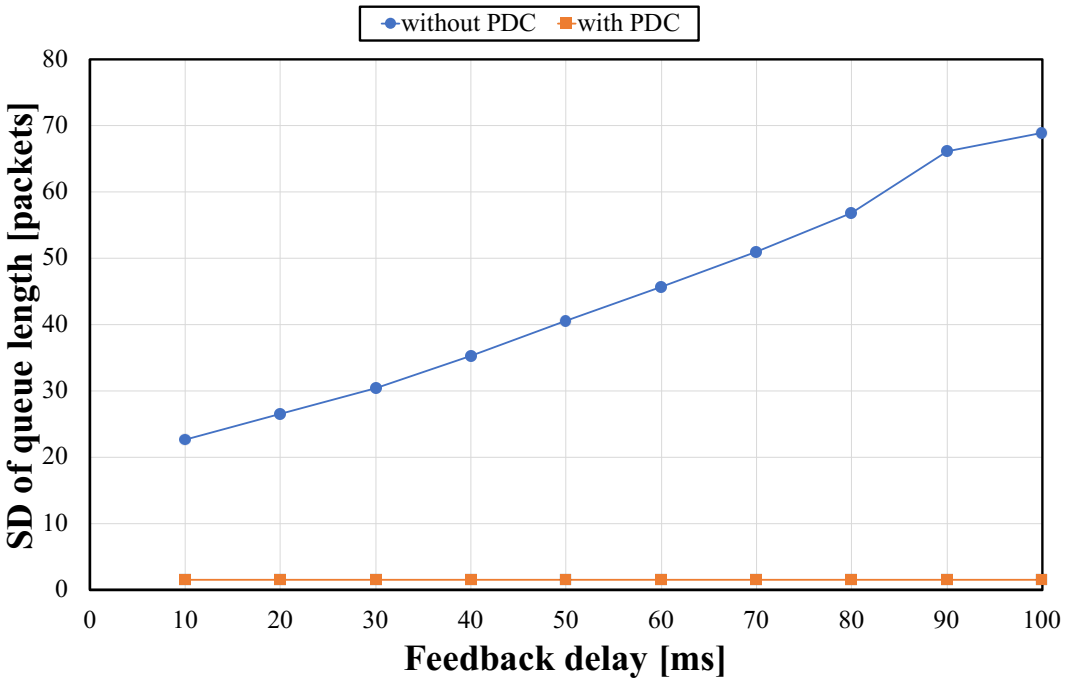
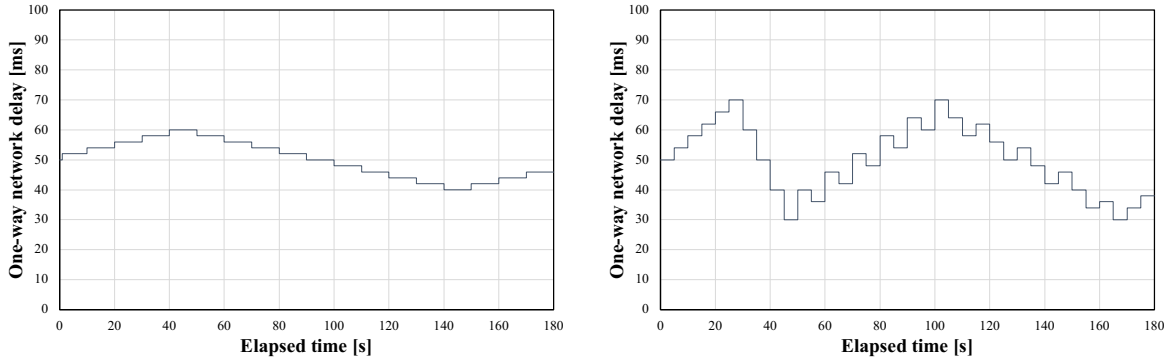


Fig. 3-12 Comparison of SD values (t_1 = 60 ms).



(a) Delay fluctuation type A (b) Delay fluctuation type B

Fig. 3-13 Two types of delay fluctuations.

TCP sessions 100, the value of N was changed and the simulation was repeated. Assuming that the original controller can detect a change in the actual number of TCP sessions, the value of $N_{\mathrm{n_c}}$ was kept equal to N.

Figure 3-16 shows the simulation results when $N_{\mathrm{n_m}} = 100$ and $N = N_{\mathrm{n_c}} = 60$, 80, 100, 120, and 140. Since the target queue length q_{cmd} was set to 100 packets, the ideal response of the system had its average queue length close to 100 packets. However, as Fig. 3-16 shows, the average queue length differed from the ideal response when mismatch error between $N_{\mathrm{n_m}}$ and $N_{\mathrm{n_c}} = N$ occurred. The stability of the system, i.e., the queue length oscillation, remained nearly constant, even if mismatch occurred. Thus, the effect of model mismatch can be discussed by focusing on the average queue length.

Figure 3-17 shows the results of average queue lengths when $N_{\mathrm{n_m}} = 100$, and N and $N_{\mathrm{n_c}}$ were set to 60, 80, 100, 120, or 140. The average queue lengths were calculated using all simulation results after 10 s out of the total simulation duration of 180 s in order to avoid overshoot occurring at the beginning of the simulations. As Fig. 3-17 shows, the average queue length would be nearly equal to 100 packets when $N_{\mathrm{n_c}} = 100$, which was equal to $N_{\mathrm{n_m}}$. Therefore, it can be assumed that when $N_{\mathrm{n_c}} = N_{\mathrm{n_m}}$, the system will return the ideal average queue length.

The same simulations with $N_{\mathrm{n_m}} = 80$ and 120 were performed in order to confirm this

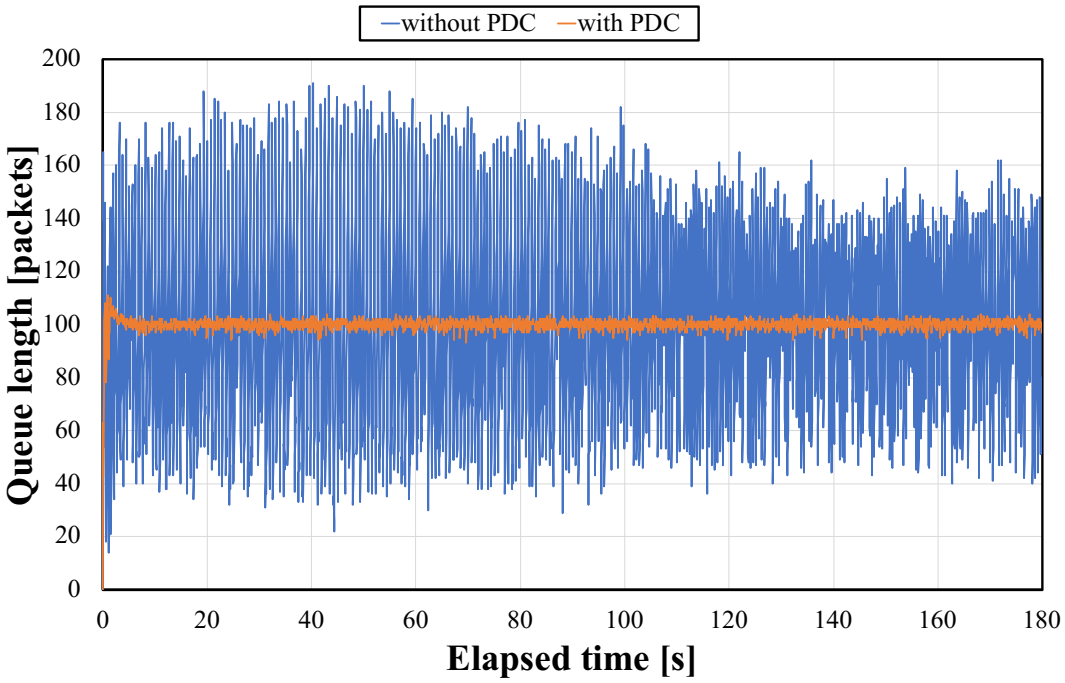


Fig. 3-14 Simulation results for (t_1, t_2) =Type A.

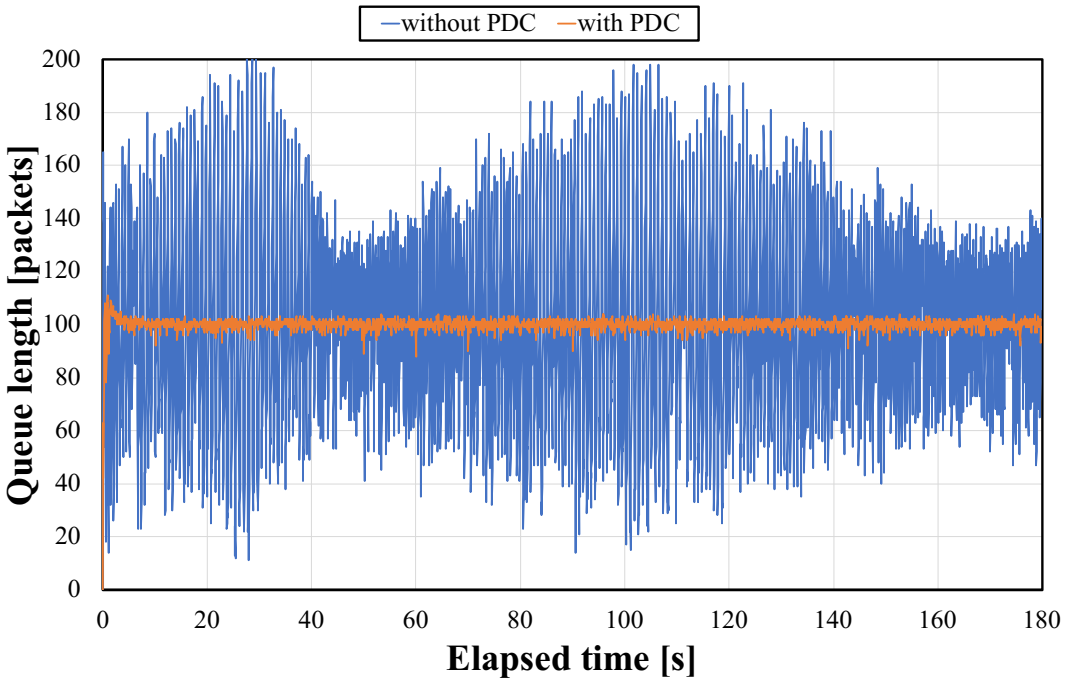


Fig. 3-15 Simulation results for (t_1, t_2) =Type B.

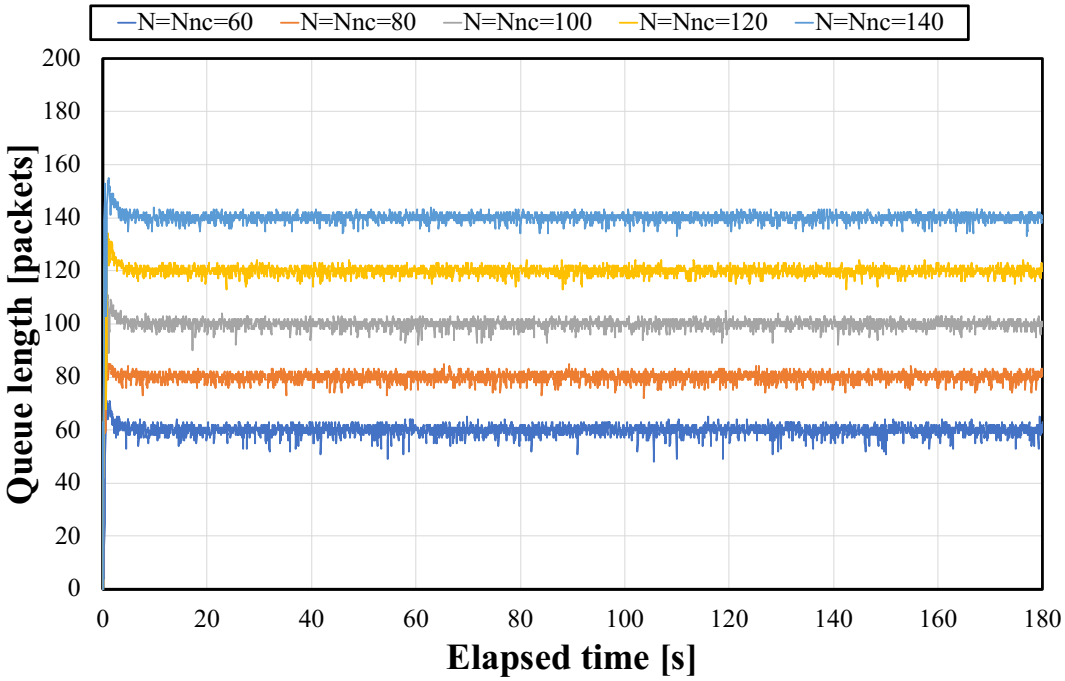


Fig. 3-16 Simulation results with mismatch ($N = N_{nc}$).

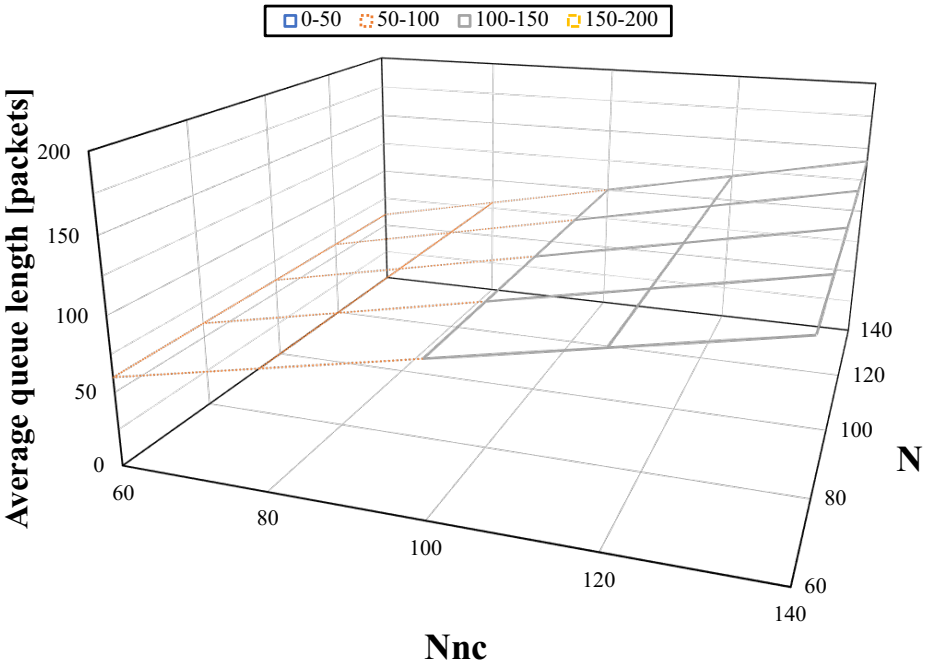


Fig. 3-17 Average queue length (N_{nm}=100).

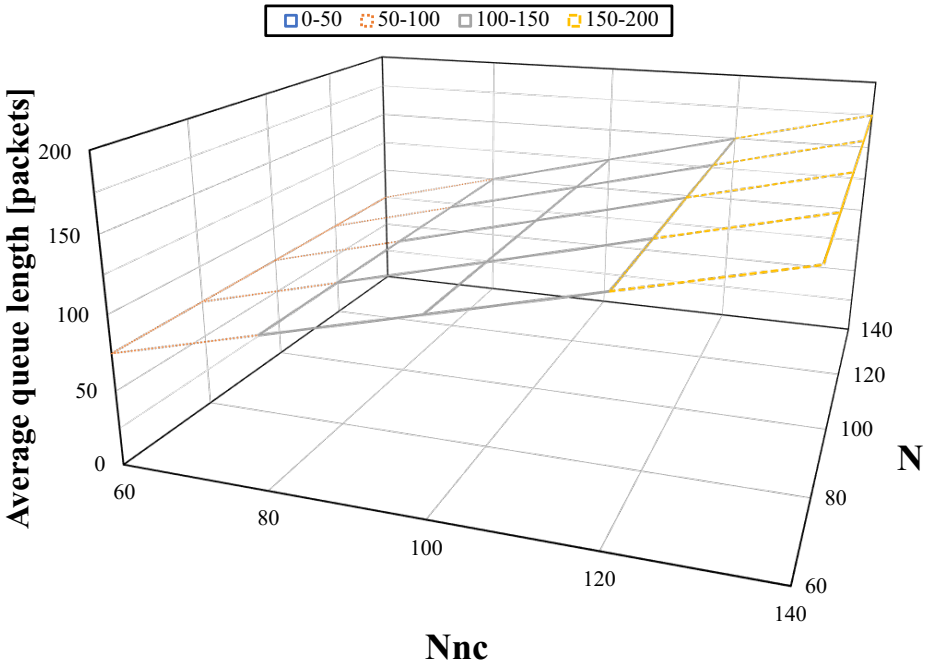


Fig. 3-18 Average queue length (N_{n_m} = 80).

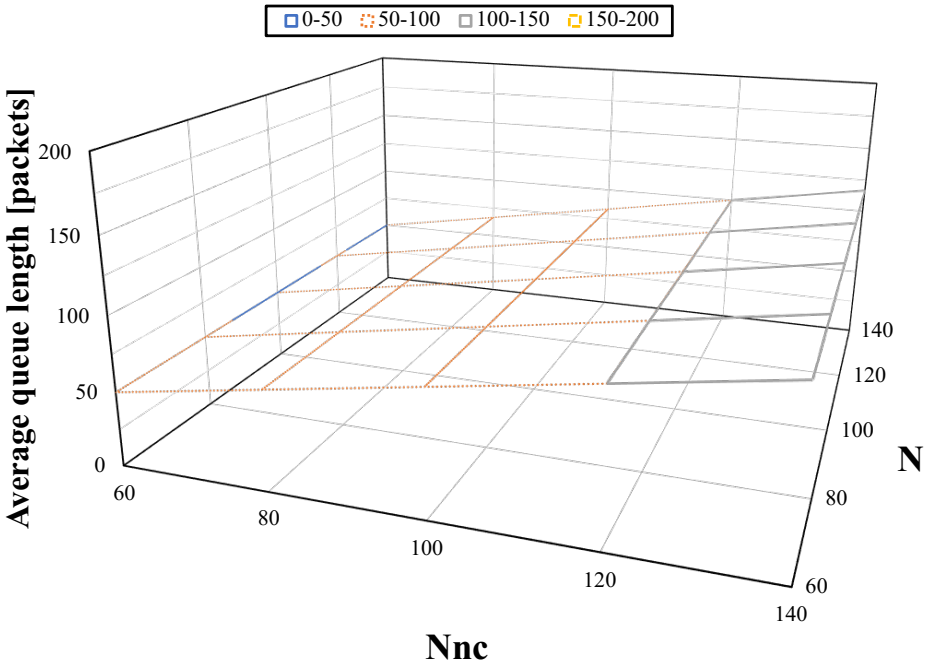


Fig. 3-19 Average queue length (N_{n_m} = 120).

assumption. Figures 3-18 and 3-19 each shows the results of average queue lengths when N_{n_m} = 80 and 120, respectively. In Fig. 3-18, the average queue length is nearly equal to 100 packets when $N_{n_c} = 80$, which is equal to N_{n_m}. In Fig. 3-19, the average queue length is nearly equal to 100 packets when $N_{n_c} = 120$, which is equal to N_{n_m}. The value of N in both figures does not clearly affect the average queue length.

These simulation results confirm that $N_{n_c} = N_{n_m}$ is suitable, regardless of the value of N.

3.4 Summary

This chapter proposed a remote congestion controller with the butterfly-shaped PDC for time-delay compensation in the TCP/AQM network. The major novelty of this study is the application of the butterfly-shaped PDC, which was originally proposed for motion control, to the non-linear TCP/AQM network control system. The simulation results showed that the proposed controller with the butterfly-shaped PDC effectively stabilized the TCP/AQM network even if the system included time-varying delays. The proposed congestion control system with PDC may have a model mismatch between the actual system and the controller model, and the ratio of the controller model and the original controller proportionally affects the output of the system. It was verified that by matching the parameters of the original controller to that of the controller model, the effect of model mismatch can be excluded even if the parameters do not match that of the actual system. Future work includes considering the situation where multiple routers are controlled simultaneously with different controller models.

Chapter 4

Robust Dead Time Compensation for High-Latency Networks

4.1 Background

Congestion control based on AQM is negatively impacted by the modeling error of the system and the time delay caused by the RTT of the TCP flows. Recently proposed AQM based on control theory is no exception, and there are some studies focusing on compensating either the modeling error or time delay. However, simultaneous compensation of the modeling error and time delay has not been studied for TCP/AQM networks. The related studies on compensation methods for the time delay or modeling error in TCP/AQM networks are described below.

As previously mentioned in chapters 1 and 2, the receiver host notifies the corresponding sender host of congestion when it detects the packet loss, and the sender host scales the sending window size down based on it. This means that after sending a packet, the sender host cannot scale the sending window size down until it receives the notification with a specific delay of RTT. Thus, TCP/AQM networks include a time delay element such as RTT between the senders and receivers, which affects the performance of the control system [92]. Several methods have

been proposed to compensate for the effect of this built-in delay of the TCP/AQM network, and the most famous method is implementing the SP [85, 87, 89]. However, a TCP/AQM congestion control system using an SP cannot cope with fluctuations of the network parameters, such as the number of TCP connections. Because such fluctuations are likely to happen in the TCP/AQM network, this effect must be considered in the design of the AQM scheme.

The robust congestion controller design that uses a disturbance observer (DOB) is an effective way to cope with the parameter fluctuations [78]. DOB has been widely utilized in the field of motion control [93]. It can estimate parameter variations as disturbances, and the estimated disturbance is fed back for robust control. The modeling error can be included in the disturbance as well, making the modeling much easier. On the other hand, the TCP/AQM congestion control system using the DOB does not have a time delay compensator. The DOB-based controller can be effectively used when the time delay is small. Because of its effectiveness, the DOB has been used for congestion control in AQM with small time delay [63, 79, 94, 95, 96]. However, a significant time delay in the TCP/AQM network will cause serious degradation in the performance and result in instability of the network. Therefore, the effects of time delay should be considered in the design of the AQM scheme.

This chapter proposes a novel TCP/AQM congestion control scheme supporting TCP flows by implementing disturbance compensation and time delay compensation in an integrated manner and avoids serious congestions under large RTT. The proposed control system consists of a PD controller, DOB with an artificial delay, and an SP. By utilizing the DOB with an artificial delay, the disturbance of the system can be suppressed even if the TCP/AQM network includes a time delay element. Implementation of the SP will negate the effect of the time delay element by using the plant model. A feature of the SP is that it needs a reasonably approximated plant model to function ideally. Because the disturbance, including the modeling error, can be suppressed by the DOB, the SP can obtain a well-approximated plant model by utilizing the same model employed by the DOB. Kato *et al.* [97] proposed a motion control system with the DOB and SP to compensate for modeling error and time delay. However, unlike the motion control system,

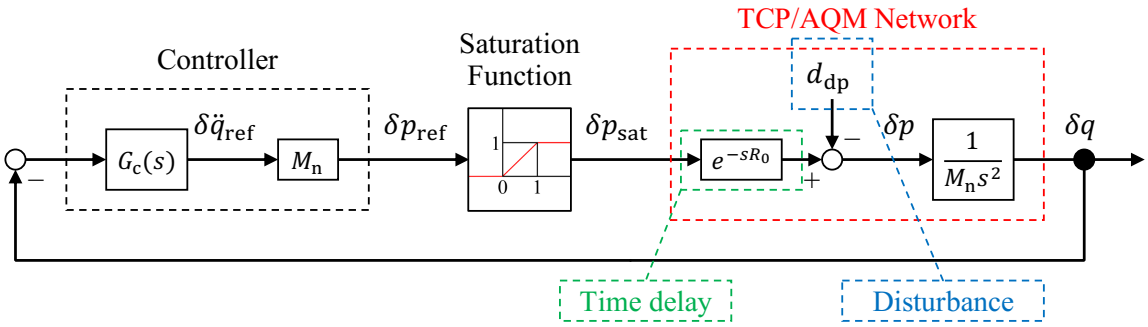


Fig. 4-1 TCP/AQM network with a saturation function.

the TCP/AQM network congestion control system is subject to input saturation of packet drop probability. The existence of saturation function makes a direct introduction of the DOB and SP impossible. In this study, this input saturation is taken into account and a novel TCP/AQM network congestion control system with the DOB and SP is designed, to compensate for the disturbance and time delay simultaneously.

4.1.1 Saturation Function

As shown in Fig. 2-11, the controller of the control-theory based TCP/AQM congestion control system calculates the reference packet drop probability δp_{ref} from δq. The fact that δp_{ref} is in the unit of probability means that its value must be saturated to be in the range of 0 to 1. Figure 4-1 is the block diagram shown in Fig. 2-11 with the saturation function explicitly indicated, where δp_{sat} denotes the reference packet drop probability after the saturation function. In addition, Fig. 4-1 clarifies the position of disturbance and time delay which are going to be discussed in the following sections.

The saturation function alters the value of δp_{ref} as follows:

$$\delta p_{\mathrm{sat}} = \begin{cases} 0 & (\delta p_{\mathrm{ref}} < 0) \\ \delta p_{\mathrm{ref}} & (0 \leqq \delta p_{\mathrm{ref}} \leqq 1) \\ 1 & (1 < \delta p_{\mathrm{ref}}) \,. \end{cases} \tag{4.1}$$

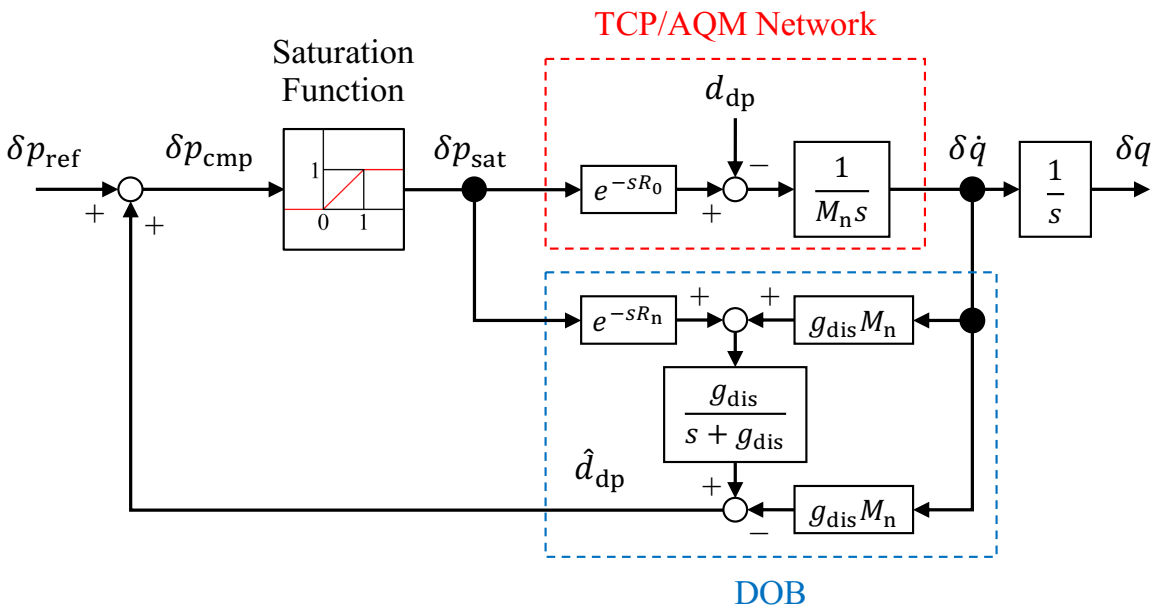


Fig. 4-2 TCP/AQM network with the DOB.

The existence of this saturation function induces a problem when implementing DOB and SP in an integrated manner, which will be discussed afterwards in this chapter.

4.1.2 DOB

As indicated in 4-1, the disturbance of the TCP/AQM network control system is denoted as d_{dp}. This disturbance includes the parameter fluctuation, modeling error, coexistence of non-TCP flows, etc. In order to suppress the effect of the disturbance d_{dp}, the DOB [93] has been implemented as a disturbance compensator. Because the TCP/AQM network has an input delay in its system, the DOB must consider the time delay element [98]. By compensating the disturbance, the DOB attempts to make the actual TCP/AQM network behave identical to the nominal model.

Figure 4-2 shows the TCP/AQM network with the DOB, where $\delta\dot{q}$ and δp_{cmp} each denotes the queue velocity and the reference packet drop probability including the compensation signal from the DOB, respectively. In addition, $\hat{d}_{\mathrm{dp}}$, R_{n}, and g_{dis} denote the compensation signal output

from the DOB, the nominal time delay, and the cut-off frequency of the DOB, respectively.

By assuming that $R_n = R_0$, the DOB estimates the value of d_{dp} through the low pass filter, as (4.2).

$$\hat{d}_{dp} = \frac{g_{dis}}{s + g_{dis}} d_{dp}. \tag{4.2}$$

The DOB suppresses d_{dp} by modifying the reference packet drop probability as follows:

$$\delta p_{cmp} = \delta p_{ref} + \hat{d}_{dp}. \tag{4.3}$$

Because the actual input to the TCP/AQM network is the probability, its value must be between 0 and 1. The saturation function will alter the value of δp_{cmp} based on this rule, as expressed as (4.4)

$$\delta p_{sat} = \begin{cases} 0 & \left(\delta p_{cmp} < 0\right) \\ \delta p_{cmp} & \left(0 \leqq \delta p_{cmp} \leqq 1\right) \\ 1 & \left(1 < \delta p_{cmp}\right). \end{cases} \tag{4.4}$$

By assuming that the error due to saturation does not occur, i.e., $\delta p_{sat} = \delta p_{cmp}$, the relationship at the point where the disturbance d_{dp} joins the control signal can be obtained as (4.5)

$$\begin{aligned} M_n s \delta \dot{q} &= e^{-sR_0} \left(\delta p_{sat} + \hat{d}_{dp}\right) - d_{dp} \\ &= e^{-sR_0} \delta p_{cmp} + e^{-sR_0} \frac{g_{dis}}{s + g_{dis}} d_{dp} - d_{dp} \\ &= e^{-sR_0} \delta p_{cmp} - \left(\frac{s + g_{dis}\left(1 - e^{-sR_0}\right)}{s + g_{dis}}\right) d_{dp} \\ &= e^{-sR_0} \delta p_{cmp} - \left(H + L\left(1 - e^{-sR_0}\right)\right) d_{dp}, \end{aligned} \tag{4.5}$$

where the high-pass filter $H = \frac{s}{s+g_{dis}}$ and the low-pass filter $L = \frac{g_{dis}}{s+g_{dis}}$. Using (4.5), the equivalent

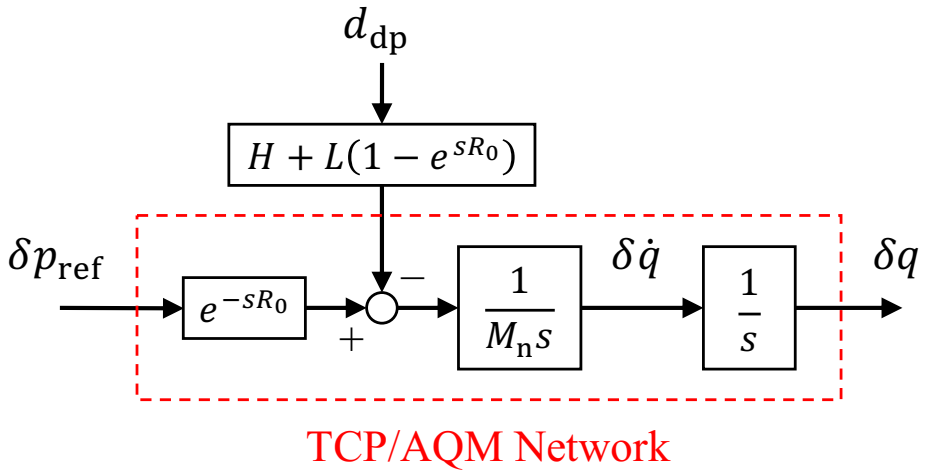


Fig. 4-3 Equivalent block diagram of Fig. 4-2 when $\delta p_{\mathrm{sat}} = \delta p_{\mathrm{cmp}}$.

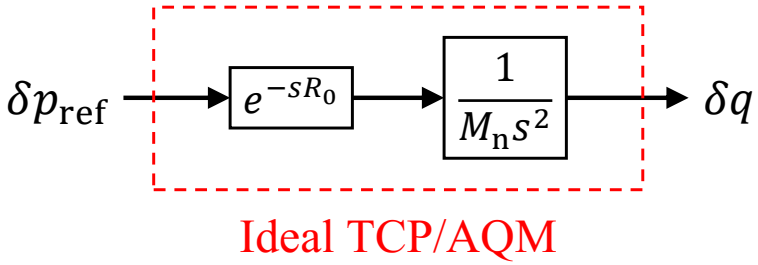


Fig. 4-4 Ideal TCP/AQM network with full suppression of disturbance.

block diagram of Fig. 4-2 can be obtained as shown in Fig. 4-3. Ideally, if $g_{\mathrm{dis}} \to \infty$ and $R_0 \to 0$, the disturbance is suppressed completely, i.e., the TCP/AQM network model perfectly converges on the nominal inertia model of the TCP/AQM network M_{n}. The block diagram of an ideal TCP/AQM network with its disturbance suppressed perfectly is shown in Fig. 4-4. The robustness of the system with DOB against disturbance has been discussed in previous studies [63, 78].

4.1.3 SP

In the feedback control, the control command of the following sampling time is determined according to a deviation of the current command and the response. When a time delay element

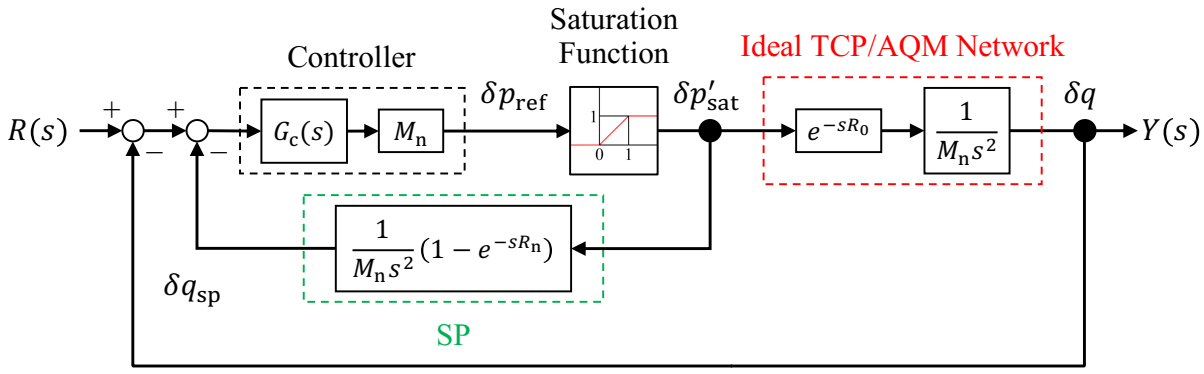


Fig. 4-5 Time delay compensation by the SP.

exists in the feedback loop, the response will be delayed for the same period, inducing a degradation of the control performance. The SP excludes the time delay element outside of the feedback loop by using the control plant model and time delay model [85].

The block diagram of a control system of the ideal TCP/AQM network with SP implemented is shown in Fig. 4-5, where $R(s)$ and $Y(s)$ denote the input and output of the system in the Laplacian domain. In addition, $\delta p'_{sat}$ and δq_{sp} denote the input and output signal of the SP. The signal $\delta p'_{sat}$ is also the input signal to the ideal TCP/AQM network. As shown in Fig. 4-5, the SP utilizes the TCP/AQM model and the delay model for its transfer function. By assuming that the error due to saturation does not occur, i.e., $\delta p'_{sat} = \delta p_{ref}$, the transfer function of the entire system can be expressed as follows:

$$\frac{Y(s)}{R(s)} = \frac{G_c(s)e^{-sR_0}}{s^2 + G_c(s)\left(1 + e^{-sR_0} - e^{-sR_n}\right)}. \tag{4.6}$$

If the plant model is identical to the actual control plant and the nominal delay R_n equals the actual delay R_0, the equivalent block diagram of Fig. 4-5 can be obtained as shown in Fig. 4-6. The transfer function of (4.6) can be also transformed into (4.7)

$$\frac{Y(s)}{R(s)} = \frac{G_c(s)}{s^2 + G_c(s)}e^{-sR_0}. \tag{4.7}$$

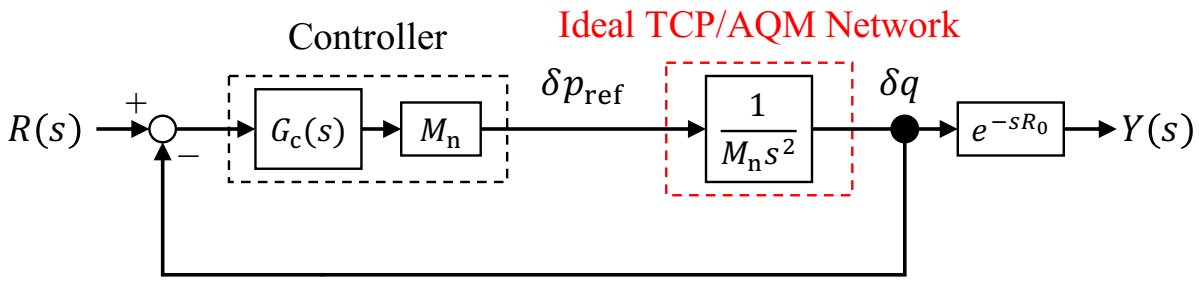


Fig. 4-6 Equivalent block diagram of Fig. 4-5.

As Fig. 4-6 indicates, the time delay element can be excluded from the feedback loop. This means that the characteristic equation of the control system does not include the time delay element, which will enable the design of a controller without considering the effect of the time delay.

4.2 Control System Design

The proposed control system is designed by implementing DOB and SP at the same time. This section describes the design procedures of the proposed control system. The PD controller is utilized for the controller of the system.

4.2.1 AQM Using PD Controller

The calculation of PD controller is nearly identical to that of PID controller, the only difference being whether integral gain K_i exists or not. When the PD controller is utilized, $\delta\ddot{q}_{ref}$ is calculated as (4.8)

$$\begin{aligned}\delta\ddot{q}_{ref} &= G_c(s)\delta q \\ &= \left(K_p + K_d s\right)(q_0 - q)\,.\end{aligned} \tag{4.8}$$

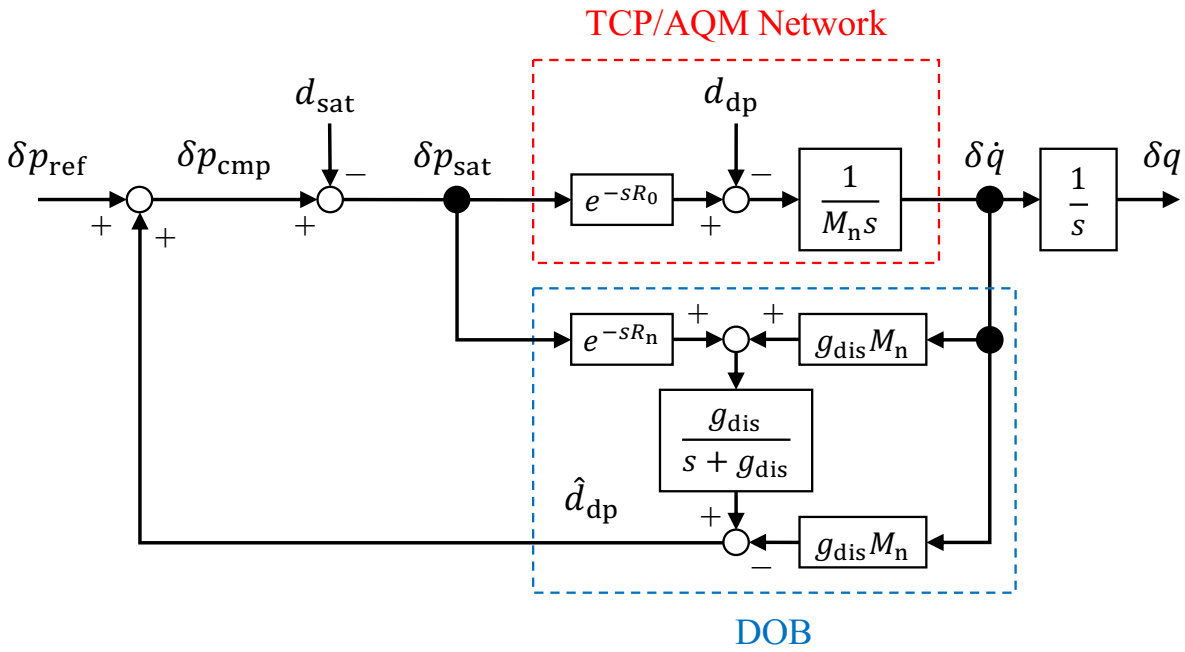


Fig. 4-7 TCP/AQM network with the DOB and saturation as a disturbance d_{sat}.

Thus, using (2.24) and (4.8), δp_{ref} calculated by the PD controller can be derived as (4.9)

$$\begin{aligned}\delta p_{\mathrm{ref}} &= M_{\mathrm{n}} G_{\mathrm{c}}(s)\delta q \\ &= M_{\mathrm{n}}\left(K_{\mathrm{p}} + K_{\mathrm{d}}s\right)(q_0 - q)\,.\end{aligned} \tag{4.9}$$

4.2.2 Implementation of DOB and SP in an Integrated Manner

The integrated implementation of DOB and SP to a linear control system has been discussed by Kato *et al.* in the past [97]. However, when implementing the DOB and SP in an integrated manner in the TCP/AQM network congestion control system, its nonlinear characteristics prevents the proper functioning of the simple independent implementation of the two compensators. Because the input to the TCP/AQM network is the packet drop probability, the saturation function must occur right before the TCP/AQM network. Because of this design restriction, the compensation signal from the DOB must be added before the input signal passes through the saturation function, as shown in Fig. 4-2. In the previous section, the saturation function was

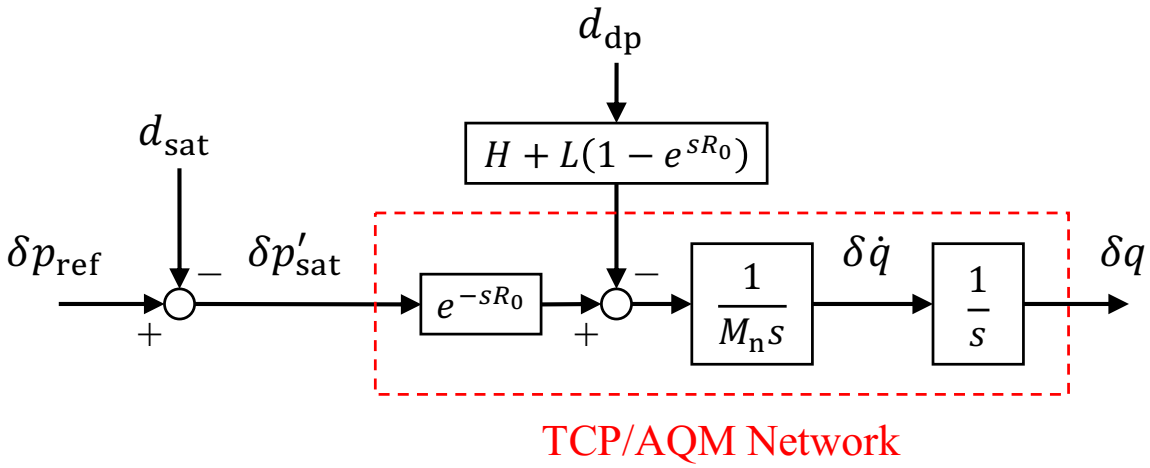


Fig. 4-8 Equivalent block diagram of Fig. 4-7.

ignored and it was assumed that $\delta p_{\mathrm{sat}} = \delta p_{\mathrm{cmp}}$ whereas, in reality, there are adequate chances that δp_{cmp} becomes lower than 0 or higher than 1 according to (4.4). The effect of the error occurring at the saturation function can be expressed as (4.10)

$$\delta p_{\mathrm{sat}} = \delta p_{\mathrm{cmp}} - d_{\mathrm{sat}}, \tag{4.10}$$

where d_{sat} denotes the saturation error. By using (4.10) as an equivalent method of expressing the saturation function, the block diagram shown in Fig. 4-2 can be redrawn as Fig. 4-7, and its equivalent block diagram is shown in Fig. 4-8. The signal alteration caused by the saturation function in Fig. 4-8 can be expressed as (4.11)

$$\delta p'_{\mathrm{sat}} = \delta p_{\mathrm{ref}} - d_{\mathrm{sat}}. \tag{4.11}$$

When the SP is implemented in the TCP/AQM congestion control system with the DOB shown in Fig. 4-7, the SP requires the saturated input signal without compensation by the DOB, $\delta p'_{\mathrm{sat}}$, as shown in Fig. 4-5. However, the required input $\delta p'_{\mathrm{sat}}$ is not explicitly shown in Fig. 4-7, since the compensation signal from the DOB $\hat{d}_{\mathrm{dp}}$ is added to δp_{ref} before the saturation occurs. Therefore, our proposed method calculates the required input signal $\delta p'_{\mathrm{sat}}$ based on the actual

input signal δp_{sat} and $\hat{d}_{\mathrm{dp}}$ and inputs the estimated $\delta p'_{\mathrm{sat}}$ into the SP.

4.2.3 Proposed Control System

Using (4.3), (4.10), and (4.11), $\delta p'_{\mathrm{sat}}$ can be rewritten as (4.12)

$$\begin{aligned} \delta p'_{\mathrm{sat}} &= \delta p_{\mathrm{ref}} - d_{\mathrm{sat}} \\ &= \delta p_{\mathrm{ref}} + \hat{d}_{\mathrm{dp}} - d_{\mathrm{sat}} - \hat{d}_{\mathrm{dp}} \\ &= \delta p_{\mathrm{cmp}} - d_{\mathrm{sat}} - \hat{d}_{\mathrm{dp}} \\ &= \delta p_{\mathrm{sat}} - \hat{d}_{\mathrm{dp}}. \end{aligned} \tag{4.12}$$

Considering this relationship, the proposed TCP/AQM network congestion control system which consists of the PD controller, DOB, and SP was designed. Figure 4-9 shows the block diagram of the proposed TCP/AQM network congestion control system, where $G_{\mathrm{PD}}(s)$ denotes the transfer function of PD controller as shown in (4.8), and δp_{sp} denotes the input signal to the SP, which is expressed as follows:

$$\delta p_{\mathrm{sp}} = \delta p_{\mathrm{sat}} - \hat{d}_{\mathrm{dp}} = \delta p'_{\mathrm{sat}}. \tag{4.13}$$

The input to the SP in the proposed method is equal to the required input signal $\delta p'_{\mathrm{sat}}$.

4.3 Performance Evaluation

In this section, first the stability analysis using Nyquist diagram is presented. After that, the simulation setup and multiple patterns of simulation results are shown. The results are compared with those of the conventional methods, namely the control systems using RED[29], controlled delay (CoDel) [99], proportional integral controller enhanced (PIE) [100], PID controller, and PD controller with DOB. For simplicity, the proposed method is denoted as PD+DOB+SP, and the five aforementioned conventional methods are denoted as RED, CoDel, PIE, PID, and

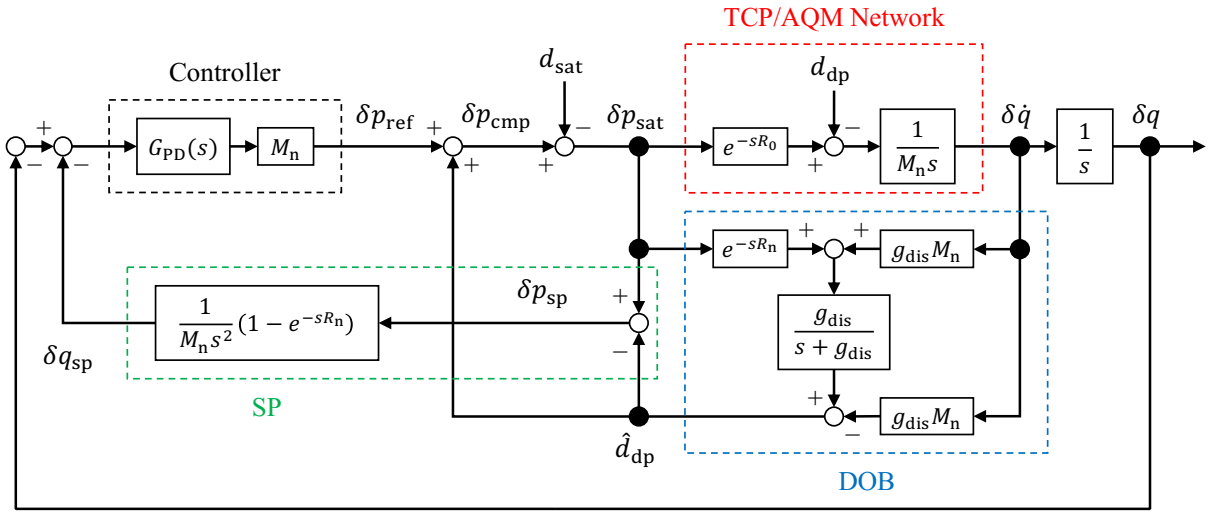


Fig. 4-9 Block diagram of the proposed control system.

PD+DOB, respectively.

4.3.1 Stability Analysis Using Nyquist Diagram

Figures 4-10 and 4-11 show the Nyquist diagrams of the loop transfer functions when R_{n} values were set to 20 ms and 100 ms, respectively. The proposed PD+DOB+SP is compared with PID and PD+DOB. It was assumed that $R_0 = R_{\mathrm{n}}$ and $d_{\mathrm{sat}} = 0$. In addition, d_{dp} was set to 0 in PID and fully suppressed by DOB in PD+DOB and the proposed PD+DOB+SP. The control gains were set to the same values as those in the simulations. The proportional, integral, and derivative gains of PID were set to 900, 700, and 55, respectively. The proportional and derivative gains of PD+DOB and PD+DOB+SP were set to 900 and 60, respectively.

In Fig. 4-10, all the three methods were stable under this condition of $R_{\mathrm{n}} = 20$ ms. The proposed PD+DOB+SP method resulted in larger gain and phase margins than any other methods. In Fig. 4-11, PID and PD+DOB were unstable under this condition of $R_{\mathrm{n}} = 100$ ms, while the proposed PD+DOB+SP was stable. The results indicated that the proposed PD+DOB+SP effectively compensated for the effect of the time delay element and stabilized the system, which was also evident from the closed-loop transfer function expressed by (4.7).

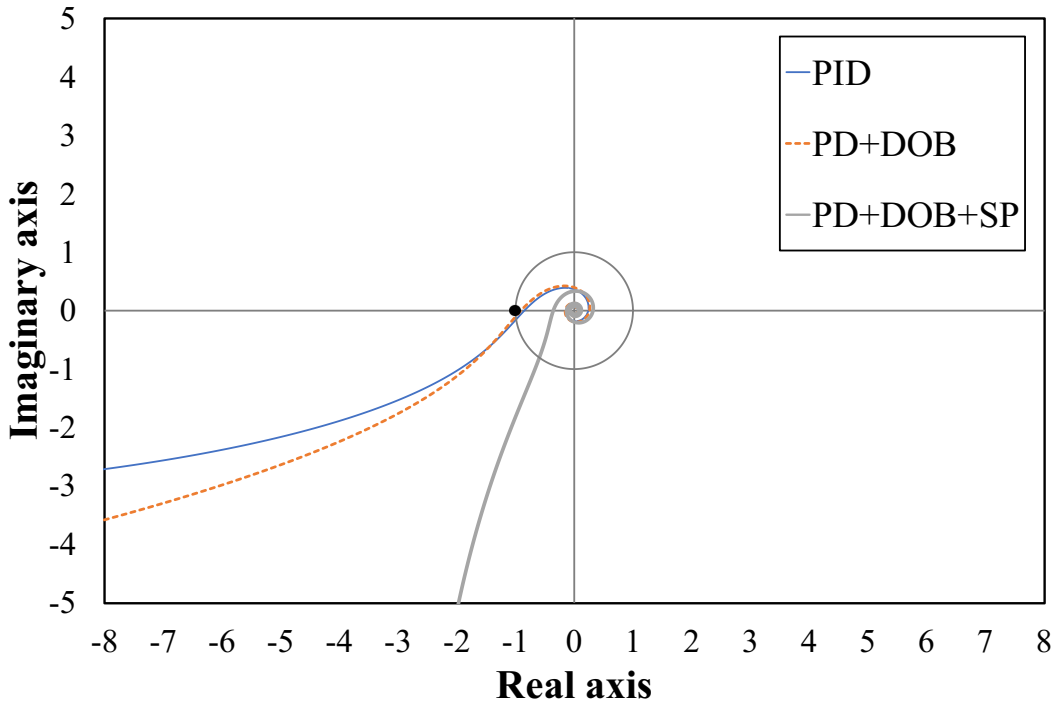


Fig. 4-10 Nyquist diagram (R_n = 20 ms).

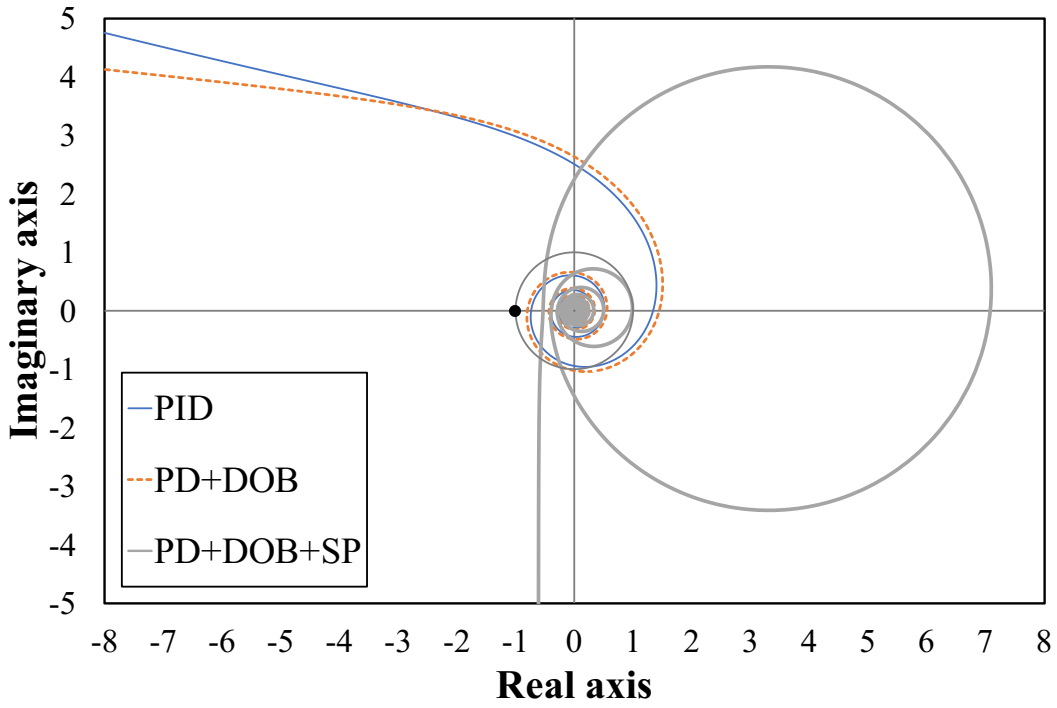


Fig. 4-11 Nyquist diagram (R_n = 100 ms).

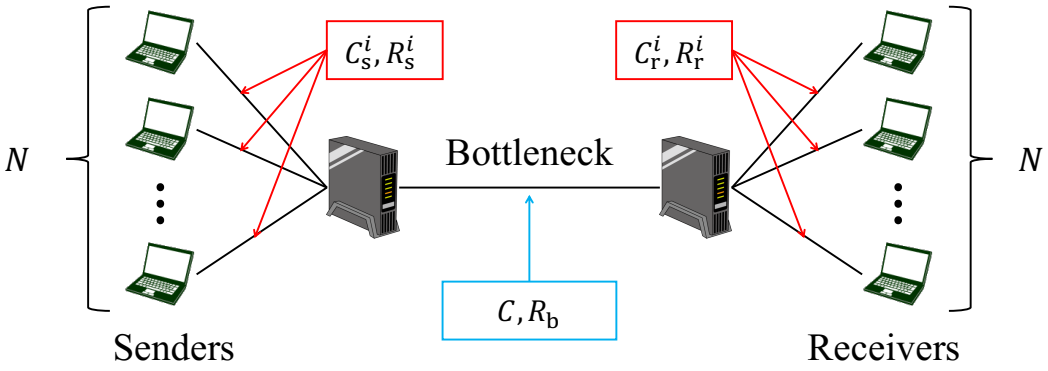


Fig. 4-12 Simulation topology.

4.3.2 Simulation Setup

Simulations were performed using the network simulator ns-2. The dumbbell shaped network topology shown in Fig. 4-12 was utilized in the simulations. In Fig. 4-12, C_s^i and R_s^i denote the link capacity and link latency between the sender side bottleneck link router node and the ith ($i = 1, 2, \cdots, N$) sender node. Similarly, C_r^i and R_r^i denote the link capacity and link latency between the receiver side bottleneck link router node and the ith ($i = 1, 2, \cdots, N$) receiver node. The bottleneck link latency is denoted as R_b; thus, the propagation delay of the ith TCP session T_p^i can be calculated as shown in (4.14)

$$T_p^i = 2\left(R_s^i + R_b + R_r^i\right). \tag{4.14}$$

Hereon, if the propagation delays of all TCP sessions are equal, the propagation delay is denoted as T_p.

The network parameters used in the simulations are shown in Table 4-1, and these are used unless mentioned otherwise. Thus, the propagation delay is set to 100 ms, unless another value is specifically mentioned. In RED [29], the queue weight, minimum threshold, maximum threshold, and maximum packet drop probability were set to 0.002, 50 packets, 150 packets, and 0.02, respectively. In CoDel [101], the interval and target were set to 116 ms and 14.4 ms,

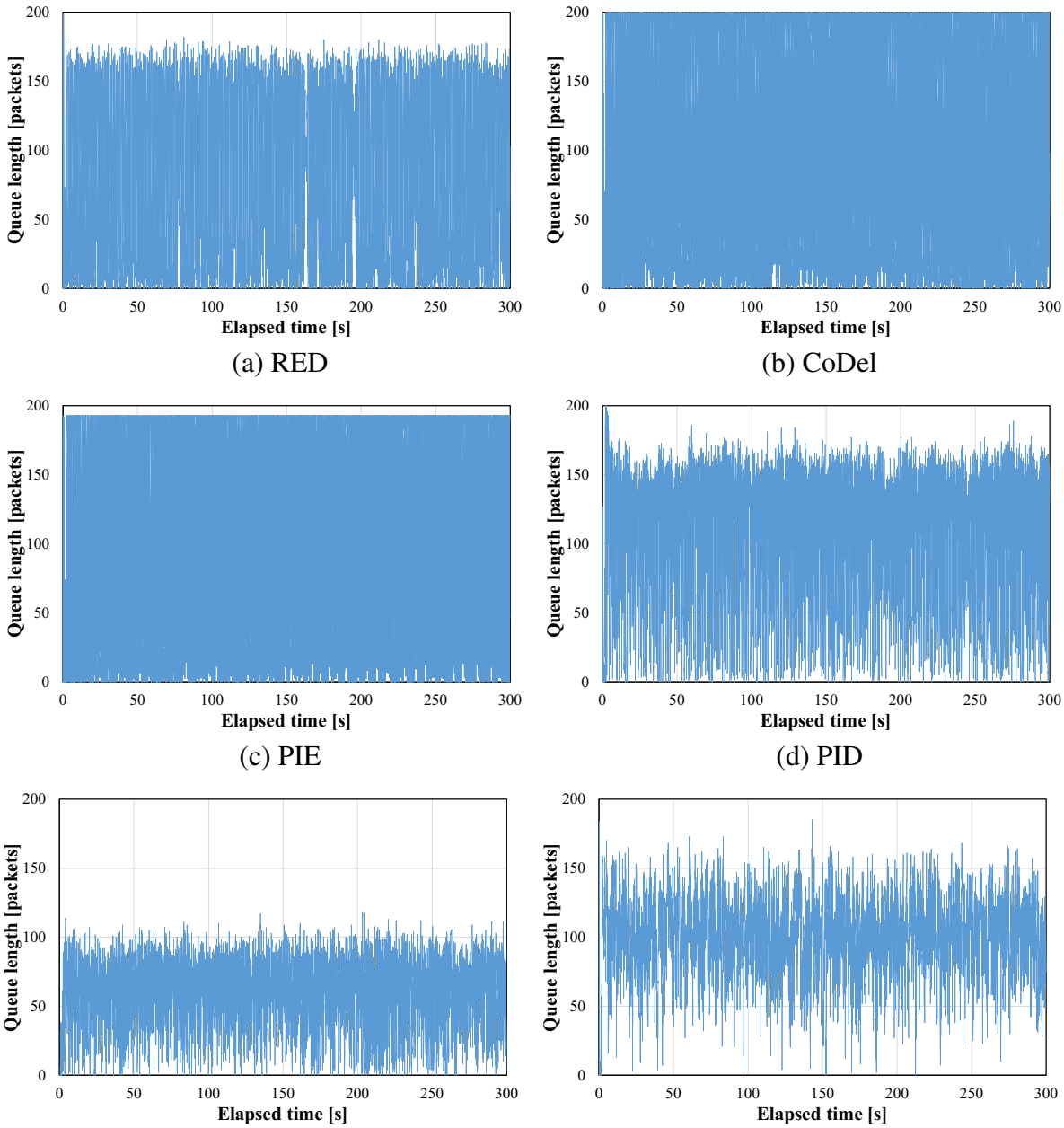
Table 4-1 Network parameters.

Number of TCP sessions N	100
Sender side link capacity C_{s}^i $(i = 1, 2, \cdots, N)$	100 Mbps
Receiver side link capacity C_{r}^i $(i = 1, 2, \cdots, N)$	100 Mbps
Bottleneck link capacity C	100 Mbps
Sender side link latency R_{s}^i $(i = 1, 2, \cdots, N)$	20 ms
Receiver side link latency R_{r}^i $(i = 1, 2, \cdots, N)$	20 ms
Bottleneck link latency	10 ms
Nominal number of TCP sessions N_{n}	100
Nominal bottleneck link capacity model C_{n}	100 Mbps
Nominal RTT delay model R_{n}	100 ms
Packet size	1040 bytes
Maximum window size $wnd_{\max}$	20 packets
Simulation duration	300 s
Router buffer size	200 packets
Target queue length q_0	100 packets
Control period	0.001 s

respectively. In PIE [102], the update interval, reference latency, and maximum burst allowance were set to 8 ms, 16 ms, and 16 ms, respectively. The parameters for CoDel and PIE were heuristically adjusted so as to have the highest throughput while maintaining the average queue length close to the target queue length of 100 packets. The parameters used in the PD controller, PID controller, and DOB are shown in Table 4-2. These control parameters were set by referring to [91]. All simulations were performed without using ECN.

4.3.3 Queue Length Fluctuation

Figure 4-13 shows the queue length fluctuations of RED, CoDel, PIE, PID, PD+DOB, and PD+DOB+SP for the whole simulation duration of 300 s. The queue length was obtained every 0.05 s. Figure 4-14 shows the first 25 s of queue length fluctuations, along with the queue length smoothed using EMA. The smoothing factor for EMA was set to 0.01. The smoothed queue

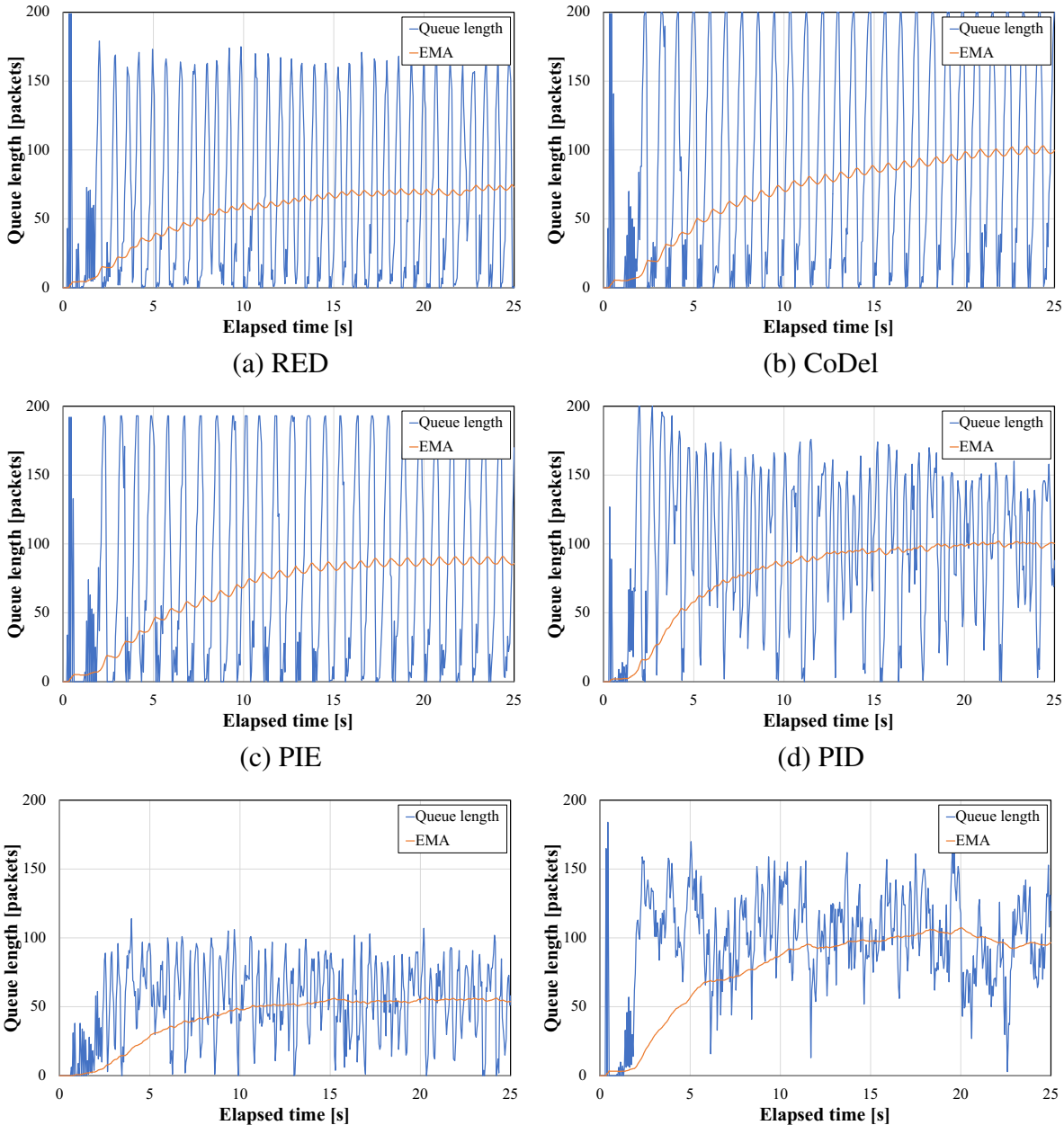


(a) RED (b) CoDel

(c) PIE (d) PID

(e) PD+DOB (f) PD+DOB+SP

Fig. 4-13 Queue length fluctuations.



(a) RED (b) CoDel

(c) PIE (d) PID

(e) PD+DOB (f) PD+DOB+SP

Fig. 4-14 Queue length fluctuations during the first 25 s.

Table 4-2 Control parameters for PID controller, PD controller, and DOB.

$K_{\mathrm{p}}^{\mathrm{pid}}$	Proportional gain for PID controller	900
$K_{\mathrm{i}}^{\mathrm{pid}}$	Integral gain for PID controller	700
$K_{\mathrm{d}}^{\mathrm{pid}}$	Derivative gain for PID controller	55
$K_{\mathrm{p}}^{\mathrm{pd}}$	Proportional gain for PD controller	900
$K_{\mathrm{d}}^{\mathrm{pd}}$	Derivative gain for PD controller	60
g_{dif}	Cut-off frequency for PD and PID controllers	50 rad/s
g_{dis}	Cut-off frequency for DOB	50 rad/s

length was utilized for evaluating the responsiveness of each method.

To evaluate the network performance of each method, the average throughput, average goodput, and fairness index of goodput were obtained. The average throughput is defined as the time average of total TCP throughput on the bottleneck link. The average goodput is defined as the time average of total throughput excluding retransmission of packets, taking the amount of packet loss into consideration. This excludes the retransmitted packets from the number of total packets received, denoting how much packet drop occurred. The fairness index is a number that shows the fairness of the goodput of each TCP session in unit of %. Jain's fairness index [103] is utilized in this thesis, and its calculation equation is as follows;

$$\text{Fairness} = \frac{\left(\sum x_i\right)^2}{n\left(\sum {x_i}^2\right)} * 100 \qquad [\%], \tag{4.15}$$

where x_i denotes the ith value of the variable that the fairness is calculated with.

To evaluate the control performance of each method, the average queue length, SD of the queue length, maximum queue length, and the total number of empty buffer samples were utilized. The average queue length was measured to evaluate not only the control performance but also the average queueing delay. The SD of the queue length was calculated to consider the queue length oscillation and jitter caused by the oscillation. The maximum queue length can be used to consider the occurrence of buffer overflow and the maximum queueing delay.

Table 4-3 Summary of results in Figs. 4-13 and 4-14.

	RED	CoDel	PIE
Average throughput [Mbps]	97.11	98.85	98.92
Average goodput [Mbps]	96.76	98.49	98.55
Fairness index of goodput [%]	99.96	92.84	84.49
Average queue length [packets]	72.57	100.53	100.62
SD of queue length [packets]	60.71	73.05	70.91
Maximum queue length [packets]	182	200	193
Total number of empty buffer samples	499	306	349
Rise time of the smoothed queue length [s]	11.55	13.65	18.70
Response delay time of the smoothed queue length [s]	5.00	5.20	5.85
	PID	PD+DOB	PD+DOB+SP
Average throughput [Mbps]	99.63	99.80	99.99
Average goodput [Mbps]	99.23	99.24	99.55
Fairness index of goodput [%]	99.94	99.94	99.93
Average queue length [packets]	100.07	57.78	100.28
SD of queue length [packets]	48.45	25.64	27.73
Maximum queue length [packets]	189	118	185
Total number of empty buffer samples	70	51	2
Rise time of the smoothed queue length [s]	8.95	10.15	8.15
Response delay time of the smoothed queue length [s]	4.20	5.05	4.45

The total number of empty buffer samples is defined as the number of measured samples when the queue length is equal to 0, which indicates the duration when the buffer of the bottleneck router is empty. The buffer overflow and empty buffer should be avoided to efficiently utilize the bottleneck link capacity.

To evaluate the responsiveness of each method, the rise time and response delay time of the smoothed queue length were utilized. Both the rise time and response delay time were derived by using EMA of queue length, as shown in Fig. 4-14. The rise time is defined as the time required for rising of the smoothed queue length from 10% to 90% of the average queue length. The response delay time is defined as the time required for rising of the smoothed queue length

from the initial value to 50% of the average queue length.

The average throughput, average goodput, fairness index of goodput, average queue length, SD of queue length, maximum queue length, the total number of empty buffer samples, rise time of the smoothed queue length, and response delay time of the smoothed queue length for each method are listed in Table 4-3. The average throughput, average queue length, SD of queue length, maximum queue length, and the total number of empty buffer samples were measured except for the first 10 s of the simulation duration, to eliminate the surges induced by the TCP algorithm. The ideal behavior of the AQM congestion control system is to maintain the queue length stable at the target queue length without any fluctuation and fully utilize the bottleneck link capacity.

As listed in Table 4-3, the proposed PD+DOB+SP showed the highest average throughput and goodput among the six methods, followed by PD+DOB and PID. The relationship between the throughput and goodput, which indicated the amount of packet loss, showed a similar tendency for each method. RED method showed the highest fairness index of goodput, while PID, PD+DOB, and proposed PD+DOB+SP methods had nearly identical fairness index values. The fairness indexes of the four methods were notably higher than those of CoDel and PID. Maintaining the relatively high fairness index while raising the throughput and goodput, the proposed PD+DOB+SP was confirmed to provide the best network performance of the six methods.

PID had the average queue length closest to 100 packets, while the proposed PD+DOB+SP also kept the average queue length within the range of 100 to 101 packets. This means that PID and PD+DOB+SP provided better control performance with respect to target tracking. However, PID provided a larger SD of the queue length than the proposed PD+DOB+SP, which induced a larger jitter in the queue. In contrast, PD+DOB had the smallest SD of the queue length, followed by the proposed PD+DOB+SP. This means that PD+DOB and PD+DOB+SP provided a smaller jitter. However, PD+DOB provided the worst control performance with respect to target tracking. In addition, only CoDel generated buffer overflow among all the methods. The

proposed PD+DOB+SP showed the smallest number of samples with the buffer being empty among all the six methods. It was confirmed that the proposed PD+DOB+SP provided the best control performance with no occurrence of buffer overflow and the least occurrence of empty buffer.

The proposed PD+DOB+SP showed the shortest rise time, followed by PID and PD+DOB. The PID showed the shortest response delay time, followed by the proposed PD+DOB+SP. It was confirmed that PID and the proposed PD+DOB+SP provided better responsiveness than the other four methods.

Therefore, the proposed PD+DOB+SP was proven to be the best from the perspectives of not only network performance but also control performance with respect to target tracking and responsiveness. This indicates that the implementation of the proposed PD+DOB+SP method will be beneficial for the bottleneck routers. One of the technical difficulties of implementing PD+DOB+SP to the router is the computation cost of it compared to the other methods. The calculations done in all six methods consist of the standard four arithmetic operations. In addition, the amount of calculations is nearly the same amongst all six methods, meaning that the calculation cost would not vary significantly based on which methods to implement. On the other hand, while RED, CoDel, PIE, and PID methods only need a few variables to be stored, PD+DOB method needs to store some certain length of variable array to record the queue lengths in the past, since the DOB has an artificial delay function inside. Moreover, the proposed PD+DOB+SP needs two of the same arrays to be stored since the SP also has an artificial delay function. This will make the proposed PD+DOB+SP consuming more memory of the router compared to the other five methods, making the computation cost higher. However, since the queue length is a simple integer, it is safe to say that the increased computation cost at the router would not give a notable impact on the computational load at the router, considering the performance of the current routers. If every number stored in the arrays is assumed to have an 8-byte data size (which is the size of a double-precision floating point format variable) each and the artificial delay is 1 s, the total amount of additional memory needed is about 16 KB under

the control period of 0.001 s. That additionally required memory size would be nearly negligible in the modern era. From these considerations, it can be said that the proposed PD+DOB+SP method can be implemented with no notable increase in computation costs.

In the following sections, the evaluation of link utilization is mainly presented, which is defined as the ratio of throughput to bottleneck link capacity, since the link utilization is the best parameter representing the network performance of AQM. PID, PD+DOB, and PD+DOB+SP are compared since RED, CoDel, and PIE showed worse results than PID, PD+DOB, and PD+DOB+SP with respect to network and control performances.

4.3.4 Changing the Bottleneck Link Capacity

Figure 4-15 shows the link utilization ratio of the three methods when the bottleneck link capacity C and the nominal bottleneck link capacity model C_{n} were changed simultaneously from 20 to 140 Mbps. The maximum value of C was set considering the throughput upper bound of the TCP flow. A single TCP session can only send up to 20 packets at once owing to the limitation of the maximum window size shown in Table 4-1. Because the sender host starts sending the next packet after the ACK is returned from the receiver hosts, it takes an entire RTT to start sending the second set of 20 packets after the first. In other words, the maximum throughput of a single TCP session in packets/s unit can be derived by dividing this maximum window size by the RTT. The theoretical maximum throughput of the system can be obtained by converting the unit to Mbps and multiplying by the number of TCP sessions. The theoretical maximum throughput of the system can be calculated from the maximum window size $wnd_{\max}$, packet size, number of TCP sessions N, and actual RTT R, as shown in (4.16)

$$\frac{wnd_{\max} * packetsize * 8 * N}{R\ [\mathrm{s}]} \qquad [\mathrm{bps}]. \tag{4.16}$$

By using the parameters shown in Table 4-1 and (4.16), the possible maximum total throughput can be approximated to 166 Mbps. This value decreases even when the queue exists, due to

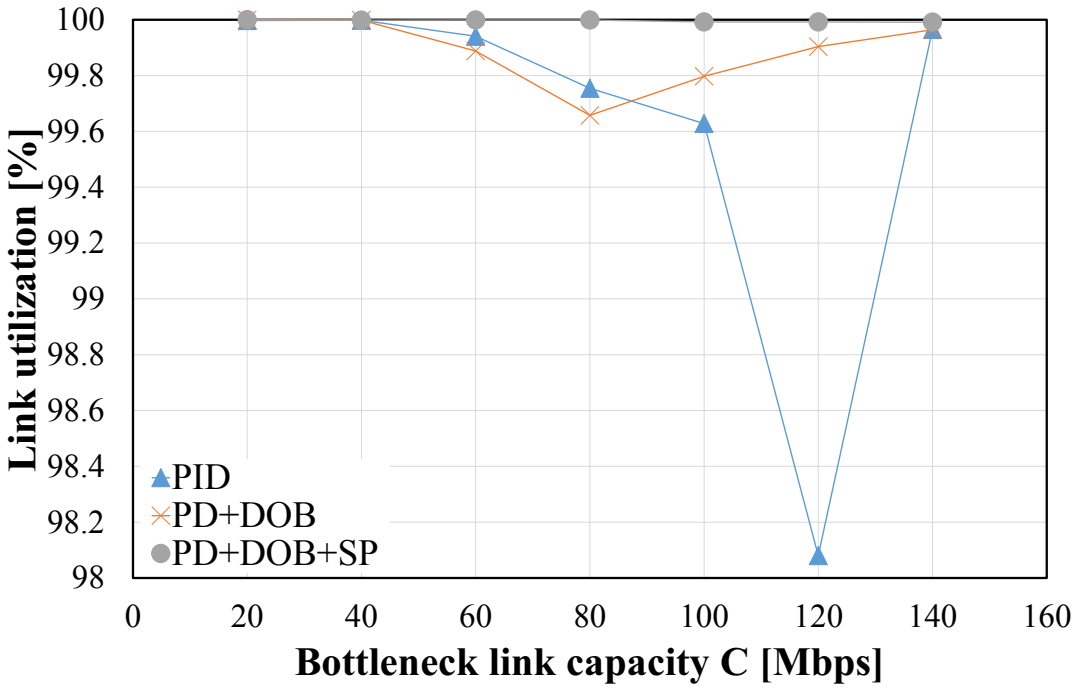


Fig. 4-15 Simulation results when C was changed with matching model.

the queueing delay. Thus, the upper bound value of C was set to 140 to ensure that the system would not surpass the abovementioned throughput limit.

Figure 4-16 shows the link utilization ratio when C was changed while C_n was maintained at the nominal value of 100 Mbps. This means that the model comprising the controller, DOB, and SP differs in value from the actual network. Hereon, this is called "changing the parameter with model mismatch."

The link utilization generally drops as C rises because the inertia of the system increases. This relationship can be confirmed by referring to (2.22) and (2.23). However, at a certain point, this relationship reverses and link utilization starts to increase, as implied by the results of PID and PD+DOB shown in Figs. 4-15 and 4-16. This is because C becomes sufficiently large for the congestion to reduce. On the other hand, the proposed PD+DOB+SP maintains nearly maximum link utilization ratio at any given value of C, demonstrating its time-delay compensation capability, even with mismatching model value of C_n.

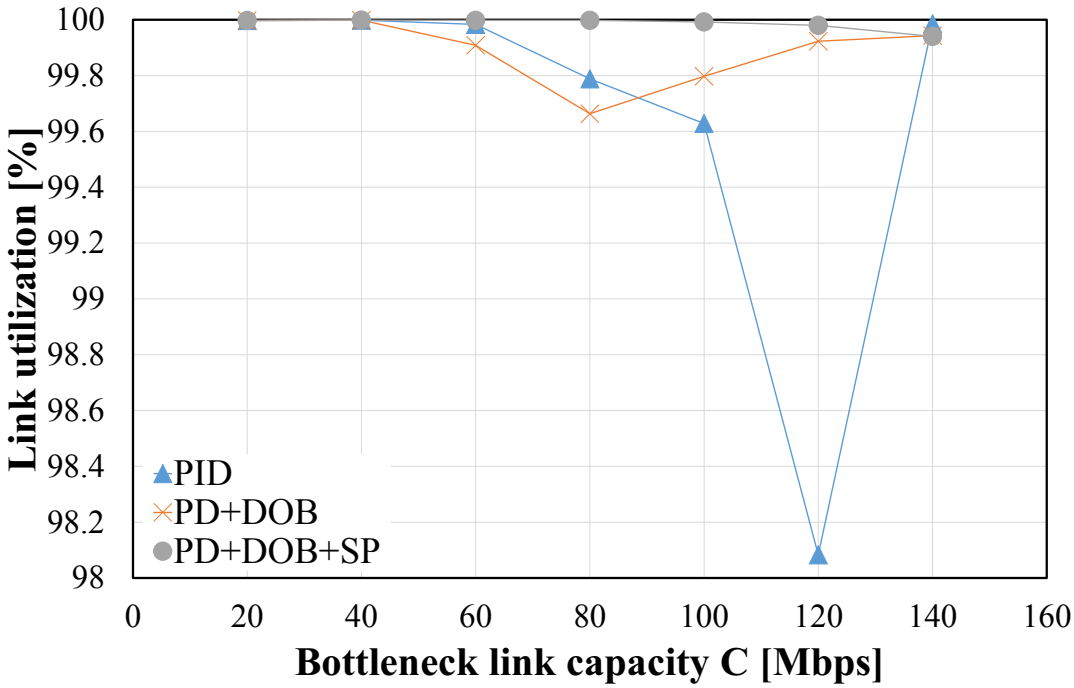


Fig. 4-16 Simulation results when C was changed with model mismatch.

4.3.5 Changing the Number of TCP Sessions

Figure 4-17 shows the link utilization ratio when the number of TCP sessions N and the nominal number of TCP sessions N_n were changed simultaneously from 80 to 200. Figure 4-18 shows the link utilization ratio when N was changed while N_n was maintained at the nominal value of 100. The lower bound of the value of N was set to 80 owing to its maximum throughput. By using (4.16), the maximum throughput when $N = 60$ was calculated to be 99.84 Mbps, which is smaller than C. Thus, N was set to values higher than 60 to ensure that the queue would be generated.

The link utilization generally drops as N drops because of the enlarging inertia of the system. This relationship can be confirmed by referring to (2.22) and (2.23). However, at a certain point, this relationship reverses and the link utilization increases, as implied by the results of PID and PD+DOB shown in Figs. 4-17 and 4-18. This is because congestion is less likely to occur when N is sufficiently small. On the other hand, the proposed PD+DOB+SP maintains

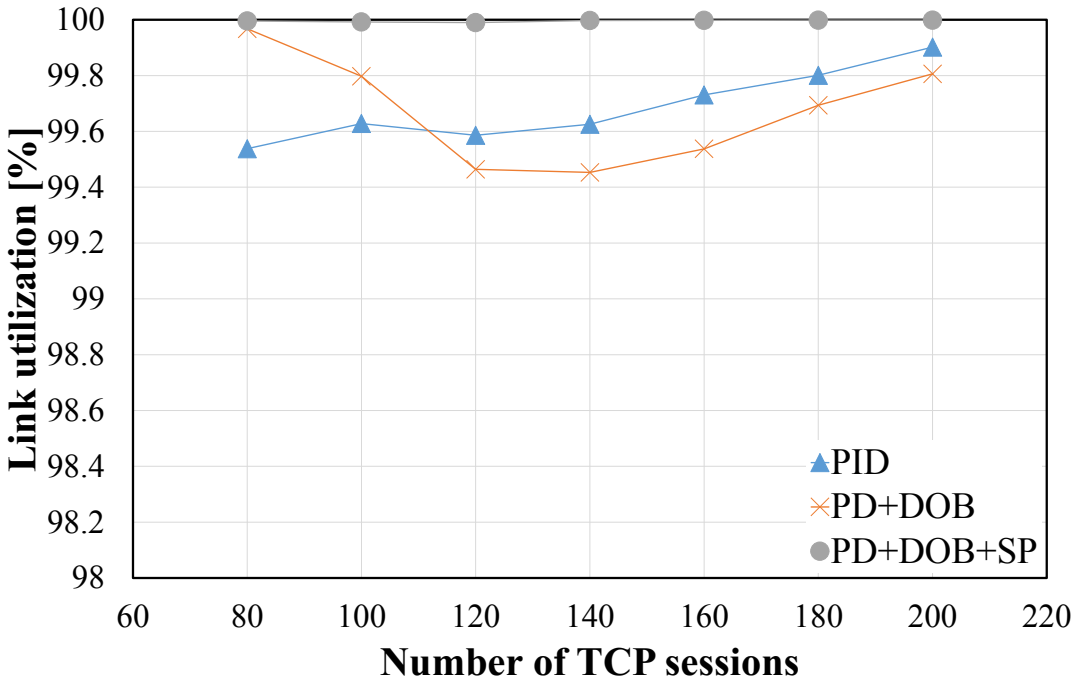


Fig. 4-17 Simulation results when N was changed with matching model.

nearly maximum link utilization ratio at any given value of N, demonstrating its time-delay compensation capability, even with mismatching model value of N_{n}.

4.3.6 Changing the RTT

Figure 4-19 shows the link utilization ratio when the propagation delay T_{p} and nominal RTT delay model R_{n} were changed simultaneously from 20 to 140 ms. The alteration of T_{p} was achieved by changing the value of R_{s}^{i} $(i = 1, 2, \cdots, N)$. Figure 4-20 shows the link utilization ratio when T_{p} was changed while R_{n} was maintained at the nominal value of 100 ms. The upper bound of the value of T_{p} was set to 140 ms owing to its maximum throughput limit. When $T_{\mathrm{p}} = 160$, the maximum throughput was calculated to be approximately 104 Mbps by using (4.16). At this point, the maximum throughput limit is greater than C although this is because the effect of queueing delay is ignored. If there are 100 packets of queue when $C = 100$ Mbps, the queueing delay would be approximately 8 ms. This will lengthen the total RTT, increasing

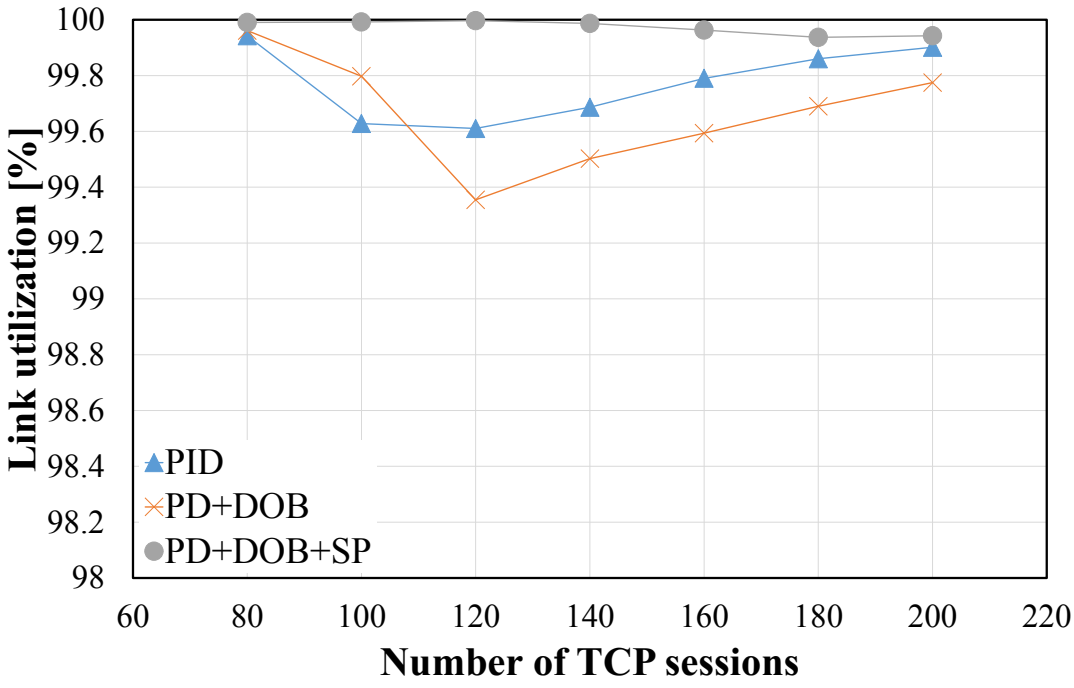


Fig. 4-18 Simulation results when N was changed with model mismatch.

the denominator of (4.16) and lowering the maximum throughput to 99.04 Mbps. Thus, T_p was set to values lower than 160 ms to ensure that adequate queue would be generated to discuss the queue control performance.

As Figs. 4-19 and 4-20 show, the proposed PD+DOB+SP maintains the highest link utilization ratio among the three methods under any given value of T_p. PD+DOB tends to generally deliver better performance than PID by compensating the time delay element with the DOB, but its compensation effect appears less effective than that of PD+DOB+SP.

4.3.7 Mixture of UDP Flows

Figure 4-21 shows the link utilization ratio of all TCP sessions when UDP flows coexist. It was assumed that the UDP flows represent the applications and services, such as video and voice, unresponsive to congestion control. The UDP flows can be considered as disturbance for the system. The UDP flows were implemented in the same manner as the TCP sessions shown in

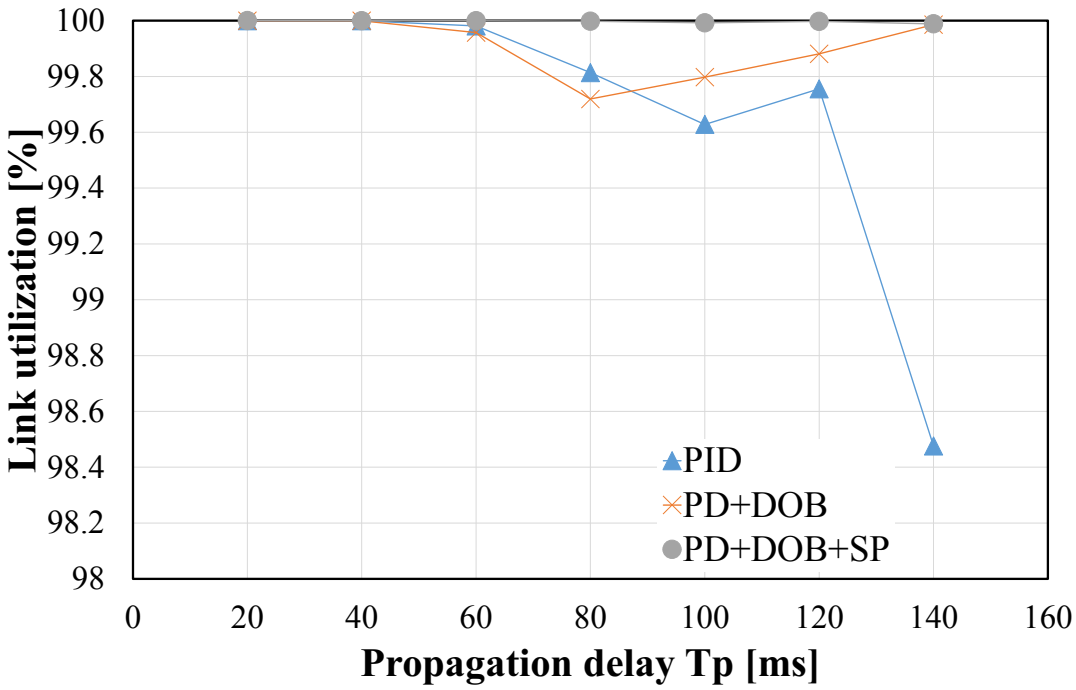


Fig. 4-19 Simulation results when T_p was changed with matching model.

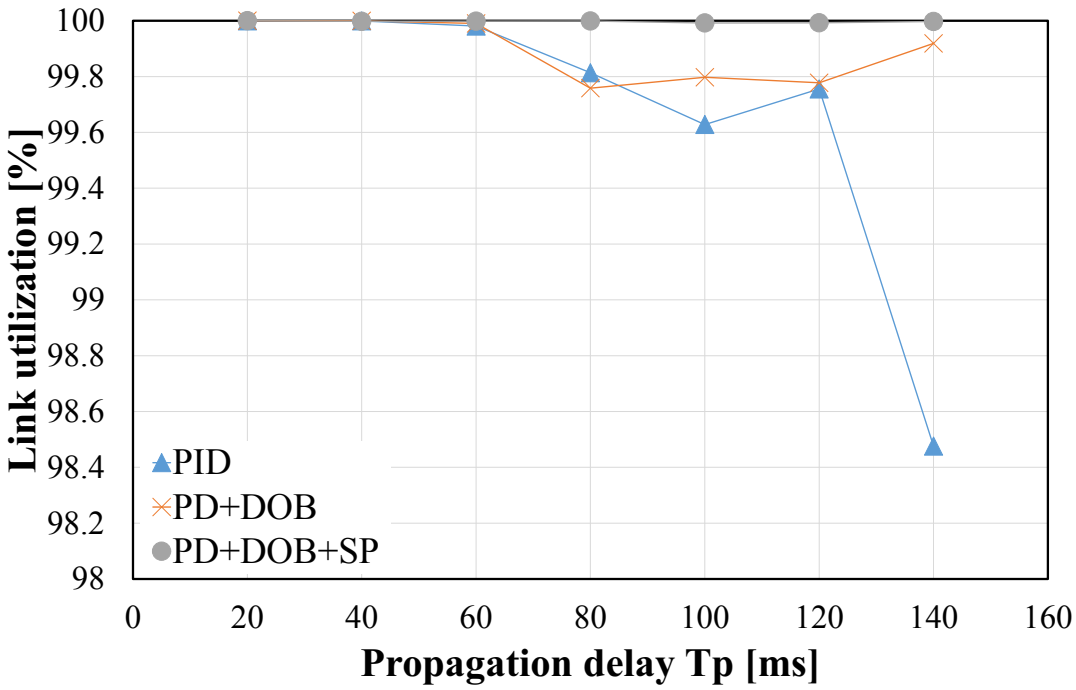


Fig. 4-20 Simulation results when T_p was changed with model mismatch.

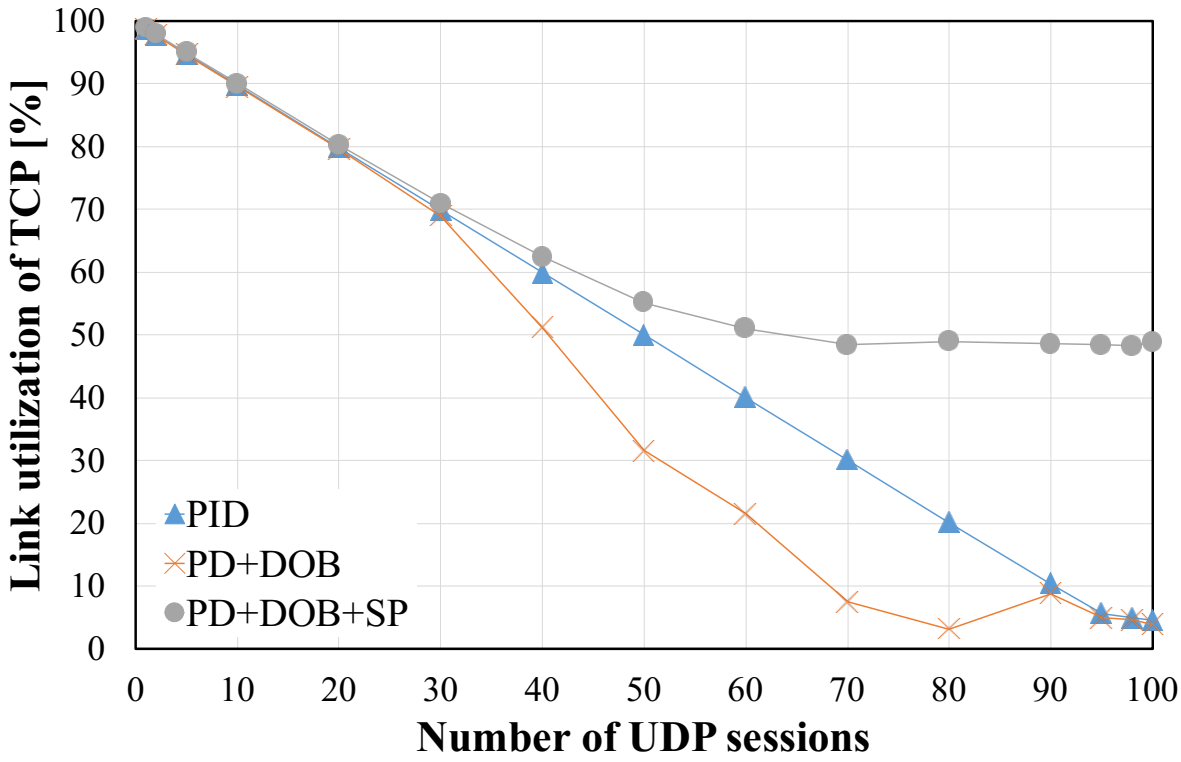


Fig. 4-21 Simulation results when UDP flows coexist.

Fig. 4-12, maintaining a simple dumbbell shaped network topology. Each UDP flow utilized in the simulation attempts to send the packets at the constant bitrate of 1 Mbps. Thus, for example, if there are 10 UDP sessions coexisting, the UDP session sources would attempt to send the total load of 10 Mbps to the bottleneck link router. The simulations were conducted with 1, 2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 95, 98, and 100 UDP sessions.

Unlike TCP, UDP traffic does not have its sending rate limited by the receiving of a packet loss. In fact, the host sending the UDP flow does not even wait for the ACK signal to return from the receiver host. Thus, the coexistence of the UDP flows creates a different situation when compared with the network without UDP flows. From Fig. 4-21, PID and PD+DOB appear to be affected by the different characteristics of the UDP flows, with the total throughput of the TCP flows being extremely low. On the other hand, the proposed PD+DOB+SP succeeded in maintaining a relatively higher throughput than the other two methods. From this result, it can

be said that the proposed PD+DOB+SP can maintain higher throughput even when UDP flows coexist.

4.4 Summary

In this chapter, the AQM congestion control system which uses a PD controller, DOB, and SP in an integrated manner was proposed. The major novelty of the proposed control system is the accomplishment of the simultaneous implementation of DOB and SP considering the effect of the saturation function. The effectiveness of the proposed method was validated by performing simulations using ns-2. The simulations were performed under multiple setups by changing the parameters that mainly affect the behavior of the TCP/AQM network. The simulation results showed that the proposed PD+DOB+SP method generally achieved the highest throughput when compared with the conventional methods. The simulation results with model mismatch also showed that the proposed method maintained its high throughput, demonstrating simultaneous compensation of time delay and model mismatch. In addition, the proposed method maintained a relatively higher throughput of TCP flows than the two conventional methods, PID and PD+DOB, when UDP flows coexisted in the same network. It was confirmed that the proposed PD+DOB+SP has improved performance when compared with the conventional methods. Future works include the implementation of ECN to improve link utilization and simulations using more complicated scenarios such as those employing multiple bottleneck routers.

Introducing machine learning approaches into AQM is one of the promising solutions to optimize control parameters. If the controller gains, cut-off frequencies, and nominal parameters in the proposed method are adaptively optimized according to network conditions, its performance may be further improved. The real-time optimization of control parameters in the proposed AQM controller is an issue to be addressed.

Chapter 5

Adaptive Target Queue Length Generation for QoS-Aware Control

5.1 Background

AQM based on control theory utilizes a parameter called the target queue length, i.e., a command parameter that the system attempts to keep the actual queue length close to. The specific value of the target queue length has not been discussed enough in the past studies, and its value has been traditionally set to one-half of the buffer size or lower [49, 56, 58]. However, changing its value would affect the performance of the TCP/AQM network.

By fully utilizing the buffer size, the bottleneck router drops smaller quantity of packets, which reduces the number of packet retransmissions of the TCP flows. If the throughput is the same, fewer retransmissions mean more efficient data transmission. This could be achieved by raising the target queue length and the average queue length. If the system utilizes a router that supports deep buffering, there would be nearly infinite data space to buffer packets and no complicated AQM technique would be needed. However, when considering a situation of improving the communication efficiency under a sensor network with low-cost routers,

such a costly router with nearly infinite buffer size would be impossible. Considering the implementation of IoT network routers, the technique of fully utilizing the limited buffer size and raising the data transmission efficiency.

On the other hand, lowering the target queue length and the average queue length is beneficial in terms of lower communication delay. Bufferbloat [105] has been considered to be a problem since the demand for faster communications in terms of lower latency has been considered recently. Shortening the queue length will result in smaller queueing delay, which will be beneficial on some occasions where smaller communication latency is preferred over efficient data transmission.

Na *et al.* [106] and Zhang *et al.* [107] presented a research about tuning the target queue length. These studies focused on controlling the target queue length according to the average packet drop probability and determining whether to aim for low latency or low packet loss ratio based on that information. The methods presented in these studies need the threshold for determining low load or high load, making the system not scalable for a variety of network conditions and withholding a risk of performing undesired behavior, such as aiming for low latency when the higher goodput is the priority.

In this chapter, the novel algorithm which dynamically generates a target queue length for the control-theory based AQM is proposed. The proposed algorithm has two modes; loss-aware mode and delay-aware mode; and which to use is predetermined. In the loss-aware mode, the algorithm attempts to fully utilize the buffer size of the bottleneck router by raising the target queue length, lowering the packet loss ratio. Methods utilizing constant monitor time [73], variable monitor time with an approximation curve plot [74], and variable monitor with dual EMA peak detection method are presented and compared with the conventional method of constant target queue length. In the delay-aware mode, the algorithm attempts to decrease the target queue length, lowering the queueing delay while maintaining the full utilization of the bottleneck link capacity. A proposed method utilizing variable monitor time with dual EMA peak detection method is compared with the conventional method of constant target queue length.

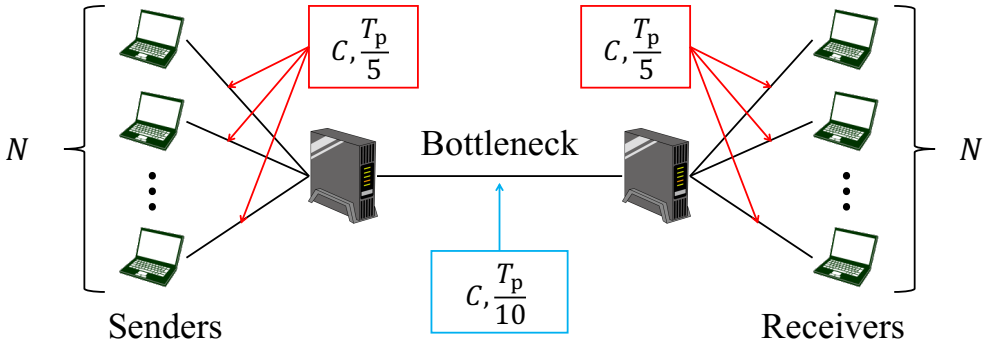


Fig. 5-1 Simulation topology.

The modes to be utilized for each buffer is predetermined and will not be automatically switched dynamically. The proposed dual EMA methods for both modes do not need the network model to function. As the controller for the system, a PID controller is utilized.

5.2 Effects of Change in Target Queue Length

In this section, the systematic behavior of queue length, the advantages of raising and lowering the target queue length, and some technical issues are described. The simulation results shown in this section were obtained using the network topology shown in Fig. 5-1. In addition, all simulations done in this section utilized a PID controller as an AQM controller, where proportional gain K_p, integral gain K_i, derivative gain K_d, and cutoff frequency for pseudo-derivative calculation g_{dif} were set to 900, 700, 55, and 50 rad/s, respectively. Network parameters, i.e., the number of TCP sessions N, bottleneck link capacity C, and propagation delay T_p, are mentioned for each presented simulation.

As shown in chapter 2, TCP/AQM network congestion control system based on control theory utilizes the target queue length q_0 as the control input. In this chapter, this variable is redefined as q_{cmd}, since this is the command input of the control-theory based TCP/AQM network congestion control system. The system attempts to keep the actual queue length close to the value of q_{cmd}. Thus, if q_{cmd} is raised or lowered, the average of the actual queue length

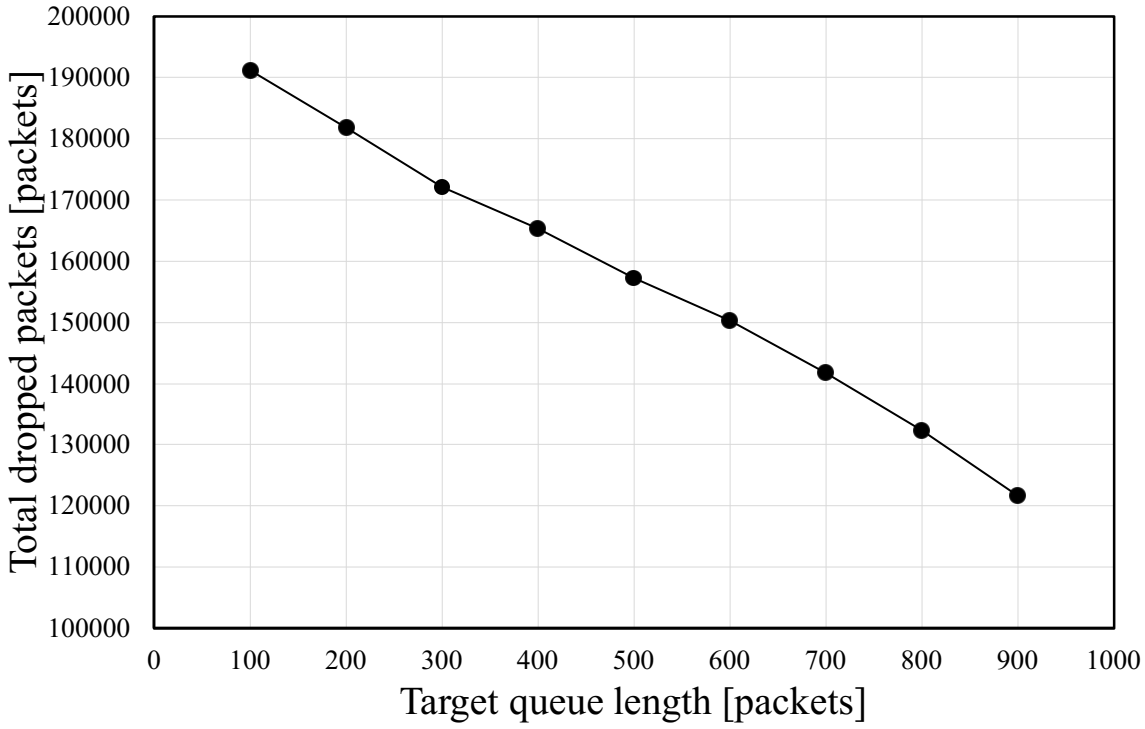


Fig. 5-2 Relationship between q_{cmd} and quantity of dropped packets.

tends to follow its fluctuation. However, the oscillation of queue length is the unavoidable factor in AQM [108]. Because of this unavoidable oscillation, simply observing the average queue length and buffer size of the router and increasing or decreasing q_{cmd} proportionally may induce undesired overshoot or undershoot, resulting in buffer overflow or underflow. The proposed algorithm raises or lowers q_{cmd} to control the average queue length, while considering this fact and avoiding buffer overflow or underflow. The proposed algorithm has two modes; loss-aware mode and delay-aware mode. Each mode raises or lowers q_{cmd}, respectively.

5.2.1 Raising Target Queue Length and Goodput

Figure 5-2 shows the quantity of dropped packets during the simulation duration under different q_{cmd}. These simulations were performed with N, C, T_{p}, buffer size q_{lim}, and simulation duration set to 1000, 100 Mbps, 200 ms, 1000 packets, and 120 s, respectively. The first

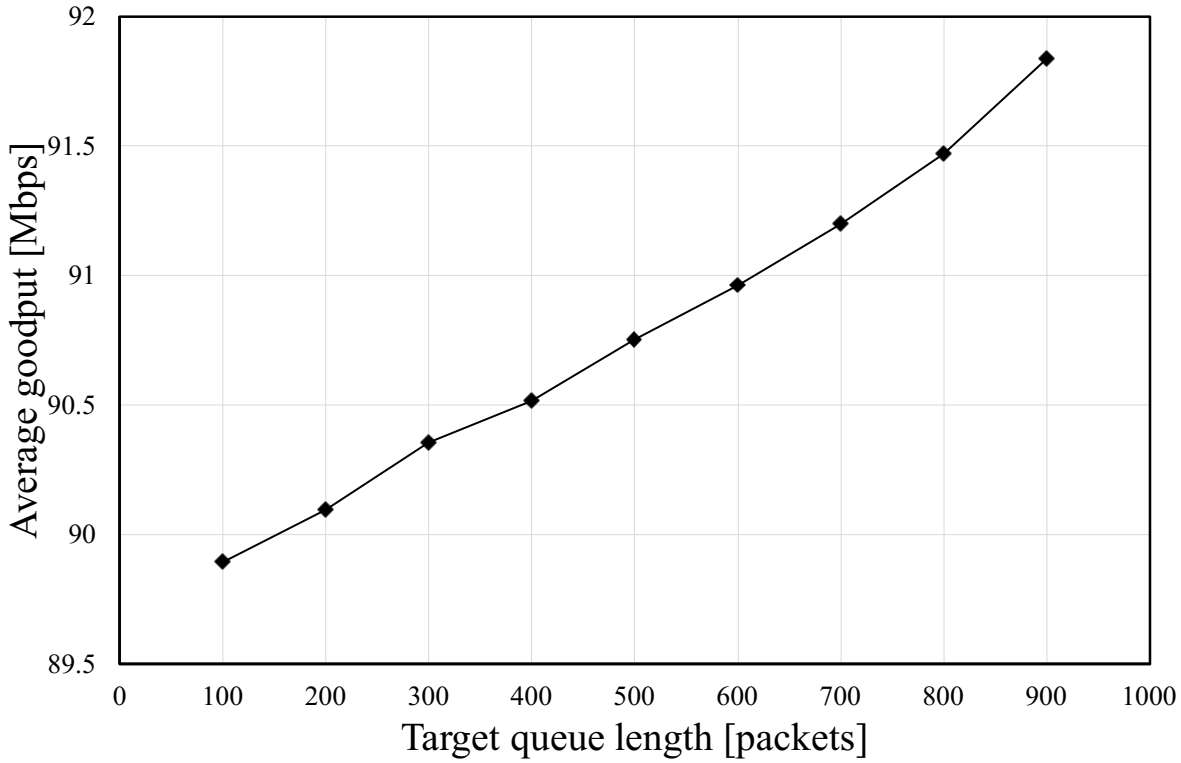


Fig. 5-3 Relationship between q_{cmd} and goodput.

TCP session started their communication at 0 s, and the second session and onwards started their communication every 0.01 s. This was performed in order to avoid unintentional surge induced by a large number of TCP sessions starting communications at the same instant. In this simulation, all the 1000 TCP sessions kept communicating until the simulation ended. Figure 5-2 shows that the system with higher q_{cmd} had a smaller quantity of dropped packets, meaning less packet retransmissions were performed. When packet retransmissions occur, the opportunity to send the same quantity of new packets are lost, which will decrease the amount of the total application data conveyed to the receiver hosts. Thus, when a smaller number of packets are dropped, the more efficient communication can be expected.

Figure 5-3 shows the goodput under different q_{cmd}, under the exact same simulation setups with Fig. 5-2. The goodput is defined as the total amount of packets received by the receiver host per unit time excluding the retransmitted packets. From the result shown in Fig. 5-3, a

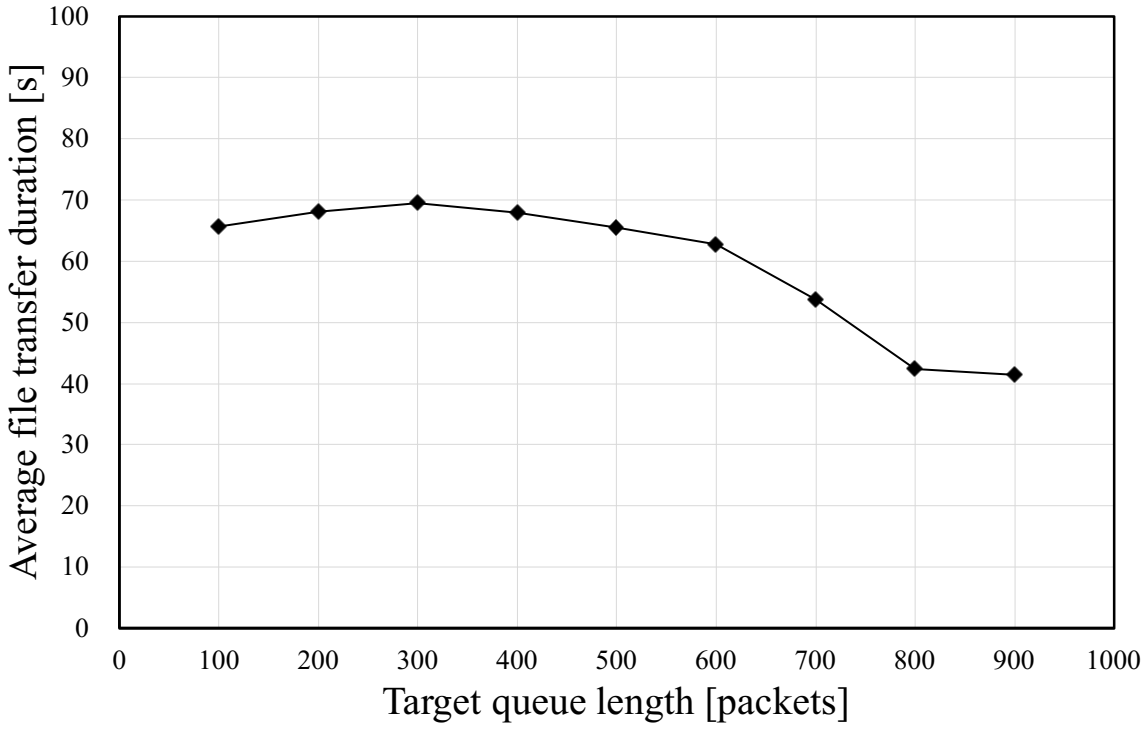


Fig. 5-4 Relationship between q_{cmd} and file transfer duration.

higher q_{cmd} tended to provide a higher goodput, since there were fewer packet retransmissions occurring. This means that setting a higher q_{cmd} will not only efficiently utilize the designed buffer size of the bottleneck router but also increases the utilization of the bottleneck link.

Figure 5-4 shows the average duration needed to transfer files under different q_{cmd}. The simulation setups are mostly same with Figs. 5-2 and 5-3, except that all the 1000 TCP sessions performed a file transmission of 1 MB in size and stopped communicating afterwards. Figure 5-4 shows that a higher q_{cmd} needed shorter duration to send the same quantity of packets to receiver hosts. The transfer duration when q_{cmd} varies from 100 to 300 does not seem to follow this characteristic, but this is assumed to be the result of random discard done by the AQM functioning luckily for q_{cmd} = 100 and 200 under this simulation setups.

In summary, raising q_{cmd} reduces the total amount of dropped packets, which induces fewer packet retransmissions resulting in a higher goodput. The drawback of raising q_{cmd} is that a

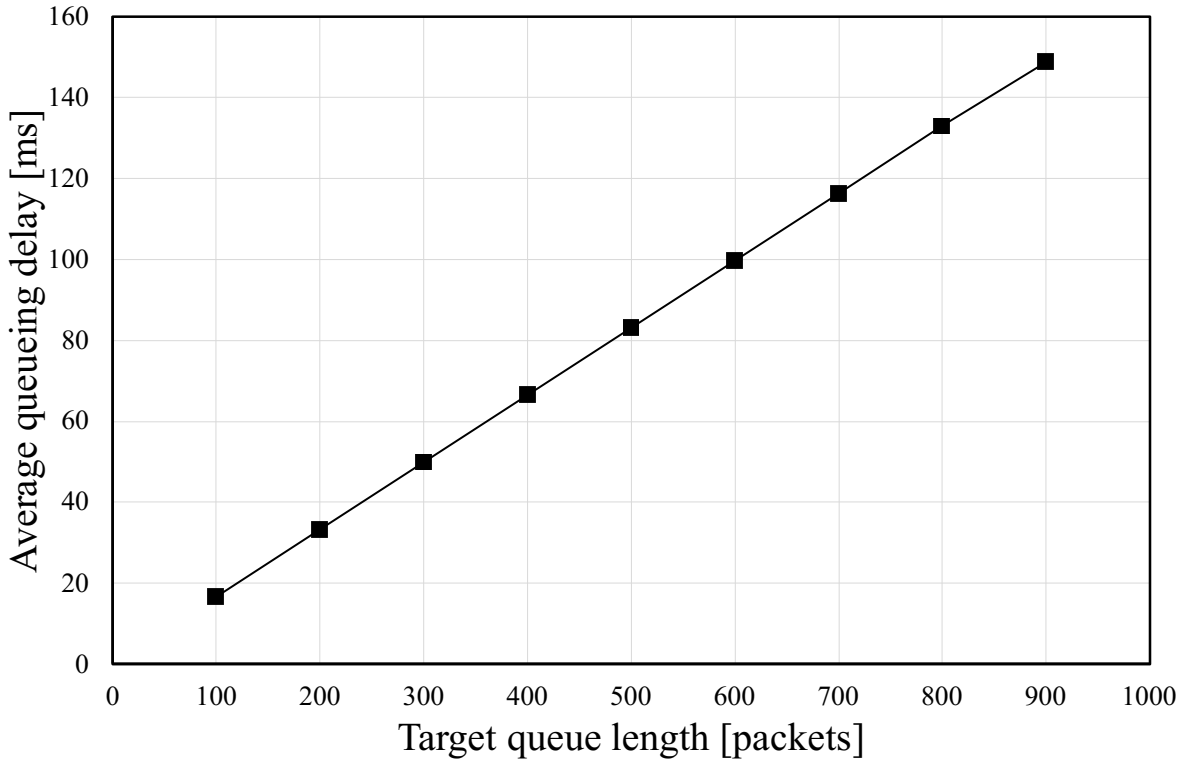


Fig. 5-5 Relationship between q_{cmd} and average queueing delay.

higher average queue length induces a larger queueing delay, which can be detrimental in some cases. Raising q_{cmd} should be beneficial for the communication system which gives preference to a higher goodput over a lower communication latency or has a large propagation delay that the effect of queueing delay is nearly negligible. Raising q_{cmd} may be suitable for online data backup and high-latency networks such as marine and satellite communications.

5.2.2 Lowering Target Queue Length and Queueing Delay

When the queue is created in the buffer of the bottleneck link router, the emergence of queueing delay is inevitable. A large queue length in the buffer may critically increase the RTT of the TCP flows. To reduce the average queueing delay, the average queue length must be lowered, since the queueing delay is changed depending on the queue length and bottleneck link capacity. This can be accomplished by lowering q_{cmd}.

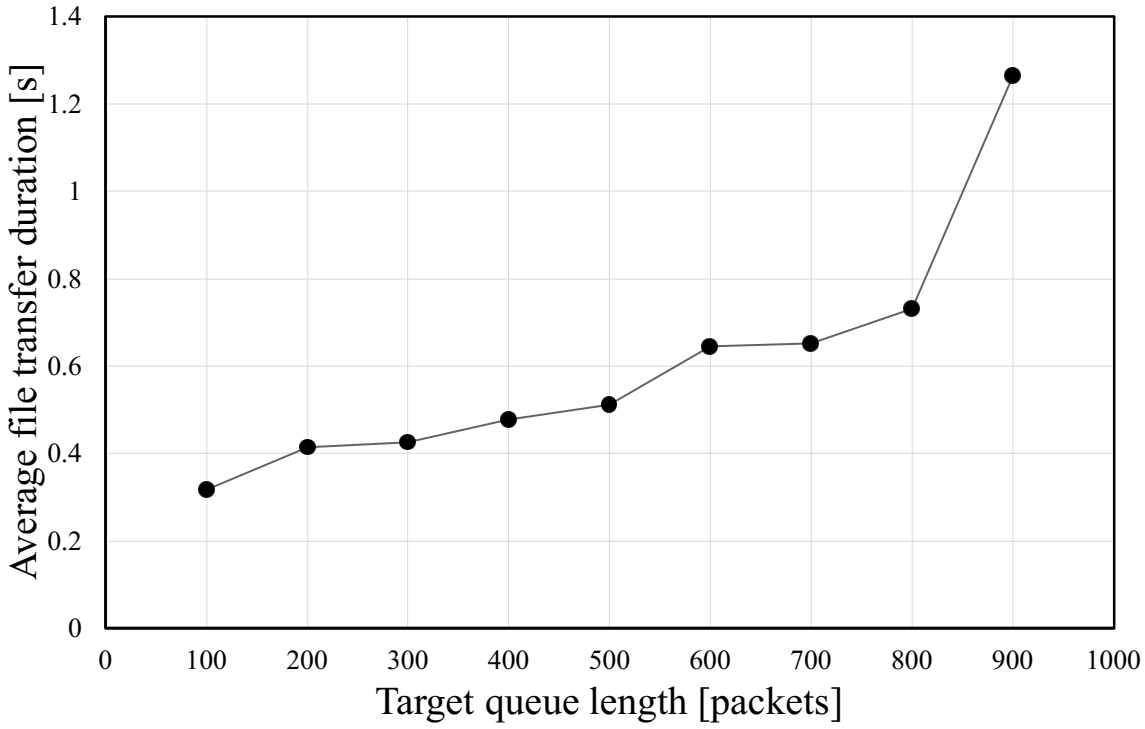


Fig. 5-6 Relationship between q_{cmd} and transfer duration of small-sized files.

Figure 5-5 shows the average queueing delay under different q_{cmd}. These simulations were performed with N, C, T_{p}, q_{lim}, and simulation duration set to 100, 50 Mbps, 50 ms, 1000 packets, and 120 s, respectively. In this simulation, the first 90 TCP sessions were the background traffic flows which kept communicating until the simulation ended, while the other 10 TCP sessions performed a file transmission of 10 KB in size and stopped communicating afterwards. The first background TCP flow session started their communication at 0 s, and the second session and onwards started their communication every 0.01 s. The first data file transmitting TCP flow session started their communication at 30 s, and the second session and onwards started their communication every 5 s. Figure 5-5 shows that the average queueing delay is proportional to the value of q_{cmd}. This is fairly easy phenomenon to comprehend, since the control theory based AQM maintains the queue length close to q_{cmd}; thus, changing q_{cmd} proportionally affects the average queue length, which is proportional to the queueing delay.

The packet arriving at the bottleneck router tends to be sent out faster if the queueing delay is short, compared to when it is long. Figure 5-6 shows the average duration needed to transfer files with different q_{cmd}, under the exact same simulation setups with Fig. 5-5. As shown in Fig. 5-6, the smaller q_{cmd} tends to need less time to transfer a small sized data file of 10 KB. There are two reasons why the relationship is not linear. The first reason is because of the function of AQM selecting the packets to drop at random, not selecting the drop candidate according to the load proportion completely. The second reason is because the plot of $q_{\mathrm{cmd}} = 900$ packets spiked up since the file transmitting TCP sessions became the victim of packet disposal and had to retransmit multiple times, which was caused by the buffer overflow.

In summary, lowering q_{cmd} reduces the queueing delay, resulting in more responsive system and shorter duration for the transmission of small-sized files. The drawback of lowering q_{cmd} is that the packets are more actively dropped to maintain the queue length, which results in lower goodput, as shown in Fig. 5-3. Lowering q_{cmd} should be beneficial for the traffic flow which gives preference to a shorter communication latency over a higher goodput in a long term, has a small propagation delay so the effect of queueing delay is magnified in contrast, or transmitting a very small-sized file consisted of a few packets. Lowering q_{cmd} may be suitable for online live communications and network with UDP traffic coexisting.

5.2.3 Buffer Overflow

When the queue length reaches the buffer size q_{lim} of the router, buffer overflow occurs. This is the situation where the buffer of the router is full ($q(t) = q_{\mathrm{lim}}$), where no more incoming packets can be received by the router; thus, all of them will be dropped. Avoiding the buffer overflow is the original purpose of AQM, since the DropTail queue was used as the basic process at the router. With the DropTail queue, the routers can only detect congestion after buffer overflow, which causes mass packet disposal and global synchronization. In order to deal with these problems, the implementation of AQM for stable queue control was proposed.

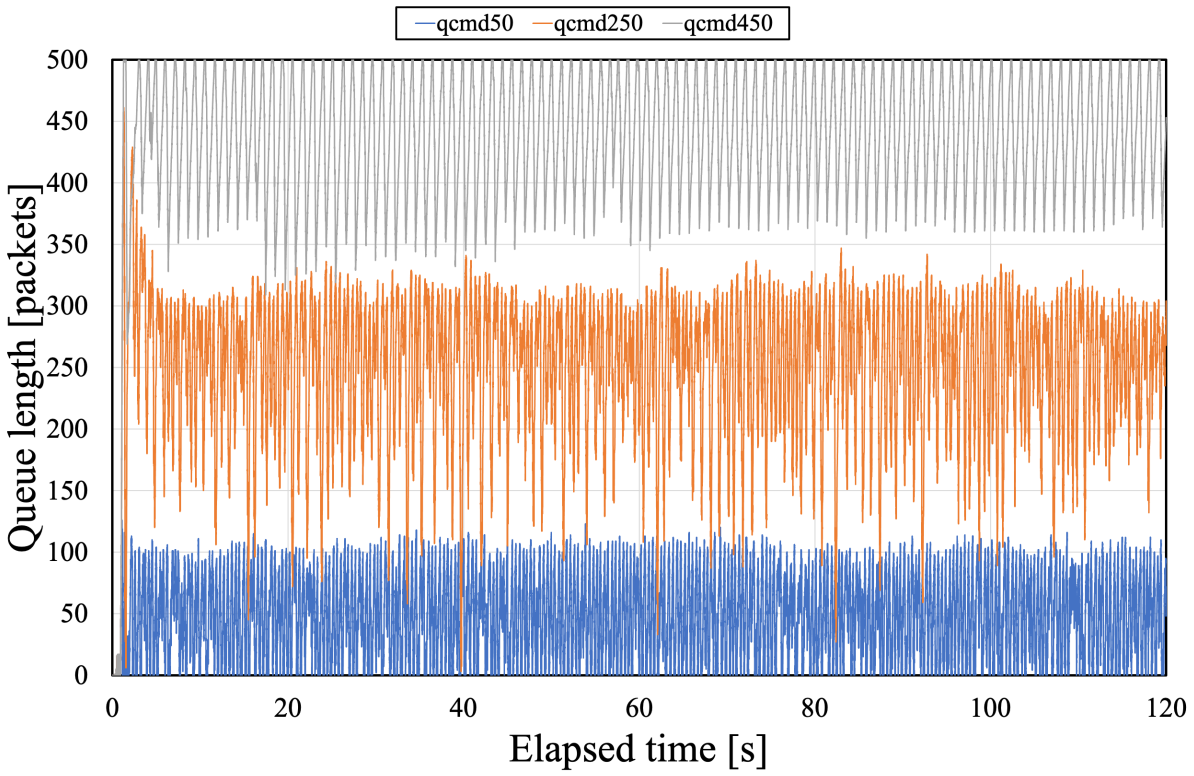


Fig. 5-7 Queue length fluctuation ($q_{\mathrm{cmd}} = 50, 250, 450$).

However, when q_{cmd} is raised to an improperly high value, the buffer overflow may be induced. Figure 5-7 shows the queue length fluctuations when q_{cmd} is set to 50, 250, and 450 packets. These simulations were performed with N, C, T_{p}, q_{lim}, and simulation duration set to 100, 100 Mbps, 100 ms, 500 packets, and 120 s, respectively. The three selected values for q_{cmd} denote low, medium ($= q_{\mathrm{lim}}/2$), and high values, respectively. As shown in Fig. 5-7, when $q_{\mathrm{cmd}} = 450$ packets, the queue length repeatedly reached q_{lim}. The fact that the queue length reaches q_{lim} means there are buffer overflows occurring.

One of the major reasons why the buffer overflow should be avoided is because it induces unfairness amongst the packets to be dropped. Figure 5-8 shows the SD of received packets amongst all TCP receiver hosts, under the exact same simulation setups with Fig. 5-7. As this figure shows, the simulation result when $q_{\mathrm{cmd}} = 450$ packets has a high SD value compared to the other q_{cmd}. SD of received packets indicates how distributed the number of received packets

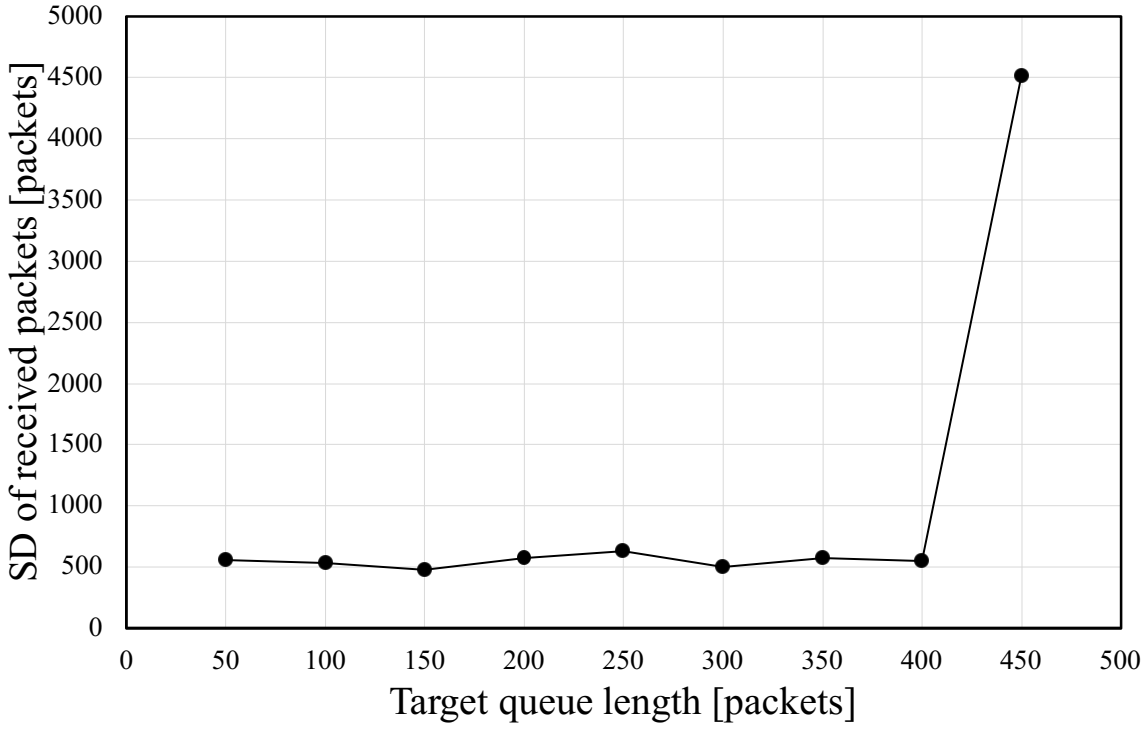


Fig. 5-8 Relationship between q_{cmd} and SD of received packets.

in the simulation duration were amongst all TCP receiver hosts. The higher SD means the total number of received packets of each TCP receiver host differs drastically. Thus, from the result shown in Fig. 5-8, it can be stated that the TCP sessions were very unfairly treated when q_{cmd} = 450 packets. This is caused by the nature of buffer overflow and DropTail procedure dropping the packets. As long as the queue length does not reach q_{lim}, the packet drop procedure is performed based on the calculation of AQM, and the candidate packets to be dropped are selected randomly. However, when the queue length reaches q_{lim}, the DropTail procedure additionally drops packets without any randomness. Thus, the aforementioned unfairness is triggered, resulting in obvious distribution shown in Fig. 5-8.

Considering this fact, the buffer overflow should be avoided in order to achieve a stable congestion control. Avoiding the buffer overflow is a critical factor to consider with the loss-aware mode, since raising q_{cmd} has a risk of raising the maximum queue length up to q_{lim}.

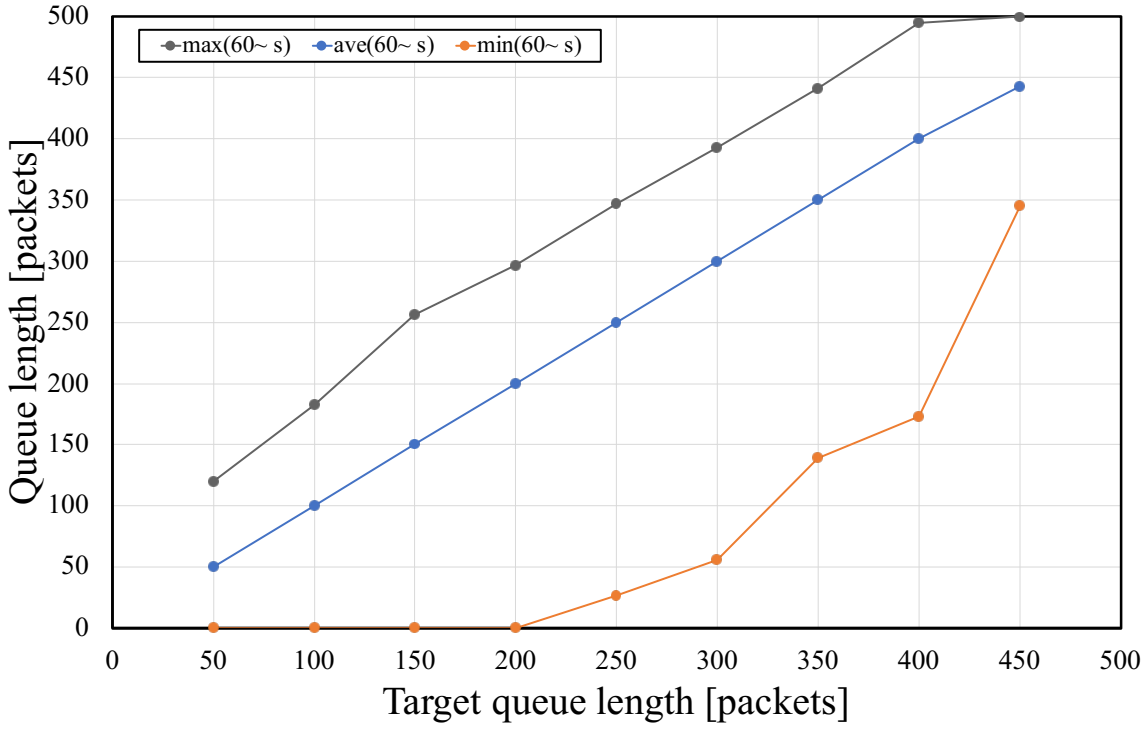


Fig. 5-9 Relationship between q_{cmd} and maximum, minimum, and average queue length.

5.2.4 Buffer Underflow

When there is no queue in the buffer of the bottleneck link router, the bottleneck link is not fully utilized. In such a situation, the throughput would be smaller than the bottleneck link capacity C.

Figure 5-9 shows the maximum, minimum, and average queue length with different q_{cmd}, under the exact same simulation setups with Figs. 5-7 and 5-8. The maximum, minimum, and average queue length were all calculated based on the queue length after 60 s in the simulation duration. As Fig. 5-9 shows, the minimum queue length reached 0 when q_{cmd} was set to 200 packets or lower. Whether the minimum queue length reaches 0 or not is a crucial factor for AQM, since a buffer underflow means there are no packets waiting to be sent through the bottleneck link, which is equal to the non-maximum usage of bottleneck link capacity.

Figure 5-10 shows the throughput with different q_{cmd}, under the exact same simulation setups

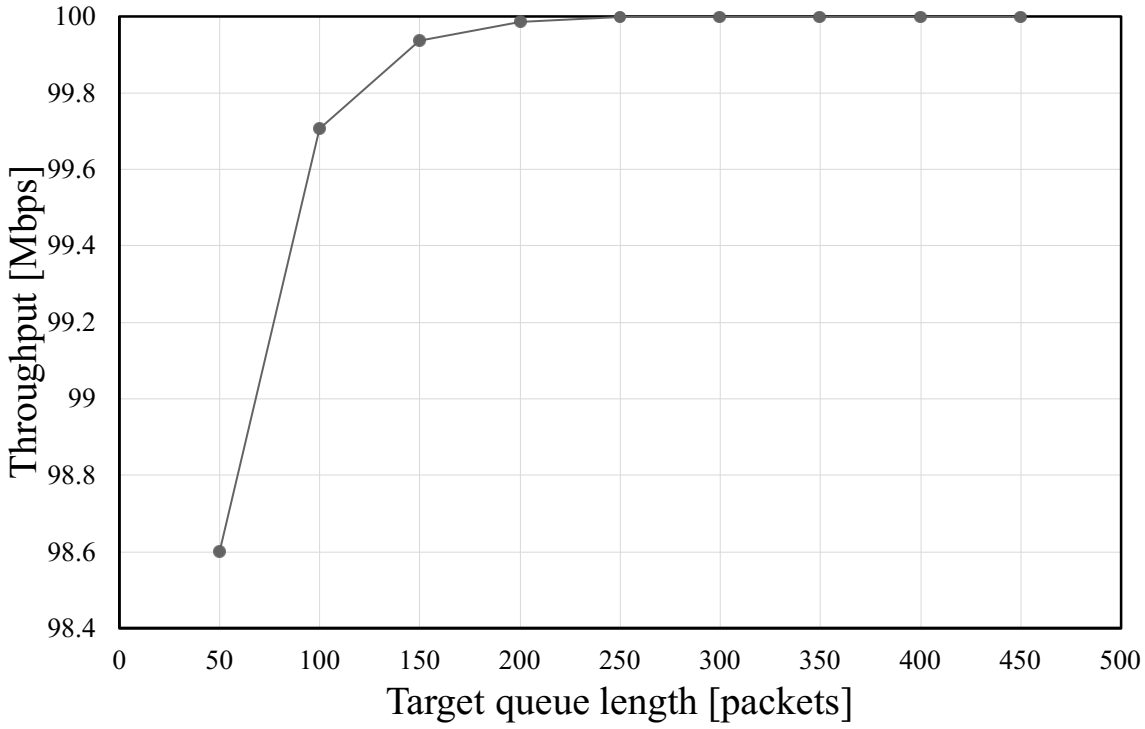


Fig. 5-10 Relationship between q_{cmd} and throughput.

with Figs. 5-7, 5-8, and 5-9. As this figure shows, the throughput of the system was equal to 100 Mbps, which is the designed value of C, when q_{cmd} is set to 250 packets or higher. However, the throughput started dropping from q_{cmd} = 200 packets and kept dropping at an accelerating rate as q_{cmd} lowered. The values of q_{cmd} that recorded a throughput lower than 100 Mbps are identical to the ones that had buffer underflow shown in Fig. 5-9.

This relationship of low throughput and buffer underflow must be taken into account for designing the algorithm. Avoiding the buffer underflow is a critical factor to consider with the delay-aware mode, since lowering q_{cmd} has a risk of lowering the minimum queue length down to 0.

5.2.5 Independent Stable State

TCP session changes their window size in order to efficiently utilize the link capacity. Generally, the window size is increased when there are no packet drops detected and is decreased when the packet drop is detected. However, there are a limit to the window size, defined as maximum window size $wnd_{\max}$. This means that the maximum amount of data a single TCP communication sender can send at once before receiving any ACK packets can be derived by (5.1),

$$wnd_{\max} * packetsize * 8 \qquad [\text{bits}], \tag{5.1}$$

where $packetsize$ denotes the data size of a single packet including the header information in bytes. Dividing (5.1) by the RTT R gives the maximum throughput a single TCP session can accomplish theoretically. Thus, the theoretical maximum throughput of the system can be calculated as shown in (4.16).

If the maximum throughput calculated by (4.16) becomes equal to C, the system becomes completely stable with a 100% link utilization and no packet drops. Under some certain scenarios, the queue length may reach a value where $R(t)\left(\fallingdotseq \frac{q(t)}{C(t)} + T_{\mathrm{p}}\right)$ satisfies the above-mentioned conditions.

Figure 5-11 shows the queue length fluctuation when q_{cmd} is set to 100, 250, and 400 packets. These simulations were performed with N, C, T_{p}, q_{lim}, and simulation duration set to 40, 50 Mbps, 100 ms, 500 packets, and 120 s, respectively. With this setup, the queue length became stable at a value of just below 200 packets when q_{cmd} = 250 and 400 packets. This is because the queueing delay at the queue length just below 200 packets satisfies the aforementioned relationship between the maximum throughput and C. In such a scenario, the AQM controller cannot raise the queue length higher than that value, and the queue length becomes stable regardless of the target queue length. This state is defined as “independent stable state” in this thesis.

The existence of this independent stable state must be taken into consideration in order to

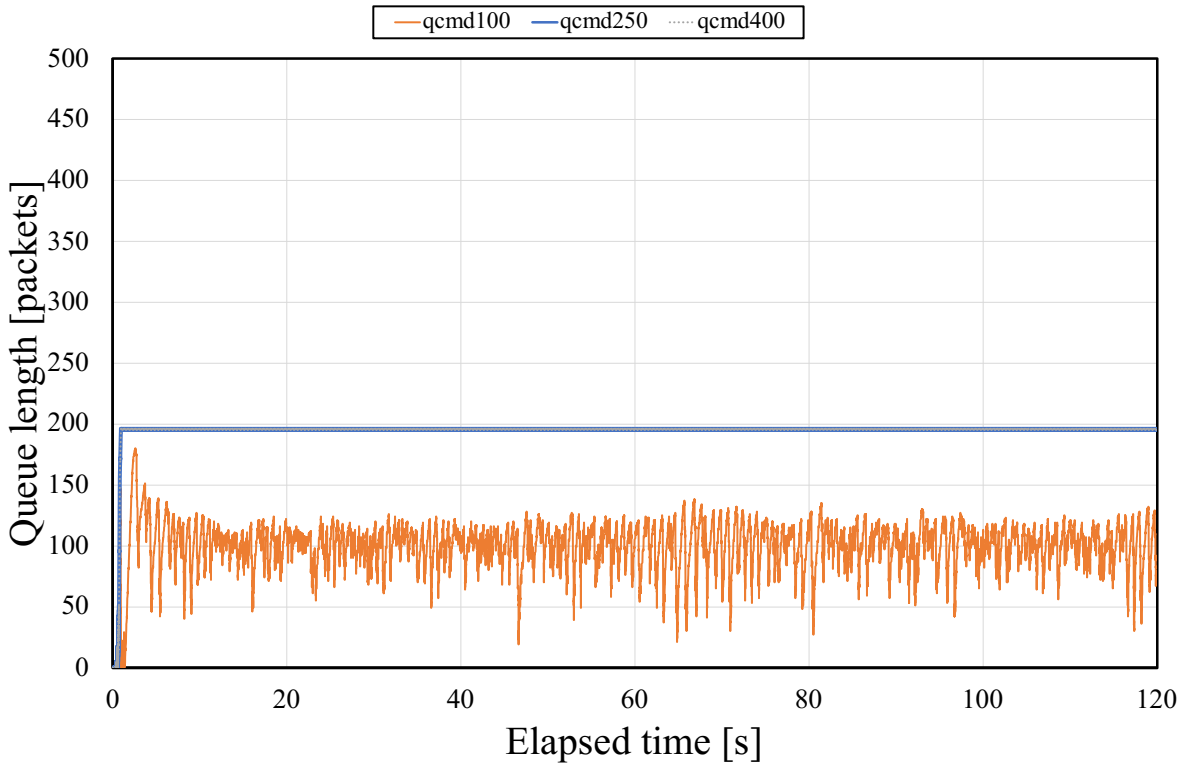


Fig. 5-11 Behavior of queue length in independent stable state.

design the algorithm, especially in the loss-aware mode. When the system is in the independent stable state, there are no packets dropped by the system while the bottleneck link capacity being fully utilized. This situation itself is an ideal state desired by the loss-aware mode algorithm, so the algorithm should attempt to stay in the independent stable state. However, if the new TCP sessions join the network, their load would break the independent stable state and the queue length would start to rise again. In such a scenario, if the value of q_{cmd} is not intentionally maintained to some certain value, the system may induce buffer overflow multiple times. The same situation can happen when the system has sudden increment of TCP sessions while the buffer is completely empty.

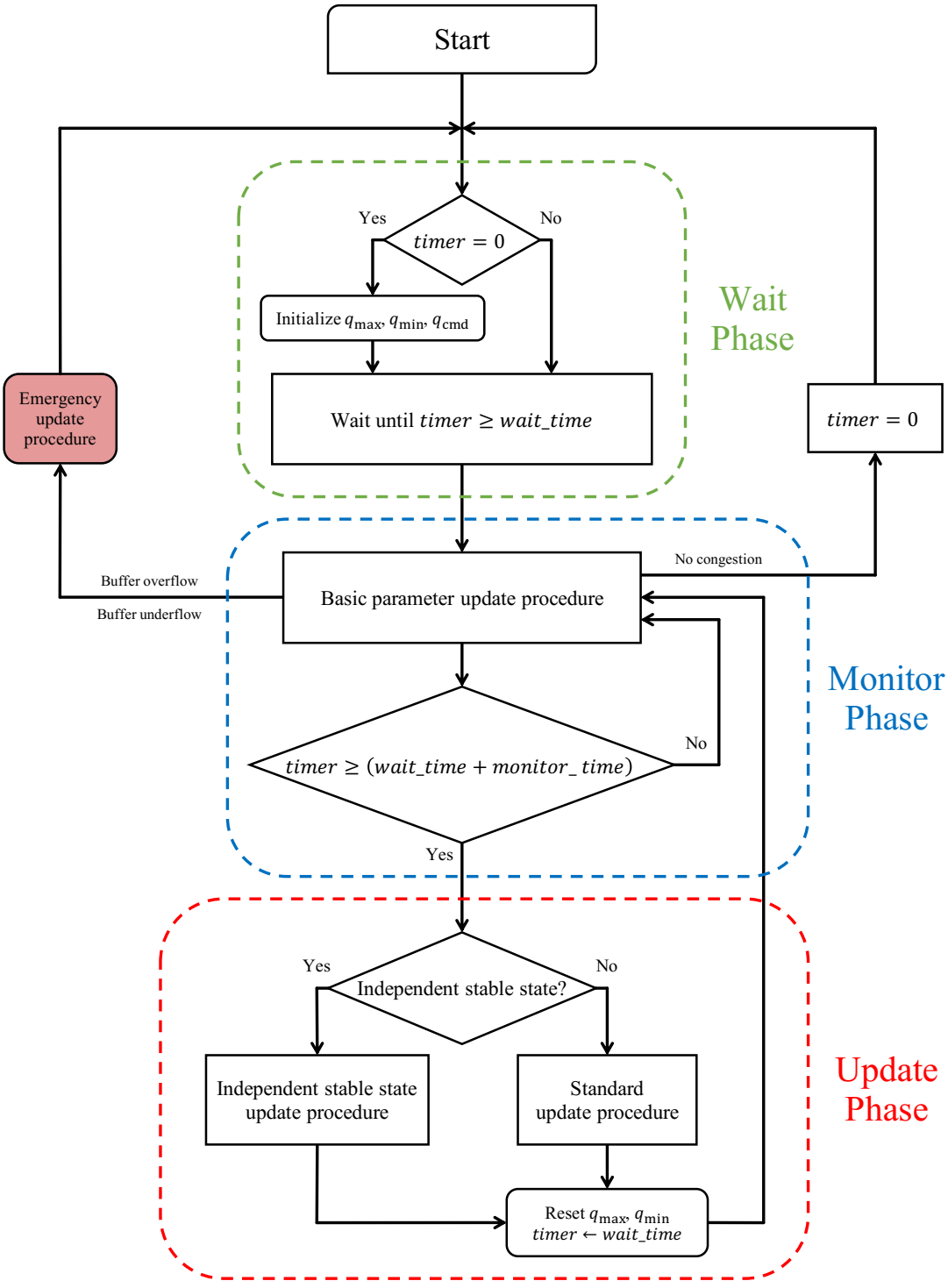


Fig. 5-12 The diagram of the algorithm flow.

5.3 Adaptive Target Queue Length Generation

In this section, the proposed algorithm which controls q_{cmd} dynamically is explained. The overall flow of the algorithm is shown in Fig. 5-12. The algorithm first starts from the "Wait Phase", and transits to "Monitor Phase" and "Update Phase". Through the multiple procedures, the value of q_{cmd} is dynamically controlled.

The algorithm has two modes, i.e., the Loss-aware mode and Delay-aware mode. The Loss-aware mode attempts to raise the q_{cmd} from its initial value while avoiding buffer overflow. The Delay-aware mode attempts to lower the q_{cmd} from its initial value while avoiding buffer underflow. The mode to be utilized for a router is predetermined and will not be automatically switched dynamically. The procedures shown in Fig. 5-12 may differ between the two modes, which would be described later.

The algorithm runs in the same sampling period with the controller. In this thesis, all the sampling periods are set to 0.001 s.

The parameters used in the algorithm are as follows:

q_{th} ≐ Threshold queue length [packets],

q_{guard} ≐ Guard queue length [packets],

q_{max} ≐ Maximum queue length [packets],

q_{min} ≐ Minimum queue length [packets],

$timer$ ≐ Elapsed time count [samples],

$wait\_time$ ≐ Wait phase duration [samples],

$monitor\_time$ ≐ Monitor phase duration [samples],

$monitor\_time_{\text{min}}$ ≐ The minimum $monitor\_time$ [samples],

EMA_1 ≐ EMA of queue length (initialized as 0) [packets],

EMA_2 ≐ EMA of EMA_2 (initialized as 0) [packets],

ind_flag ≐ Independent stable state flag (initialized as 0),

emg_flag ≐ Emergency update flag (initialized as 0).

Some parameters were not utilized for all the proposed algorithms.

5.3.1 Wait Phase

The algorithm starts from this phase. This phase is designed to prevent the algorithm from updating q_{max}, q_{min}, and q_{cmd}. If the algorithm enters this phase with $timer = 0$, the following parameter initializations are performed;

$$q_{\text{max}} = 0, \tag{5.2}$$

$$q_{\text{min}} = q_{\text{lim}}, \tag{5.3}$$

$$q_{\text{cmd}} = q_{\text{lim}}/2. \tag{5.4}$$

This phase last until $timer$ reaches the duration predefined by the parameter $wait\_time$. If $timer \geq wait\_time$, the algorithm enters the "Monitor Phase".

5.3.2 Monitor Phase

After $wait\_time$ has passed, the algorithm enters this phase. In this phase, the "Basic parameter update procedure" is performed at every sampling time. This is continued until the duration defined by the parameter $monitor\_time$ passes, i.e., $timer \geq wait\_time + monitor\_time$. After that, the algorithm enters the "Update Phase".

If a buffer overflow was detected in Loss-aware mode, the algorithm performs the Emergency update procedure. If a buffer underflow was detected in Delay-aware mode, the algorithm performs the Emergency update procedure. The detail about the Emergency update procedure will be described at section 5.3.4.

5.3.2.1 Basic Parameter Update Procedure

This procedure updates the value of $q_{\max}$ and $q_{\min}$ at every sampling time. The current queue length q is observed and compared with that of $q_{\max}$ and $q_{\min}$. If $q > q_{\max}$, the algorithm substitutes q for $q_{\max}$. If $q < q_{\min}$, the algorithm substitutes q for $q_{\min}$. This procedure combined with the Monitor Phase duration is shown in Algorithm 1. The value of $q_{\max}$ is utilized in the Loss-aware mode, and the value of $q_{\min}$ is utilized in the Delay-aware mode.

Algorithm 1 Basic parameter update procedure

```
while timer < (wait_time + monitor_time) do
  if q > q_max then
    q_max ← q.
  end if
  if q < q_min then
    q_min ← q.
  end if
end while
```

5.3.2.2 Queue Length Monitor Time

The duration of the Monitor Phase is defined as $monitor\_time$, and the Basic parameter update procedure is performed at every sampling time during this period. There are multiple methods of defining $monitor\_time$. Three methods, i.e., Constant Monitor Time (CMT) method [73], Variable Monitor Time with Approximation Curve (VMTwAC) method [74], and Variable Monitor Time with Dual EMA (VMTwDEMA) method, are proposed in this chapter. The CMT method and VMTwAC method were designed only to function with Loss-aware mode, while VMTwDEMA method functions with both the Loss-aware mode and Delay-aware mode.

5.3.2.2.1 Constant Monitor Time The first method, i.e., CMT method, is the very basic method of predetermining $monitor\_time$ [73]. The value of $monitor\_time$ was defined to be 10 s, i.e., 10000 samples. This value was selected to make the Monitor Phase long enough so its duration would be longer than the queue length oscillation period. This duration was utilized regardless of the conditions of the TCP/AQM network; thus, having no scalability amongst different network conditions.

5.3.2.2.2 Variable Monitor Time with Approximation Curve The second method, i.e., VMTwAC method, utilizes an approximation curve to estimate a queue oscillation period [74]. As it can be seen in Figs. 5-7 and 5-11, the queue length tends to oscillate and its oscillation period is heavily affected by the delay of the TCP/AQM network, which is RTT R.

In order to obtain the relationship between the queue oscillation period and RTT, numerous simulations were performed with N, C, q_{lim}, and simulation duration were set to 100, 50 Mbps, 2000 packets, and 60 s, respectively. The value of T_{p} and q_{cmd} was changed in the range of 20 to 200 ms and 200 to 1000 packets, respectively. From the simulation results, the approximation curve shown in Fig. 5-13 was obtained. The calculated RTT on the horizontal axis of Fig. 5-13 is the estimated RTT calculated from q_{cmd} and the nominal model of RTT R_{n}, which is equal to the actual propagation delay T_{p} if it is assumed to have no modeling errors. The equation of the

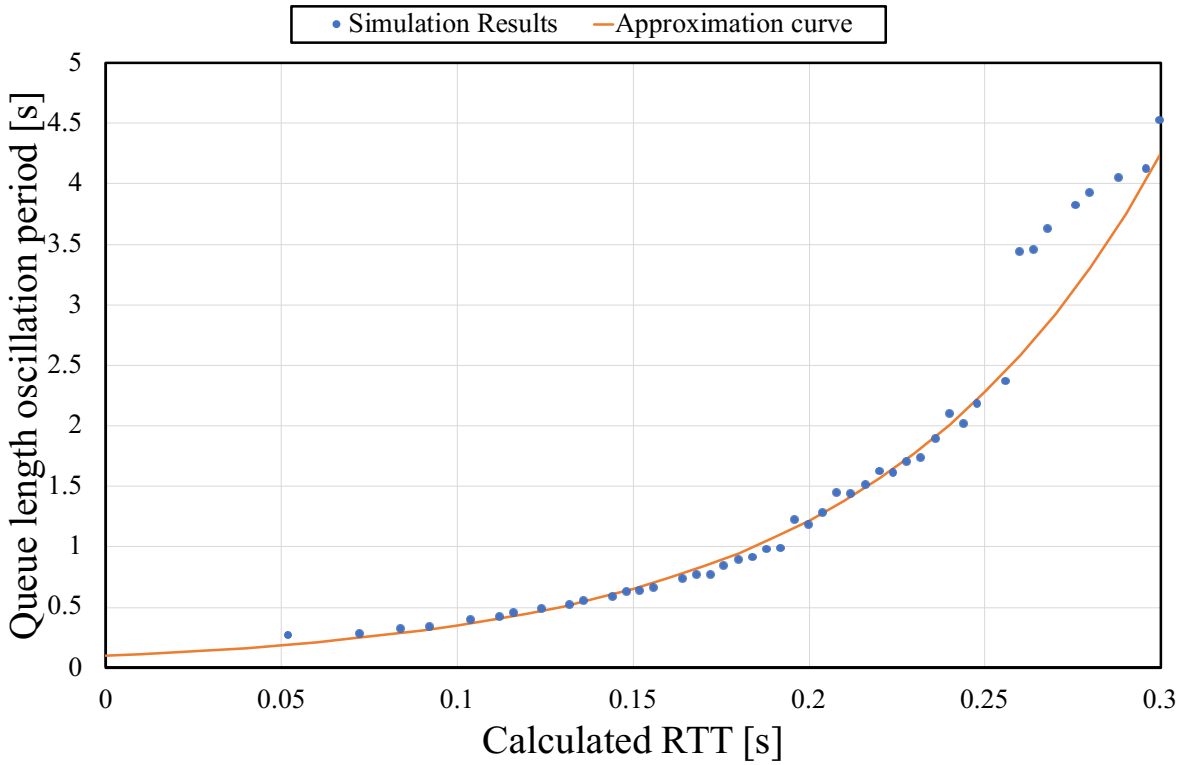


Fig. 5-13 The approximation curve.

approximation curve is as follows;

$$\text{queue length oscillation period} \fallingdotseq 0.1e^{0.0125R_{\text{calc}}}, \tag{5.5}$$

where R_{calc} denotes the calculated RTT. By assuming the average queue length is generally close or equal to q_{cmd}, the average queueing delay could be calculated by converting the unit of q_{cmd} to bits and dividing it by C. Thus, the value of R_{calc} can be derived by adding the calculated queueing delay with R_{n}. The duration of $monitor\_time$ is defined as the doubled duration of the queue oscillation period, which means there would be two queue oscillation peaks during every $monitor\_time$. Thus, the calculation would be as follows;

$$monitor\_time = 0.2e^{0.0125\left(T_{\text{p}}+\frac{q_{\text{cmd}}*8*pkt\_size}{C_{\text{Mbps}}}\right)}, \tag{5.6}$$

where pkt_size denote the size of a single packet in unit of bytes, and C_{Mbps} denotes the value of C in Mbps which equals to $C/1000000$.

This method attempts to change the duration according to the TCP/AQM network conditions, making the algorithm respond faster while avoiding misdetection due to making $monitor\_time$ unsuitably short. However, there are still some issues remaining with this method. The two major issues of this method are the lack of consideration about N and needs for an accurate model. As shown in (5.6), $monitor\_time$ is determined based on the values of R_{n}, q_{cmd}, pkt_size, and C. However, the queue length oscillation period is also affected by the number of TCP sessions N, since it affects the inertia of the TCP/AQM network as shown in (2.22). Thus, this method cannot consider the effect of different value of N while it actually affects the queue oscillation period. The second issue is that this method heavily depends on the multiple network model parameters. Thus, there is a probability of the modeling error severely degrading the efficiency in this method.

5.3.2.2.3 Variable Monitor Time with Dual EMA The third method is VMTwDEMA method, where two EMAs are utilized for determining $monitor\_time$. The first EMA, denoted as EMA_1, is an EMA of queue length with smoothing factor $\alpha = 0.0005$. The second EMA, denoted as EMA_2, is an EMA of EMA_1 with the same smoothing factor $\alpha = 0.0005$. Thus, the calculation would be as follows;

$$EMA_1(k) = q(k)\alpha + EMA_1(k-1)(1-\alpha)\,, \tag{5.7}$$

$$EMA_2(k) = EMA_1(k)\alpha + EMA_2(k-1)(1-\alpha)\,, \tag{5.8}$$

where $q(k)$, $EMA_1(k)$, and $EMA_2(k)$ denote q, EMA_1, and EMA_2 at k^{th} sample, respectively. These two EMAs are calculated throughout the algorithm procedure, regardless of the current phase the algorithm is in.

Figure 5-14 shows the queue length and two EMAs when q_{cmd}, N, C, T_{p}, q_{lim}, and simulation

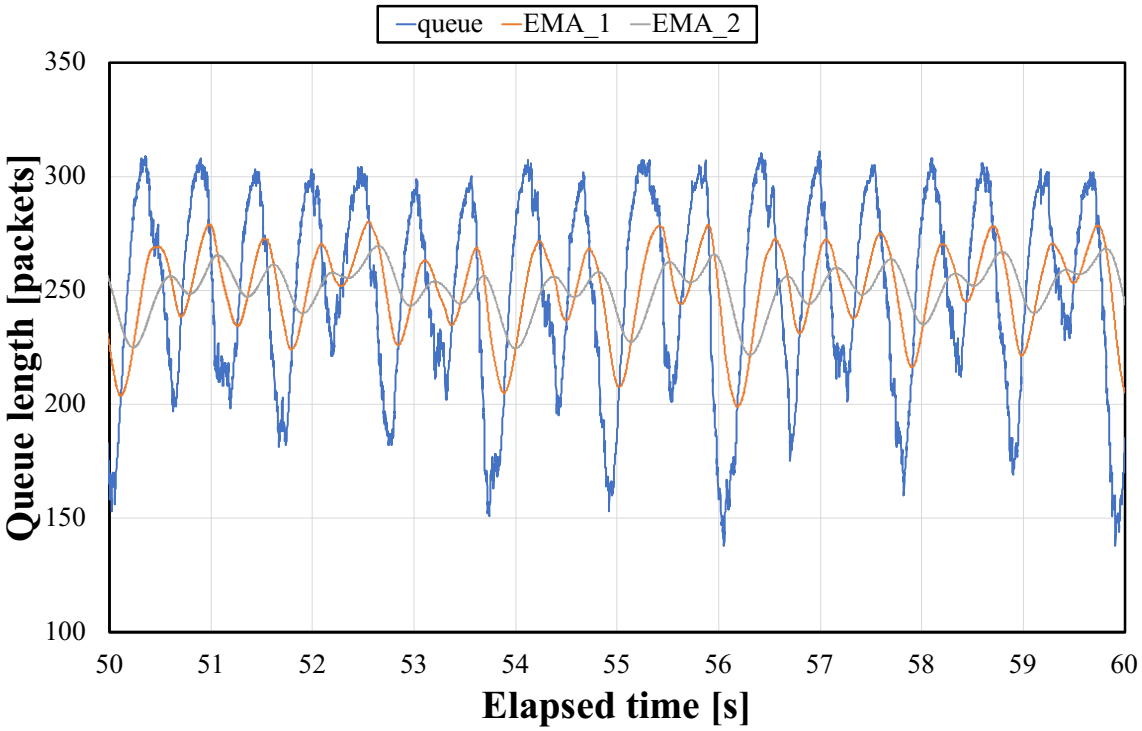


Fig. 5-14 Queue length and two EMAs.

duration were set to 250 packets, 100, 50 Mbps, 100 ms, 500 packets, and 120 s, respectively. As it can be seen in Fig. 5-14, EMA_1 tended to follow the queue length but was slightly delayed and had narrower oscillation. This is since taking the EMA calculation is mathematically identical to applying a digital low-pass filter. This relationship can also be observed between EMA_1 and EMA_2.

Utilizing these characteristics, the upper and lower peak of the queue length oscillation can be detected by the magnitude relationship between EMA_1 and EMA_2. When the magnitude relationship changed from $EMA_1 \geq EMA_2$ to $EMA_1 < EMA_2$ at two consecutive sampling times, that indicates there was an upper peak of the queue length oscillation. On the other hand, when the magnitude relationship changed from $EMA_1 < EMA_2$ to $EMA_1 \geq EMA_2$ at two consecutive sampling times, that indicates there was a lower peak of the queue length oscillation. This relationship can be confirmed by observing the queue length fluctuations in Fig. 5-14. This

characteristic is due to the existence of phase delay between EMA_1 and EMA_2.

Along with the calculation of EMAs, the algorithm always observes the magnitude relationship between EMA_1 and EMA_2 and stores the magnitude relationship datum of the previous sampling time. By calculating the current magnitude relationship and comparing it with that of the previous sampling time, the algorithm can detect whether the inversion of magnitude relationship has occurred or not. If an inversion has been detected, the algorithm take note that the queue length oscillation has passed an upper or lower peak based on which inversion of magnitude relationship has happened. The Loss-aware mode takes record of detected upper peaks, and the Delay-aware mode takes record of detected lower peaks. The duration of $monitor\_time$ is defined as the time needed for detecting three peaks corresponding to a selected mode. This number three was increased from the number two which was utilized in approximation curve method, in order to avoid misdetections.

In addition, the minimum duration of the Monitor Phase $monitor\_time_{\min}$ is predetermined. If the duration passed in the Monitor Phase is larger than $monitor\_time_{\min}$ and three peaks corresponding to the mode are detected, the algorithm exits the Monitor Phase and enters Update Phase. The whole flow of the Monitor Phase using VMTwDEMA is shown in Algorithm 2.

5.3.3 Update Phase

After performing the Basic parameter update procedure for duration of $monitor\_time$, the algorithm enters “Update Phase”.

If VMTwDEMA method is utilized, the algorithm first checks if the system is in the independent stable state or not, as shown in Fig. 5-12. If the algorithm determined that the system is in the independent stable state, the algorithm performs the “Independent stable state procedure”. The actual calculation done in the procedure differs depending on the mode and the independent stable state flag ind_flag. If the algorithm determined that the system is not in the independent stable state, the algorithm performs the “Standard update procedure”.

Algorithm 2 Monitor Phase using VMTwDEMA

```
while timer ≥ wait_time do
  Calculate EMA1(timer).
  Calculate EMA2(timer).
  if EMA1(timer) ≥ EMA2(timer) then
    if EMA1(timer − 1) < EMA2(timer − 1) then
      An upper peak detected.
    end if
  end if
  if EMA1(timer) < EMA2(timer) then
    if EMA1(timer − 1) ≥ EMA2(timer − 1) then
      A lower peak detected.
    end if
  end if
  if timer ≥ (wait_time + monitor_time_min) then
    if Loss-aware mode then
      if Detected upper peaks ≥ 3 then
        Exit Monitor Phase and enter Update Phase.
      end if
    else {Delay-aware mode}
      if Detected lower peaks ≥ 3 then
        Exit Monitor Phase and enter Update Phase.
      end if
    end if
  end if
end while
```

If CMT method or VMTwAC method is utilized, the algorithm cannot detect the independent stable state and the Standard update procedure is always performed in this phase. Thus, at the decision symbol of the beginning of the Update Phase in Fig. 5-12, the route of No is always selected with CMT and VMTwAC methods.

5.3.3.1 Standard Update Procedure

In this procedure, the algorithm compares $q_{\max}$ or $q_{\min}$ with the threshold queue length q_{th}, and increase or decrease q_{cmd} based on the difference. The calculation differs depending on the mode of the algorithm.

5.3.3.1.1 Loss-Aware Mode In the Loss-aware mode, the algorithm compares $q_{\max}$ with the threshold queue length q_{th} defined as follows;

$$q_{\text{th}} = q_{\text{lim}} - q_{\text{guard}}, \tag{5.9}$$

where q_{guard} is the designed guard queue length to avoid the buffer overflow and is set to $q_{\text{lim}}/20$ in this thesis. In other words, $q_{\text{th}} = q_{\text{lim}} * 19/20$ in the Loss-aware mode.

If $q_{\max}$ is exactly equal to q_{th}, then the current q_{cmd} is maintained. If $q_{\max}$ is larger than q_{th}, the algorithm calculates the difference and subtracts that value from q_{cmd}, thus lowering the queue length. If $q_{\max}$ is smaller than q_{th}, the algorithm calculates the difference and adds $1/3$ of its value to q_{cmd}. This coefficient of $1/3$ functions as an attenuator in order to avoid the queue length spiking up instantly to reach q_{lim}, inducing buffer overflow.

After the update of q_{cmd} is performed, the algorithm checks if q_{cmd} is in the range from the initial value $q_{\text{lim}}/2$ to the threshold q_{th}. If q_{cmd} is outside the range, its value is saturated to be in the range, which means q_{cmd} would not be lower than $q_{\text{lim}}/2$ nor higher than q_{th} in the Loss-aware mode. Finally, $q_{\max}$ is reset to 0, $q_{\min}$ is reset to q_{lim}, $timer$ is reset to $wait\_time$, and the algorithm reverts to the Monitor Phase.

The whole flow of the Standard update procedure of the Loss-aware mode is shown in Algorithm 3.

Algorithm 3 Standard update procedure of the Loss-aware mode

if $q_{\max} > q_{\text{th}}$ **then**
  $q_{\text{cmd}} \leftarrow q_{\text{cmd}} - (q_{\max} - q_{\text{th}})$.
else if $q_{\max} < q_{\text{th}}$ **then**
  $q_{\text{cmd}} \leftarrow q_{\text{cmd}} + (q_{\text{th}} - q_{\max})\ /3$.
end if
if $q_{\text{cmd}} < q_{\text{lim}}/2$ **then**
  $q_{\text{cmd}} \leftarrow q_{\text{lim}}/2$.
else if $q_{\text{cmd}} > q_{\text{th}}$ **then**
  $q_{\text{cmd}} \leftarrow q_{\text{th}}$.
end if
$q_{\max} \leftarrow 0$.
$q_{\min} \leftarrow q_{\text{lim}}$.
$timer \leftarrow wait\_time$.
Revert to the Monitor Phase.

5.3.3.1.2 Delay-Aware Mode In the Delay-aware mode, the algorithm compares $q_{\min}$ with the threshold queue length q_{th} defined as follows;

$$q_{\text{th}} = q_{\text{guard}}, \tag{5.10}$$

where q_{guard} is same with the Loss-aware mode, which means that $q_{\text{th}} = q_{\text{lim}} * 1/20$ in the Delay-aware mode.

If $q_{\min}$ is exactly equal to q_{th}, then the current q_{cmd} is maintained. If $q_{\min}$ is smaller than q_{th}, the algorithm calculates the difference and adds that value to q_{cmd}, thus raising the queue length. If $q_{\min}$ is larger than q_{th}, the algorithm calculates the difference and subtracts $1/3$ of its value from q_{cmd}. This coefficient of $1/3$ functions as an attenuator in order to avoid the queue length plunging down instantly to reach 0, inducing buffer underflow.

After the update of q_{cmd} is performed, the algorithm checks if q_{cmd} is in the range from the threshold q_{th} to the initial value $q_{\text{lim}}/2$. If q_{cmd} is outside the range, its value is saturated to

be in the range, which means q_{cmd} would not be lower than q_{th} nor higher than $q_{\mathrm{lim}}/2$ in the Delay-aware mode. Finally, q_{max} is reset to 0, q_{min} is reset to q_{lim}, $timer$ is reset to $wait\_time$, and the algorithm reverts to the Monitor Phase.

The whole flow of the Standard update procedure of the Delay-aware mode is shown in Algorithm 4.

Algorithm 4 Standard update procedure of the Delay-aware mode

if $q_{\mathrm{min}} < q_{\mathrm{th}}$ **then**
  $q_{\mathrm{cmd}} \leftarrow q_{\mathrm{cmd}} + (q_{\mathrm{th}} - q_{\mathrm{min}})$.
else if $q_{\mathrm{min}} > q_{\mathrm{th}}$ **then**
  $q_{\mathrm{cmd}} \leftarrow q_{\mathrm{cmd}} - (q_{\mathrm{min}} - q_{\mathrm{th}})\ /3$.
end if
if $q_{\mathrm{cmd}} > q_{\mathrm{lim}}/2$ **then**
  $q_{\mathrm{cmd}} \leftarrow q_{\mathrm{lim}}/2$.
else if $q_{\mathrm{cmd}} < q_{\mathrm{th}}$ **then**
  $q_{\mathrm{cmd}} \leftarrow q_{\mathrm{th}}$.
end if
$q_{\mathrm{max}} \leftarrow 0$.
$q_{\mathrm{min}} \leftarrow q_{\mathrm{lim}}$.
$timer \leftarrow wait\_time$.
Revert to the Monitor Phase.

5.3.3.2 Independent Stable State Procedure

If the TCP/AQM network enters the independent stable state, the standard update procedure may function in an unintended manner. The following is an example scenario in the Loss-aware mode:

1. The network enters independent stable state and q_{max} stays at 200 packets, while $q_{\mathrm{lim}} = 500$ packets and currently $q_{\mathrm{cmd}} = 350$ packets.
2. The algorithm enters the Update Phase, the difference is calculated as $q_{\mathrm{th}} - q_{\mathrm{max}} = 275$, and q_{cmd} is updated to 442 packets.
3. The algorithm reverts to the Monitor Phase, but q_{max} is still 200 packets, regardless of the raised q_{cmd}.

4. In the next Update Phase, q_{cmd} is raised and saturated to be 475 packets, which is equal to the possible maximum value q_{th}.

Thus, when the system enters independent stable state, the algorithm keeps raising q_{cmd} until it reaches the maximum value q_{th}. This is undesired because when the new TCP sessions join the network, their load resolves the independent stable state, raises the queue length, and induces buffer overflow. Since q_{cmd} is set to the possible maximum value at that moment, a longer duration is needed to resolve the buffer overflow. In order to avoid such situations, the independent stable state must be detected, and specific procedure must be performed.

The following independent stable state procedures includes both detection process and update process, which are expressed separately in Fig. 5-12. The independent stable state procedures were implemented for the VMTwDEMA method, where the other two proposed methods, i.e., the CMT method and VMTwAC method, do not include these functions.

5.3.3.2.1 Loss-Aware Mode In the Loss-aware mode, whether the system is in the independent stable state or not is determined by observing the relationship between q_{max} and q_{cmd}. If the AQM congestion control system has a very small steady state error, the queue length oscillates above and below q_{cmd}, which means $q_{\mathrm{max}} > q_{\mathrm{cmd}}$. However, if $q_{\mathrm{max}} \leq q_{\mathrm{cmd}}$, the algorithm considers the situation may be in the independent stable state and sets the independent stable state flag ind_flag to 1 from the initial value of 0. At the first detection of the independent stable state, instead of performing the standard update procedure for q_{cmd}, q_{cmd} is kept the same value, and the algorithm reverts back to Monitor Phase.

At the second consecutive detection, which means $q_{\mathrm{max}} \leq q_{\mathrm{cmd}}$ while $ind\_flag = 1$, the algorithm recognizes that the system is in the independent stable state, and sets q_{cmd} as q_{max}. By performing this procedure, the algorithm maintains the independent stable state while setting q_{cmd} as low as possible. This shortens the duration needed to resolve the buffer overflow when new TCP sessions join the network and induces buffer overflow.

If $q_{\mathrm{max}} > q_{\mathrm{cmd}}$ while $ind\_flag = 1$, that indicates that the system may have resolved the

independent stable state. In such a situation, ind_flag is updated to 0, but q_{cmd} is kept the same value for that Update Phase. If the relationship of $q_{\max} > q_{\mathrm{cmd}}$ is still kept for the next Update Phase, the standard update procedure is performed.

The saturation of q_{cmd} is also performed after setting q_{cmd} as $q_{\max}$ after the independent stable state recognition. Thus, q_{cmd} does not become lower than $q_{\mathrm{lim}}/2$ even after the detection of the independent stable state.

The whole flow of the independent stable state procedure of the Loss-aware mode is shown in Algorithm 5.

Algorithm 5 Independent stable state procedure of the Loss-aware mode

```
if ind_flag = 0 then
  if q_max > q_cmd then
    Perform standard update procedure.
  else {q_max ≤ q_cmd}
    ind_flag ← 1.
  end if
else {ind_flag = 1}
  if q_max > q_cmd then
    ind_flag ← 0
  else {q_max ≤ q_cmd}
    q_cmd ← q_max.
  end if
end if
q_max ← 0.
q_min ← q_lim.
timer ← wait_time.
Revert to the Monitor Phase.
```

5.3.3.2.2 Delay-Aware Mode In the Delay-aware mode, the detection of whether the system is in independent stable state or not is determined in the same manner with the Loss-aware mode. However, since the Delay-aware mode attempts to lower the q_{cmd}, a different update procedure must be performed.

The Delay-aware mode generally can perform the Standard update procedure even if the independent stable state was detected, since the process of lowering the q_{cmd} itself may resolve

the independent stable state. However, under some specific scenarios, there is a possibility of the Standard update procedure of the Delay-aware mode algorithm may function in an unintended manner. The following is an example scenario in the Delay-aware mode:

1. The network enters independent stable state while $q_{\max} = 40$ packets, $q_{\min} = 37$ packets, $q_{\lim} = 500$ packets, and $q_{\mathrm{cmd}} = 41$ packets.
2. The algorithm enters the Update Phase, the difference is calculated as $q_{\min} - q_{\mathrm{th}} = 12$, and q_{cmd} is updated to 37 packets.
3. The algorithm reverts to the Monitor Phase, but the network is still in the independent stable state, $q_{\max} = 40$ packets, and $q_{\min} = 37$ packets, regardless of the lowered q_{cmd}.
4. The algorithm enters the Update Phase again, the same calculations are done, and q_{cmd} is updated to 33 packets.
5. The algorithm reverts to the Monitor Phase again, but the network is still in the independent stable state.
6. The algorithm enters the Update Phase for the third time, the same calculations are done, and q_{cmd} is updated to 29 packets.
7. The network finally reacts to the lowered q_{cmd}, but the current $q_{\mathrm{cmd}} = 29$ packets are too small compared to $q_{\max} = 40$ packets and $q_{\min} = 37$ packets.
8. The $q_{\mathrm{cmd}} = 29$ packets plunges down the queue length to 0, inducing buffer underflow.

Thus, when the system enters independent stable state at some certain queue length, the Delay-aware mode algorithm keeps lowering q_{cmd} until it is too low, that the buffer underflow is induced when the network starts reacting to the AQM controller. This is because there is a possibility of independent stable state being triggered while $q_{\max} > q_{\mathrm{cmd}}$. In order to avoid such situations, the algorithm should avoid continuously lowering q_{cmd} when it has a risk of inducing buffer underflow. In other words, the algorithm should define the range of queue length for $q_{\min}$, so that when $q_{\min}$ is in that range the Standard update procedure would not be performed. In this thesis, this range was defined as from q_{guard} to $q_{\mathrm{guard}} * 2$.

The saturation of q_{cmd} is also performed after setting q_{cmd} as q_{max} after the independent stable state recognition. Thus, q_{cmd} does not become lower than q_{guard} even after the detection of the independent stable state.

The whole flow of the independent stable state procedure of the Delay-aware mode is shown in Algorithm 6.

Algorithm 6 Independent stable state procedure of the Delay-aware mode

if $ind\_flag = 0$ **then**
  if $q_{\mathrm{max}} > q_{\mathrm{cmd}}$ **then**
    Perform standard update procedure.
  else if $q_{\mathrm{min}} > q_{\mathrm{guard}} * 2$ **then**
    Perform standard update procedure.
    $ind\_flag \leftarrow 1$
  else
    $ind\_flag \leftarrow 1$
  end if
else $\{ind\_flag = 1\}$
  if $q_{\mathrm{max}} > q_{\mathrm{cmd}}$ **then**
    $ind\_flag \leftarrow 0$
  else if $q_{\mathrm{min}} > q_{\mathrm{guard}} * 2$ **then**
    Perform standard update procedure.
  else
    $q_{\mathrm{cmd}} \leftarrow q_{\mathrm{max}}$
  end if
end if
$q_{\mathrm{max}} \leftarrow 0.$
$q_{\mathrm{min}} \leftarrow q_{\mathrm{lim}}.$
$timer \leftarrow wait\_time.$
Revert to the Monitor Phase.

5.3.4 Emergency Update Procedure

In the Loss-aware mode, the buffer overflow must be avoided while raising q_{cmd}. In the Delay-aware mode, the buffer underflow must be avoided while lowering q_{cmd}. However, the system may induce buffer overflow or underflow due to the change of network conditions. The algorithm must instantly mitigate the buffer overflow or underflow.

The instant decrement or increment of q_{cmd} to mitigate the buffer overflow or underflow is called the "Emergency update procedure". This is performed during the Wait Phase or Monitor Phase.

5.3.4.1 Loss-Aware Mode

In the Loss-aware mode, when the buffer overflow is detected, which means $q = q_{\text{lim}}$, q_{cmd} is updated to the average of the current q_{cmd} and the initial value $q_{\text{lim}}/2$ at an instant. After that, the algorithm turns the emergency update flag emg_flag to 1 from the initial value of 0 to indicate that there was an Emergency update procedure performed recently. Then, the algorithm resets q_{max} to 0, q_{min} to q_{lim}, $timer$ to $wait\_time * 4/5$, and reverts back to the Wait Phase. This means that the algorithm stays in the Wait Phase for the duration of $wait\_time/5$. This procedure is performed in order to avoid recording the high queue length right after the emergency update procedure as q_{max} and lowering q_{cmd} too much afterwards.

After the emergency update procedure, if the algorithm passes through Monitor Phase and Update Phase normally without triggering another Emergency update procedure, emg_flag is reset to 0. However, if another buffer overflow is detected in Monitor Phase while $emg\_flag = 1$, the algorithm resets $timer$ to 0 and restarts from the Wait Phase. In that case, q_{max}, q_{min}, and q_{cmd} will also be reset to their initial values, as mentioned in 5.3.1 and shown in Fig. 5-12.

The whole flow of the Emergency update procedure of the Loss-aware mode is shown in Algorithm 7.

5.3.4.2 Delay-Aware Mode

In the Delay-aware mode, when the buffer underflow is detected, which means $q = 0$, the value of q_{cmd} is updated to the average of the current q_{cmd} and the initial value $q_{\text{lim}}/2$ at an instant. After that, the algorithm turns the emergency update flag emg_flag to 1. Then, the algorithm resets q_{max} to 0, q_{min} to q_{lim}, $timer$ to $wait\_time * 4/5$, and reverts back to the Wait Phase.

Algorithm 7 Emergency update procedure of the Loss-aware mode

```
if q = q_lim then
  if emg_flag = 0 then
    q_cmd ← (q_cmd + q_lim/2) /2.
    emg_flag ← 1.
    timer ← wait_time ∗ 4/5.
  else {emg_flag = 1}
    timer ← wait_time.
  end if
  q_max ← 0.
  q_min ← q_lim.
  Revert to the Wait Phase.
end if
if Standard update procedure is performed and emg_flag = 1 then
  emg_flag ← 0.
end if
```

This procedure is performed in order to avoid recording the low queue length right after the emergency update procedure as $q_{\min}$ and raising q_{cmd} too much afterwards.

After the emergency update procedure, if the algorithm passes through Monitor Phase and Update Phase normally without triggering another emergency update, emg_flag is reset to 0. However, if another buffer underflow is detected in Monitor Phase while $emg\_flag = 1$, the algorithm resets $timer$ to 0 and restarts from the Wait Phase. In that case, $q_{\max}$, $q_{\min}$, and q_{cmd} will also be reset to their initial values, as mentioned in 5.3.1 and shown in Fig. 5-12.

The whole flow of the Emergency update procedure of the Delay-aware mode is shown in Algorithm 8.

5.3.5 No Congestion Detection

If the total throughput that the communications through the bottleneck router can achieve is smaller than the bottleneck link capacity, there would be no congestion occurring. In another words, the queue length would be generally equal to 0. Since the proposed VMTwDEMA method utilizes the values of two EMAs to determine its monitor time, the method needs the

Algorithm 8 Emergency update procedure of the Delay-aware mode

```
if q = 0 then
  if emg_flag = 0 then
    q_cmd ← (q_cmd + q_lim/2) /2.
    emg_flag ← 1.
    timer ← wait_time ∗ 4/5.
  else {emg_flag = 1}
    timer ← wait_time.
  end if
  q_max ← 0.
  q_min ← q_lim.
  Revert to the Wait Phase.
end if
if Standard update procedure is performed and emg_flag = 1 then
  emg_flag ← 0.
end if
```

magnitude relationship between EMA_1 and EMA_2 to change in order to enter the Update Phase. However, if the queue length falls to 0 at some point and does not increase, the inversion of magnitude relationship will never occur afterwards. This is troublesome especially for the Loss-aware VMTwDEMA method because the algorithm would be unable to update the q_{cmd} once the TCP/AQM network enters this situation of no congestion. That may lead into the same scenario described in section 5.3.3.2.

In order to deal with this problem, VMTwDEMA method has a specific function to determine if there are no congestion occurring. If the queue length is equal to 0 for decently long duration, the values of two EMAs will eventually become lower than 1. Using this characteristic, the algorithm determines the network to have no congestion occurring if queue length is 0 and both EMA_1 and EMA_2 are lower than 1.0 at the same time. If the variables satisfy that requirements, the algorithm resets q_{max}, q_{min}, q_{cmd}, $timer$, ind_flag, and emg_flag to their initial values and reverts to Wait Phase. By performing this procedure, the algorithm resets itself to its initial state and waits for the congestion to occur.

This detection is only performed in the Monitor Phase. The whole flow of this detection of

no congestion is shown in Algorithm 9.

Algorithm 9 No Congestion Detection procedure for VMTwDEMA

if in Monitor Phase **then**
  if $q = 0$ **and** $EMA_1 < 1.0$ **and** $EMA_2 < 1.0$ **then**
    $q_{\mathrm{cmd}} \leftarrow q_{\mathrm{lim}}/2$.
    $q_{\mathrm{max}} \leftarrow 0$.
    $q_{\mathrm{min}} \leftarrow q_{\mathrm{lim}}$.
    $timer \leftarrow 0$.
    $ind\_flag \leftarrow 0$.
    $emg\_flag \leftarrow 0$.
  end if
end if

5.4 Performance Evaluation

In this section, the performances of the proposed algorithms are evaluated. All simulations were performed using Network Simulator 2 (NS-2).

5.4.1 Simulation Setup

The dumbbell shaped network topology shown in Fig. 5-1 was utilized in the simulations. As shown in Fig. 5-1, all the links had the same link capacity C. The bottleneck link had a link latency of $T_{\mathrm{p}}/10$, while all the other links had a link latency of $T_{\mathrm{p}}/5$. This was designed to make the RTT equal to the propagation delay T_{p} if there were no queueing delays.

The network parameters used in the simulations are shown in Table 5-1. The parameters used in the PID controller are shown in Table 5-2. These control parameters were set by referring to [91]. All the simulations were performed without using ECN. The parameters not mentioned here, such as N, C, T_{p}, and q_{lim} are specified in each section.

Table 5-1 Network parameters.

Packet size	1040 bytes
Maximum window size $wnd_{\max}$	20 packets
Simulation duration	120 s
Control period	0.001 s
$wait\_time$	5000 samples (5 s)
$monitor\_time_{\min}$	1000 samples (1 s)
Smoothing factor α for VMTwDEMA method	0.005

Table 5-2 Control parameters for PID controller.

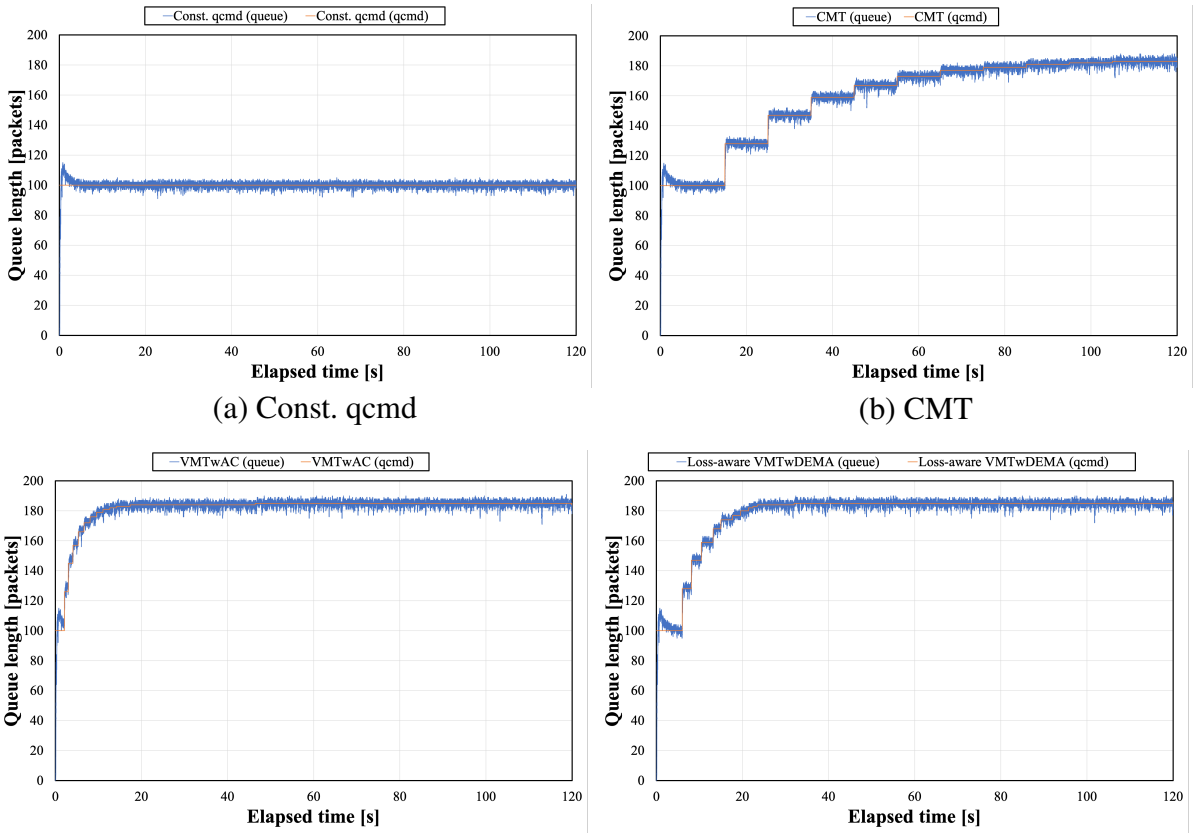
K_{p}	Proportional gain for PID controller	900
K_{i}	Integral gain for PID controller	700
K_{d}	Derivative gain for PID controller	55
g_{dif}	Cut-off frequency for PID controller	50 rad/s

5.4.2 Loss-Aware Mode

The proposed loss-aware mode algorithm was evaluated by the simulations. The three methods, i.e., CMT, VMTwAC, and VMTwDEMA, were presented as the proposed Loss-aware mode algorithm. The effectiveness of the three proposed methods were evaluated by comparing with the conventional method of using constant target queue length. In addition, the effectiveness in terms of converging speed and independent stable state detection of the three proposed methods were also evaluated. For simplicity, the conventional method is denoted as "Const. qcmd", and the three proposed algorithms are denoted as "CMT", "VMTwAC", and "Loss-aware VMTwDEMA", respectively.

5.4.2.1 Comparison Under Different Buffer Size

Figures 5-15, 5-16, and 5-17 show the fluctuations of the queue length and q_{cmd} of the four methods under different buffer size q_{lim}. Both the queue length and q_{cmd} was obtained every 1 ms. Figure 5-15 shows the simulation results under relatively small buffer size, where the values



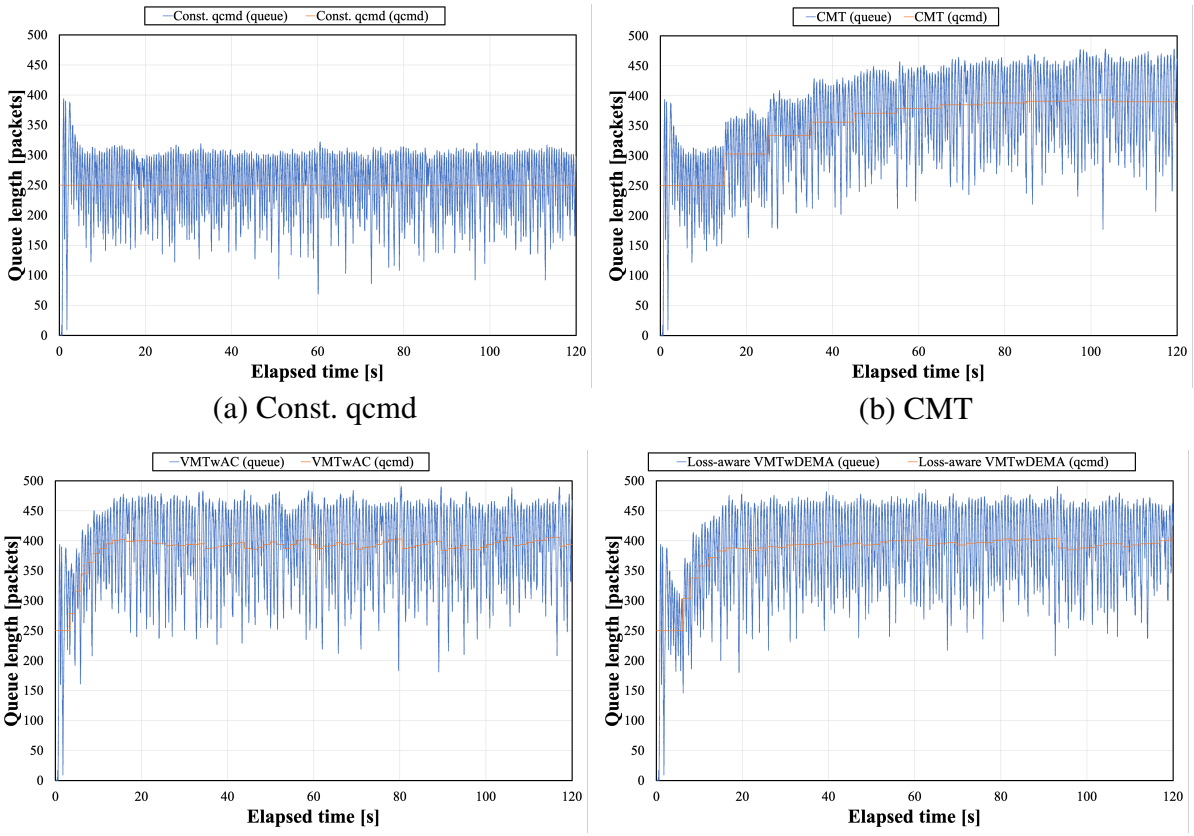
(a) Const. qcmd (b) CMT

(c) VMTwAC (d) Loss-aware VMTwDEMA

Fig. 5-15 Queue length fluctuations when q_{lim} = 200 packets.

Table 5-3 Summary of the results in Fig. 5-15.

	Const. qcmd	CMT
Average queue length [packets]	100.00	179.99
Maximum queue length [packets]	105	188
Rise time of the smoothed queue length [s]	21.887	54.258
Average goodput [Mbps]	9.34	9.34
Fairness index of goodput [%]	94.48	98.29
	VMTwAC	Loss-aware VMTwDEMA
Average queue length [packets]	185.00	184.99
Maximum queue length [packets]	191	190
Rise time of the smoothed queue length [s]	24.030	28.258
Average goodput [Mbps]	9.37	9.36
Fairness index of goodput [%]	98.28	97.78



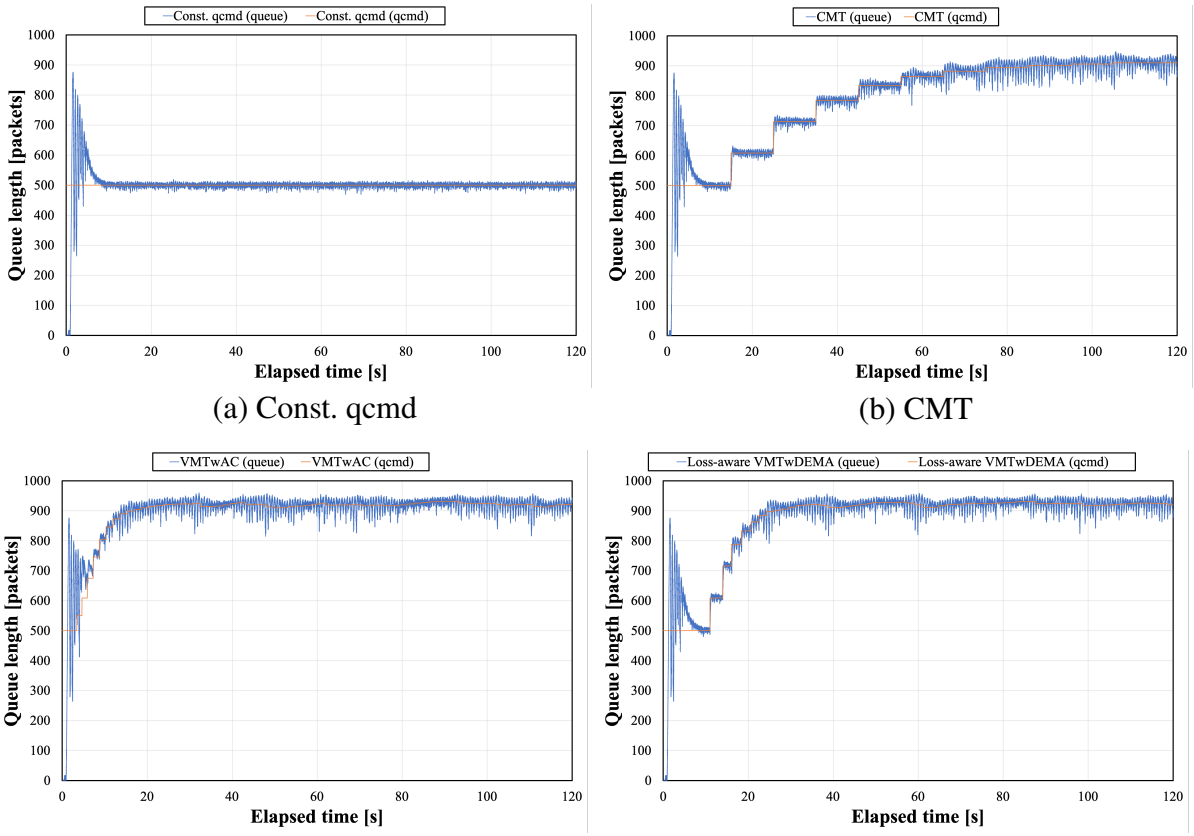
(a) Const. qcmd (b) CMT

(c) VMTwAC (d) Loss-aware VMTwDEMA

Fig. 5-16 Queue length fluctuations when q_{lim} = 500 packets.

Table 5-4 Summary of the results in Fig. 5-16.

	Const. qcmd	CMT
Average queue length [packets]	250.03	388.47
Maximum queue length [packets]	322	478
Rise time of the smoothed queue length [s]	21.518	47.205
Average goodput [Mbps]	49.55	49.59
Fairness index of goodput [%]	99.65	99.72
	VMTwAC	Loss-aware VMTwDEMA
Average queue length [packets]	393.96	396.54
Maximum queue length [packets]	491	491
Rise time of the smoothed queue length [s]	22.693	26.861
Average goodput [Mbps]	49.61	49.62
Fairness index of goodput [%]	99.56	99.50



(a) Const. qcmd (b) CMT

(c) VMTwAC (d) Loss-aware VMTwDEMA

Fig. 5-17 Queue length fluctuations when q_{lim} = 1000 packets.

Table 5-5 Summary of the results in Fig. 5-17.

	Const. qcmd	CMT
Average queue length [packets]	500.00	896.54
Maximum queue length [packets]	516	948
Rise time of the smoothed queue length [s]	20.533	53.904
Average goodput [Mbps]	96.18	96.89
Fairness index of goodput [%]	99.57	99.55
	VMTwAC	Loss-aware VMTwDEMA
Average queue length [packets]	921.88	923.12
Maximum queue length [packets]	957	959
Rise time of the smoothed queue length [s]	25.364	32.072
Average goodput [Mbps]	97.13	97.06
Fairness index of goodput [%]	99.61	99.56

of N, C, T_{p}, and q_{lim} were each set to 100, 10 Mbps, 20 ms, and 200 packets, respectively. Figure 5-16 shows the simulation results under medium buffer size, where the values of N, C, T_{p}, and q_{lim} were each set to 100, 50 Mbps, 100 ms, and 500 packets, respectively. Figure 5-17 shows the simulation results under a large buffer size, where the values of N, C, T_{p}, and q_{lim} were each set to 500, 100 Mbps, 100 ms, and 1000 packets, respectively. From the results shown in Figs. 5-15, 5-16, and 5-17, it can be confirmed that the three proposed methods successfully raised q_{cmd} without inducing any buffer overflow, while their converging speed differs.

Tables 5-3, 5-4, and 5-5 show the five evaluation index data under the three different simulation setups shown in Figs. 5-15, 5-16, and 5-17, respectively. The first two data, i.e., average queue length and maximum queue length, denote the time average of queue length and the maximum value of queue length, respectively. These values were obtained based on the queue length of the second half of the simulation duration, which means the data during the first 60 s were not used for calculating average nor maximum queue lengths. The rise time is defined as the time taken for rising of the smoothed queue length from 10% to 90% of the average queue length. The smoothed queue length was derived by taking the EMA of queue length with the smoothing factor of 0.0001 and was derived in independently from the calculation utilized in the proposed VMTwDEMA method. The average goodput is defined as the time average of the total throughput excluding the retransmitted packets. The fairness index of the goodput is a number that shows the fairness of the goodput of each TCP session in unit of %. Jain's fairness index [103] is utilized in this thesis, and its calculation equation shown in (4.15).

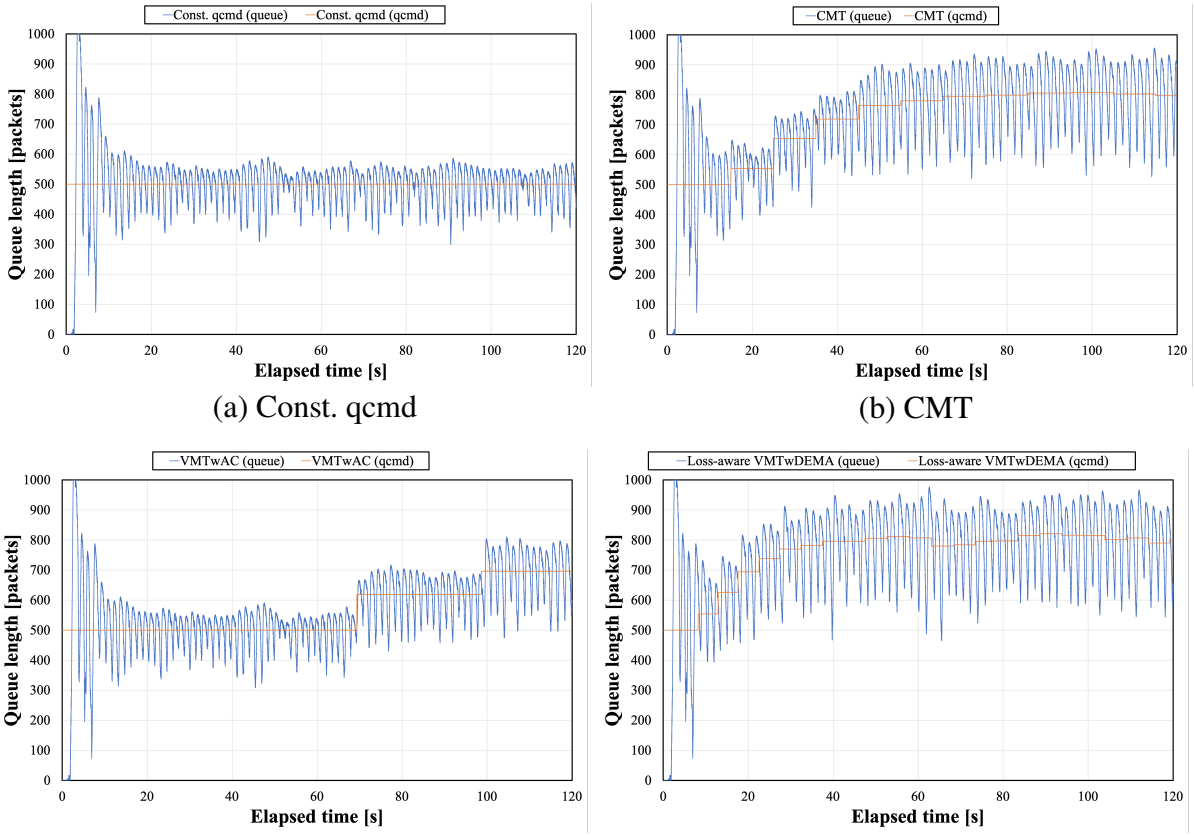
As shown in Tables 5-3, 5-4, and 5-5, the three methods using the Loss-aware mode algorithms had their average and maximum queue lengths being higher than the conventional Const. qcmd method, raising their average goodputs as well. Comparing the rise time of the three proposed methods, the VMTwAC method tends to have the fastest converging speed, followed by the proposed Loss-aware VMTwDEMA method, and the CMT method being the slowest. This is because the duration of $monitor\_time$ of the VMTwAC method was defined as the doubled duration of the queue oscillation period, meaning the algorithm attempts to observe

two upper peaks of the queue oscillation, while Loss-aware VMTwDEMA attempts to observe three of them. On the other hand, the CMT method had its monitor phase duration set to 10 s, which is too long under these simulation setups, resulting in the slowest converging speed of the three methods. The fairness index varied depending on the utilized method and the network setup in the range of 94 % to 99.5%. A significant difference between the four methods in term of fairness was not observed.

5.4.2.2 Performance Under High-Latency Network

Figure 5-18 shows the queue length fluctuations of the four methods where N, C, T_{p}, and q_{lim} were each set to 500, 50 Mbps, 300 ms, and 1000 packets, respectively. This simulation setup was designed to simulate the communication in a high-latency network. As shown in Fig. 5-18, the three proposed methods had their average and maximum queue length being higher than the conventional Const. qcmd method. However, unlike the simulation results shown in Figs. 5-15, 5-16, and 5-17, the Loss-aware VMTwDEMA method converged the fastest while the VMTwAC method became the slowest. This is because the approximation curve utilized in the algorithm, shown in (5.6), failed to approximate an accurate queue length oscillation period under large T_{p} and derive a duration too large for the monitor phase.

Table 5-6 shows the five evaluation index data under the simulation setup shown in Fig. 5-18. Tables 5-7 and 5-8 show the five evaluation index data under the same simulation setup with Table 5-6, while the value of N were set to 750 and 1000, respectively. From the results shown in Tables 5-6, 5-7, and 5-8, it can be confirmed that the three proposed methods successfully raised the average queue length and increased the average goodput. The Loss-aware VMTwDEMA method had the fastest converging speed and highest average goodput of the three proposed method, while the VMTwAC method had slowest converging speed and lowest average goodput. The characteristic of the approximation curve not functioning as intended under a high-latency network is reflected in the results. The fairness index varied amongst the methods and the



(a) Const. qcmd (b) CMT

(c) VMTwAC (d) Loss-aware VMTwDEMA

Fig. 5-18 Queue length fluctuations under T_p = 300 ms, N = 500.

Table 5-6 Summary of results in Fig. 5-18.

	Const. qcmd	CMT
Average queue length [packets]	499.52	799.60
Maximum queue length [packets]	586	956
Rise time of the smoothed queue length [s]	18.531	49.332
Average goodput [Mbps]	48.32	48.54
Fairness index of goodput [%]	99.11	98.93
	VMTwAC	Loss-aware VMTwDEMA
Average queue length [packets]	626.47	801.49
Maximum queue length [packets]	811	977
Rise time of the smoothed queue length [s]	74.695	33.911
Average goodput [Mbps]	48.41	48.55
Fairness index of goodput [%]	99.10	98.98

Table 5-7 Summary of results under T_p = 300 ms, N = 750.

	Const. qcmd	CMT
Average queue length [packets]	500.02	909.50
Maximum queue length [packets]	509	943
Rise time of the smoothed queue length [s]	19.085	58.120
Average goodput [Mbps]	47.27	47.61
Fairness index of goodput [%]	98.57	98.72
	VMTwAC	Loss-aware VMTwDEMA
Average queue length [packets]	655.47	939.57
Maximum queue length [packets]	756	953
Rise time of the smoothed queue length [s]	75.898	31.000
Average goodput [Mbps]	47.39	47.73
Fairness index of goodput [%]	98.65	98.71

Table 5-8 Summary of results under T_p = 300 ms, N = 1000.

	Const. qcmd	CMT
Average queue length [packets]	500.00	912.84
Maximum queue length [packets]	508	941
Rise time of the smoothed queue length [s]	19.161	56.688
Average goodput [Mbps]	46.57	46.87
Fairness index of goodput [%]	97.30	97.54
	VMTwAC	Loss-aware VMTwDEMA
Average queue length [packets]	656.15	942.66
Maximum queue length [packets]	755	951
Rise time of the smoothed queue length [s]	76.005	30.400
Average goodput [Mbps]	46.65	46.94
Fairness index of goodput [%]	97.46	98.06

network setups in the range of 97 % to 99%. A significant difference between the four methods in term of fairness was not observed.

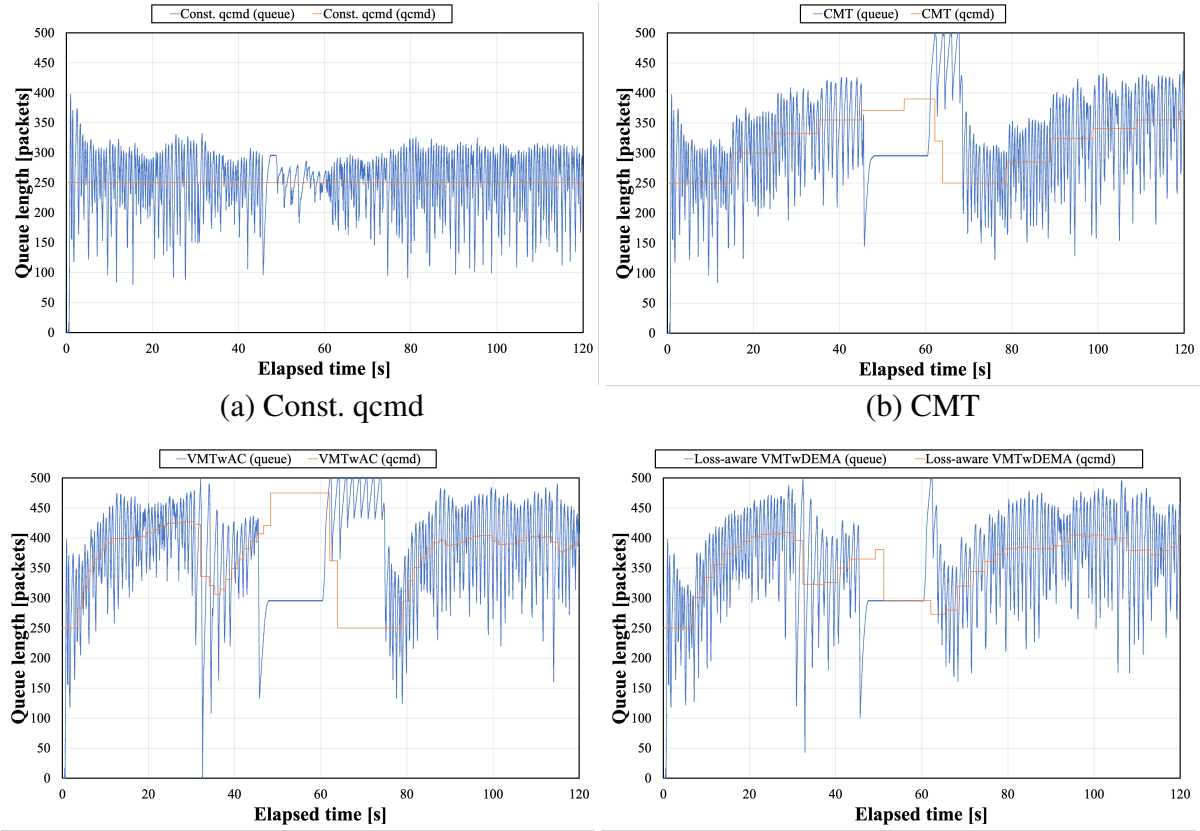
5.4.2.3 Independent Stable State Detection

Figure 5-19 shows the queue length fluctuations of the four methods where N changes depending on time. The value of other parameters; C, T_{p}, and q_{lim}; were each set to 50 Mbps, 100 ms, and 500 packets, respectively. The value of N was set to 75 at the beginning of the simulation, and its value decreased to 60 at the simulation time of 30 s. The value of N was decreased again to 45 at the simulation time of 45 s, and returned to 60 at the simulation time of 60 s. After that, N was increased back to 75 at the simulation time of 75 s, and maintained that value for the rest of simulation duration. These simulation setups were designed to simulate the TCP/AQM network communication with fluctuating number of TCP sessions.

As Fig. 5-19 shows, when q_{cmd} was set to a value equal to or greater than approximately 300 packets, the TCP/AQM network entered the independent stable state at 45 s. Since the CMT method and VMTwAC method did not have any detection algorithm of independent stable state, they keep raising q_{cmd} while the system is in the independent stable state. This induced multiple samples of buffer overflow as soon as N increased at 60 s. Table 5-9 shows the total number of buffer overflow samples of the simulation result shown in Fig. 5-19. The Loss-aware VMTwDEMA method reduced the buffer overflow samples compared the other two proposed methods which induced a large quantity of buffer overflow samples.

5.4.3 Delay-Aware Mode

The proposed Delay-aware VMTwDEMA method is evaluated. A method using constant target queue length is utilized as the conventional method. For simplicity, the proposed algorithm is denoted as “Delay-aware VMTwDEMA” and the conventional method is denoted as “Const. qcmd”.



(a) Const. qcmd (b) CMT

(c) VMTwAC (d) Loss-aware VMTwDEMA

Fig. 5-19 Queue length fluctuations when N fluctuated between 45 to 75.

Table 5-9 Total number of buffer overflow samples in Fig. 5-19.

	Const. qcmd	CMT
Total number of buffer overflow samples	0	992
	VMTwAC	Loss-aware VMTwDEMA
Total number of buffer overflow samples	2234	219

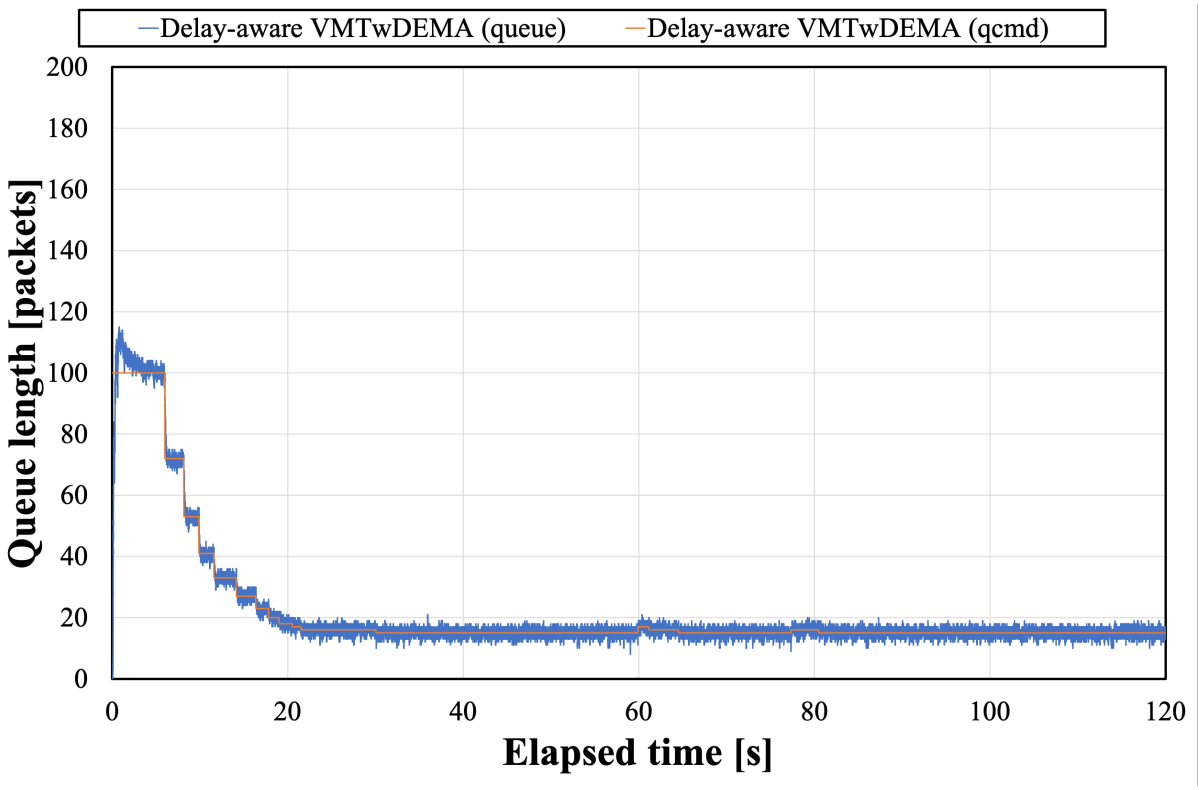


Fig. 5-20 Queue length fluctuation of Delay-aware VMTwDEMA method (q_{lim} = 200 packets).

5.4.3.1 Comparison Under Different Buffer Size

Figures 5-20, 5-21, and 5-22 show the fluctuations of the queue length and q_{cmd} of the proposed Delay-aware VMTwDEMA method under different buffer size q_{lim}. Figure 5-20 shows the simulation results under the same simulation setups with Fig. 5-15, where the values of N, C, T_{p}, and q_{lim} were each set to 100, 10 Mbps, 20 ms, and 200 packets, respectively. Figure 5-21 shows the simulation results under the same simulation setups with Fig. 5-16, where the values of N, C, T_{p}, and q_{lim} were each set to 100, 50 Mbps, 100 ms, and 500 packets, respectively. Figure 5-22 shows the simulation results under the same simulation setups with Fig. 5-17, where the values of N, C, T_{p}, and q_{lim} were each set to 500, 100 Mbps, 100 ms, and 1000 packets, respectively.

From the results shown in Figs. 5-20, 5-21, and 5-22, it can be confirmed that the proposed

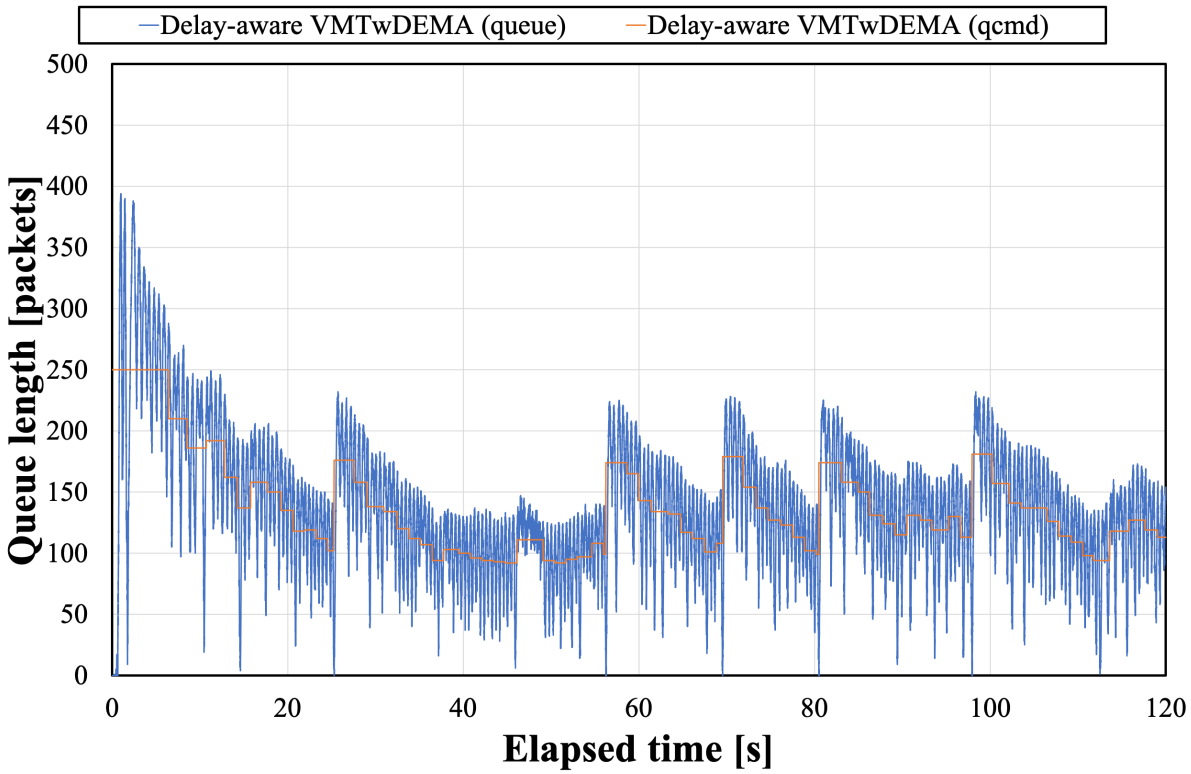


Fig. 5-21 Queue length fluctuation of Delay-aware VMTwDEMA method (q_{lim} = 500 packets).

Delay-aware VMTwDEMA method lowered q_{cmd} and average queue length. Although it generally avoids buffer underflow, queue length seldom reaches 0 when the queue length oscillation amplitude is large, as shown in Fig. 5-21. However, even in such a situation, the algorithm immediately reacted to the buffer underflow and raised the q_{cmd}, suppressing the effect of buffer underflow to the minimum.

Table 5-10 shows the evaluation index data under the three different simulation setups shown in Figs. 5-20, 5-21, and 5-22. The data of Const. qcmd method are derived from the simulation data shown in Figs. 5-15, 5-16, and 5-17. The average queue length and minimum queue length denote the time average of queue length and the minimum value of queue length, respectively. These values were obtained based on the queue length of the second half of the simulation duration, which means the data during the first 60 s were not used for calculating average nor maximum queue length. The average throughput of TCP flows is defined as the time average of

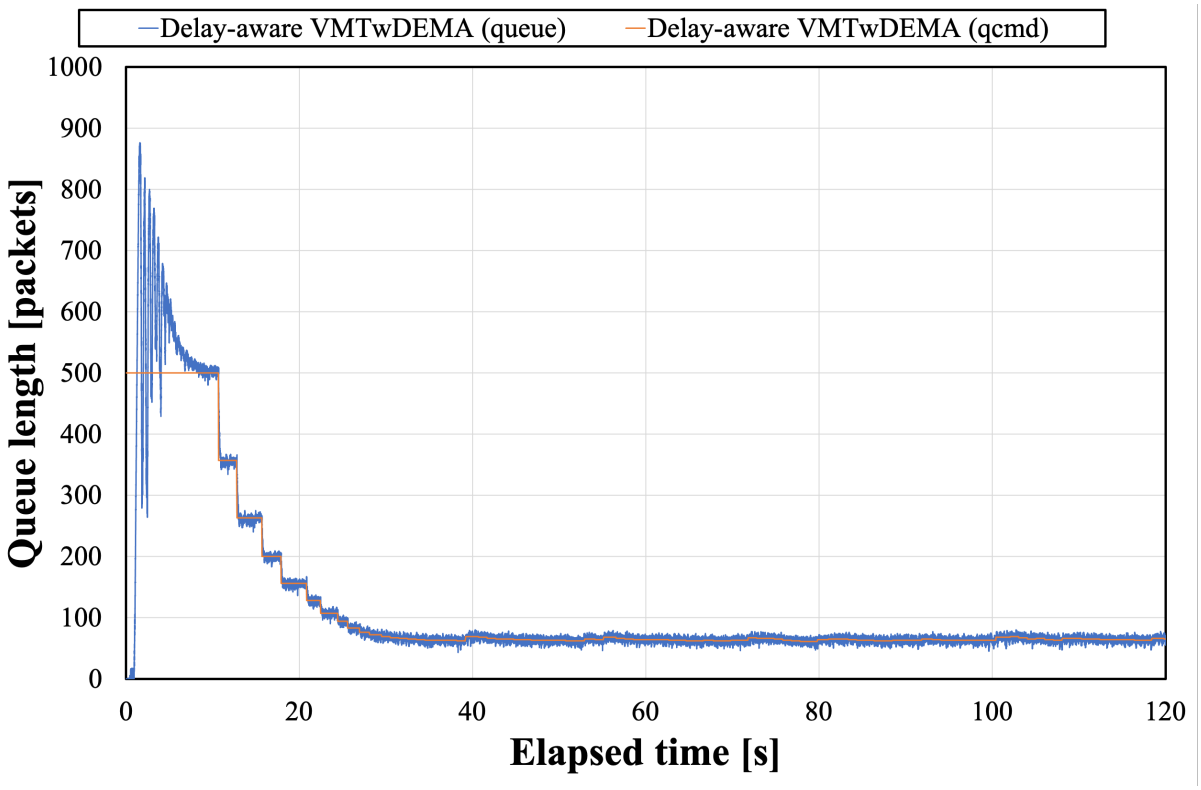


Fig. 5-22 Queue length fluctuation of Delay-aware VMTwDEMA method (q_{lim} = 1000 packets).

the throughput of TCP sessions, which does not include the throughput of non-TCP flows which are utilized in the following section. This value is also calculated based on the queue length of the second half of the simulation duration. The total number of buffer underflow samples is the number of the samples with queue length of 0 in the second half of the simulation duration.

As shown in Table 5-10, it can be confirmed that the proposed Delay-aware VMTwDEMA method successfully lowered q_{cmd}, lowering the average queue length lower than the conventional Const. qcmd method. The proposed Delay-aware VMTwDEMA method generally avoids buffer underflow, and had only 7 samples, which corresponds to 7 ms duration, of buffer underflow under q_{lim} = 500 packets. Even in such a situation, the average throughput differed by 0.001 Mbps compared to the conventional Const. qcmd method, which was only a few packets worth of difference.

Table 5-10 Comparison of the simulation results of Const. qcmd and Delay-aware VMTwDEMA under different buffer sizes.

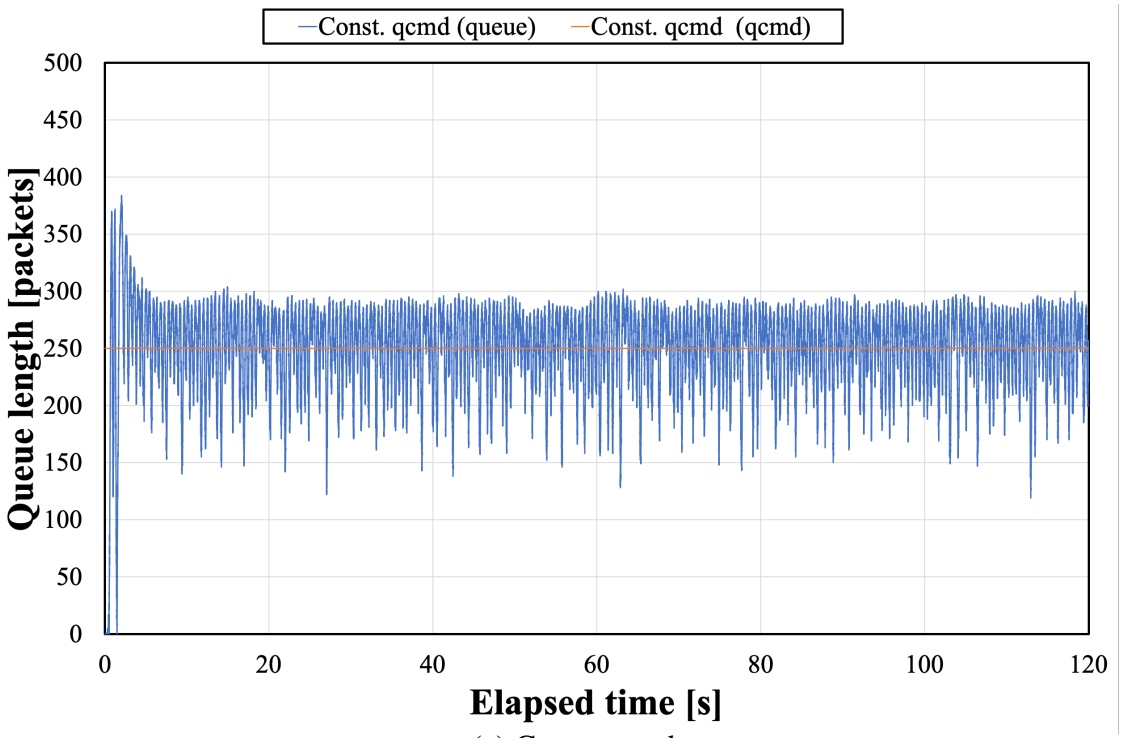
q_{lim}	200	500	1000
Average queue length [packets] (Const. qcmd)	100.00	250.03	500.00
Average queue length [packets] (Delay-aware VMTwDEMA)	15.15	131.92	63.96
Minimum queue length [packets] (Const. qcmd)	92	69	469
Minimum queue length [packets] (Delay-aware VMTwDEMA)	9	0	45
Average throughput of TCP flows [Mbps] (Const. qcmd)	9.999	49.999	99.998
Average throughput of TCP flows [Mbps] (Delay-aware VMTwDEMA)	9.999	49.998	99.998
Total number of buffer underflow samples (Const. qcmd)	0	0	0
Total number of buffer underflow samples (Delay-aware VMTwDEMA)	0	7	0

5.4.3.2 Latency of UDP Packets

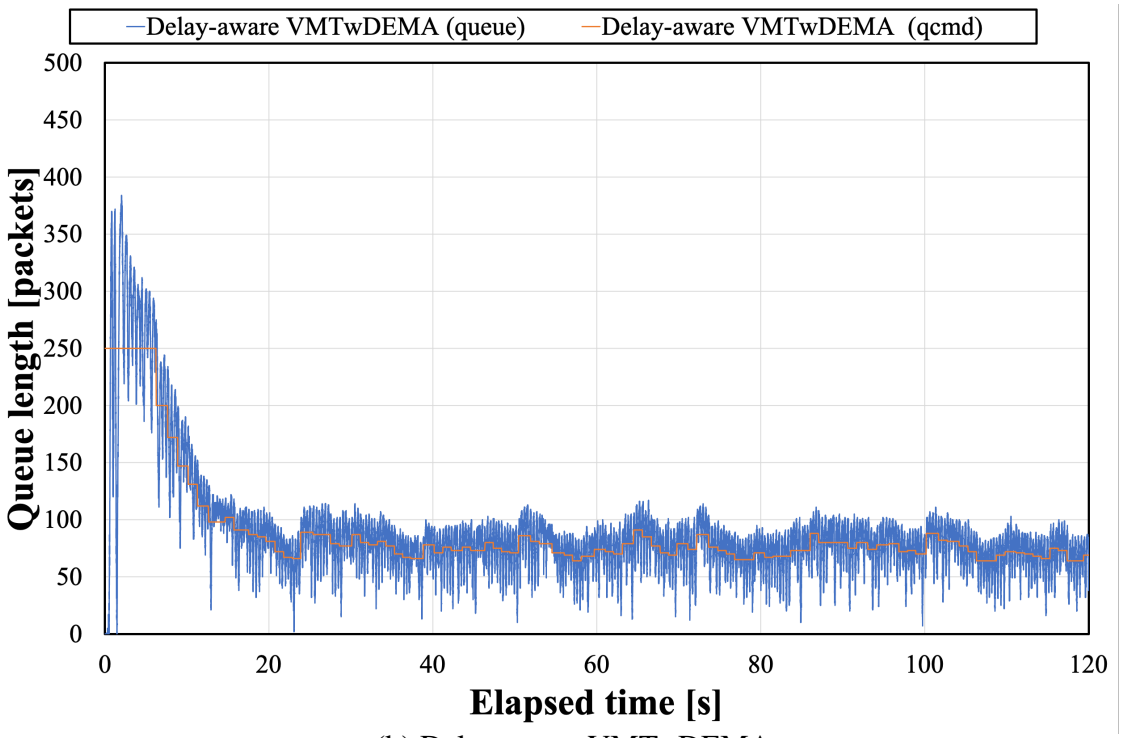
Figure 5-23 shows the queue length fluctuations of the two methods when an UDP traffic flow coexisted in the TCP/AQM network. The single UDP traffic started communication at 60 s of simulation duration and kept sending packets with a constant bit rate of 1 Mbps. The values of N, C, T_{p}, and q_{lim} were set to 100, 50 Mbps, 80 ms, and 500 packets, respectively. The general behavior of queue length fluctuation did not vastly change by including a single UDP flow, and the proposed Delay-aware VMTwDEMA method successfully lowered q_{cmd} and average queue length, lowering the queueing delay.

Table 5-11 shows the evaluation index data under the same simulation setups used for Fig. 5-23, expect with multiple values of T_{p}. In addition to the evaluation index data shown in Table 5-10, Table 5-11 shows the average UDP packet propagation delay and calculated UDP packet queueing delay. The average UDP packet propagation delay denotes the average duration of all UDP packets to be conveyed from the sender host to the receiver host. The estimated UDP packet queueing delay denotes the value of average UDP packet propagation delay subtracted by $T_{\mathrm{p}}/2$ which is equal to the total forward propagation latency. This value shows the pure additional delay caused by the queue.

As shown in Table 5-11, the proposed Delay-aware VMTwDEMA method lowered q_{cmd}



(a) Const. qcmd



(b) Delay-aware VMTwDEMA

Fig. 5-23 Queue length fluctuations when an UDP traffic flow coexists.

Table 5-11 Summary of simulation data when an UDP traffic flow coexists.

T_p	20	40	60	80	100
Average queue length [packets] (Const. qcmd)	250.05	250.12	250.02	250.06	249.92
Average queue length [packets] (Delay-aware VMTwDEMA)	37.59	41.56	53.44	74.25	118.75
Minimum queue length [packets] (Const. qcmd)	217	191	157	119	92
Minimum queue length [packets] (Delay-aware VMTwDEMA)	20	13	14	7	0
Average throughput of TCP flows [Mbps] (Const. qcmd)	49.051	49.039	49.0267	49.021	49.012
Average throughput of TCP flows [Mbps] (Delay-aware VMTwDEMA)	49.110	49.068	49.049	49.033	49.018
Total number of buffer underflow samples (Const. qcmd)	0	0	0	0	0
Total number of buffer underflow samples (Delay-aware VMTwDEMA)	0	0	0	0	17
Average UDP packet propagation delay [s] (Const. qcmd)	0.052	0.062	0.072	0.082	0.092
Average UDP packet propagation delay [s] (Delay-aware VMTwDEMA)	0.017	0.027	0.039	0.053	0.075
Estimated UDP packet queueing delay [s] (Const. qcmd)	0.042	0.042	0.042	0.042	0.042
Estimated UDP packet queueing delay [s] (Delay-aware VMTwDEMA)	0.007	0.007	0.009	0.013	0.025
T_p	120	140	160	180	200
Average queue length [packets] (Const. qcmd)	249.81	250.14	251.00	250.63	249.19
Average queue length [packets] (Delay-aware VMTwDEMA)	164.87	202.74	232.88	250.16	249.19
Minimum queue length [packets] (Const. qcmd)	58	42	0	0	0
Minimum queue length [packets] (Delay-aware VMTwDEMA)	0	0	0	0	0
Average throughput of TCP flows [Mbps] (Const. qcmd)	49.012	49.010	48.998	48.962	48.878
Average throughput of TCP flows [Mbps] (Delay-aware VMTwDEMA)	48.989	48.971	48.989	48.962	48.878
Total number of buffer underflow samples (Const. qcmd)	0	0	37	166	343
Total number of buffer underflow samples (Delay-aware VMTwDEMA)	103	123	59	160	343
Average UDP packet propagation delay [s] (Const. qcmd)	0.102	0.112	0.122	0.132	0.142
Average UDP packet propagation delay [s] (Delay-aware VMTwDEMA)	0.092	0.106	0.121	0.132	0.142
Estimated UDP packet queueing delay [s] (Const. qcmd)	0.042	0.042	0.042	0.042	0.042
Estimated UDP packet queueing delay [s] (Delay-aware VMTwDEMA)	0.032	0.036	0.041	0.042	0.042

and the average queue length. When T_p got larger, the queue length oscillation amplitude got larger, as it can be seen with the relationship between the average and minimum queue length of Const. qcmd method. Thus, the algorithm got limited with how much it lowered q_{cmd} under a higher propagation delay, being not much effective when T_p = 160, 180, or 200 ms. However, the proposed Delay-aware VMTwDEMA method did not degrade the communication efficiency in such situations, giving no negative effect even under high-latency network scenario.

On the other hand, the reduction of UDP packet propagation delay by the proposed Delay-aware VMTwDEMA method is notable, especially when T_p is small. This is because the proposed algorithm decreased the average queue length, lowering the average queueing delay as well. The proposed Delay-aware VMTwDEMA method also raised the throughput under a small T_p, as shown in Table 5-11. This is be because the smaller queueing delay made the control delay in the TCP/AQM network R smaller and made the system become more stable.

5.5 Summary

In this chapter, the algorithm that dynamically generated the target queue length considering the QoS for a control theory based AQM controller was proposed. The algorithm attempts to raise the goodput or lower the communication delay of the system, by raising or lowering the average queue length, respectively. In addition to simply raising or lowering the target queue length, the algorithm needed to consider avoiding buffer overflows or buffer underflows in the process.

The effectiveness of the proposed method was validated by performing simulations using ns-2. The simulations were performed under multiple scenarios by changing the parameters that mainly affect the behavior of the TCP/AQM network. The three proposed Loss-aware mode methods, i.e., CMT, VMTwAC, and Loss-aware VMTwDEMA, all showed their ability to raise the goodput. The Loss-aware VMTwDEMA method showed multiple improvements compared to the other two methods. The proposed Delay-aware mode method, i.e., Delay-aware

VMTwDEMA, showed its ability to lower the communication latency. The simulation results showed that the proposed algorithm successfully raised or lowered the target queue length and increased goodput or decreased queueing delay, achieving communication with improved QoS.

The proposed method of using an algorithm to control the target queue length can improve the QoS of the network communication without constructing any additional infrastructures or applying a new communication protocol. The additional calculations that the proposed algorithms require are estimated to be less than 100 times, including both comparisons and standard four arithmetic operations, per 0.001 s. In addition, the additional variables that the proposed algorithms require to store are about a dozen, which would be a minuscule portion of the memory built in the router. Considering the calculation capability of routers used in the modern Internet society, these additional computation costs would be negligibly small. This small computation cost would be beneficial for implementing the proposed methods to low-cost low-powered routing devices, such as a mobile router capable of managing numerous IoT devices for example.

Future works include the implementation of ECN to improve link utilization and simulations using more complicated scenarios such as those employing multiple bottleneck routers.

Chapter 6

Conclusion

This thesis proposed a remote TCP/AQM congestion control system with butterfly-shaped PDC, a robust congestion control system with tolerance for high-latency network, and adaptive target queue length generation algorithms for QoS-aware congestion control.

In chapter 3, the remote congestion control system with butterfly-shaped PDC was proposed. The delay compensator butterfly-shaped PDC was implemented to the remote TCP/AQM network congestion control system in order to compensate for the network delay induced by the NCS. The simulation results showed that the proposed controller with the butterfly-shaped PDC effectively stabilized the TCP/AQM network even if the system included time-varying delays.

In chapter 4, the AQM congestion control system using a PD controller, DOB, and SP in an integrated manner was proposed. The DOB was implemented to compensate for the control disturbances such as modeling errors and parameter fluctuations, and the SP was implemented to compensate for the control delays of the high-latency TCP/AQM network. The technical difficulty with the integrated implementation of the DOB and SP to the nonlinear TCP/AQM network was discussed, and the method to overcome the difficulty was presented. The simulation results showed that the proposed PD+DOB+SP method generally achieved the highest throughput when compared with the conventional methods.

In chapter 5, the algorithm that adaptively generates the target queue length in order to

improve the QoS was proposed. The proposed algorithm had two modes, i.e., the Loss-aware mode and Delay-aware mode. The proposed Loss-aware mode algorithm successfully lowered the packet loss ratio, increasing the goodput of the TCP/AQM network. The proposed Delay-aware mode algorithm successfully lowered the queueing delay, decreasing the communication latency of the network.

This research contributes for improving the QoS of AQM congestion control, making it support the diversified communication services. Along with the arise of the IoT technology, demands for efficient online communications are increasing. The online services under a high-latency network can increase in the future. The preference of users may be more efficient goodput or smaller communication latency. Therefore, the proposed methods are important for improving the QoS in the future of Internet communication technologies.

Currently, the raising demands for live online contents combined with electronic devices with relatively high communication capabilities tend to shift everyone attention to the "low-latency" communications. However, the electronic contents that can be created is increasing their size rapidly. Virtual reality (VR) is the current example, and this may be followed by holographic or 3D movies that can be replayed at home. If a new communication protocol like TCP but being compatible with such a large-sized content emerge in the future, all the techniques proposed in this thesis may be applied in the same form. Such a new-generation TCP would be like the packet size being 1 MB, routers having buffering capacity of 1 GB, and the link capacity being in TB order. The proposed AQM scheme would still be applied to such a scenario, where only some parameters need to be proportionally magnified. In addition, the high-latency network environment would keep on existing in the future, such as marine communications and interstellar communications. The technique of delay compensation would be useful in a same manner in the future.

References

[1] Ministry of Internal Affairs and Communications of Japan,
"Result of communication use trend research in Reiwa 1st,"
https://www.soumu.go.jp/menu_news/s-news/01tsushin02_02000148.html, May 2020.

[2] Statistics Bureau, Ministry of Internal Affairs and Communications of Japan,
"Overview of population estimation result,"
https://www.stat.go.jp/data/jinsui/2.html, October 2020.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash,
"Internet of things: A survey on enabling technologies, protocols, and applications,"
IEEE Communications Surveys & Tutorials, Vol. 17, No. 4, pp. 2347–2376, Fourthquarter 2015.

[4] J. Granjal, E. Monteiro, and J. Sá Silva,
"Security for the internet of things: A survey of existing protocols and open research issues,"
IEEE Communications Surveys & Tutorials, Vol. 17, No. 3, pp. 1294–1312, Thirdquarter 2015.

[5] S. Bera, S. Misra, and A. V. Vasilakos,
"Software-defined networking for internet of things: A survey,"
IEEE Internet of Things Journal, Vol. 4, No. 6, pp. 1994–2008, December 2017.

[6] A. Molina-Garcia, J. A. Fuentes, E. Gomez-Lazaro, A. Bonastre, J. C. Campelo, and J. J. Serrano,
"Development and assessment of wireless sensor and actuator network for heating and cooling loads,"
IEEE Transactions on Smart Grid, Vol. 3, No. 3, pp. 1192–1202, September 2012.

[7] J. Postel,
Internet Protocol, document RFC 791, September 1981.

[8] S. Deering and R. Hinden,
Internet Protocol, Version 6 (IPv6) Specification, document RFC 8200, July 2017.

[9] J. Arkko and M. Townsley,
IPv4 Run-Out and IPv4-IPv6 Co-Existence Scenarios, document RFC 6127, May 2011.

[10] Ministry of Internal Affairs and Communications of Japan,
"Collection and calculation of the Internet traffics in our country,"
https://www.soumu.go.jp/menu_news/s-news/01kiban04_02000171.html, July 2020.

[11] J. Postel,
Transmission control protocol, document RFC 793, September 1981.

[12] K. Chou,
"Periodic observation report,"
Internet Infrastructure Review (IIR), Vol. 48, pp. 4–9, September 2020.

[13] K. Chou,
"Broadband traffic report,"
Internet Infrastructure Review (IIR), Vol. 32, pp. 28–33, August 2016.

[14] J. Postel,
User Datagram Protocol, document RFC 768, August 1980.

[15] S. Kent and R. Atkinson,
IP Encapsulating Security Payload (ESP), document RFC 2406, November 1998.

[16] R. Shorten, C. King, F. Wirth, and D. Leith,
"Modelling TCP congestion control dynamics in drop-tail environments,"
Automatica, Vol. 43, No. 3, pp.441–449, March 2007.

[17] A. De Vendictis, A. Baiocchi, and M. Bonacci,
"Analysis and enhancement of TCP Vegas congestion control in a mixed TCP Vegas and TCP Reno network scenario,"
Performance Evaluation, Vol. 53, No. 3, pp. 225–253, August 2003.

[18] V. Jacobson,
"Congestion avoidance and control,"
ACM SIGCOMM Computer Communication Review, Vol. 18, No. 4, pp. 314–329, August 1998.

[19] S. Low, F. Paganini, and J. Doyle,
"Internet congestion control: An analytical perspective,"
IEEE Control Systems Magazine, Vol. 22, No. 1, pp. 28–43, February 2002.

[20] H. Wang, H. Xin, D. S. Reeves, and K. G. Shin,
"A simple refinement of slow-start of TCP congestion control,"
in *Proceedings of ISCC 2000. Fifth IEEE Symposium on Computers and Communications (ISCC)*, July 2000, pp. 98–105.

[21] W. Lautenschlaeger and A. Francini,
"Global synchronization protection for bandwidth sharing TCP flows in high-speed links,"
in *Proceedings of IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, July 2015, pp. 1–8.

[22] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang,
Recommendations on Queue Management and Congestion Avoidance in the Internet, document RFC 2309, April 1998.

[23] R. Adams,
"Active queue management: A survey,"
IEEE Communications Surveys and Tutorials, Vol. 15, No. 3, pp. 1425–1476, Thirdquarter 2013.

[24] V. Kushawaha and R. Gupta,
"Congestion control for high-speed wired networks: A systematic literature review,"
Journal of Network and Computer Applications, Vol. 45, pp 62–78, October 2014.

[25] H. Wang, M. J. Cheng, and Z. H. Tian,
"Robust buffer management mechanism in quality of service routers,"
Journal of Shanghai Jiaotong University (Science), Vol. 16, No. 4, pp. 452–458, August 2011.

[26] L. Le, J. Aikat, K. Jeffay, and F. D. Smith,
"The effects of active queue management and explicit congestion notification on web performance,"
IEEE/ACM Transactions on Networking, Vol. 15, No. 6, pp. 1217–1230, December 2007.

[27] L. Pingping and Z. Lianying,
"The Research of Adaptive Network Congestion Control Algorithm Based on AQM,"
in *Proceedings of the 2009 International Forum on Information Technology and Applications (IFITA)*, May 2009, pp. 123–125.

[28] T. Wen, C. Chen, Z. Ding, and T. C. Yang,
"A novel AQM scheme for wireless networks with BER estimation,"
in *Proceedings of the 17th IFAC World Congress*, July 2008, Vol. 41, No. 2, pp. 2919–2924.

[29] S. Floyd and V. Jacobson,
"Random early detection gateways for congestion avoidance,"
IEEE/ACM Transactions on Networking, Vol. 1, No. 4, pp. 397–413, August 1993.

[30] T. Bonald, M. May, and J. C. Bolot,
"Analytic evaluation of RED performance,"
in *Proceedings of the International Conference on Computer Communications (INFOCOM)*, March 2000, Vol. 3, pp. 1415–1424.

[31] X. Chen, S. Wong, and C. K. Tse,
"Adding randomness to modeling internet TCP-RED systems with interactive gateways,"
IEEE Transactions on Circuits and Systems II: Express Briefs, Vol. 57, No. 4, pp. 300–304, April 2010.

[32] S. K. Mohapatra, S. K. Bisoy, and P. K. Dash,
"Stability analysis of active queue management techniques,"
in *Proceedings of the 2015 International Conference on Man and Machine Interfacing (MAMI)*, December 2015, pp. 1–6.

[33] S. Patel, B. Gupta, and V. Sharma,
"Throughput analysis of AQM schemes under low-rate denial of service attacks,"
in *Proceedings of the 2016 International Conference on Computing, Communication and Automation (ICCCA)*, April 2016, pp. 551–554.

[34] S. Floyd, R. Gummadi, and S. Shenker,
"Adaptive RED: An algorithm for increasing the robustness of RED's active queue management,"
http://www.icir.org/floyd/red.html , August 2001.

[35] D. Lin and R. Morris,
"Dynamics of random early detection,"
ACM SIGCOMM Computer Communication Review, Vol. 27, No. 24, pp. 127–137, October 1997.

[36] C. M. Patel,
"URED: Upper threshold RED an efficient congestion control algorithm,"
in *Proceedings of the 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, July 2013, pp. 1–5.

[37] C. Wang, J. Liu, B. Li, K. Sohraby, and Y. T. Hou,
"LRED: A robust and responsive AQM algorithm using packet loss ratio measurement,"
IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 1, pp. 29–43, January 2007.

[38] Q. Xu, and J. Sun,
"A simple active queue management based on the prediction of the packet arrival rate,"
Journal of Network and Computer Applications, Vol. 42, pp. 12–20, June 2014.

[39] K. Chavan, R. G. Kumar, M. N. Belur, and A. Karandikar,
"Robust active queue management for wireless networks,"
IEEE Transactions on Control Systems Technology, Vol. 19, No. 6, pp. 1630–1638, November 2011.

[40] K. Ramakrishnan, S. Floyd, and D. Black,
The Addition of Explicit Congestion Notification (ECN) to IP, document RFC 3168, September 2001.

[41] S. Floyd,
"TCP and explicit congestion notification,"
ACM SIGCOMM Computer Communication Review, Vol. 24, No. 5, pp. 1–23, October 1994.

[42] A. Kuzmanovic,
"The power of explicit congestion notification,"
ACM SIGCOMM Computer Communication Review, Vol. 35, No. 4, pp. 61–72, August 2005.

[43] W. Feng, K. G. Shin, D. D. Kandlur, and D. Saha,
"The BLUE active queue management algorithms,"
IEEE/ACM Transactions on Networking, Vol. 10, No. 4, pp. 513–528, August 2002.

[44] W. Chen, Y. Li, and S. Yang,
"An average queue weight parameterization in a network supporting TCP flows with RED,"
in *Proceedings of the 2007 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, April 2007, pp. 590–595.

[45] S. Woo and K. Kim,
"Tight upper bound for stability of TCP/RED systems in AQM routers,"
IEEE Communications Letters, Vol. 14, No. 7, pp. 682–684, July 2010.

[46] V. Misra, W. Gong, and D. Towsley,
"A fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED,"
ACM SIGCOMM Computer Communication Review, Vol. 30, No. 4, pp. 151–160, October 2000.

[47] C. V. Hollot, V. Misra, D. Towsley, and W. Gong,
"Analysis and design of controllers for AQM routers supporting TCP flows,"
IEEE Transactions on Automatic Control, Vol. 47, No. 6, pp. 945–959, June 2002.

[48] J. Aweya, M. Ouellette, and D. Y. Montuno,
"A control theoretic approach to active queue management,"
Computer Networks, Vol. 36, No. 2–3, pp. 203–235, July 2001.

[49] X. Deng, S. Yi, G. Kesidis, and C. R. Das,
"A control theoretic approach for designing adaptive AQM schemes,"
in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, December 2003, pp. 2947–2951.

[50] D. Aqrawal and F. Granelli,
"Redesigning an active queue management system,"
in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, November 2004, pp. 702–706.

[51] A. Haider, H. Sirisena, and K. Pawlikowwski,
"PID based congestion control algorithms for AQM routers supporting TCP/IP flows,"
IEICE Transactions on Communications, Vol. E87-B, No. 3, pp. 548–555, March 2004.

[52] F. Ren and C. Lin,
"Speed up the responsiveness of active queue management system,"
IEICE Transactions on Communications, Vol. E86-B, No. 2, pp. 630–636, February 2004.

[53] B. A. Sadek, T. E. Houssaine, and C. Noreddine,
"A robust PID controller for active queue management framework in congested routers,"
in *Proceedings of the 2017 International Conference on Intelligent Systems and Computer Vision (ISCV)*, April 2017, pp. 1–6.

[54] Q. Chen and O. W. W. Yang,
"On Designing Self-Tuning Controllers for AQM Routers Supporting TCP Flows Based on Pole Placement,"
IEEE Jounal on Selected Areas in Communications, Vol. 22, No. 10, pp. 1965–1974, December 2004.

[55] M. Y. Waskasi, M. J. Yazdanpanah, and N. Yazdani,
"A new active queue management algorithm based on neural networks PI,"
in *Proceedings of the 16th IFAC Triennial World Congress*, July 2005, Vol. 38, No. 1, pp. 1–6.

[56] J. Sun, G. Chen, K. Ko, S. Chan, and M. Zukerman,
"PD-controller: A new active queue management scheme,"
in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, December 2003, pp. 3103–3107.

[57] P. K. Dash and S. K. Bisoy,
"Analysis of AQM router of network supporting multiple TCP flows,"
in *Proceedings of the 2014 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, December 2014, pp. 1–5.

[58] S. K. Bisoy and P. K. Pattnaik,
"Design of feedback controller for TCP/AQM networks,"
Engineering Science and Technology, an International Journal, Vol. 20, No. 1, pp. 116–132, February 2017.

[59] H. C. Cho, M. S. Fadali, and H. Lee,
"Neural network control for TCP network congestion," in *Proceedings of the 2005 American Control Conference (ACC)*, June 2005, pp. 3480–3485.

[60] M. H. Y. Moghaddam,
"A fuzzy active queue management mechanism for Internet congestion control,"
in *Proceedings of the Third International Workshop on Advanced Computational Intelligence (IWACI)*, August 2010, pp. 203–208.

[61] Y. H. Aoul, A. Nafaa, D. Negru, and A. Mehaoua,
"FAFC: Fast adaptive fuzzy AQM controller for TCP/IP networks,"
in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, November 2004, pp. 1319–1323.

[62] N. E. Fezazi, S. B. Alaoui, F. E. Haoussi, E. H. Tissir, and T. Alvarez,
"A dynamic anti-windup AQM for congestion control in Internet,"
in *Proceedings of the IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, November 2016, pp. 1–6.

[63] H. Chibana, M. Tadokoro, D. Murayama, K. Suzuki, and R. Kubo,
"Robustness evaluation of disturbance observer-based active queue management supporting TCP flows,"
IEICE Communications Express, Vol. 3, No. 10, pp. 311–316, October 2014.

[64] A. Ghasempour and T. K. Moon,
"Optimizing the number of collectors in machine-to-machine advanced metering infrastructure architecture for Internet of Things-based smart grid,"
in *Proceedings of 2016 IEEE Green Technologies Conference (GreenTech)*, pp. 51–55, April 2016.

[65] H. Park, J. Park, and J. Kim,
"Network infrastructure for Giga internet service: Trial deployment and prospects,"
in *Proceedings of Digest of the 9th International Conference on Optical Internet (COIN*

2010), pp. 1–3, July 2010.

[66] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson,
"TCP Vegas: New Techniques for Congestion Detection and Avoidance,"
ACM SIGCOMM Computer Communication Review, Vol. 24, No. 4, pp. 24–35, October 1994.

[67] Y. Chan and H. Lee,
"A hybrid congestion control for TCP over high speed networks,"
in *Proceedings of 2012 Sixth International Conference on Genetic and Evolutionary Computing*, pp. 530–533, August 2012.

[68] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. K. Karn and F. Abdessamia,
"Network traffic classification techniques and comparative analysis using machine learning algorithms,"
in *Proceedings of 2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 2451–2455, October 2016.

[69] Q. Xu, G. Ma, K. Ding, and B. Xu,
"An adaptive active queue management based on model predictive control,"
IEEE Access, Vol. 8, pp. 174489–174494, September 2020.

[70] Y. Su, L. Huang, and C. Feng,
"QRED: A Q-Learning-based Active Queue Management Scheme,"
Journal of Internet Technology, Vol. 19, No. 4, pp. 1169–1178, July 2018.

[71] R. Hotchi and R. Kubo,
"Remote congestion control using model-free butterfly-shaped perfect delay compensator for active queue management supporting TCP flows,"
Nonlinear Theory and Its Applications (NOLTA), IEICE, Vol. 10, No. 2, pp. 157–172, April 2019.

[72] R. Hotchi, H. Chibana, T. Iwai, and R. Kubo,
"Active queue management supporting TCP flows using disturbance observer and Smith predictor,"
IEEE Access, Vol. 8, pp. 173401–173413, September 2020.

[73] R. Hotchi and R. Kubo,
"Active queue management supporting TCP flows using dynamically controlled target queue length,"
in *Proceedings of the 2018 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, May 2018, pp. 77–78.

[74] R. Hotchi and R. Kubo,
"Update cycle recalculation in active queue management using dynamically controlled target queue length,"
in *Proceedings of the 2019 International Symposium on Nonlinear Theory and Its Applications (NOLTA)*, December 2019, pp. 389–392.

[75] M. Allman, V. Paxson, and W. Stevens,
TCP Congestion Control, document RFC 2581, April 1999.

[76] T. Henderson, S. Floyd, and A. Gurtov,
The NewReno Modification to TCP's Fast Recovery Algorithm, document RFC 6582, April 2012.

[77] S. Bensley, D. Thaler, and P. Balasubramanian
Data Center TCP (DCTCP): TCP Congestion Control for Data Centers, document RFC 8257, October 2017.

[78] R. Kubo, J. Kani, and Y. Fujimoto,
"Advanced Internet congestion control using a disturbance observer,"
in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, December 2008, pp. 1–5.

[79] H. Chibana, M. Yoshino, M. Tadokoro, D. Murayama, K. Suzuki, and R. Kubo,
"Disturbance-observer-based active queue management with time delay using software-defined networking controller,"
in *Proceedings of the 41st Annual Conference of the IEEE Industrial Electronics Society (IECON)*, November 2015, pp. 1049–1054.

[80] J. M. Zhang and S. Q. Wang,
"Networked control system design and implementation,"
in *Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC)*, November 2002, pp. 750–753.

[81] F. L. Lian, J. R. Moyne, and D. M. Tilbury,
"Performance evaluation of control networks: Ethernet ControlNet and DeviceNet,"
IEEE Control Systems Magazine, Vol. 21, No. 1, pp. 66–83, February 2001.

[82] J. Baillieul and P. J. Antsaklis,
"Control and communication challenges in networked real-time systems,"
Proceedings of the IEEE, Vol. 95, No. 1, pp. 9–28, January 2007.

[83] W. Zhang, M. S. Branicky, and S. M. Phillips,
"Stability of networked control systems,"
IEEE Control Systems Magazine, Vol. 21, No. 1, pp. 84–99, February 2001.

[84] E. Joelianto,
"Networked control systems: Time delays and robust control design issues,"
in *Proceedings of the 2nd International Conference on Instrumentation Control and Automation (ICA)*, November 2011, pp. 16–25.

[85] O. J. M. Smith,
"A controller to overcome dead time,"
ISA Journal, Vol. 6, No. 2, pp. 28–33, February 1959.

[86] A. Bahill,
"A simple adaptive Smith-predictor for controlling time-delay systems: A tutorial,"
IEEE Control Systems Magazine, Vol. 3, No. 2, pp. 16–22, May 1983.

[87] Y. Li, K. Ko, and G. Chen,
"A Smith predictor-based PI-controller for active queue management,"
IEICE Transactions on Communications, Vol. E88-B , No. 11, pp. 4293–4300, November 2005.

[88] C. L. Lai and P. L. Hsu,
"Design the remote control system with the time-delay estimator and the adaptive Smith predictor,"
IEEE Transactions on Industrial Informatics, Vol. 6, No. 1, pp. 73–80, February 2010.

[89] H. Ohsaki, H. Yamamoto, and M. Imase,
"SPRED: Active queue management mechanism for wide-area networks,"
in *Proceedings of the 2007 International Symposium on Applications and the Internet*

(SAINT '07), January 2007, pp. 531–537.

[90] C. L. Lai and P. L. Hsu,
"The butterfly-shaped feedback loop in networked control systems for the unknown delay compensation,"
IEEE Transactions on Industrial Informatics, Vol. 10, No. 3, pp. 1746–1754, August 2014.

[91] R. Kubo, J. Kani, and Y. Fujimoto,
"Congestion control in TCP/AQM networks using a disturbance observer,"
IEEJ Transactions on Industry Applications, Vol. 129, No. 6, pp. 541–547, June 2009. (in Japanese)

[92] L. Khoshnevisan, F.R. Salmasi, and V. Shah-Mansouri,
"Robust queue management for TCP-based large round trip time networks with wireless access link,"
in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, March 2015, pp. 1309–1313.

[93] K. Ohnishi, M. Shibata, and T. Murakami,
"Motion control for advanced mechatronics,"
IEEE/ASME Transactions on Mechatronics, Vol. 1, No. 1, pp. 56–67, March 1996.

[94] M. Lin, T. Ren, H. Yuan, and M. Li,
"The congestion control for TCP network based on input/output saturation,"
in *Proceedings of the 29th Chinese Control and Decision Conference (CCDC)*, May 2017, pp. 1166–1171.

[95] P. Wang, C. Zhu, and X. Yang,
"A novel AQM algorithm based on feedforward model predictive control,"
International Journal of Communication Systems, Vol. 31, No. 12, article e3711, August 2018.

[96] L. Khoshnevisan, X. Liu, and F.R. Salmasi,
"Predictive sliding-mode congestion control for wireless access networks with singular and non-singular control gain,"
IET Control Theory & Applications, Vol. 14, No. 13, pp. 1722–1732, August 2020.

[97] A. Kato, A. Muis, and K Ohnishi,
"Robust network motion control system based on disturbance observer,"
Automaika, Vol. 47, No. 1–2, pp. 5–10, January 2006.

[98] K. Hong and K. Nam,
"A load torque compensation scheme under the speed measurement delay,"
IEEE Transactions on Industrial Electronics, Vol. 45, No. 2 pp. 283–290, April 1998.

[99] K. Nichols and V. Jacobson,
"Controlling queue delay,"
ACM Queue, Vol. 10, No. 5, pp. 1–15, May 2012.

[100] R. Pan, P. Natarajan, C. Piglione, M.S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg,
"PIE: A lightweight control scheme to address the bufferbloat problem,"
in *Proceedings of the 14th IEEE International Conference on High Performance Switching and Routing (HPSR)*, Jul. 2013, pp. 148–155.

[101] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar,
Controlled delay active queue management,
document RFC 8289, January 2018.

[102] R. Pan, P. Natarajan, F. Baker, and G. White,
Proportional integral controller enhanced (PIE): A lightweight control scheme to address the bufferbloat problem,
document RFC 8033, February 2017.

[103] R. Jain, D. M. Chiu, and W. R. Hawe,
"A quantitative measure of fairness and discrimination for resource allocation in shared computer systems,"
DEC Research Report, TR–301, September 1984.

[104] T. Qi and H. Wang,
"PID sliding mode controller design and application to active queue management,"
in *Proceedings of 2016 35th Chinese Control Conference (CCC)*, July 2016, pp. 6917–6922.

[105] J. Gettys,
"Bufferbloat: Dark buffers in the Internet,"
IEEE Internet Computing, Vol. 15, No. 3, p. 96, May 2011.

[106] Z. Na and Q. Guo,
"An improved AQM scheme with adaptive reference queue threshold,"
in *Proceedings of the 6th International ICST Conference on Communications and Networking in China (CHINACOM)*, August 2011, pp. 589–593.

[107] H. Zhang, L. Song, B. Fan, and J. Jiang,
"Optimal buffer management algorithm with auto-tuning reference queue length,"
in *Proceedings of 2008 10th IEEE International Conference on High Performance Computing and Communications*, September 2008, pp. 418–424.

[108] S. H. Low, F. Paganini, J. Wang, and J. C. Doyle,
"Linear stability of TCP-RED and a scalable control,"
Computer Networks, Vol. 43, No. 5, pp. 633–647, December 2003.

Achievements

Journals (First Author)

[1] Ryosuke Hotchi and Ryogo Kubo,
"Remote congestion control system using model-free butterfly-shaped perfect delay compensator for active queue management supporting TCP flows,"
Nonlinear Theory and Its Applications, IEICE, Vol. 10, No. 2, pp. 157–172, April 2019.

[2] Ryosuke Hotchi, Hosho Chibana, Takanori Iwai, and Ryogo Kubo,
"Active queue management supporting TCP flows using disturbance observer and smith predictor,"
IEEE Access, Vol. 8, pp. 173401–173413, September 2020.

Journals (Co-author)

[1] Takaharu Yamanaka, Kenta Yamada, Ryosuke Hotchi, and Ryogo Kubo,
"Simultaneous time-delay and data-loss compensation for networked control systems with energy-efficient network interfaces,"
IEEE Access, Vol. 8, pp. 110082–110092, June 2020.

[2] Ryuta Sakamoto, Takahiro Shobudani, Ryosuke Hotchi, and Ryogo Kubo,
"QoE-aware stable adaptive video streaming using proportional-derivative controller for MPEG-DASH,"
IEICE Transactions on Communications, accepted for publication.

International Conferences (First Author)

[1] Ryosuke Hotchi, Hosho Chibana, and Ryogo Kubo,
"Active queue management using remote congestion controller with model-free butterfly-shaped perfect delay compensator,"
Proceedings of the 31st International Technical Conference on Circuit/Systems, Computers and Communications, ITC-CSCC 2016, Okinawa, Japan, pp. 755–758, July 10–13, 2016.

[2] Ryosuke Hotchi and Ryogo Kubo,
"Analysis of controller mismatch in AQM with butterfly-shaped perfect delay compensator,"
Proceedings of the 2017 International Symposium on Nonlinear Theory and Its Applications, NOLTA 2017, Cancun, Mexico, pp. 58–61, December 4–7, 2017.

[3] Ryosuke Hotchi and Ryogo Kubo,
"Active queue management supporting TCP flows using dynamically controlled target queue length,"
Proceedings of the IEEE International Conference on Consumer Electronics-Taiwan, ICCE-TW 2018, Taichung, Taiwan, pp. 77–78, May 19–21, 2018.

[4] Ryosuke Hotchi and Ryogo Kubo,
"Update cycle recalculation in active queue management using dynamically controlled target queue length,"
Proceedings of the 2019 International Symposium on Nonlinear Theory and Its Applications, NOLTA 2019, Kuala Lumpur, Malaysia, pp. 389–392, May 2–6, 2019.

International Conferences (Co-author)

[1] Ryuta Sakamoto, Takahiro Shobudani, Ryosuke Hotchi, and Ryogo Kubo,
"QoE assessment considering user preferences in PD-based stable adaptive streaming for

MPEG-DASH,"
Proceedings of the 2019 International Symposium on Nonlinear Theory and Its Applications, NOLTA 2019, Kuala Lumpur, Malaysia, pp. 389–392, May 2–6, 2019.

[2] Kento Aida, Kenta Yamada, Ryosuke Hotchi, and Ryogo Kubo,
"Dynamic redundant path selection for tamper-tolerant networked control,"
Proceedings of the 2019 International Symposium on Nonlinear Theory and Its Applications, NOLTA 2019, Kuala Lumpur, Malaysia, pp. 389–392, May 2–6, 2019.

[3] Takaharu Yamanaka, Kenta Yamada, Ryosuke Hotchi, and Ryogo Kubo,
"Time-delay and data-loss compensation using communication disturbance observer for energy-efficient networked control systems,"
Proceedings of the 2019 International Symposium on Nonlinear Theory and Its Applications, NOLTA 2019, Kuala Lumpur, Malaysia, pp. 389–392, May 2–6, 2019.

Domestic Conferences (First Author)

[1] Ryosuke Hotchi, Hosho Chibana and Ryogo Kubo,
"Model-free delay compensation using butterfly-shaped PDC for AQM with fluctuating control delays,"
Proceedings of the 2016 IEICE Society Conference, Vol. 2, p. 266, September 20–23, 2016, Hokkaido, Japan. (in Japanese)

[2] Ryosuke Hotchi, Hosho Chibana and Ryogo Kubo,
"Time-delay compensation for AQM using disturbance observer,"
Proceedings of the 3rd IEICE Communication Quality Workshop, p. 43, January 21, 2017, Osaka, Japan. (in Japanese)

[3] Ryosuke Hotchi and Ryogo Kubo,
"Effect of controller model parameter mismatch in AQM with butterfly-shaped PDC,"
Proceedings of the 2017 IEICE Society Conference, Vol. 2, p. 184, September 12–15, 2017, Tokyo, Japan. (in Japanese)

[4] Ryosuke Hotchi and Ryogo Kubo,
"A study of queue length control in TCP/AQM networks considering buffer capacity,"
IEICE Technical Report, Communication Quality, Vol. 118, No. 140, CQ2018–39, pp. 45–50, July 19–20, 2018, Miyagi, Japan. (in Japanese)

[5] Ryosuke Hotchi and Ryogo Kubo,
"A study of updating cycle of target queue length in AQM,"
Proceedings of the 2018 IEICE Society Conference, Vol. 2, p. 208, September 11–14, 2018, Ishikawa, Japan. (in Japanese)

Domestic Conferences (Co-author)

[1] Kenta Yamada, Ryosuke Hotchi, and Ryogo Kubo,
"Information hiding for networked control systems using steganography,"
IEICE Technical Report, Communication Quality, Vol. 118, No. 302, CQ2018–73, pp. 55–60, November 15–16, 2018, Ishikawa, Japan. (in Japanese)

[2] Kento Aida, Kenta Yamada, Ryosuke Hotchi, and Ryogo Kubo,
"Dynamic redundancy switching method of network paths for tamper-tolerant networked control,"
IEICE Technical Report, Communication Systems, Vol. 199, No. 6, CS2019–1, pp. 1–6, April 18-19, 2019, Osaka, Japan. (in Japanese)

[3] Ryuta Sakamoto, Takahiro Shobudani, Ryosuke Hotchi, and Ryogo Kubo,
"QoE-aware mitigation of bitrate fluctuation in MPEG-DASH,"
IEICE Technical Report, Communication Quality, Vol. 119, No. 7, CQ2019–1, pp. 1–6, April 18-19, 2019, Osaka, Japan. (in Japanese)

[4] Takaharu Yamanaka, Kenta Yamada, Ryosuke Hotchi, and Ryogo Kubo,
"Time-delay and data-loss compensation using communication disturbance observer for

networked control systems with power-saving function,"
IEICE Technical Report, Communication Quality, Vol. 119, No. 61, CQ2019–32, pp. 107–112, May 30–31, 2019, Hiroshima, Japan. (in Japanese)

[5] Ryuta Sakamoto, Ryosuke Hotchi, and Ryogo Kubo,
"Stable bitrate selection method for improving user QoE in MPEG-DASH,"
Proceedings of the 2020 IEICE General Conference, Vol. 2, pp. S–92–S–93, March 17–20, 2020, Hiroshima, Japan. (in Japanese)

[6] Koji Ochi, Ryuta Sakamoto, Ryosuke Hotchi, and Ryogo Kubo,
"A time-delay compensation technique for QoE-aware adaptive video streaming,"
IEICE Technical Report, Communication Systems, Vol. 120, No. 75, CS2020–9, pp. 33–38, June 25–26, 2020, Online. (in Japanese)

[7] Keisuke Tajima, Ryuta Sakamoto, Ryosuke Hotchi, and Ryogo Kubo,
"Waiting time control considering QoE fluctuation in web browsing,"
IEICE Technical Report, Communication Quality, Vol. 120, No. 76, CQ2020–16, pp. 77–82, June 25–26, 2020, Online. (in Japanese)

Awards

[1] Finalist of NOLTA 2017 Best Student Paper Award,
in *The 2017 International Symposium on Nonlinear Theory and Its Applications, NOLTA 2017*, December 7th, 2017, Cancun, Mexico.