Doctoral Dissertation

A Study on Performance Improvement of TCP using Forward Error Correction

Yurino Sato

March, 2019

Graduate School of Environmental Engineering

The University of Kitakyushu

Preface

The growth of the Internet has led to provide a wide variety of network services such as multimedia services, e-commerce, online banking and trading, social network services, and online games. Our desire in network services has changed significantly as the Internet grows. In particular, there is a great demand for real-time and delay-sensitive services to improve the quality of life. The quality of such services depends especially on transmission delays. It has been improved with the development of broadband network technologies, though the improvement will have limitations due to physical distance between communication ends. Therefore, further improvement should be achieved by improving the transmission efficiency of communication protocols to transmit data between the communication ends.

TCP is still commonly used as reliable data-transmission protocol although network environment for such services has significantly changed as described above. It generally estimates an available bandwidth of networks on the basis of packet losses due to congestion. Lost packets are recovered by retransmissions and the transmission rate is kept low while the lost packets are recovered. Since it needs at least “one round-trip time” to recover lost packets, a long recovery time causes the degradation of service quality. By this “reactive” control to recover lost packets, TCP has an essential problem to provide real-time and delay-sensitive services. To prevent this problem, the number of retransmissions must be kept as low as possible.

One efficient way to prevent packet losses by "proactive" control is to apply a technology called "forward error correction" (FEC). FEC enables a sender to transmit packets with redundant information to recover lost packets by the information at a receiver. The recovery success rate depends on the amount of redundant information; however, redundant information places an additional load on the network. It is typically used for UDP communication, which has a constant transmission rate, while it is difficult to adapt to TCP communication, where transmission rate changes often, because it is harder to select an appropriate redundancy level according to network conditions. For this reason, although there have been few studies on generally applying FEC to TCP operations, there have been several studies on TCP restrictively using it.

This study aims to improve TCP performance for real-time and delay-sensitive services. To achieve this, I propose schemes to apply FEC to the entire TCP operation. The effectiveness of the proposed schemes is demonstrated through simulation evaluations.

Chapter 2 introduces related works and basic knowledge which are necessary for understanding this dissertation. For example, TCP's congestion control and error correction technology are explained.

In Chapter 3, I first consider a scheme to simply apply FEC technology to the entire TCP operation. The proposed scheme dynamically controls redundancy level according to transmission rates. I investigate the fundamental characteristics of the proposed scheme focusing on the redundancy, especially including in a high-latency environment. Simulation evaluations have shown that the proposed scheme enables higher throughput than the conventional schemes, especially in high-latency environments.

In Chapter 4, I examine the characteristics of the proposed scheme in detail.

A simple application of FEC to TCP operation might not work effectively. If the redundancy is too low, lost packets might not be recovered effectively. Moreover, unnecessary retransmissions are possibly caused due to the reception of duplicate ACKs even if recovery is successful. Therefore, I extend the proposed scheme to consider the ways to limit minimum redundancy and suppress the return of duplicate ACKs. I investigate the characteristics of the proposed scheme focusing on the introduced functions in an environment where random packet losses occur. Simulation evaluations have shown that the proposed scheme improves throughput significantly by suppressing the return of duplicate ACKs and controlling minimum redundancy, especially in high-latency environments, although FEC technology cannot work effectively when simply applied to TCP operation.

In Chapter 5, I consider adapting the proposed scheme to real environments. FEC cannot recover lost packets if both of original and redundant packets are "burstily" lost in a network. In addition, when FEC recovers lost packets, congestion does not be controlled through original TCP operations. Therefore, I further extend the proposed scheme to consider the ways to interleave redundant packets from original packets and control transmission rates when recovery is successful. I investigate the characteristics of the proposed scheme focusing on these functions in a real environment where burst packet losses occur. Evaluations of various characteristics by simulation have shown that the proposed scheme offers higher TCP throughput performance than the conventional scheme by recovering lost packets effectively.

Finally, this research is concluded in Chapter 6. I hope that this dissertation will be helpful for further study in this field.

Acknowledgements

I would like to acknowledge the support and encouragement received from a number of people for several years.

First of all, I wish to express my sincere appreciation to Associate Professor Hiroyuki Koga of The University of Kitakyushu. His constant encouragement, guidance through this research, invaluable discussions and advice have greatly helped in accomplishing the research. I also thank him for his careful reading of all papers on the research.

I would also like to express my gratitude to Professor Takeshi Ikenaga of the Kyushu Institute of Technology for his valuable comments, time and help in completing the research.

I wish to thank Associate Professor Katsuyoshi Iida of Hokkaido University for valuable comments, time and help in completing the research. His steady support has greatly helped my study.

I am very grateful to Professor Satoshi Uehara, Professor Takashi Satoh, Professor Gao Weijun, and Associate Professor Yasushi Yamazaki of The University of Kitakyushu for their advice and comments.

I extend thanks to Mr. Sho Nishimura, Mr. Kentaro Kameyama, Mr. Hiroki Mihara, Mr. Yuichi Iwashita, Mr. Makoto Shinada, Ms. Mei Yoshino, and all other members of the Network Engineering Research Group at The University of Kitakyushu for their kindly supports and valuable discussions.

Finally, my greatest appreciation goes to my family. They perpetually helped me whenever I faced various problems. Their long standing support has enable me to complete my degree.

Contents

List of Figures

List of Tables

1 Introduction

The growth of the Internet has led to provide a wide variety of network services such as multimedia services, e-commerce, online banking and trading, social network services, and online games. The Internet has been developed to support character-based transmission that is mainly used by legacy services such as e-mail and world wide web (WWW) browsing based on end-to-end basis. With the development of network technologies including high-speed broadband networks like 5G cellular networks, IEEE 802.11ac wireless LAN, and terabit optical fiber transmission, it has enabled providing higher quality multimedia services (e.g., streaming video and real-time applications) as well as will expect supporting delay-sensitive services (e.g., autonomous driving and mixed reality) in near future. The Internet is now essential to a rich and comfortable life in various fields of human society.

1.1 Real-time and Delay-sensitive Services

Our desire in network services has changed significantly as the Internet grows. In particular, there is a great demand for real-time and delay-sensitive services to improve the quality of life [1]. The quality of such services depends on not only transmission rates but also transmission delays. It has been improved with the development of broadband network technologies, though the improvement will

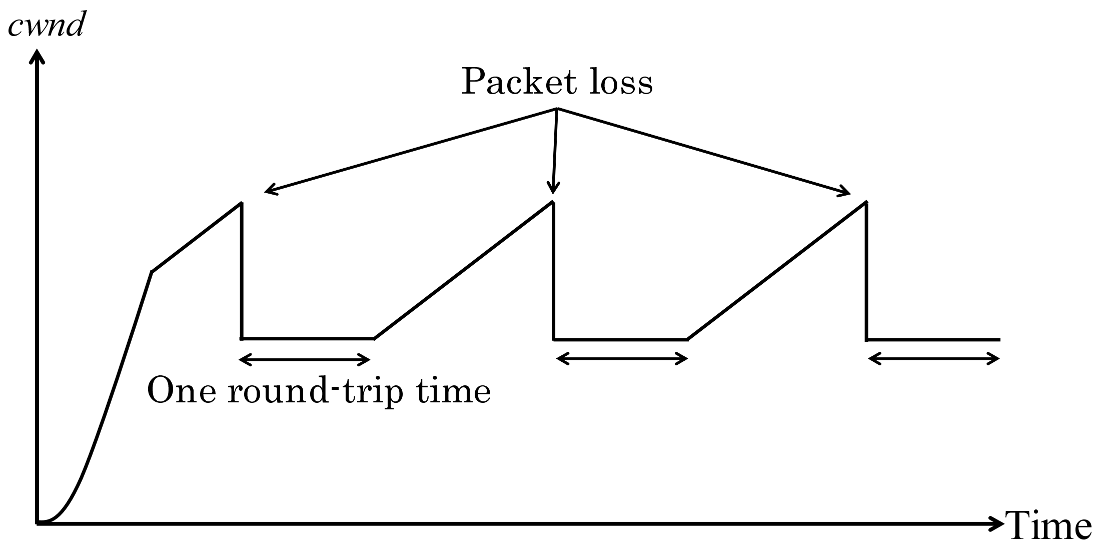


Figure 1.1: Example of transmission rate change

have limitations in terms of transmission delays due to physical distance between communication ends. Therefore, further improvement should be achieved by improving the transmission efficiency of communication protocols to transmit data between the communication ends. Although network environment for such services has significantly changed as described above, TCP [2] is still commonly used as reliable data-transmission protocol.It has been also improved according to the development of network environment, though it still has essential problems to provide such services.

1.2 Issues of Transport Protocol

TCP controls congestion by adjusting transmission rates according to network conditions [3]. It generally estimates an available bandwidth of networks on the basis of packet losses due to congestion. Lost packets are recovered by retransmissions and the transmission rate is kept low while the lost packets are recovered. It needs at least “one round-trip time” to recover lost packets. Since the recovery time grows longer, especially in high-latency environments, TCP performance is degraded. For real-time and delay-sensitive services, the long recovery time causes the degradation of service quality. An example of transmission rate change, i.e., congestion window size (*cwnd*), is shown in Figure 1.1. By this “reactive” control to recover lost packets, TCP has an essential problem to provide real-time and delay-sensitive services. To prevent this problem, the number of retransmissions must be kept as low as possible.

One efficient way to prevent packet losses by “proactive” control is to apply a technology called “forward error correction” (FEC) [4]. FEC enables a sender to transmit packets with redundant information so that lost packets can be recovered at a receiver. The success rate of recovery depends on the amount of redundant information; however, redundant information places an additional load on the network. To use FEC effectively, the amount of redundant information must be appropriately determined according to network conditions. Therefore, it is typically used for UDP [5] communication, which has a constant transmission rate, while it is difficult to adapt to TCP communication, in which transmission rate changes often. For that reason, although there have been few studies on generally applying FEC to TCP operations, there have been several studies on restrictively applying it to TCP operations.

1.3 Overview of This Dissertation

This study aims to improve TCP performance for real-time and delay-sensitive services. To achieve this, I propose schemes to apply FEC to the entire TCP operation. The effectiveness of the proposed schemes is demonstrated through simulation evaluations.

In Chapter 2, I introduce related works and basic knowledge which are necessary for understanding this dissertation. For example, TCP's congestion control and error correction technology are explained.

In Chapter 3, I first consider a scheme to simply apply FEC technology to the entire TCP operation. The proposed scheme dynamically controls redundancy level according to transmission rates. I investigate the fundamental characteristics of the proposed scheme focusing on the redundancy, especially including in a high-latency environment.

In chapter 4, I examine the characteristics of the proposed scheme in detail. A simple application of FEC to TCP operation might not work effectively. If the redundancy is too low, lost packets might not be recovered effectively. Moreover, unnecessary retransmissions are possibly caused due to the reception of duplicate ACKs even if recovery is successful. Therefore, I extend the proposed scheme to consider the ways to limit minimum redundancy and suppress the return of duplicate ACKs. I investigate the characteristics of the proposed scheme focusing on the introduced functions in an environment where random packet losses occur.

In Chapter 5, I consider adapting the proposed scheme to real environments. FEC cannot recover lost packets if both of original and redundant packets are "burstily" lost in a network. In addition, when FEC recovers lost packets, congestion does not be controlled through original TCP operations. Therefore, I further extend the proposed scheme to consider the ways to interleave redundant packets from original packets and control transmission rates when recovery is successful. I investigate the characteristics of the proposed scheme focusing on these functions in a real environment where burst packet losses occur, before concluding in Chapter 6.

The results discussed in Chapter 4 are mainly taken from [6, 7] and Chapter 5 from [8].

2 Related Works

In this chapter, I introduce related works and basic knowledge which are necessary for understanding this dissertation. TCP's congestion control and error correction technology are particularly explained.

2.1 TCP/IP

The Internet is a computer network that are interconnected multi-billion of computing devices throughout world wide. These computing device are primarily conventional desktop PCs and servers that store and transmit information such as Web page and e-mail message. In Internet technical term, these computing devices are called hosts or end systems and block of data are called packet that are transmitted among end systems. Recently, unconventional end systems such as laptop, smartphones, tablets, Web cameras and sensing devices are increasingly being connected to the Internet. End systems are connected Internet by communication links. There are many types of communication links, which are consisted of different types of physical media, including coaxial cable, copper wire, optical fiber, and wireless link. When end systems are connected Internet by different communication links, data transmits at different rates. Transmission rate of communication links are called "link bandwidth" that measures in bits/second.

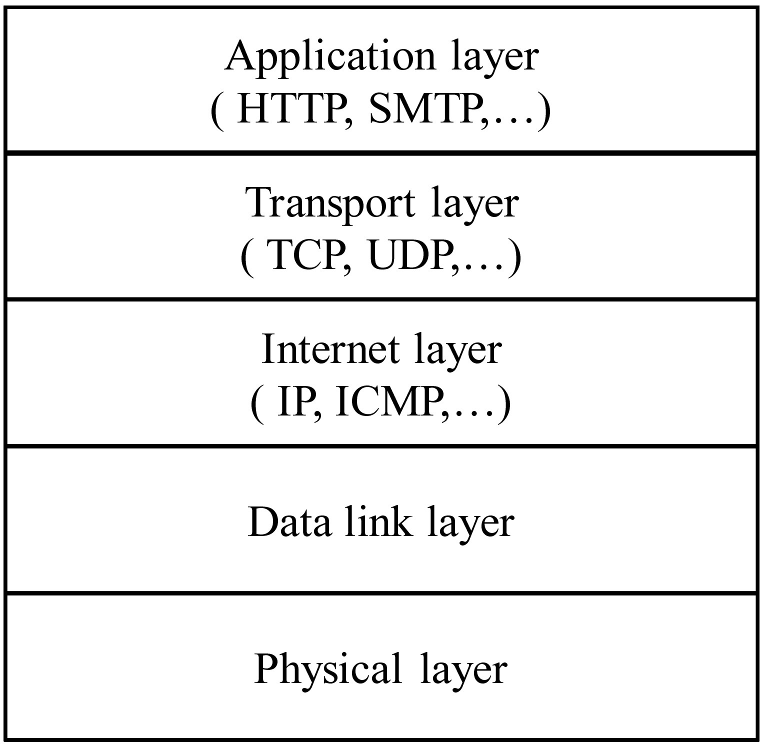


Figure 2.1: TCP/IP model

End system of Internet run many protocols that control the sending and receiving of data or information. Particularly, Transmission Control Protocol (TCP) and Internet Protocol (IP) are most important protocols in the Internet. TCP has a function of transmitting and receiving data between applications. IP modifies the format of the packets and creates path to send/receive among routers and end systems.

The Internet's principal protocols are collectively called as TCP/IP. These are hierarchically layered as shown in Figure 2.1. Packets are divided to manage in each layers while transmitting. Transport layer managed packet as segment and Internet layer manages packet as datagram.

40 years have passed since TCP/IP protocol was born. Along remarkable

progress of network technology, TCP has been variously improved. Although network users desire to speed-up the network, TCP has remarkable problems for delay-sensitive service as mentioned above.

2.1.1 Connection Oriented

TCP is a connection-oriented protocol. It needs to establish a TCP connection called a process before transmitting data between applications. When multiple processes run between end systems, each process is distinguished by unique port number on a corresponding end system. Each process can establish independent TCP connections by using port number. Each TCP connection is identified by four information; the source IP address and TCP port number, the destination IP address and port number.

2.2 TCP

TCP is flexibly designed, it can be applied to various environments using a method of communicating while estimating the character of the communication path and the congestion situation of the network. TCP uses congestion control, retransmission control, window control and flow control to cope with network congestion, packet loss, duplication packet, and packet reordering. Especially, congestion control is most important one of network technology, it increases/decreases the transmission rate according to estimate the congestion state of the network while transmitting packets. Also, TCP provides reliability by retransmission control that retransmits lost packet. Through these controls, TCP realizes efficient and highly reliable communication.

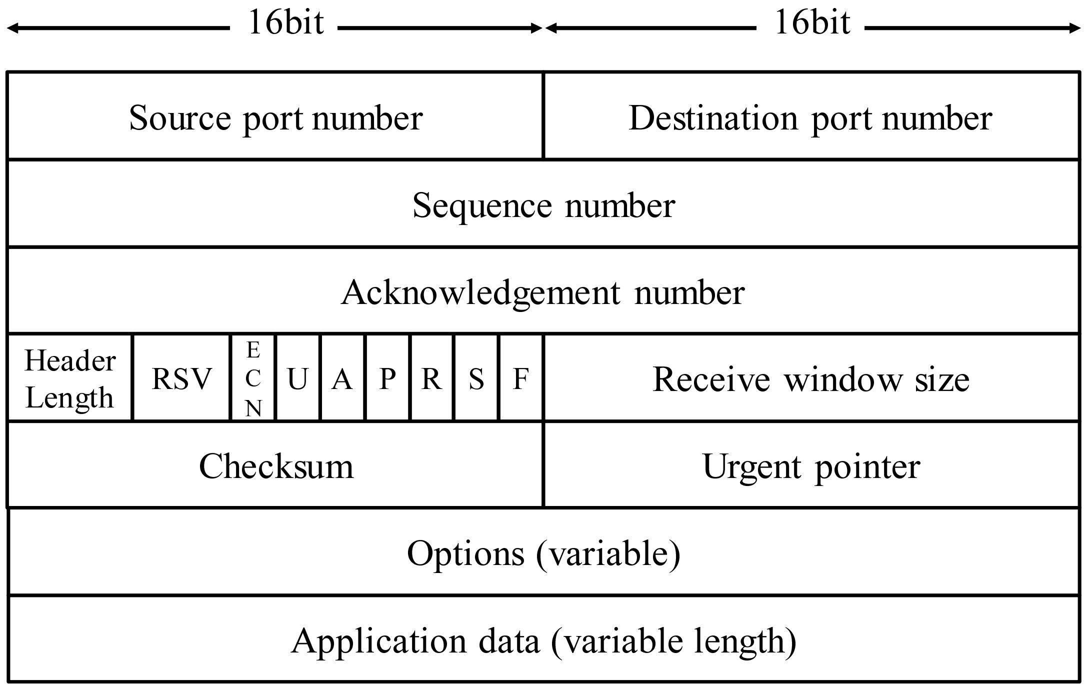


Figure 2.2: Structure of TCP header

2.2.1 TCP Header

To understand the operation of TCP, I explain structure of TCP header. Each TCP segment consists of two parts; One is a header part having 20 bytes fixed length and the other is a payload part having variable length including data. The header has information to control TCP connection. Figure 2.2 shows structure of TCP header. Here, the meaning of each field is as follows.

- Source port number (16 bits)
  - The application on the sender is identified by the only port number. Multiple TCP connections can be multiplexed by the same protocol process using port number.
- Destination port number (16 bits)

  - The application on the destination is identified by the only port number.

- Sequence number (32 bits)
  - This field means the first byte number of each TCP segment. Sequence numbers are used to reconstruct data with TCP segments and to reorder arrived TCP segments. TCP applies consecutive numbers to each byte. For example, if the sequence number of the current segment is 1000 and the length of the data segment is 1234 bytes, then the sequence number of the next segment is 2234. The sender uses this field to inform the sending status to destination.

- Acknowledgement number (32 bits)
  - The destination uses this field to inform the reception status to sender.

- Header length (4 bits)
  - This 32-bit field indicates the header length. In the usual case, the header length is fixed by 20 bytes. However, when the option field is set, the header length is variable length.

- RSV (4 bits)
  - This field is reserved and used for experimental purposes.

- ECN (2 bits)
  - A 2 bit field are allocated for use with explicit congestion notification.

- Flags (6 bits)
  - U, A, P, R, S, F is a control flag bit. This field indicates to the various information of the protocol in the TCP segment. The sender

or destination specifies the information to be carried with the control flag bit. Multiple control flag bits can be simultaneously used on the sending and receiving side. The meaning of each flag bit is as follows;

  - A (ACK)
    - Acknowledgment number field is valid.
  - F (FIN)
    - No more data from sender.
  - P (PSH)
    - Push Function. It is necessary for the destination to pass this segment as quickly as possible to the application layer.
  - R (RST)
    - Reset the connection.
  - S (SYN)
    - New connection is established. To initialize the connection, the sequence numbers are synchronized.
  - U (URG)
    - Urgent pointer field is valid.

- Receive window size (16 bits)
  - This field is used to notify the size of receive window from destination to sender. The sender controls data flow using this information.
- Checksum (16 bits)
  - This filed indicates the checksum of the TCP segment. Checksum is calculated and set by the sender, and verified at the destination. When

TCP segments are calculated the checksum using header including the destination IP address etc.

- Urgent pointer (16bits)
  - This filed indicates offset that is the last sequence number of urgent data (urgent data indicate that should be preferentially transmitted data). It is only valid when flag bit U (URG) is set.
- Options (variable)
  - This field is variable length and consists of two parts. One is contains various information for improving the performance and expansion of function. The other is a part called padding, and when TCP header length is excessively long, remaining part is set 0 to align in units of 32 bits.

2.2.2 TCP Process

TCP uses 3-way handshake, sequence number and acknowledgment, retransmission and window control, error control to provide reliable service. The 3-way handshake function is used to establish a TCP connection as shown in Figure 2.3. TCP establishes a virtual circuit between a sender and a destination before it sends or receives data. Virtual circuit is established through three phases;

Phase 1

The sender transmits a TCP segment that indicates the connection request to the destination. The sender sets the SYN bit of the control flag of the TCP header to ON and transmits it. The segment for which the SYN bit is set is usually called a SYN segment or SYN.

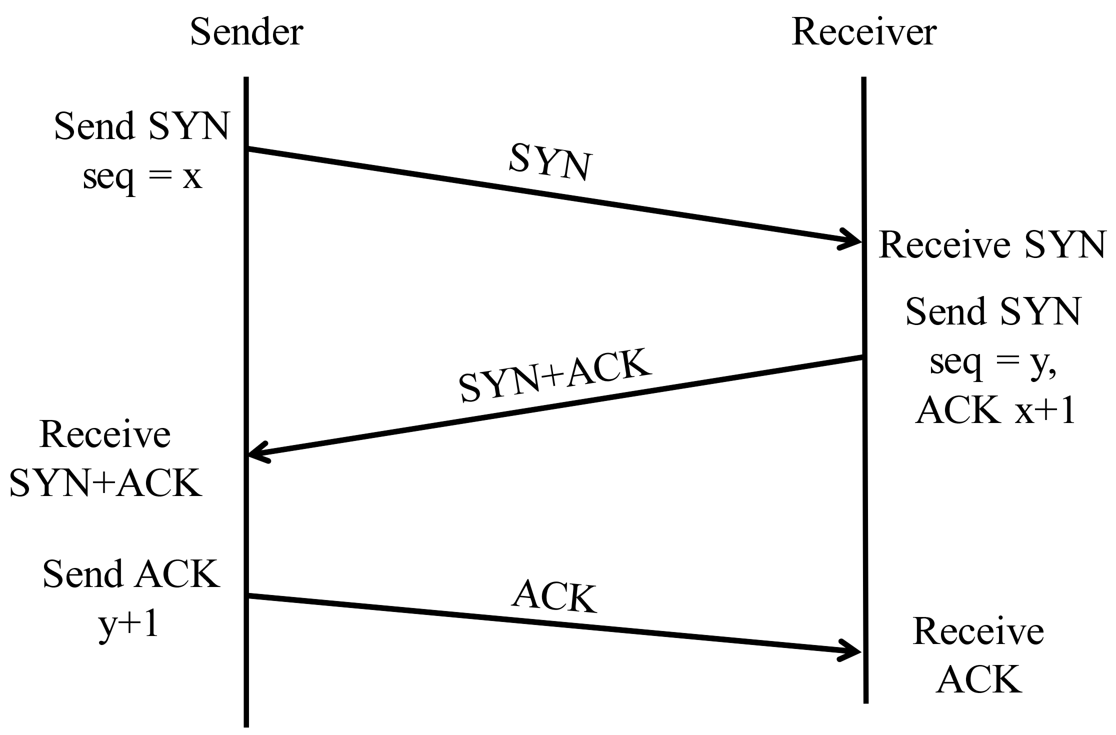


Figure 2.3: 3-way hand shake in TCP

Phase 2

When the destination accepts the connection request from the sender, it sends back the connection request to the sender together with the acknowledgment indicating acceptance. In this case, the control flags of both ACK and SYN are turned ON. The segment for which the ACK bit is set is usually called a ACK segment or ACK.

Phase 3

The sender accepts the connection request from the destination and return ACK to the destination. Then, TCP connection is established.

2.2.3 Data Transfer in TCP

When a virtual circuit is established, the sender start to transfers data to the destination. If there are multiple routes between the sender and destination, the arrival order of the transmitted TCP segments may be disturbed. Therefore, sender adds a sequence number indicating order of TCP segments and send it. The destination reorder TCP segments using this sequence number.

Acknowledgement

The destination returns an acknowledgment with an ACK number to sender for properly received segments. The ACK number is calculated from the received sequence number. Usually, it is set the number of the segment to be received next. For example, when a segment with sequence number 100 is received, the ACK number is 101.

Retransmission

The sender keeps that segment in the buffer until it receive acknowledge. Figure 2.4 shows the operation when a loss packet is retransmitted. The retransmission is performed when ACK is not received even certain period of time after the sender transmits the packet. The sender measures the elapsed time between sending a data with a particular sequence number and receiving an acknowledgment that is the sequence number of corresponding segment. This measured elapsed time

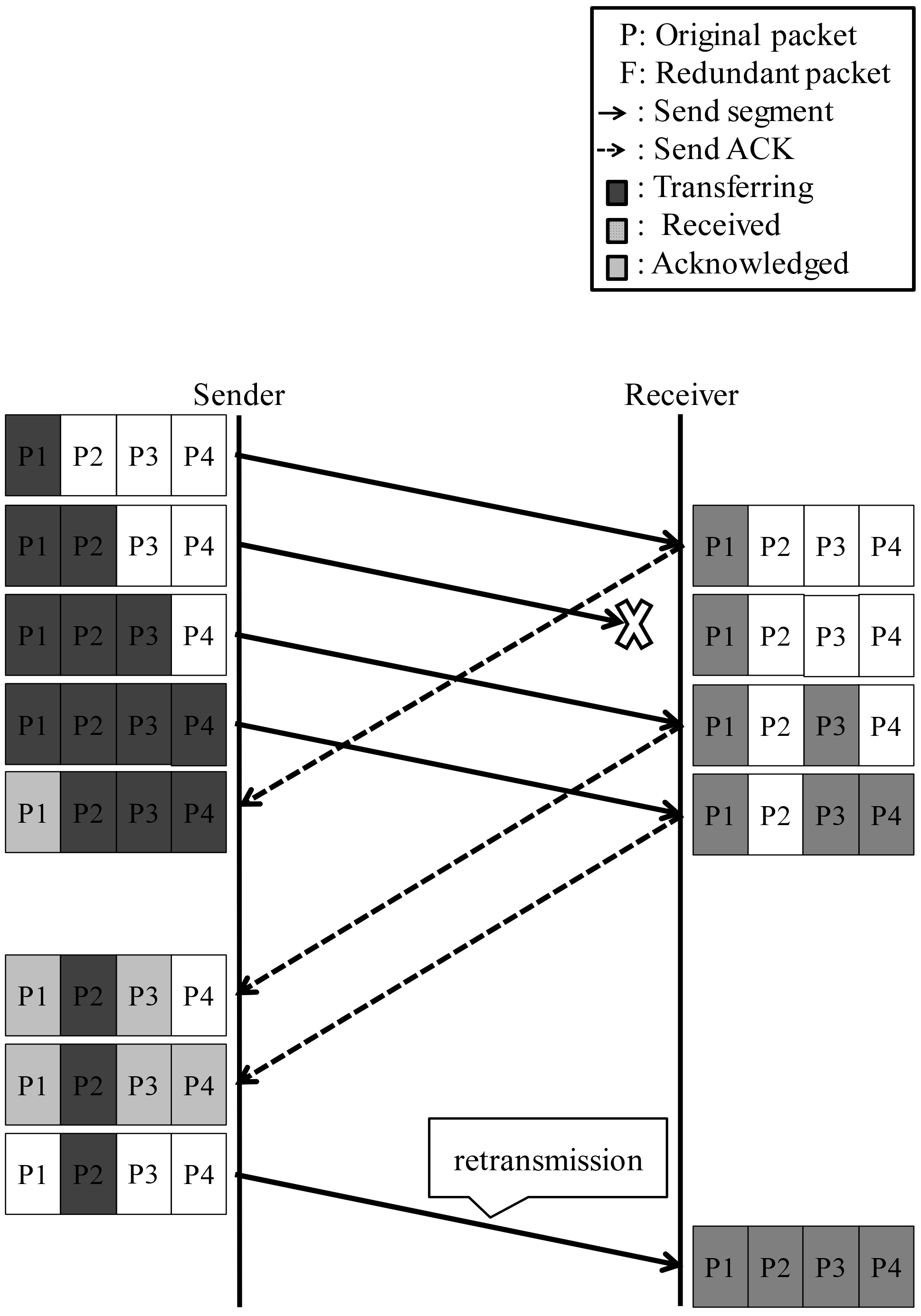


Figure 2.4: Retransmit a lost packet

is called Round Trip Time (RTT). Because 1 RTT is required for one exchange, the loss detection time depends on RTT.

However, although destination returns an acknowledgment to sender, ACK may not arrive at the sender within the waiting time depending on the congestion state of network. The sender retransmits the corresponding segment when ACK can not be received during the waiting period. Therefore, the destination re-receives the segment once received. When destination receives TCP segments with duplicate sequence number, it discards duplicating TCP segment.

In general, trigger of retransmission are as follow;

- Transmitted TCP segment was lost in transmission path.
- Acknowledgment of transmitted TCP segment was lost on transmission path.
- Acknowledgment was not returned within the waiting time due to congestion state of network.
- Checksum error occurred in received data.

Timeout

As mention above, the retransmission is performed when ACK is not received even certain period of time after the sender transmits the packet. Namely, a timer is set when a TCP segment is transmitted, and a retransmission is performed when the timer times out. The retransmission timeout time is considered to be slightly longer than the RTT between the communicating hosts.

In TCP, when the following conditions are occurred, sender judges that a TCP segment is lost and are occurred timeout;

- When three or more segments with same acknowledge number are received.

- After sending TCP segments, if ACK segment does not return to sender within waiting time.

Window Control

While TCP's simple retransmission is running, only one segment is sent at a time. New TCP segment cannot be transmitted unless acknowledgment of retransmitted segment is received. This operation leads to degrade efficiency transmission. Therefore, in TCP, the sender provides a mechanism to continue transmitting segments before receiving acknowledgment. Such method is called window control, and it is possible to transmit segments for window size at a time. Here, window size indicates amount of acceptable segments that are transmitted from the sender from the destination.

The sender can collectively send the segments within window size to the destination by using window control after establishing TCP connection. Receive buffer is free as much as the processing amount of the received segment. When the sender received ACK is sent to the sender for received segment, the sender is notified the available window size. If the receive buffer becomes full, the destination notify the window size 0 with ACK to the sender and transmission is stopped to the destination. Since the window size varies depending on the remaining amount of the receive buffer, the number of segments transmitted at once changes. For this reason, such a window control method is particularly called a sliding window control method as shown in Figure 2.5. Here the segments just have numbers, but each integer represents a whole number of byte of segment. In this example,

i) window size is 4000 bytes, segments 0 through 3999 have been sent but not acknowledged.

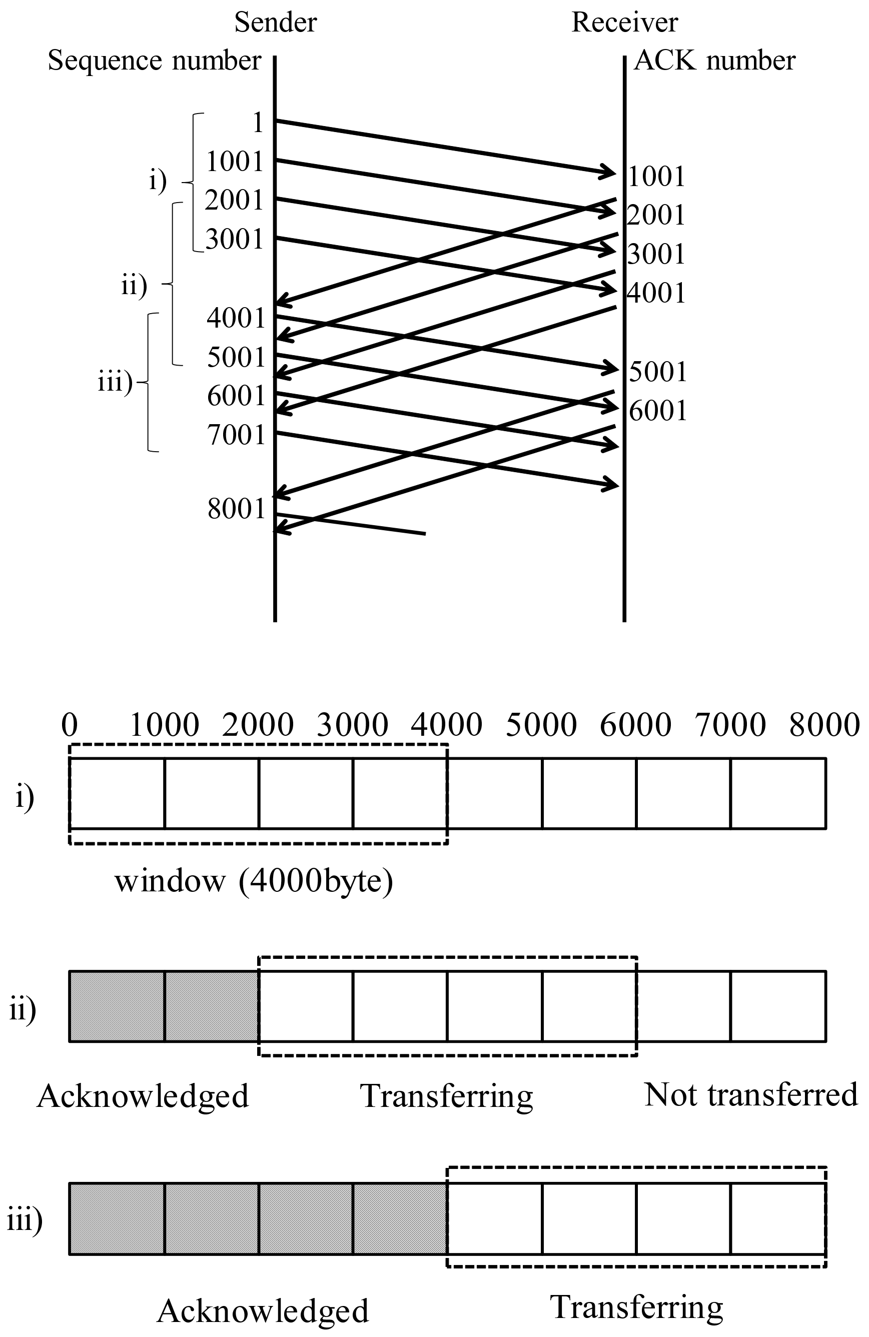


Figure 2.5: Sliding window control

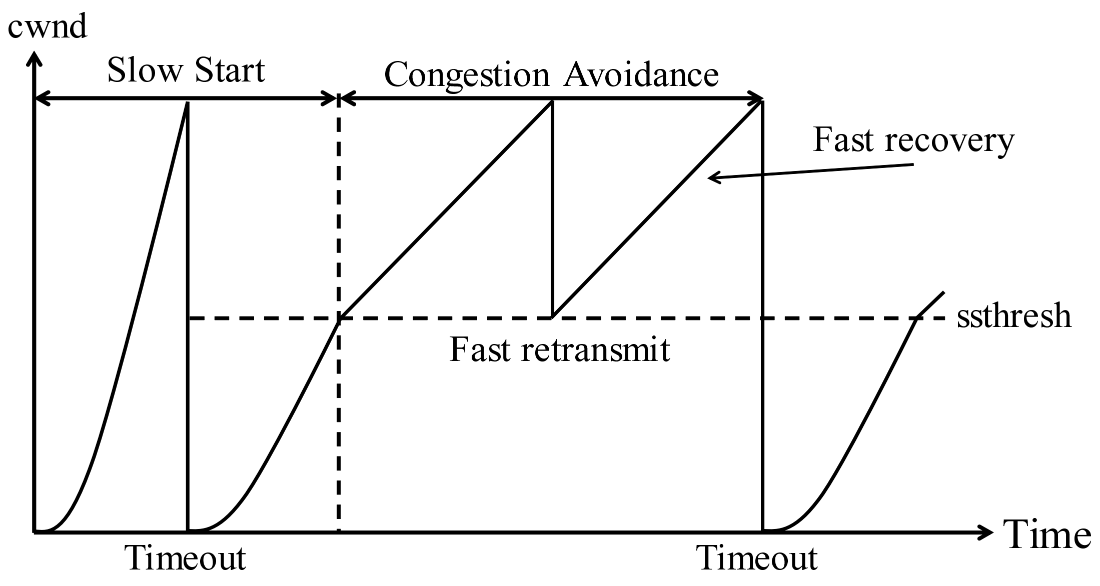


Figure 2.6: Congestion control

ii) segments 0 through 1000 have been sent and acknowledged, 2000 through 5999 have been sent but not acknowledged, and segments 6000 through 7999 are waiting to be sent.

iii) segments 2000 through 3999 have been sent and acknowledged, 4000 through 7999 have been sent but not acknowledged.

2.2.4 Congestion Control

Generrically, TCP performs congestion control with sliding window method and has the following features.

- Transmission rate of TCP segments can be adapted to available bandwidth
- Network congestion can be avoided
- Reliable communication can be provided by retransmission of lost segments

To control the transmission rate of TCP segments, the number of segments that can be sent is limited by a parameter called congestion window (*cwnd*). If the number of TCP segments for which ACK sent from the sender but ACK indicating that it was correctly accepted from the destination has not reached *cwnd*, the sender stops sending and goes into transmission wait state. To reply received status of the TCP segment with ACK to sender, sequence numbers are assigned to each TCP segments.

The main objectives of ACK control are as follows.

- To control the transfer rate, new segments are transferred when the other segment leaves the network.
- Establish a reliable TCP connection by providing a means to inform the sender that it should retransmit the segment that has not arrived at the destination.

ACK control can detect lost TCP segments and non arrival TCP segments using ACK from the destination. The sender determines the sequence number of the TCP segment that is transmitted next by ACK from the destination.

When three duplicate ACKs are received at the sender, the sender retransmits lost TCP segments. Such as the retransmission is called "fast retransmit" Even if fast retransmit succeeds, congestion window size is downwardly reduced in the recovery process. This amount of the reduction is different on TCP congestion algorithm. In the case of Tahoe, congestion window size is set 1. On the other hand, Reno and NewReno makes it half of the current congestion window size.

The basic concept of the congestion window dynamic control method is as follows and as shown in Figure 2.6. If congestion window size is small, window size is exponentially increased (Slow Start). When congestion window size exceeds a certain large value, if the congestion does not occur, congestion window size is linearly increased (Congestion Avoidance). When congestion occurs, congestion window size is remarkably reduced (Recovery).

By using this dynamic control method, when congestion occurs, this method can respond promptly and effective use bandwidth.

In congestion control, threshold value of congestion window (*ssthresh*) is defined. This threshold is usually called the slow start threshold. The size of receivable window that the destination notifies with ACK is defined as Advertise Window (*awnd*), and the size of sendable window (w) of is determined as follows;

$$w = \min(cwnd, awnd) \tag{2.1}$$

In other words, the window size informed from the destination does not directly use, the sender compare advertise window size with congestion window and adopt the smaller one.

Advertise window is the upper limit of the congestion window. Congestion window can suppress the transfer exceeding the upper limit of the network bandwidth. The process of TCP that gradually increases congestion window from initial value and reaches advertise window or the slow start threshold is called slow start.

If the sender does not have prior information about the bottleneck link, it is necessary to estimate the value of congestion window size.

Slow Start

The sender sets the initial value of *cwnd* to 1. It increments *cwnd* each time when it receives one ACK segment.

$$cwnd = cwnd + 1 \tag{2.2}$$

Then, *cwnd* doubles for every RTT and increases exponentially until TCP segments are lost. TCP segment is lost, the sender shifts to the recovery process and calculates slow start threshold. When *cwnd* exceeds the threshold value ($cwnd > ssthresh$), the sender ends the slow start process and shifts to congestion avoidance process.

Congestion Avoidance

When the sender normally receives ACKs, it increases the transmission rate and *cwnd* at a rate of 1 TCP segment per RTT. To use link bandwidth as efficiently as possible where congestion is likely to occur, congestion window size is gradually increased by using this operation. If TCP segment is lost, the sender shift to the recovery process.

Recovery

The method of controlling the congestion window when timeout or TCP segment loss occurs that varies depending on the version of TCP. TCP Tahoe restarts slow start after updating the threshold to half of congestion window size when TCP segment loss occurred, and the congestion window size to 1. In TCP Reno or TCP NewReno, the threshold and the congestion window size are updated to half when TCP segment loss occurred, and then slow start is restarted. In Section 2.2.6, I describe TCP Reno [9] which uses an option called SACK (Selective Acknowledgment) [10] among TCP.

Reactive vs Proactive

Congestion control algorithm are categorized into two ways; reactive and proactive.

In general, TCP uses reactive control that adjusts transmission rate after it detects lost packet due to sudden traffic fluctuation. Specifically, the sender control *cwnd* according to the network condition. TCP Reno employs reactive congestion control. In TCP Reno, *cwnd* is adjusted based on ACK from destination. TCP Reno decrease *cwnd*, which may be unnecessary for wireless environments. Basic TCP have been proposed as the standard Reno that uses reactive control. However, reactive control increases transfer delay by transmission of lost packets. Therefore, in reactive control, it is impossible to optimize the use of link bandwidth of the network, and the communication state of the network can not stable. To improve the affective of transfer delay, congestion control based on transfer delay is developed, but it is difficult to prevent increase of transfer delay according to sudden traffic fluctuation.

On the other hand, in proactive control, sender reallocate *cwnd* by using feedback from destination to prevent congestion. Proactive control requires prior correspondence, but it is possible to prevent increasing of transfer delay by responding in advance due to retransmission. It is necessary technology to realize a better delay-sensitive service. For this reason, I research proactive control that recover lost packets without retransmission.

2.2.5 Explicit Congestion Notification

TCP has a function that detects packet loss and retransmits lost packets. However, retransmission of lost packet leads to degradation of TCP throughput.

When TCP detects a lost packet, it judges that congestion has occurred, and degrades transmission rate. Thus, in high-delay environment, there is a problem that the congestion state of network is not accurately reflected to congestion control.

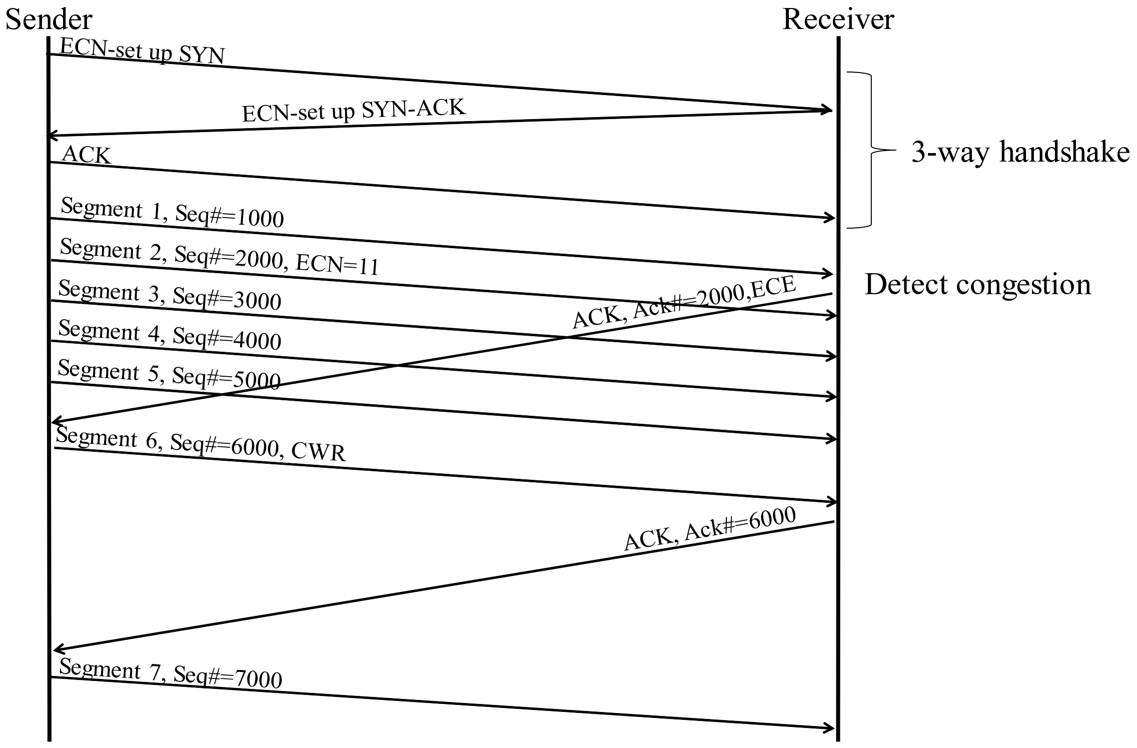


Figure 2.7: The operation of ECN control

Therefore, Explicit Congestion Notification (ECN) [11] has been proposed in which a router or a destination adjusts the window size by notifying congestion status of network to the sender. Specifically, ECN flag bit of TCP header is set. When the destination receives a packet with ECN flag, it knows that congestion is occurring on the route. The sender degrade transmission rates by received ECN.

At transport layer, the sender and destination must be able to cope with each other's ECN. The destination needs to have a function that receives a packet marked congestion state from the router and notifies congestion state to the sender. On the other hand, the sender needs to have a function to downwardly adjust transmission rate by using the notification from the destination on the congestion state.

The Addition of Explicit Congestion Notification to TCP

TCP supports ECN using 2 bit at RSV field in the following.

- ECE: ECN-Echo flag
- CWR: Congestion Window Reduced flag

ECE is used to notify the information that ECN flag is set ON in TCP segment by router CWR is flag in the TCP header that the data sender can inform the data receiver that the congestion window has been reduced The procedure of ECN control is a s follows and are showed Figure 2.7;

- In the TCP connection setup phase, the sender sends SYN segment with ECE=1, CWN=1 to the receiver
- The receiver sends back SYN-ACK segment with ECE=1, CWR=0 to the sender
- If congestion is detected or congestion is occurring, the corresponding router set ECN=11 on IP header
- The receiver receives ECN packet (11) , sends back ACK segment with ECE flag in TCP header
- The sender receives ACK segment with ECE flag, it shift to congestion avoidance process
- The sender decrease congestion window size, it set CWR flag in TCP header of next segment and send one
- The receiver receives new segment with CWR flag, it clear next ACK with ECE flag

2.2.6 Selective Acknowledgment Options

Cumulative acknowledgment notifies number of TCP segments that arrives without coming out from the start of communication, and if there is a missing in the stream, it is not notified whether or not subsequent data has arrived. When multiple packets are lost on a *cwnd*, TCP throughput is likely to decrease. Therefore, it is necessary to extend the function of acknowledgment and to discriminate normally received packets and lost packets even when multiple packets are lost during *cwnd*. This method is called a selective acknowledgment option (SACK option), which is defined in RFC 2018 and can be used by further adding an optional field to the TCP header. Specifically, when a packet losses, only a packet that has been lost can be retransmitted while transmitting next TCP segment. TCP that are applied this method is called TCP with SACK option. It wasteful traffic generation on the network can be prevented, and high utilization efficiency and throughput can be achieved.

2.2.7 TCP with SACK Option

Since the maximum size of TCP option is 40 times, the number of blocks that can be acknowledged is limited to a maximum of four. In the wideband and high-delay environment, it is conceivable to use together with the time stamp option and the window scale option, but in this case the maximum number of blocks that can be acknowledged is limited to three. When accepting a selection acknowledgment, it sends a selection acknowledgment permission option (SACK OK) to communicating party in the SYN segment at TCP connection establishment. A host that receives this option can send a selection acknowledgment to the segment sent by the other party. When the selection acknowledgment is used, the range of the reached sequence number is embedded in the selection acknowledgment field and acknowledged. In the case of sparse missing, it is possible to acknowledge up to four blocks.

2.2.8 TCP's Problems

TCP using the congestion control algorithm has been widely used to the present. However, in recent years, high throughput can not be obtained in wideband and high-delay environment. In high delayed environment, it takes time to reach the window size that can obtain a sufficient transmission rate compared to other environments. In conventional TCP, the utilization efficiency is poor and performance can not be fully utilized, so TCP with SACK option reduces *cwnd* on half by retransmission control at fast recovery when packet loss is detected.

Furthermore, while recovery, transmission rate are kept as shown in Figure 1.1. Therefore, the efficiency of communication might be lowered in a particularly high-delay environment such as an international communication. In order to prevent such this problem, it is necessary to send information to recover a lost packet in advance to the destination. Therefore, in this research, I focus on Forward Error Correction (FEC) which can respond to recovery in advance. In the next section I will describe the details of FEC.

2.3 Error Correction

Error correction is a well-known technique for correcting the error over communication by redundancy information as shown in Figure 2.8. Generally, error correction codes are converted into codewords whose k-bit information has a code length of n bits. This is called $(n,\ k)$ code. The $(n,\ k)$ code can be divided into two types; a block code and a convolutional code. Those encoded into a sequence of a certain length are called block codes. Sequentially encoded and theoretically

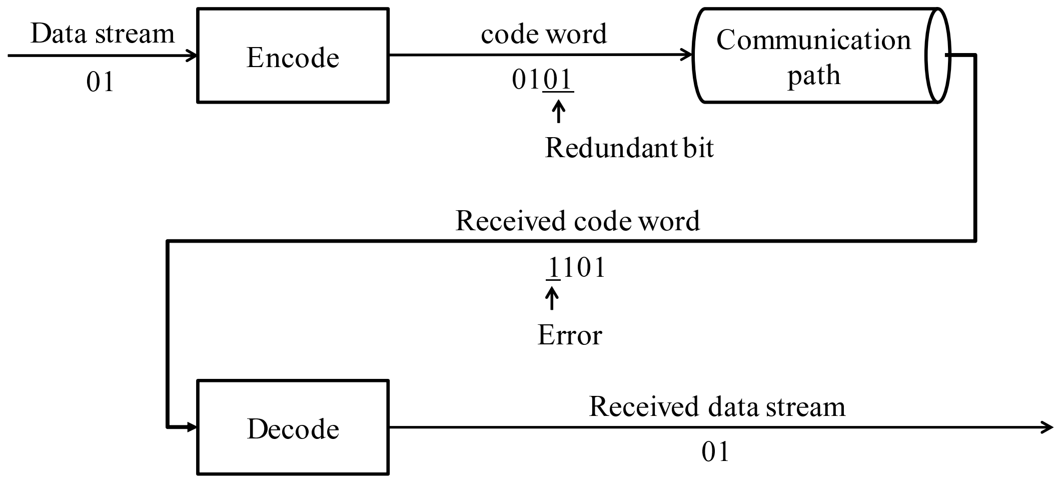


Figure 2.8: The operation of error correction

encoded into a sequence of infinite length is called a convolutional code [12].

The error correction code detects an error on the destination and corrects an error at the same time. It do not need feedback, and are used for transmission of voice, images, and the like, real time transmission is possible with less delay time. However the high error correction capability has the trade off relation between redundancy and efficiency.

Convolutional Code

While the block code is a code to be encoded in units of blocks, the convolutional code is a code for obtaining coded bits at the present time using several past bits. In block code, data stream are divided into blocks, and error correction is performed in units of the delimited frames. However, since error processing is performed after discriminating consecutive received analog signals as 1 and 0, there is a limit to error correction in a mathematically.

Block Code

A block code is obtained by dividing the encoded data into blocks with fixed length. That is, one error code block is formed for one block. Four typical examples of block codes are Hamming code, BCH code, Reed-Solomon code and parity check code. The four are described below.

- Hamming Code
  - Hamming code can perform one error correction, and error detection of 2 or less is possible using a generator polynomial with bits as coefficients. Although the design of code is simple, in the application to the radio transmission line etc. with low reliability, the error correction capability is not sufficient for the error rate of the line.
- BCH Code
  - BCH code can correct multiple error corrections. Like hamming code, BCH code performs encoding using a generator polynomial with bits as coefficients.
- Reed-Solomon Code
  - The Reed-Solomon code [13] is extended BCH code. It is complicated in design and it takes time to calculate.
- Parity Check Code
  - The parity check code calculate whether the sum bit stream is an odd number or an even number, and thereby detects an error. Such an operation method is called exclusive OR. It is simple to implement because it takes short computation time.

Block code is calculated using a fixed block length. In TCP, it is difficult to determine a fixed block length because amount transfer data are constantly fluctuated according to network condition. For this reason, in this study, the proposed scheme determine a block length based on *cwnd*.

2.3.1 Forward Error Correction

The forward error correction technique adds redundant information to the original data according to follow rule and sends one. The destination checks whether the received data complies with the rule and corrects the error according to the result.

Forward error correction technique can be realized it by using error correction code. The sender sends a redundant information from original data and transmits the redundant information. The destination can recover lost data from the redundant information.

Generally, the sender encodes a redundant information by using exclusive OR (XOR) with the all original information and transmits the redundant information after the original data. It thus needs only a few memory resources for creating redundant information, and it can calculate an encoded bit-stream using simple bitwise operations. It has lower overhead than other coding schemes, like Reed-Solomon codes [13].

The destination uses XOR to recover a lost data within a group when it receives a redundant information. This scheme cannot recover lost data when two or more data losses occur within a group. In that case, the sender retransmits the lost data through the original TCP operation. When data losses do not occur within a group, a received redundant information is simply discarded.

FEC's Problem

Generally, in the FEC technology, the recovery success rate depends on the amount of redundant information. Therefore, in order to secure high reliability, more redundant information and sufficient link bandwidth is required is required. As described above, when the redundancy is increased more than necessary, the utilization efficiency of the network band decreases. In other words, there arises a problem of trade-off between bandwidth and redundancy information that the restoration rate of the lost packet depends on the redundant information. In order to solve this problem, it is necessary to set the optimum redundancy according to network condition. Also, in the case where FEC technology is applied to a plurality of packets collectively, it is preferable that packets are continuously generated, so it is generally used for CBR communication using UDP. However, unlike UDP, TCP has complicated control that the transmission speed always fluctuates, so it is difficult to set the optimum redundancy of FEC. Therefore, in recent years, various methods related to the redundancy dynamic determination method to suppress the consumption of the network bandwidth to the minimum by applying the FEC technology as a part of TCP have been proposed. In the next section, I explain such related work.

2.4 Advanced Works

The recovery success rate depends on the redundancy level. To control redundancy, the proposed FEC algorithms are categorized into two groups: Keeping constant redundancy level independent of network conditions [14–17] and dynamically adjusting redundancy level according to network conditions [18–26].

A method for applying FEC to UDP communication, which has a constant transmission rate, was proposed [14]. AL-FEC [15] utilizes FEC for UDP to improve throughput in high-loss-rate wireless environments. These schemes improve throughput performance of UDP communication, but the present study focuses on a scheme that applies FEC to TCP communication, in which transmission rate dynamically changes.

Sakakibara et al. focus on the problem of consecutive reductions in the transmission rate caused by congestion control when multiple packets are lost consecutively [18]. To resolve this problem, they propose a method to apply FEC to TCP only when TCP detects packet losses. This method attaches redundant packets during a recovery phase; i.e., when the transmission rate is low. It thus suppresses the transmission rate reduction as well as the amount of redundant information.

Sharma et al. focus on the problem that it takes a long time to recover lost packets even when short-term burst losses occur in a high latency broadband environment [19]. This is because TCP cannot discriminate between short-term or long-term congestion when burst packet losses occur. To solve this problem, they propose a form of TCP congestion control using Explicit Congestion Notification (ECN) to identify short-term congestion. This method uses adaptive FEC with redundancy proportional to the number of lost packets only when it detects packet losses by ECN. It enables high throughput by quickly responding to burst losses in high latency broadband networks.

Baldantoni et al. propose a method using FEC which dynamically determines the redundancy to improve TCP performance in a wireless network [20]. This method calculates an appropriate level of redundancy based on the current loss rate and round-trip time. There are two types of packet loss in wireless networks: that caused by congestion and that caused by interference such as fading. TCP cannot distinguish between these types of packet loss, so it unnecessarily reduces the transmission rate even if packet losses are caused by interference. The proposed method avoids unnecessary reduction of the transmission rate by using FEC to recover packet losses caused by interference.

Seferoglu et al. propose a method to achieve fairness between TCP and media flows, specifically multimedia delivery flows for mobile devices in the next generation of wireless technology [21]. When TCP and media flows coexist in a network, media flows can easily cause packet losses and drastically reduce the transmission rate due to TCP congestion control. This method prevents media flow packet losses through adaptive FEC to maintain fairness between TCP and media flows. It dynamically determines FEC redundancy by predicting loss events based on the transmission delay time. Overall, these methods restrictively apply FEC technology to TCP only when packet losses occur or when a particular transmission rate should be maintained. In this study, I focus on the application of FEC technology to the entire operation of TCP to use FEC more effectively, especially in high latency environments. Our method uses FEC technology to suppresses frequent transmission rate reduction by avoiding retransmissions.

Tsugawa et al. propose TCP with FEC for streaming delivery services, a method that does not greatly change the transmission rate [22]. It controls FEC redundancy based on packet loss detection and maintains the transmission rate to preserve the quality of service. This method suppresses temporal reduction of the transmission rate with a change in redundancy to maintain the transmission rate needed for streaming delivery services.

FEC-ARQ [23] combines FEC with ARQ mechanisms to keep the quality of streaming services in a low-latency environment. This method is based on a packet streaming code well suited to sequential decoding and improves the total delay caused by retransmission.

LT-TCP [24] uses ECN and FEC to mitigate the effects of random packet losses over lossy wireless networks.

Moreover, TCP-IR [25] applies FEC to the TCP operation. It focuses on the problem that TCP needs at least 1 RTT when it recovers lost packets. It injects redundant packets within TCP streams, so it reduces latency of web transactions. This work enhances TCP throughput performance in low latency environments. It was extended to improve TCP throughput in high-latency environments [26]. TCP-IR encodes up to 16 packets. These works apply FEC to TCP operations only in a restricted manner.

The control within TCP is complicated and most suggestions focus on the application layer FEC. Alomost objective is to increase TCP throughput on the wireless network and to isolate the loss detection and recovery mechanism. Related work performs loss detection and recovery of TCP to reduce RTT, however any approach does not respects network condition (*cwnd*) and does not adaptable work FEC technology. In order to improve TCP's performance, dynamic adaptive FEC that respects network condition (*cwnd*) is strongly required.

2.5 Conclusion

Delay-sensitive services such as streaming and online games have increased. To satisfy the network users that use delay-sensitive services, transfer delay should be short. For that purpose, it is required to predict sudden traffic fluctuation and to provide stable communication. However, in general, TCP uses reactive control that adjusts transmission rate after it detects lost packet due to sudden traffic fluctuation. In reactive control, it is impossible to optimize the use of link bandwidth of the network, and the communication state of the network can not stable. Therefore, proactive control has been researched that detect congestion by RTT and adjust transmission rate. It is worked to improve congestion control and are not considered transfer delay. Delay-sensitive services that increase in the future complacently will no be provided because this problem still will not improve. Progress of network technology is limited and realization of high quality and stable delay sensitive services are difficult in present.

Therefore, in this research, to realize high quality and stable communication in delay-sensitive services, I propose a scheme to suppress increasing of transfer delay and reduction of transmission rate by retransmission. Namely, proactive control that uses forward error correction technology is important when lost packet is retransmitted. Delay-sensitive services have been needed dedicated line, but it satisfactorily can be performed without preparing dedicate line by the proposed scheme.

In this study, I focus on the application of FEC technology to the entire operation of TCP. This scheme only extend FEC mechanism at transport layer and do not need rebuild of protocol. It gradually spreads in network without dedicated line. For this reason, it can contribute significant cost-cutting of equipment of dedicated line. It has fairness problems that competing of another TCP version because it gradually spreads. In this research, it has been examined congestion control that is considered regard of fairness of another TCP version, this detail will be described in Chapter 5.

3 Simply Applying FEC to TCP Operation

In this chapter, I first consider a scheme to simply apply FEC technology to the entire TCP operation. The proposed scheme dynamically controls redundancy level according to transmission rates. I investigate the fundamental characteristics of the proposed scheme focusing on the redundancy, especially including in a high-latency environment.

3.1 Introduction

In this chapter, I explain the evaluation of characteristics of basic proactive control that is applied to the entire TCP operation.

Data exchange in a high-latency environment is one of delay-sensitive services. For communication over the Internet, TCP is basically used for reliable data transmission protocol. TCP adjusts its transmission rate through congestion control according to a network condition. However, packet losses significantly degrade TCP performance in a high latency environment, such as international communication. This is because the recovery of lost packets takes at least the round trip time. To prevent this, the number of retransmissions must be reduced and congestion control applied independently of the packet losses.

In this context, several methods which apply a forward error correction (FEC) technology to prevent packet losses have been proposed. The recovery success rate depends on the amount of redundant information, but redundant information places an additional load on the network. Therefore, the amount of redundant information must be appropriately determined according to network conditions. Furthermore, FEC is typically used for UDP communication, which has a constant transmission rate. It is difficult to adapt to TCP communication where transmission rates change a lot because it is harder to select an appropriate redundancy level in such an environment. For this reason, although there have been few studies on the application of FEC technology to TCP to suppress retransmission and improve performance, there have been studies on TCP restrictively using FEC technology.

In this chapter, to enable more effective use of FEC technology, I propose a scheme to apply FEC technology to the entire operation of TCP. TCP with simple FEC adapts FEC redundancy to the TCP transmission rate through effective use of network resources. Furthermore, I show the effectiveness of the scheme through simulation evaluations where I focus on high latency networks.

3.2 Overview of TCP with Simple FEC

In this section, I explain TCP with simple FEC which applies FEC technology to the entire operation of TCP while taking into account the trade-off between FEC redundancy and network efficiency. TCP with simple FEC determines the appropriate level of redundancy according to the transmission rate to improve throughput by suppressing retransmission. It aims to prevent the recovery time caused by retransmissions. Since TCP with simple FEC does not conflict with the schemes that improves the effectiveness of retransmission such as TCP with

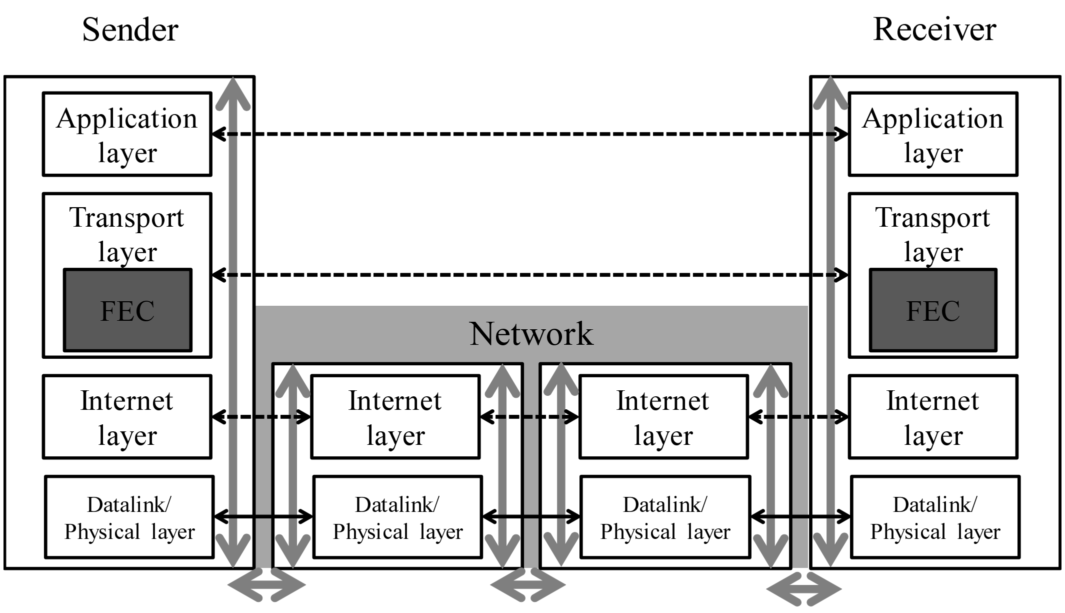


Figure 3.1: Applying FEC to TCP operation

SACK, it will work effectively with them.

First, I provide an overview of TCP wiht simpe FEC. This method provides end-to-end communication with FEC technology as shown in Figure 3.1. FEC redundancy is added and lost packets recovered in the operation of TCP. That is, a sender transmits redundant information in addition to original packets, while a receiver recovers lost packets through the redundant information.

In the simple FEC, the sender encodes a redundant packet by using exclusive OR (XOR) with the payload field of all packets in a congestion window (*cwnd*) and transmits the redundant packet after the original packets as shown in Figure 3.2. The sender creates a redundant packet from original packets and transmits the redundant packet. The destination can recover lost packets from the redundant packet. Note that recovery of lost packet is focused on; in other words, redundant packet is not added to an original packet, and redundant packets en-

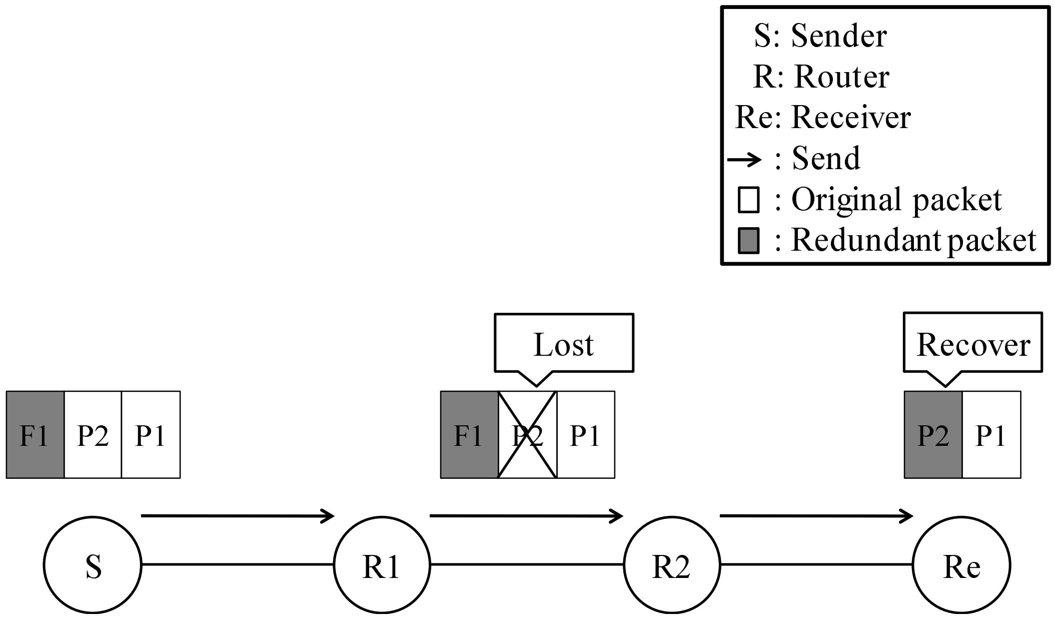


Figure 3.2: The operation of Simple FEC

coded from original packets are interjected within TCP streams. Namely, the simple FEC constantly creates redundant packets according to network conditions.

More specifically, the sender creates a redundant packet XORed with all packets in a congestion window size (*cwnd*) and transmits the redundant packet following the original packets as shown in Figure 3.3. It thus needs only a few memory resources for creating redundant packets, and it can calculate an encoded bit-stream using simple bitwise operations. It has lower overhead than other coding schemes, like Reed-Solomon codes [13]. In the case of the simple FEC, group is defined as a block of one redundant packet with corresponding original packets, and group size is defined as the number of original packets. Namely, TCP with simple FEC constantly creates redundant packets according to network conditions.

The receiver thus can use the redundant packet to recover a lost packet within

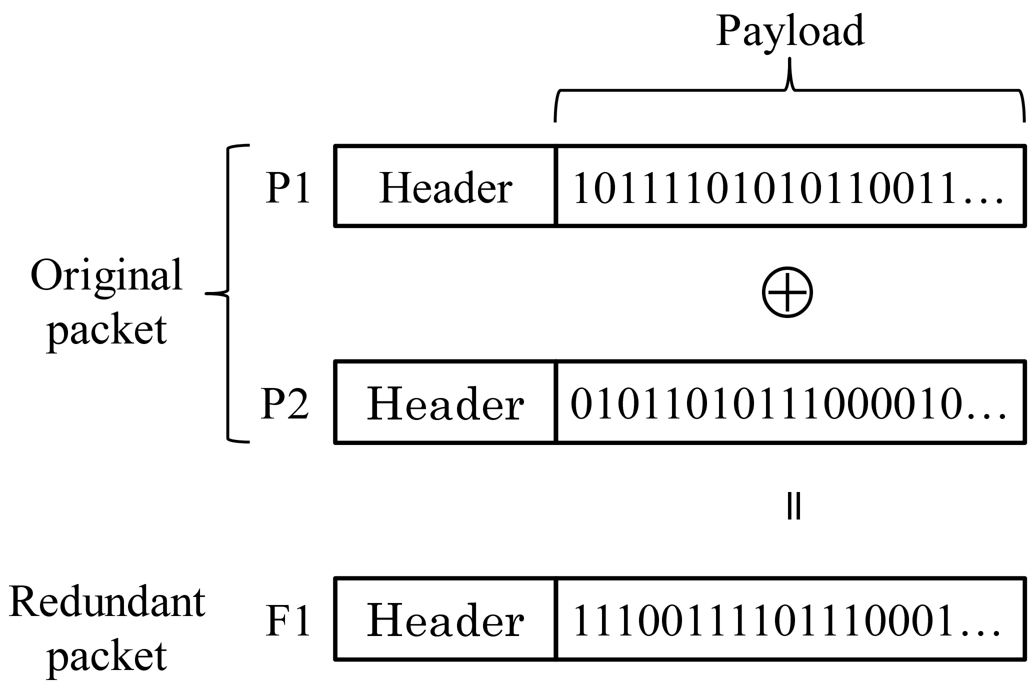


Figure 3.3: Create redundant packet

the *cwnd*. This method cannot recover lost packets when two or more packet losses occur within the *cwnd*. In that case, the sender retransmits the lost packets through the original TCP operation. The retransmitted packets are handled in the same manner as the original packets within the *cwnd*. Therefore, they can be recovered by the received redundant packet. When packet losses do not occur within a *cwnd*, a received redundant packet is simply discarded.

3.3 Redundancy Control

To recover lost packets with an FEC redundant packet, a receiver has to determine which original packets the received redundant packet corresponds to. Therefore, TCP with simple FEC groups a redundant packet and corresponding

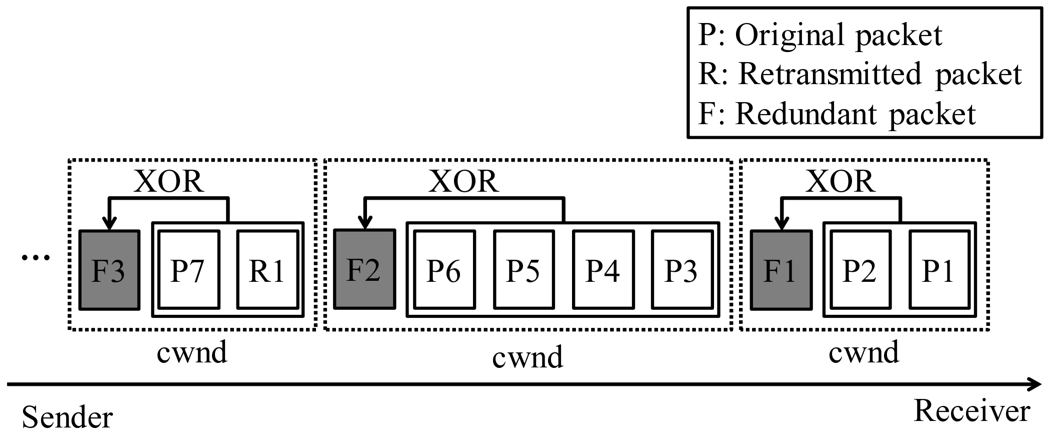


Figure 3.4: Relationship between original and redundant packets

original packets as shown in Figure 3.4. The sender creates an redundant packet XORed with all packets in a *cwnd* and transmits the redundant packet following the original packets. I define redundancy as the ratio of a redundant packet to the corresponding original packets; i.e., 1/*cwnd*. For example, the redundancy is 1/2 when *cwnd* is 2. The redundancy decreases as *cwnd* increases; thus, the redundancy is controlled according to network conditions. This means that redundancy is high when packet losses have a significant impact – i.e., at a low transmission rate – and is otherwise low. However, there is a time lag of up to 1 round trip time between the redundancy level and *cwnd* because redundancy is updated just after the sender transmits an redundant packet.

3.4 Simulation Evaluation

In this section, I compare the effectiveness of TCP with simple FEC to conventional TCP Reno with the Selective Acknowledgment Option (SACK) and SACK with FEC of static redundancy in a high-latency environment. I used the network

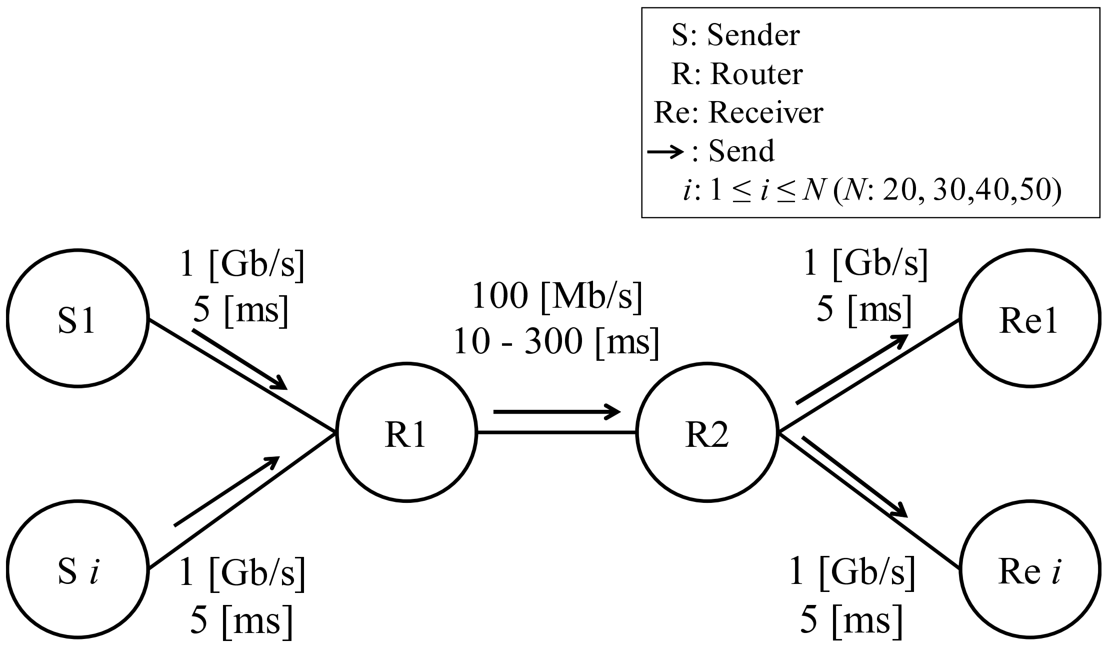


Figure 3.5: Simulation topology

simulator ns-2 ver.2.35 [27] after implementing TCP with simple FEC for the performance evaluations. In the following subsection, I describe the simulation mode and evaluation criteria.

3.4.1 Simulation Model

Figure 3.5 shows the network topology used in this simulation. In this model, each sender communicates with the corresponding receivers through routers. The simulation parameters are summarized in Table 3.1. The senders start TCP bulk transfer randomly at intervals of 0.1 seconds until 1 second after the simulation starts. I evaluate the average values over 10 simulation runs with different random seeds from 20 seconds to 220 seconds after the simulation starts to focus on the stable state performance.

Table 3.1: Simulation parameters

Simulation time	200 [s]
Number of trials	10 [times]
Buffer size	50–1000 [packet]
Number of flows	20–50
Packet size	1500 [byte]
Static redundancy	1, 10, 40

3.4.2 Evaluation Criteria

To evaluate the effectiveness of TCP with simple FEC, I use the average throughput, retransmission rate, effective recovery rate, and redundancy rate as evaluation criteria. Average throughput is defined as the average throughput of all flows. The other criteria are used to analyze the performance in detail. The retransmission rate is defined as the number of retransmitted packets as a percentage of the number of all sent packets. The effective recovery rate is defined as the number of recovered packets as a percentage of the number of redundant packets. The redundancy rate is defined as the number of redundant packets as a percentage of the number of both redundant and original packets. I evaluate these criteria while varying the number of flows, the delay time, or the buffer size.

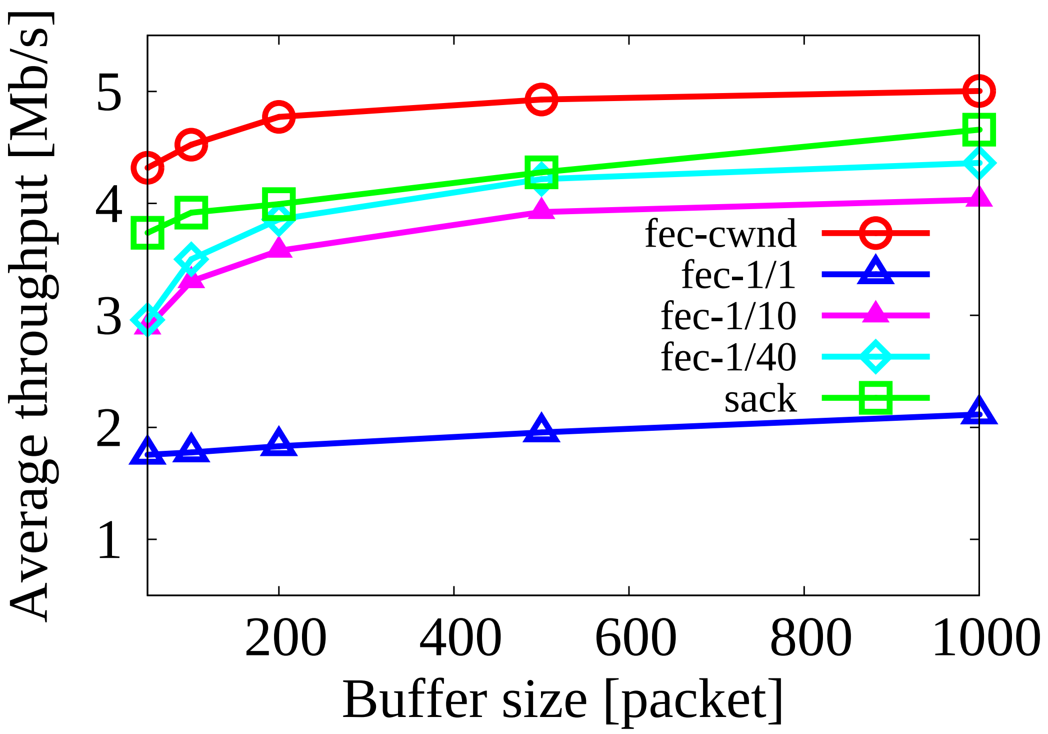


Figure 3.6: Effect of buffer size

3.5 Simulation Results

In this section, I show simulation results from a comparison of TCP with simple FEC, conventional TCP (TCP-SACK) and TCP-SACK with static FEC. First, I show the throughput with each method when the buffer size, delay time, and number of flows are varied. Next, I investigate the retransmission rate, the effective recovery rate, and the redundancy rate to analyze the results in detail.

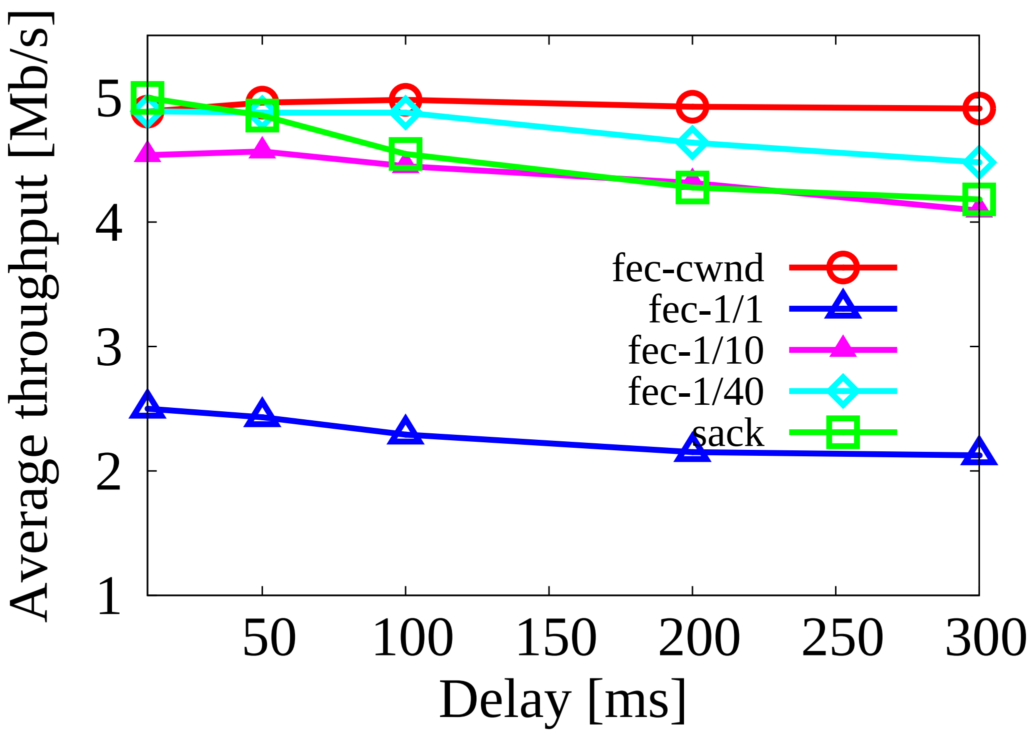


Figure 3.7: Effect of delay time

3.5.1 Effect of Buffer Size, Delay Time and Number of Flows

First, Figure 3.6 shows the average throughput of three methods when the buffer size varies from 50 to 1000 packets with a delay time of 200 ms and 20 flows. In this figure, “sack” represents the conventional SACK and “fec” represents SACK with FEC. Of the “fec” methods, “cwnd” is TCP with simple FEC where the redundancy is adapted according to *cwnd*, while the others use static redundancy. From this figure, I can see that TCP with simple FEC enables excellent throughput regardless of buffer size. In addition, the fec methods provides higher throughput than the sack method. These results are analyzed in the next sub-

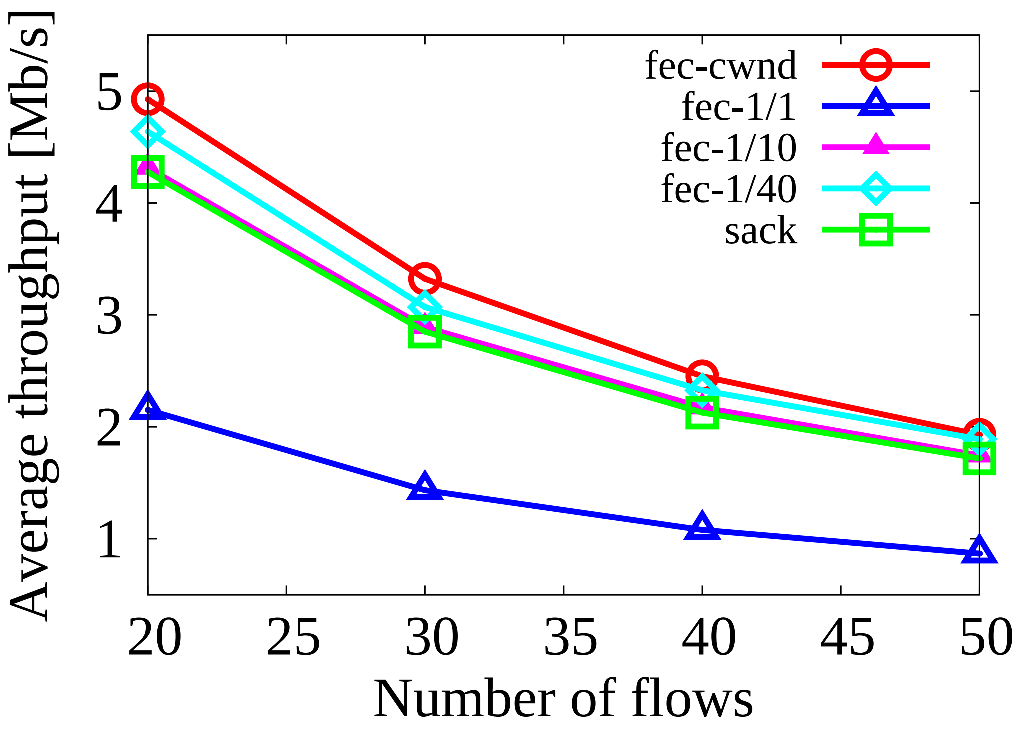


Figure 3.8: Effect of number of flows

section.

Next, Figure 3.7 shows the average throughput of three methods when the delay time varies from 10 to 300 ms with a buffer size of 500 packets and 20 flows. The results show that TCP with simple FEC provides high throughput even as the delay time increases although throughput deteriorates with the other methods.

Last, Figure 3.8 shows the average throughput when the number of flows ranges from 20 to 50 with a buffer size of 500 packets and a delay time of 200 ms. TCP with simple FEC achieves higher throughput than the other methods regardless of the number of flows.

The above results demonstrate that TCP with simple FEC can improve through-

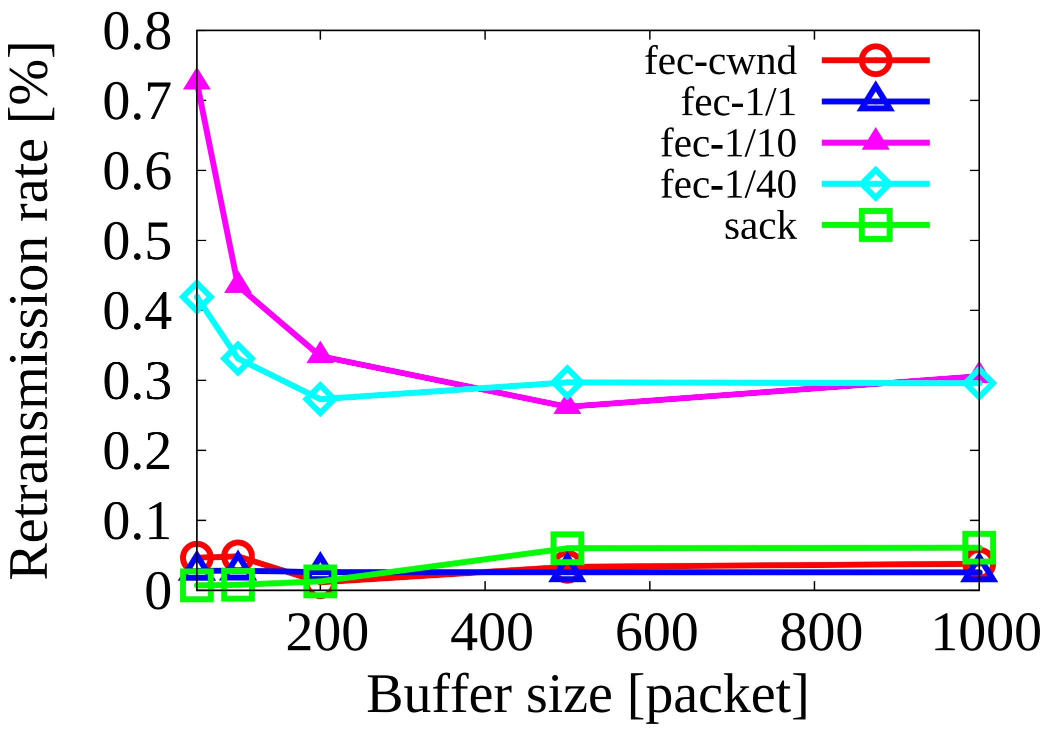


Figure 3.9: Retransmission rate

put regardless of the network parameters. In the following subsection, I analyze the results in detail.

3.5.2 Analysis of the Characteristics

In the previous subsection, I examined the throughput performance of TCP with simple FEC in various environments. In this subsection, to analyze the results in detail, I investigate the retransmission rate, the effective recovery rate, and the redundancy rate.

First, Figure 3.9 shows the retransmission rate when the buffer size is varied from 50 to 1000 packets with a delay time of 200 ms and 20 flows. From this

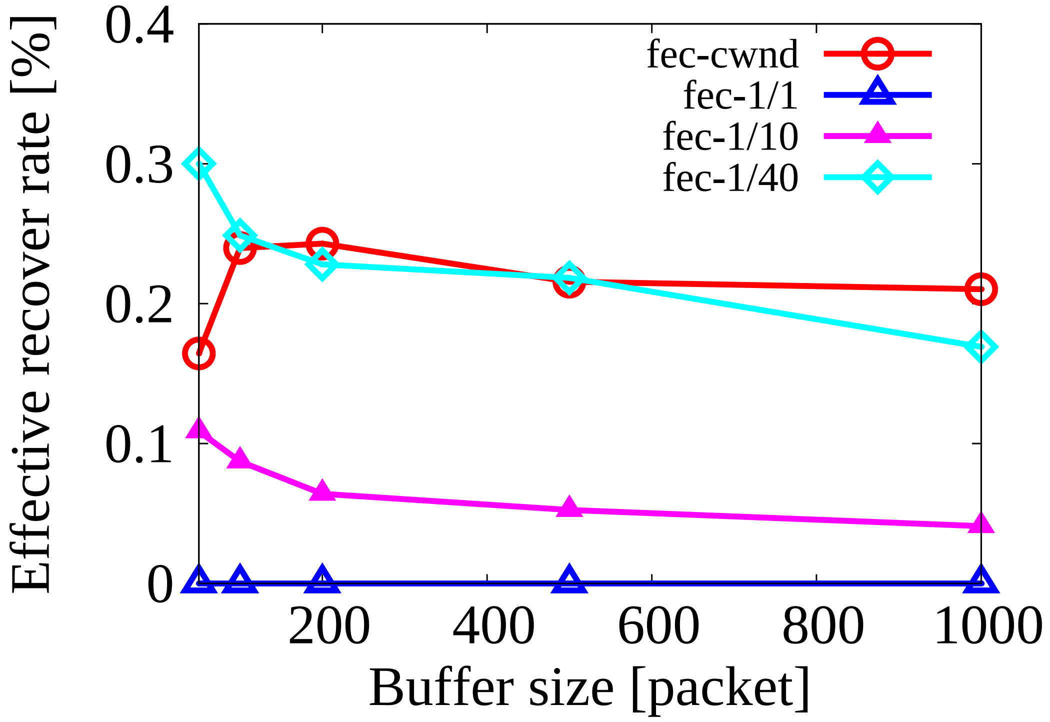


Figure 3.10: Effective recovery rate

figure, I can see that TCP with simple FEC suppresses the retransmission rate to approximately the same degree as the fec-1/1 method by adapting redundancy according to network conditions regardless of the buffer size. On the other hand, the fec-1/40 and fec-1/10 methods have a high retransmission rate because the redundancy of these methods is too low.

Next, Figure 3.10 shows the effective recovery rate when the buffer size is varied from 50 to 1000 packets with a delay time of 200 ms and 20 flows. These results show that TCP with simple FEC achieves a high recovery rate almost equal to that of the fec-1/40 method. On the other hand, the fec-1/1 and fec-1/10 methods waste redundant packets because the redundancy of these methods is too high for the network conditions.

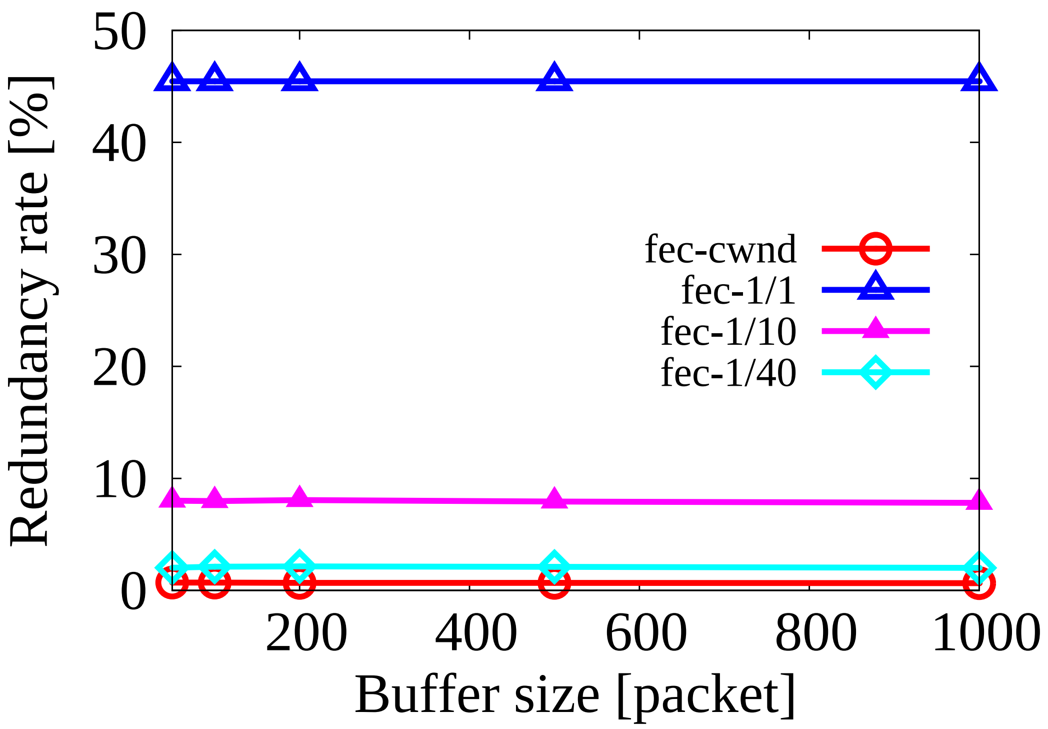


Figure 3.11: Redundancy rate

Last, I investigate the load of TCP with simple FEC on the network. Figure 3.11 shows the redundancy rate when the delay time is set to 200 ms, and the number of flows is set to 20. From this figure, I can see that TCP with simple FEC has a very low redundancy rate compared with those of the other methods. Consequently, TCP with simple FEC can recover lost packets effectively and prevent retransmission by introducing adaptive FEC without placing an intensive load on networks.

3.6 Conclusion

A Scheme is considered to simply apply FEC technology to the entire TCP operation. To improve TCP performance in such networks, TCP with simple FEC adapts FEC redundancy to the TCP transmission rate while considering the tradeoff between the recovery success rate and effective use of network resources. Simulation evaluations show that TCP with simple FEC enables higher throughput than the conventional methods, especially in high latency environments. I will consider a scheme to more appropriately determine the appropriate redundancy level.

4 Detailed Examination of TCP with FEC

In this chapter, I examine the characteristics of the proposed scheme in detail. A simple application of FEC to TCP operation might not work effectively. If the redundancy is too low, lost packets might not be recovered effectively. Moreover, unnecessary retransmissions are possibly caused due to the reception of duplicate ACKs even if recovery is successful.

4.1 Introduction

In this study, I propose a scheme to apply FEC technology to the entire TCP operation in order to prevent degradation of transmission rates as well as improve throughput. However, a simple application of FEC technology to TCP operation might not work effectively. It causes unnecessary retransmission due to the reception of duplicate ACKs even if recovery is successful as shown in Figure 4.1.

The sender creates a redundant packet XORed with the payload field of all packets in a congestion window (*cwnd*) and transmits the redundant packet following the original packets. Namely, TCP-FEC forms a group of a redundant packet and corresponding original packets. The receiver thus can use the redundant packet to recover a lost packet within the *cwnd*. This scheme cannot recover

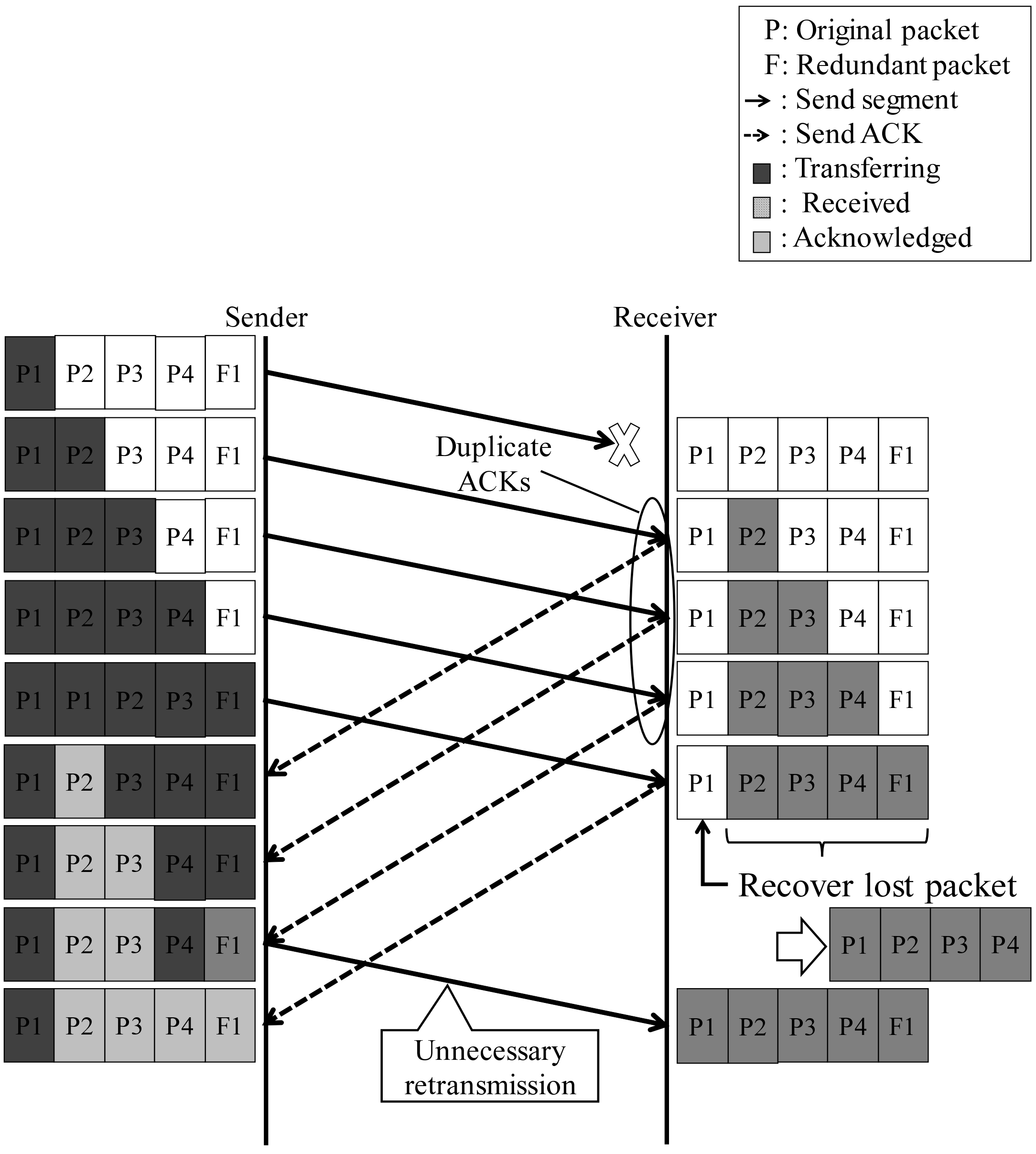


Figure 4.1: Unnecessary retransmission

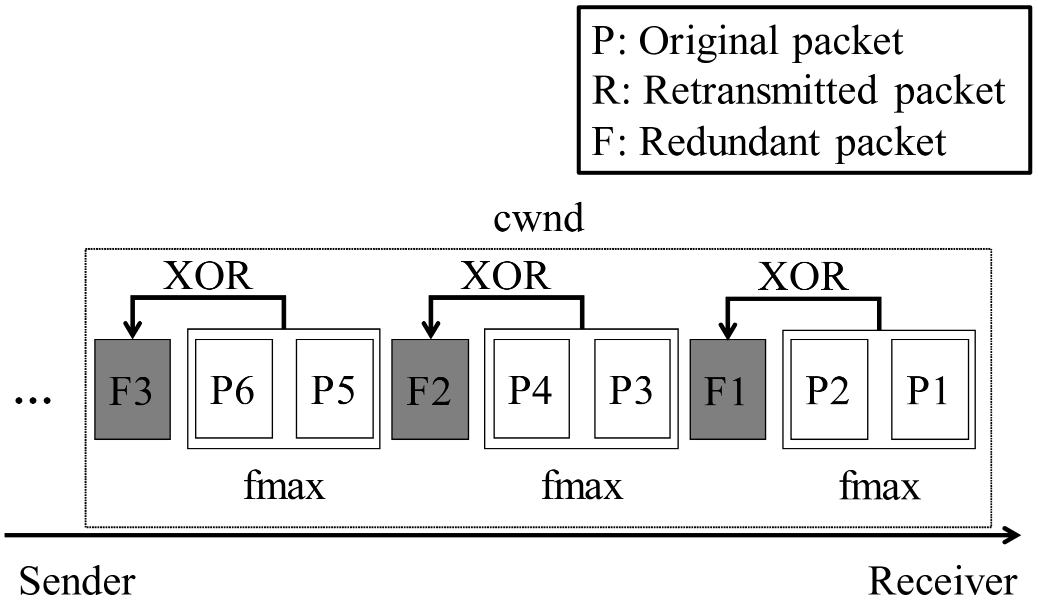


Figure 4.2: Overview of redundancy control

lost packets when two or more packet losses occur within a *cwnd*. In that case, the sender retransmits the lost packets through the original TCP operation.

Therefore, I consider three ways to control the redundancy level according to transmission rates, to limit minimum redundancy, and to suppress the return of duplicate ACKs. First, TCP-FEC introduces an upper limit to group size "fmax" to prevent inefficient operation due to too low redundancy. Next, TCP-FEC suppresses returning duplicate ACKs until it is determined that a lost packet cannot be recovered; i.e., when another original packet or a redundant packet in a group is lost. I show the effectiveness of this scheme through simulation evaluations.

4.2 Redundancy Control

When *cwnd* is large, i.e., network condition is good, redundancy should be low to prevent additional loads; otherwise, it means that network condition is bad or transmissions have just started, so that redundancy should be high to prevent packet losses. Redundancy is defined as the ratio of a redundant packet to the corresponding original packets; i.e., $1/(cwnd - 1)$. For example, redundancy is 1 when *cwnd* is 2. FEC group size is updated when the sender sends a redundant packet, i.e., at the end of an FEC group. If redundancy is too low, lost packets might not be recovered effectively. Therefore, in TCP-FEC for random loss environments, an upper limit to group size, "$fmax$", is introduced to prevent inefficient operation as shown in Figure 4.2. Redundancy is then expressed as $1/\min(cwnd - 1,\ fmax)$. The effect of $fmax$ was evaluated through simulations.

4.3 ACK Control

As mentioned above, FEC mechanisms cannot work effectively if simply applied to TCP operation. When the redundancy is particularly low, FEC mechanisms might cause unnecessary retransmission due to the reception of duplicate ACKs even if recovery succeeds and might not recover lost packets effectively.

TCP-FEC for random loss environments can recover a lost packet from a redundant packet within a group. FEC mechanisms might cause unnecessary retransmissions due to the reception of duplicate ACKs even if recovery succeeds. Therefore, TCP-FEC for random loss environments suppresses returning duplicate ACKs until it is determined that lost packets cannot be recovered; namely, when another original packet or a redundant packet in a group is lost as shown in Figure 4.3.

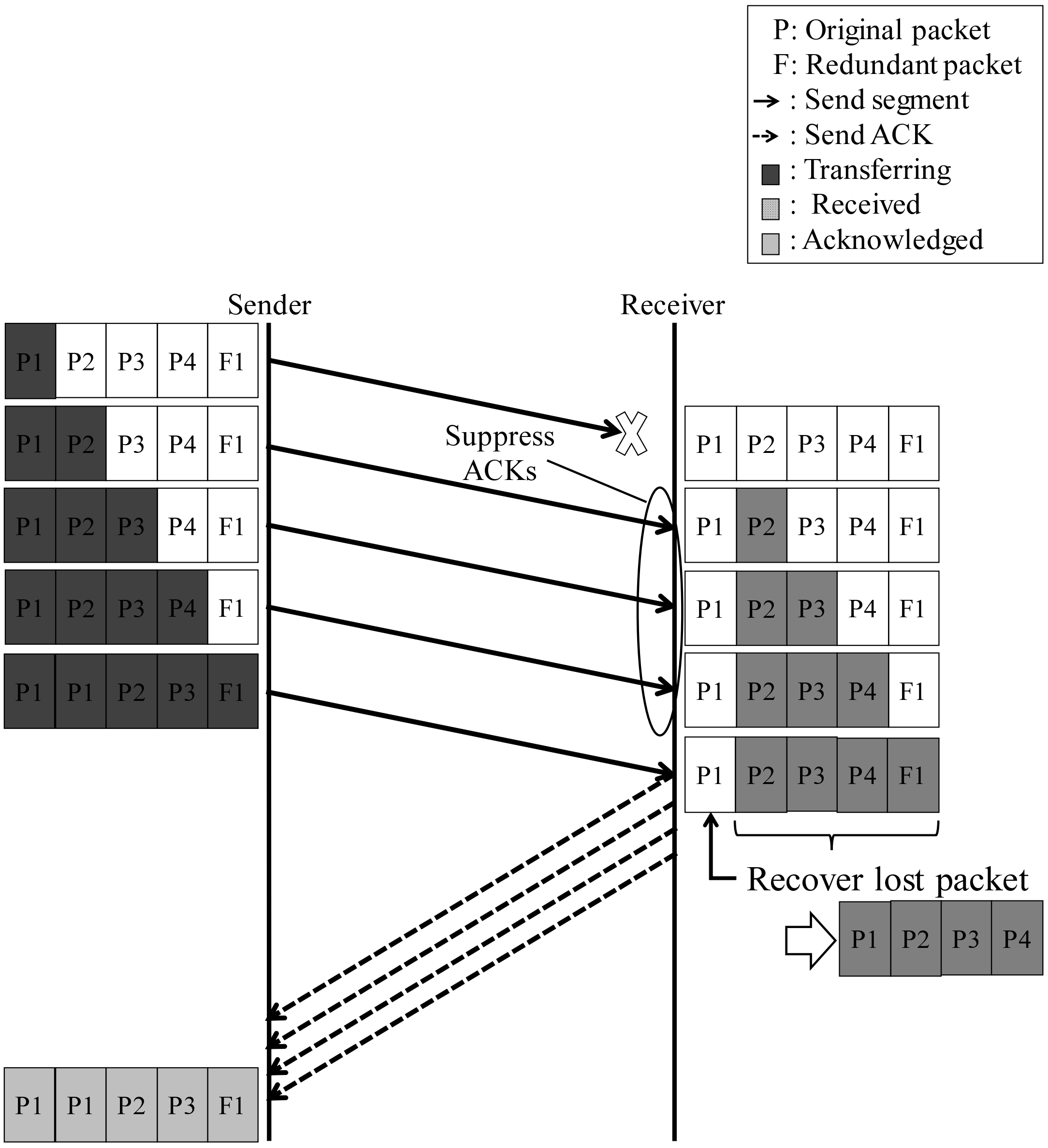


Figure 4.3: ACK control

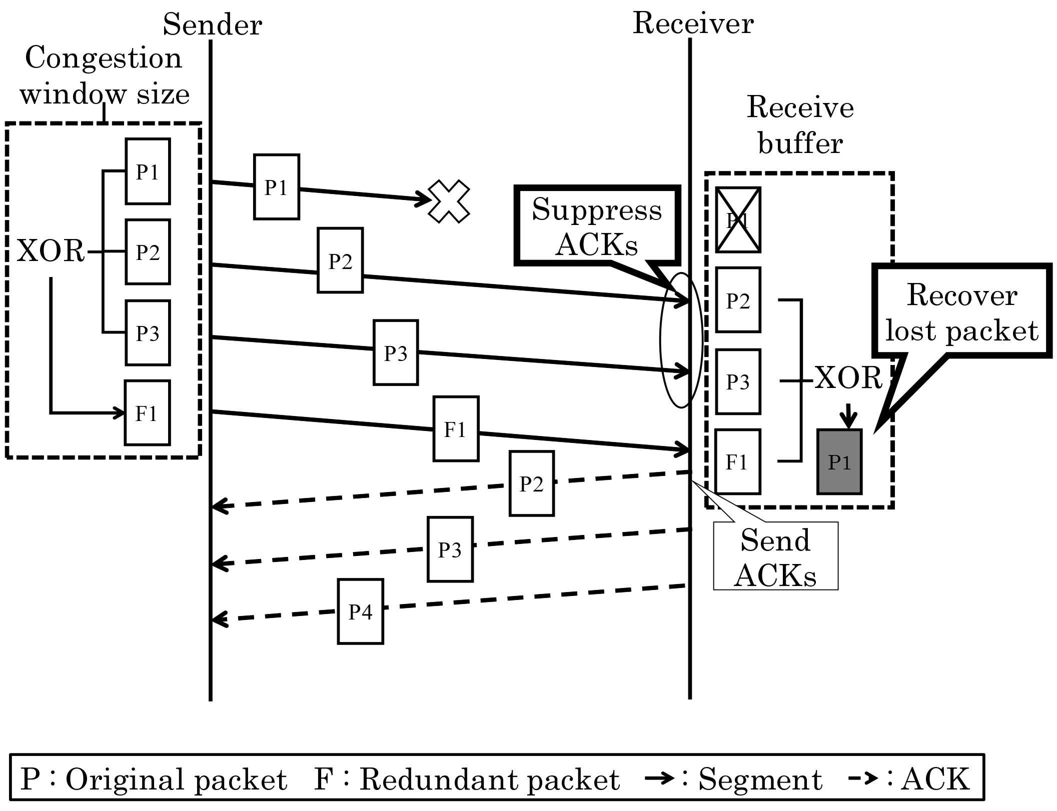


Figure 4.4: Overview of TCP with FEC

The redundancy decreases as the transmission rate increases considering network efficiency. This means that redundancy is high at a low transmission rate where packet losses have a significant impact. Figure 4.4 illustrates the operation of TCP-FEC. When destination receives TCP segments with duplicate sequence number, it discards duplicating TCP segment as shown Figure 4.5.

4.4 Simulation Evaluation

To investigate the efficiency of TCP-FEC, I evaluated it through simulation using Network Simulator ns-2 ver. 2.35 [27] after implementing TCP-FEC. In this

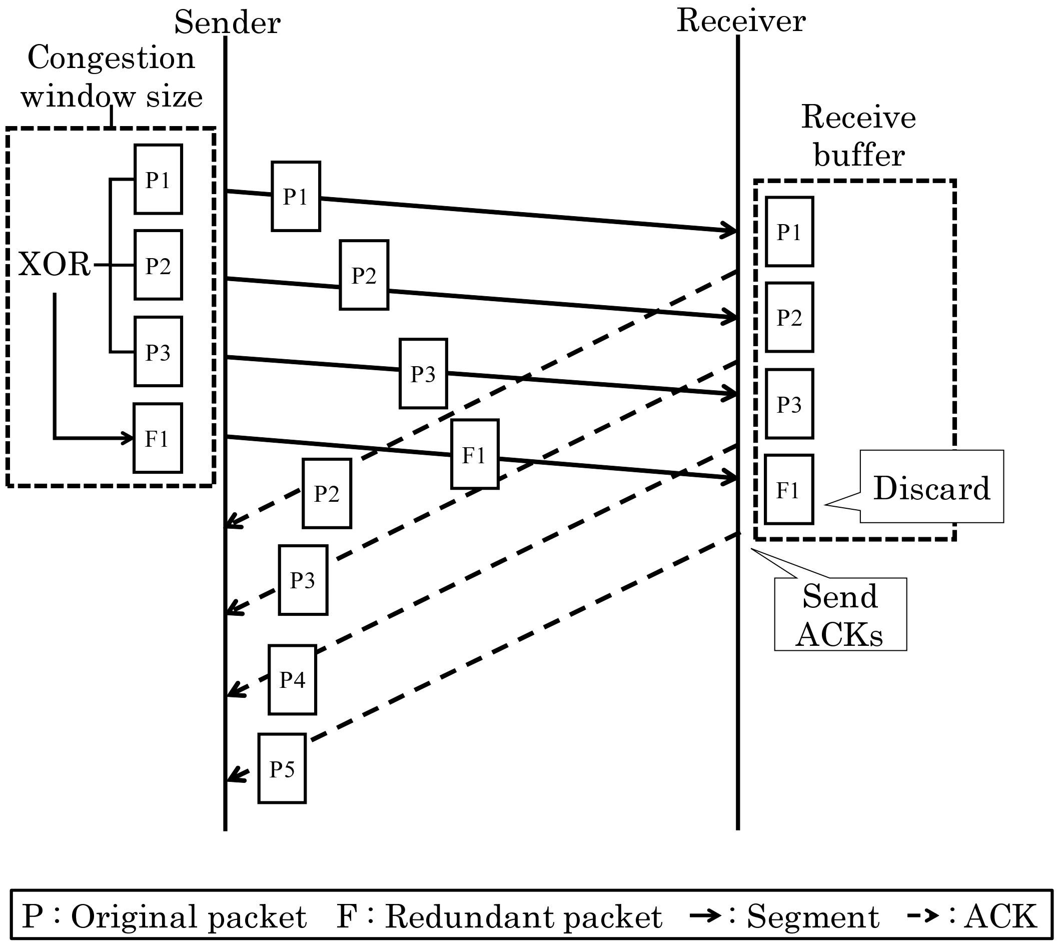


Figure 4.5: Behavior of proposed scheme without recover

simulation, a sender communicates with the corresponding receiver through two routers as show in Figure 4.6. The parameters used in the simulation are summarized in Table.4.1. I assumed that random packet loss of 0.001–1% occurs over the link between routers. The link has a bandwidth of 100 Mb/s and a delay time of 10–300 ms. Other links have a bandwidth of 100 Mb/s and a delay time of 1 ms.

The sender employing TCP with selective ACK option (sack) and TCP-FEC

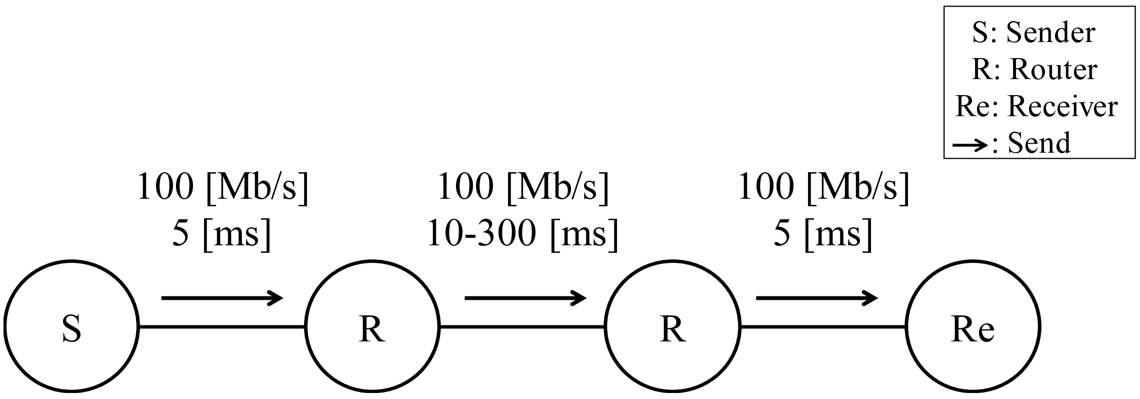


Figure 4.6: Simulation topology

Table 4.1: Simulation parameters

Simulation time	220 [s]
Buffer size	50 [packet]
Packet size	1500 [byte]
Random loss rate	0.001–1 [%]
Number of trials	1–20
Upper limit of group size (fmax)	1, 10, 40

for random loss environments transmits continuous data packets to the receiver. In TCP-FEC for random loss environments, I compared the performance of three methods: "fec" method where the redundancy is adapted according to *cwnd*, "fec+fmax *n*" methods which sets the upper limit of group size to *n* packets in addition to the "fec" method, and "fec-dup" method which does not suppress duplicate ACKs. In this simulation, the fmax value varied from 1 to 50.

To evaluate the effectiveness of TCP-FEC, I focus on the average throughput, effective recovery rate, and number of TCP timeouts and fast recoveries. The

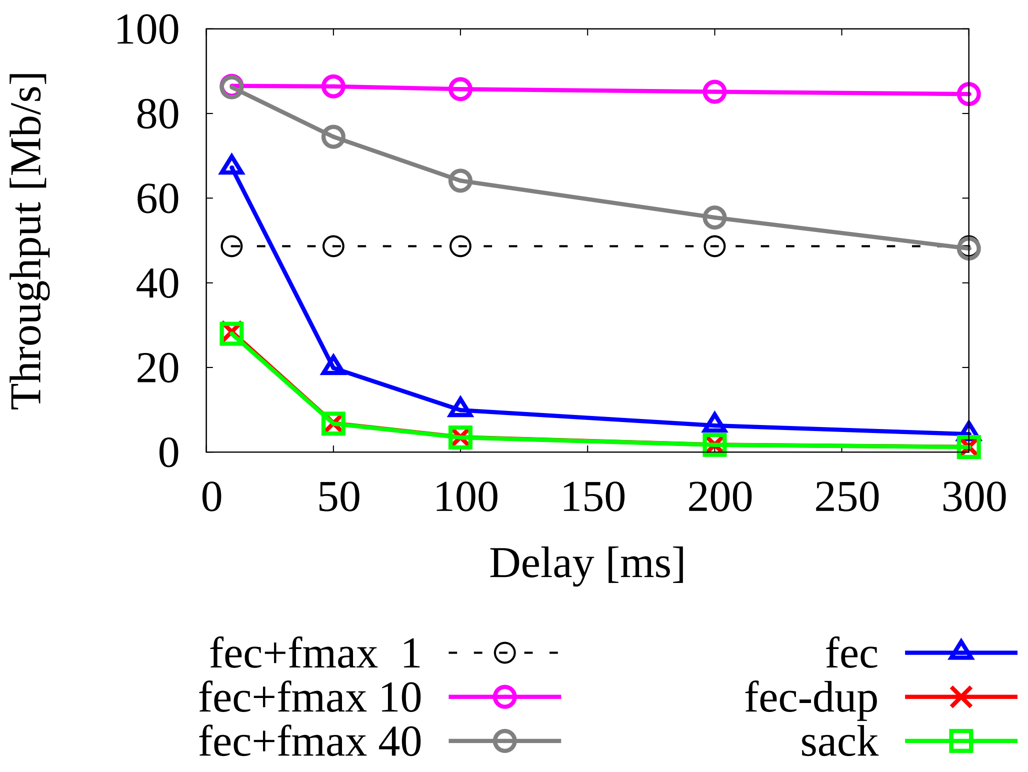


Figure 4.7: Effect of delay time (Packet loss rate = 0.05%)

effective recovery rate is defined as the ratio of recovered packets to redundant packets; e.g., the rate of 10% means that the redundant packets of 90% are not utilized effectively. I evaluated the average values over 40 simulation runs with different random seeds.

4.5 Simulation Results

Figure 4.7, 4.8, 4.9, 4.10 shows the average throughput performance when the delay time, packet loss rate, and fmax vary. From Figure 4.7, TCP-FEC achieves higher throughput than "sack" method. Especially, "fec+fmax 10" method achieves excellent throughput even as the delay time increases although throughput de-

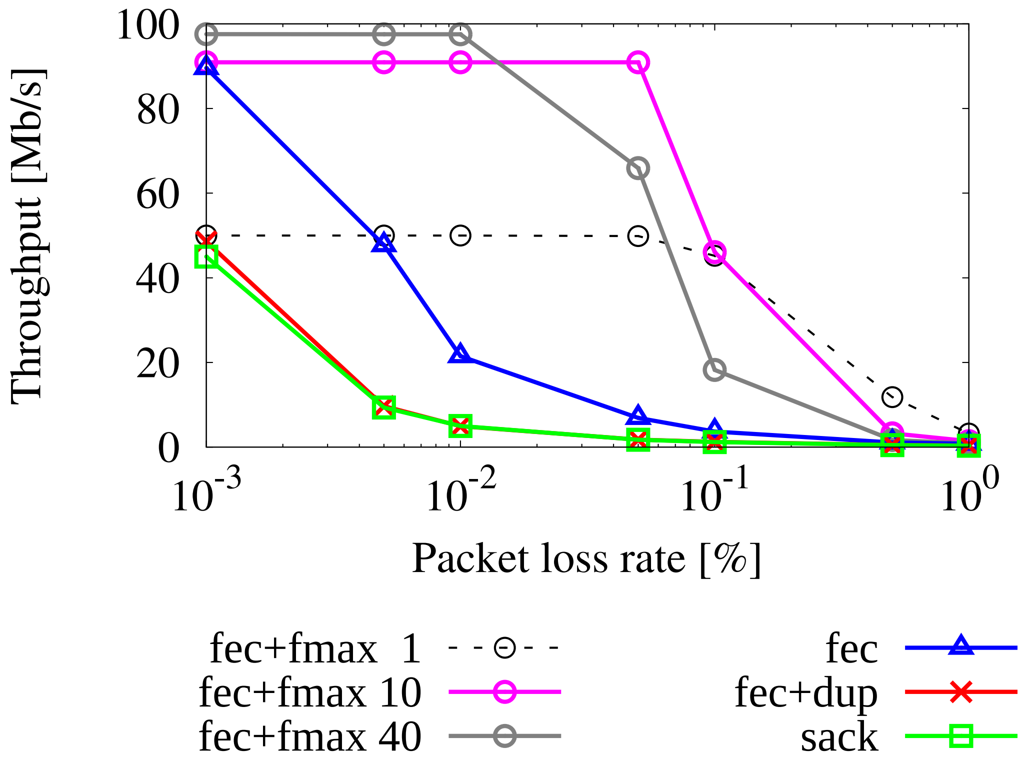


Figure 4.8: Effect of packet loss rate (Delay time = 200 ms)

teriorates with other methods. On the other hand, "fec-dup" method attains as low throughput as "sack" method. Namely, FEC technology cannot work effectively if simply applied to TCP operation. Figure 4.8 shows that TCP-FEC achieves higher throughput than "sack" method in a wide range of packet loss rates. Our scheme with large fmax achieves good throughput at a low packet loss rate, while that with small fmax does it at a high packet loss rate. To investigate the effect of fmax on throughput performance in detail, the average throughput of "fec+fmax n" methods is shown in Figures 4.9 and 4.10, when the fmax value varies. These results indicate that the packet loss rate has a large impact on the effect of fmax compared with the delay time. Since large fmax means low

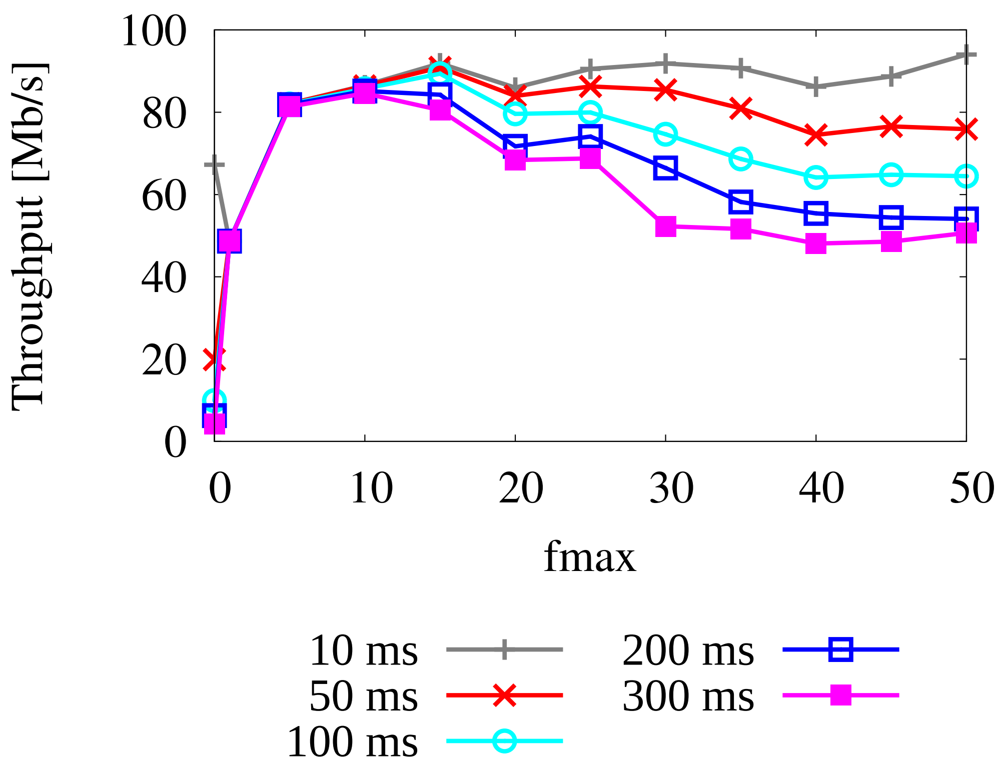


Figure 4.9: Effect of fmax (Packet loss rate = 0.05%)

redundancy, it degrades throughput significantly at a long delay time and high packet loss rate. On the other hand, small fmax causes inefficient transmission due to high redundancy. Therefore, an appropriate fmax value is 5–10 in this environment.

Let's investigate the reason for the above improvement. The number of fast recoveries and timeouts, and effective recovery rate of each method are shown in Figures 4.11, 4.12 and 4.13, when the delay time is set to 200 ms. From Figure 4.11, I can see that "fec+fmax n" and "fec" methods reduce the number of fast recoveries much more than the other methods do. However, "fec+fmax 40" and "fec" methods cause many timeouts at a high packet loss rate as shown in

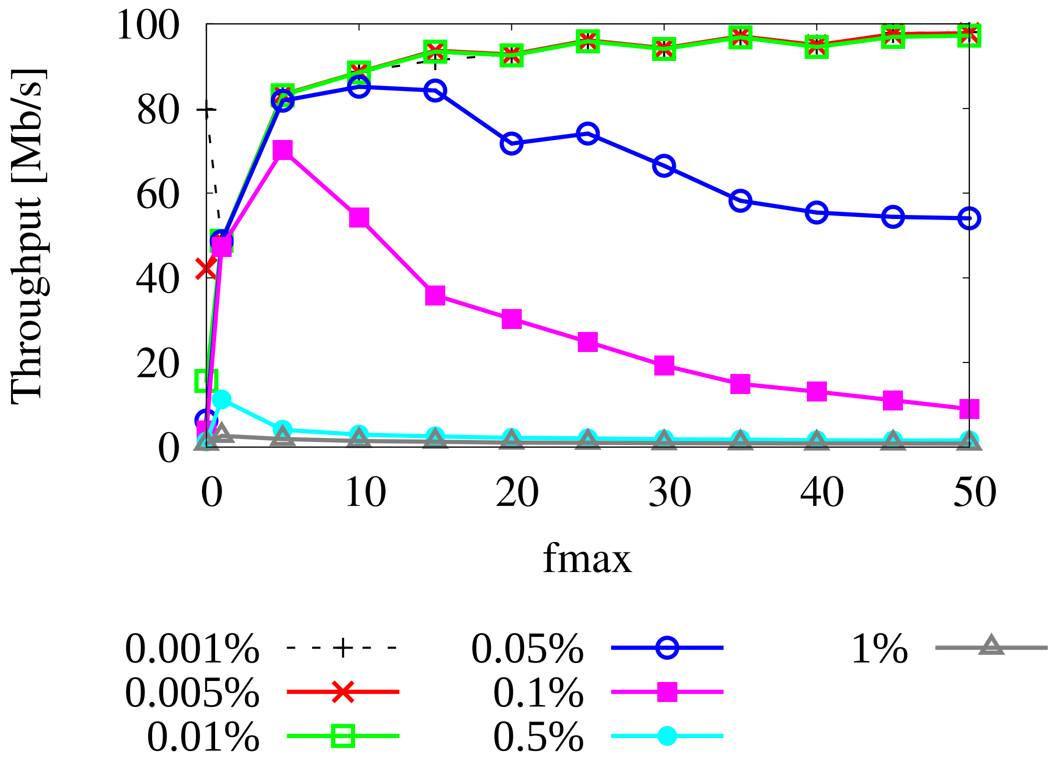


Figure 4.10: Effect of fmax (Delay time = 200 ms)

Figure 4.12. This is because the suppression time of duplicate ACKs increases as the group size increases in these methods. It causes many timeouts as well as a little throughput improvement. On the other hand, "fec+fmax 1" method drastically reduces the number of fast recoveries and timeouts, although the effective recovery rate of it is very low as shown in Figure 4.13. Clearly, "fec+fmax 10" method can recover lost packets effectively and prevent retransmission by redundant packets. Consequently, TCP-FEC improves throughput performance regardless of the delay time by setting an appropriate redundancy level according to packet loss rates.

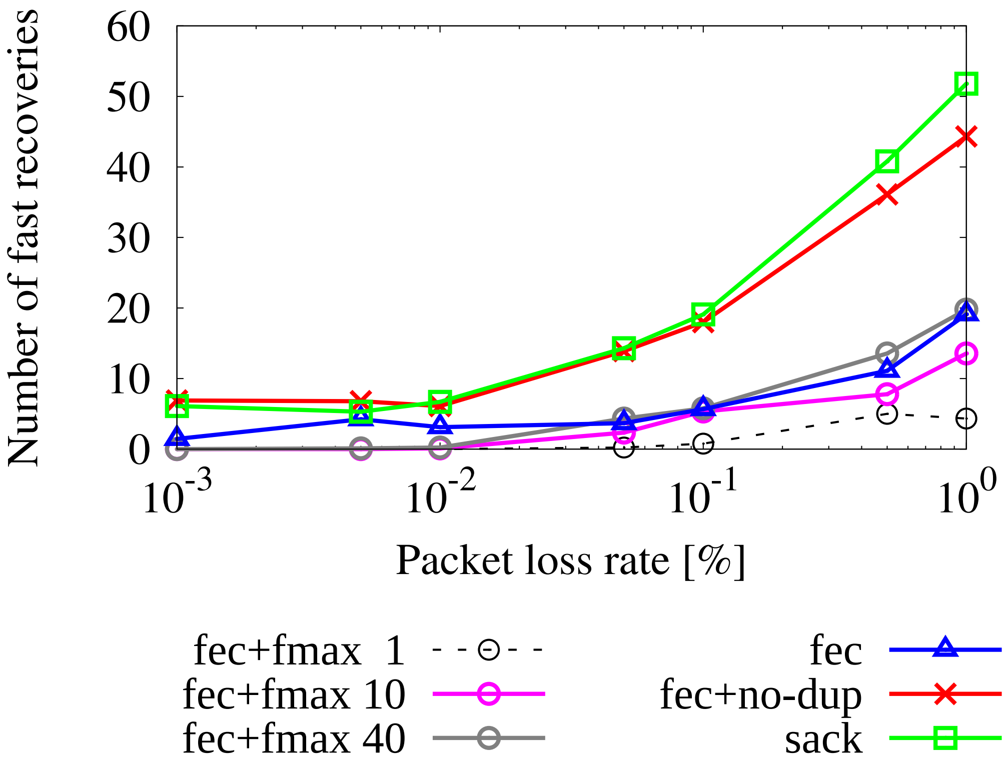


Figure 4.11: Number of fast recoveries

4.6 Conclusion

Packet losses significantly degrade TCP performance in high latency networks. To improve TCP performance in such networks, I proposed a scheme to apply FEC technology to the entire TCP operation. TCP-FEC for random loss environments consists of three mechanisms to control the redundancy level, to limit minimum redundancy, and to suppress duplicate ACKs. Simulation evaluations show that TCP-FEC for random loss environments improves throughput significantly by suppressing the return of duplicate ACKs and controlling minimum redundancy, especially in high latency environments, although FEC technology cannot work effectively when simply applied to TCP operation. In future work, I will consider

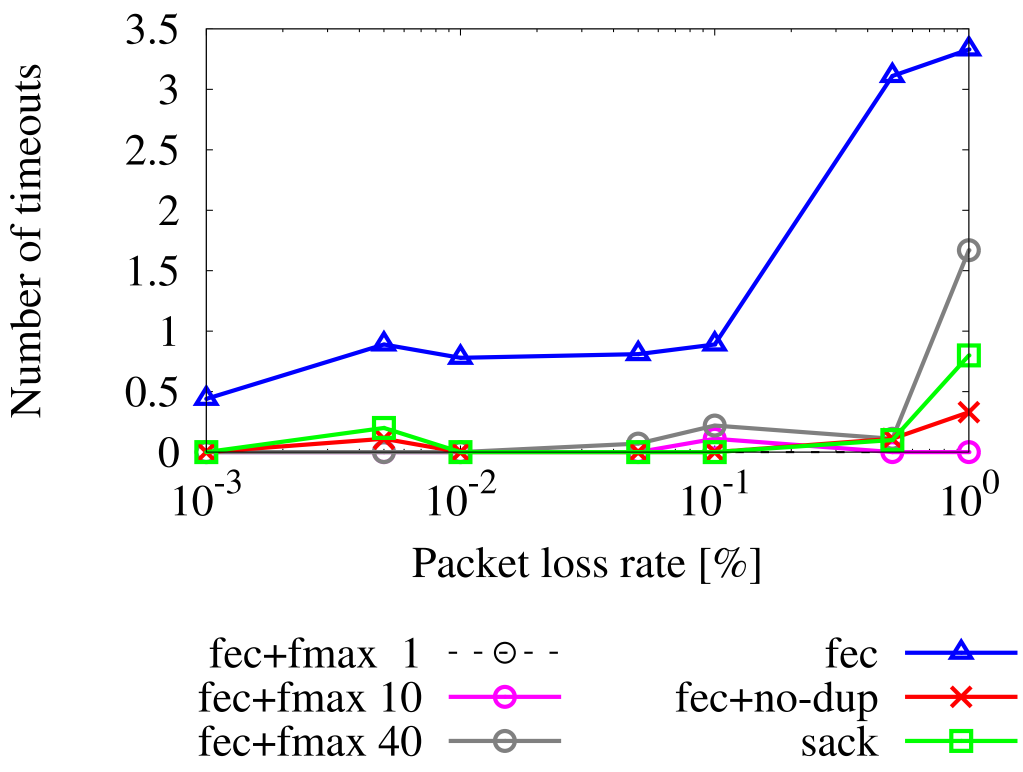


Figure 4.12: Number of timeouts

a scheme to determine the appropriate redundancy level for network conditions and to more effectively recover lost packets in a real environment, such as where bust packet losses occur.

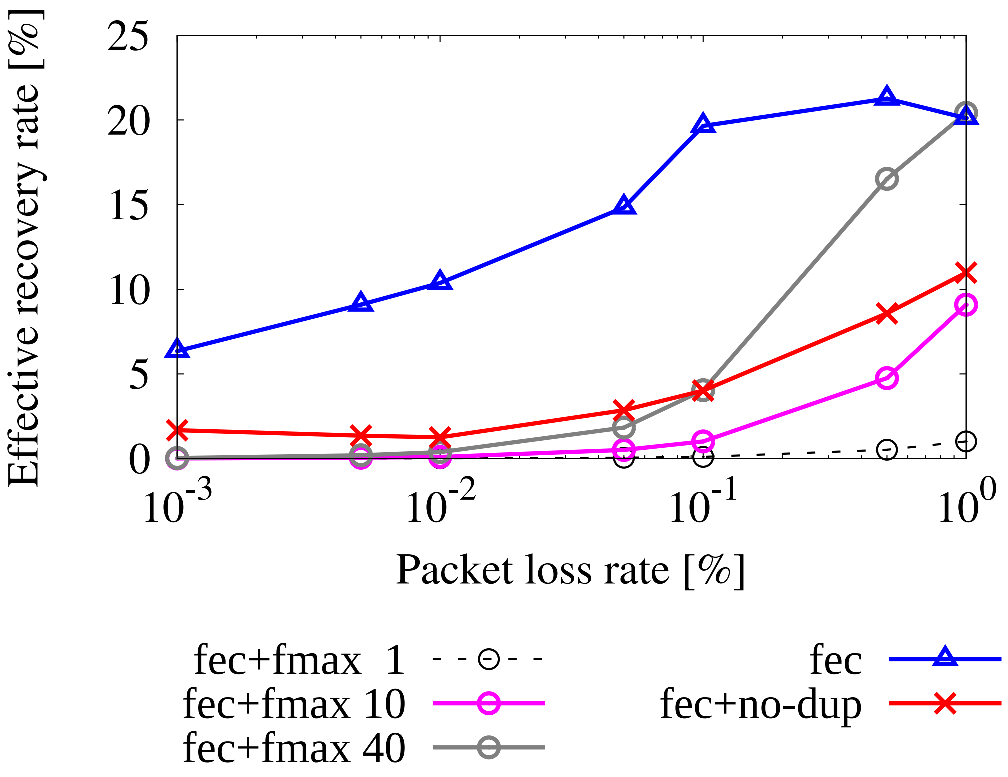


Figure 4.13: Effective recovery rate

5 Adapting TCP with FEC to Real Environment

In this chapter, I consider adapting the TCP-FEC with Interleave to real environments. FEC cannot recover lost packets if both of original and redundant packets are "burstily" lost in a network. In addition, when FEC recovers lost packets, congestion does not be controlled through original TCP operations.

5.1 Introduction

In this study, aiming to improve throughput, a scheme to apply FEC to the entire TCP operation is proposed. If the redundancy is too low, lost packets might not be recovered effectively. Moreover, unnecessary retransmissions and timeouts are possibly caused, due to the reception of duplicate ACKs and lack of congestion avoidance, respectively, even if recovery is successful.

On the Internet, packet losses commonly occur "burstily" rather than randomly as shown in Figure 5.1. FEC cannot recover lost packets if both of original and redundant packets are "burstily" lost in a network. In addition, when FEC recovers lost packets, congestion does not be controlled through original TCP operations. Therefore, a scheme to control transmission rates when recovery is successful and interleave redundant packets from original packets is proposed

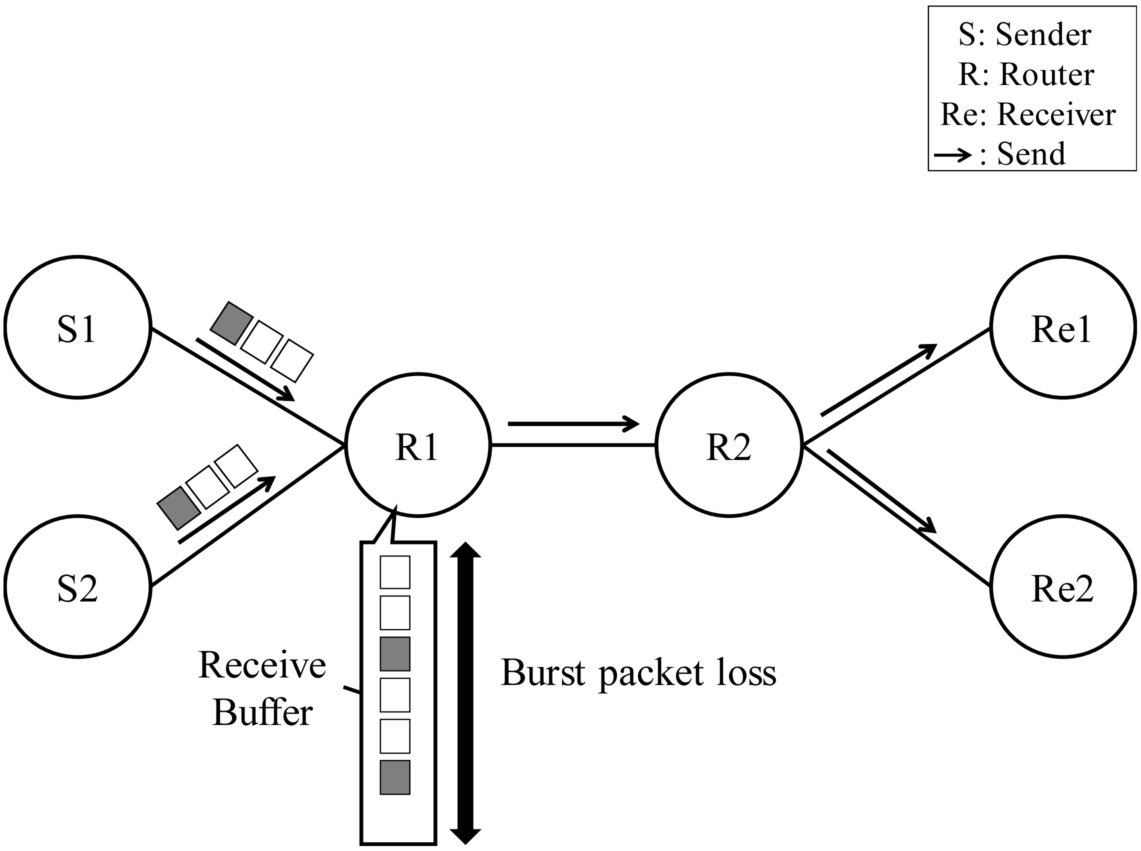


Figure 5.1: Burst packet loss on a bottleneck link

hereafter. The effectiveness of this scheme is demonstrated through simulation evaluations in burst loss environments such as actual environment.

5.2 Congestion Control

TCP-FEC for burst loss environments controls congestion based on the conventional scheme such as TCP NewReno, which generally performs by detecting packet losses. Since FEC mechanisms can recover lost packets, congestion does not be controlled through original TCP operations in spite of excessive transmission rate as shown in Figure 5.2. To avoid congestion, TCP-FEC with Interleave

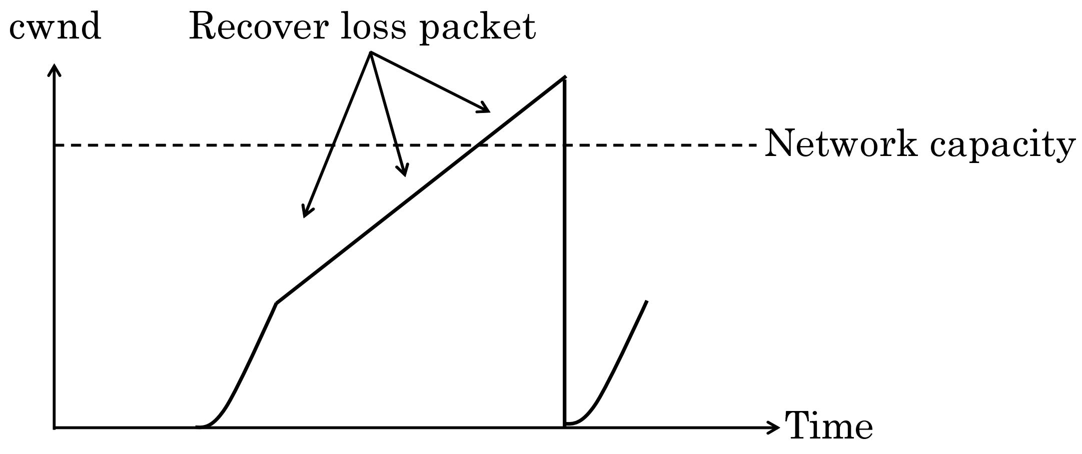


Figure 5.2: TCP-FEC without congestion avoidance

for burst loss environments uses ECN to notify the sender that lost packets have been successfully recovered as shown in Figure 5.3. Specifically, the receiver adds ECN information to a returning ACK when it recovers a lost packet. The sender decreases transmission rate when it receives the ACK with ECN. Transmission rate, i.e., *cwnd*, is calculated by using a reduction factor, "r" ($0 < r \leq 1$), as

$$cwnd = r * cwnd. \tag{5.1}$$

For example, the reduction factor of TCP NewReno [28] is 0.5, and that of CUBIC [29] is 0.8. The effect of the reduction factor on throughput performance was evaluated through simulations.

5.3 Interleave Control

On the Internet, packet losses commonly occur "burstily" rather than randomly. In random loss environment, TCP-FEC can effectively recover the lost packet as shown in Figure 5.4. On the other hand, transmitting a redundant packet

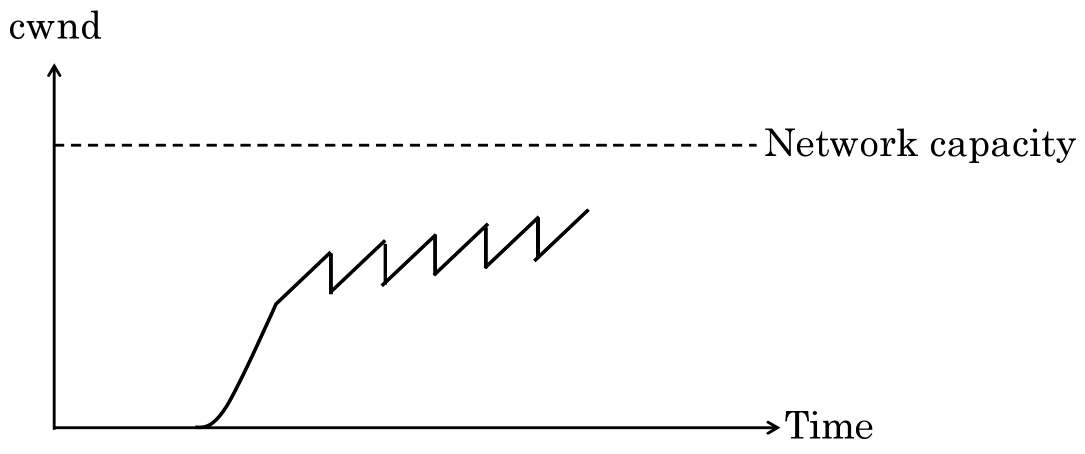


Figure 5.3: Concept of congestion control in TCP-FEC

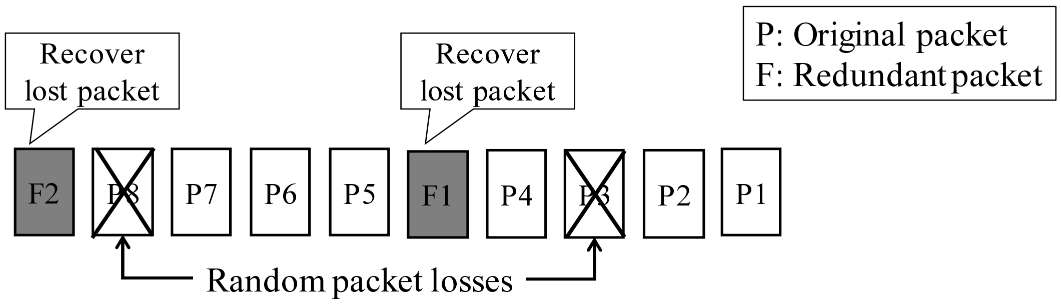


Figure 5.4: Recover the lost packet in random loss environments

following original packets might cause unsuccessful recovery due to burst packet losses as shown in Figure 5.5. Therefore, as shown in Figure 5.6, TCP-FEC with Interleave for burst loss environments interleaves each packet of a group with packets of other groups to prevent multiple packet losses belonging to the same group. The number of groups within a *cwnd* is defined as "g," and each group consists of the same number of packets to be interleaved. Namely, with TCP-FEC for burst loss environments, number of groups is constant, and group size is dynamically controlled. Group size is calculated as shown in Figure 5.7. When

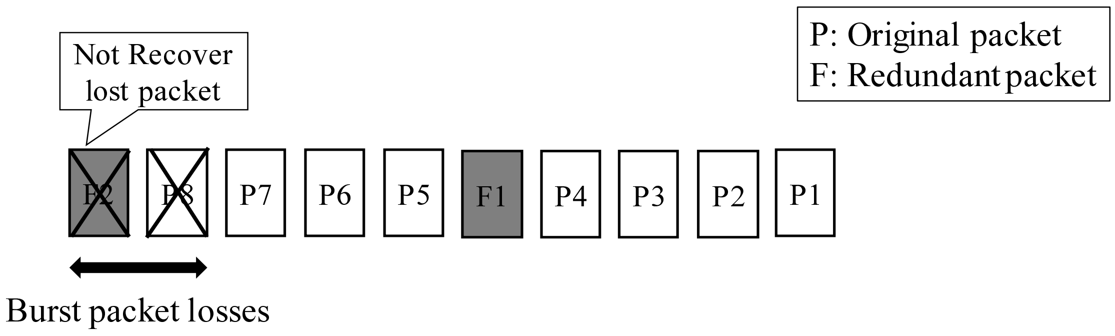


Figure 5.5: Recover the lost packet in burst loss environments

cwnd is less than $2 * g$, the number of groups is set to 1; i.e., the packets cannot be interleaved. The large number of groups will have high tolerance to burst packet losses. However, timeouts will be caused by duplicate ACK suppression because interleaving elongates maximum ACK suppression time. The effect of the number of groups was evaluated through simulations as described in the following.

5.4 Simulation Experiment

The effectiveness of TCP-FEC with Interleave for burst loss environments in the case of burst loss environments was evaluated through simulation using Network Simulator ns-3 [30] after its implementation. Note that TCP-FEC with Interleave for burst loss environments is based on simple TCP NewReno to focus on the effect of recovery by FEC.

The parameters used in the simulation are summarized in Table 5.1. As shown in Figure 5.8, a sender communicates with the corresponding receiver; that is, the sender transmits continuous data packets to the receiver. Since TCP throughput in burst loss environments is focused on in this study, it is assumed that the bottleneck link has a bandwidth of 50 Mb/s and a delay time of 50–200 ms.

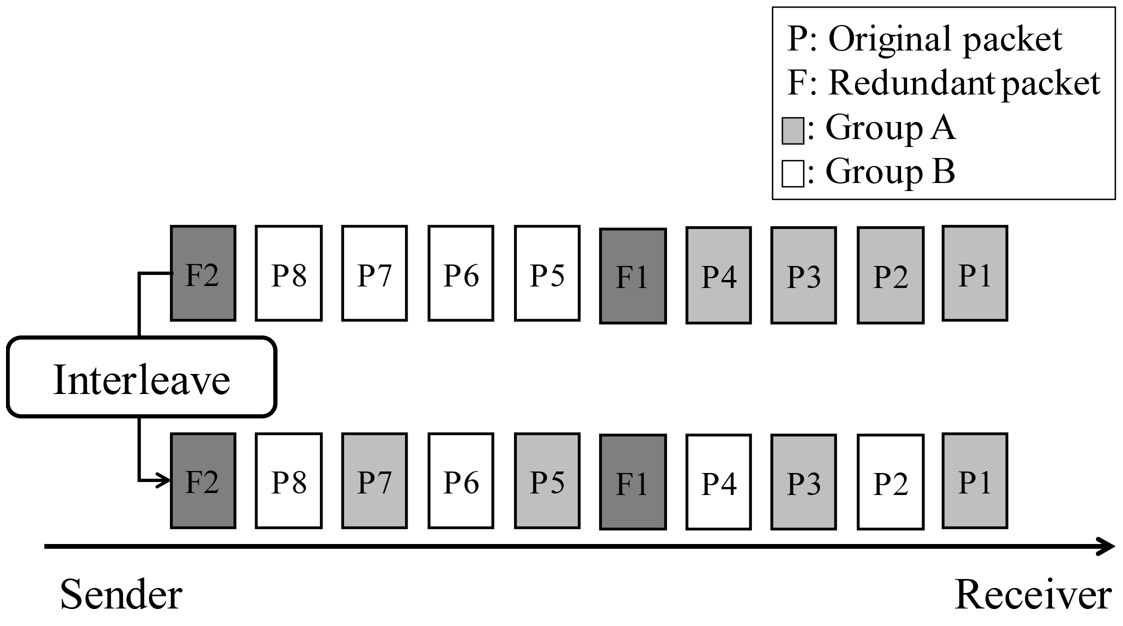


Figure 5.6: Interleave control

Other links have a bandwidth of 100 Mb/s and a delay time of 1 to 5 ms.

In this simulation, the upper limit of group size (l) was varied from 0 to 40. l of "0" means that TCP-FEC with Interleave for burst loss environments does not limit group size. The reduction factor of transmission rates (r) was varied from 0.5 to 1.0. r of 1.0 means that TCP-FEC with Interleave for burst loss environments does not decrease transmission rate when it recovers a lost packet. Note that r of 0.5 intends the same value of conventional scheme and r of 0.7 intends a middle value of 0.5–1.0. Number of groups (g) was varied from 1 to 5. g of "1" means that TCP-FEC with Interleave for burst loss environments does not interleave redundant packets.

Performance of TCP-FEC with Interleave for burst loss environments, focusing on total throughput, compared with that of the conventional TCP NewReno , was evaluated. Moreover, the characteristics of TCP-FEC with Interleave for

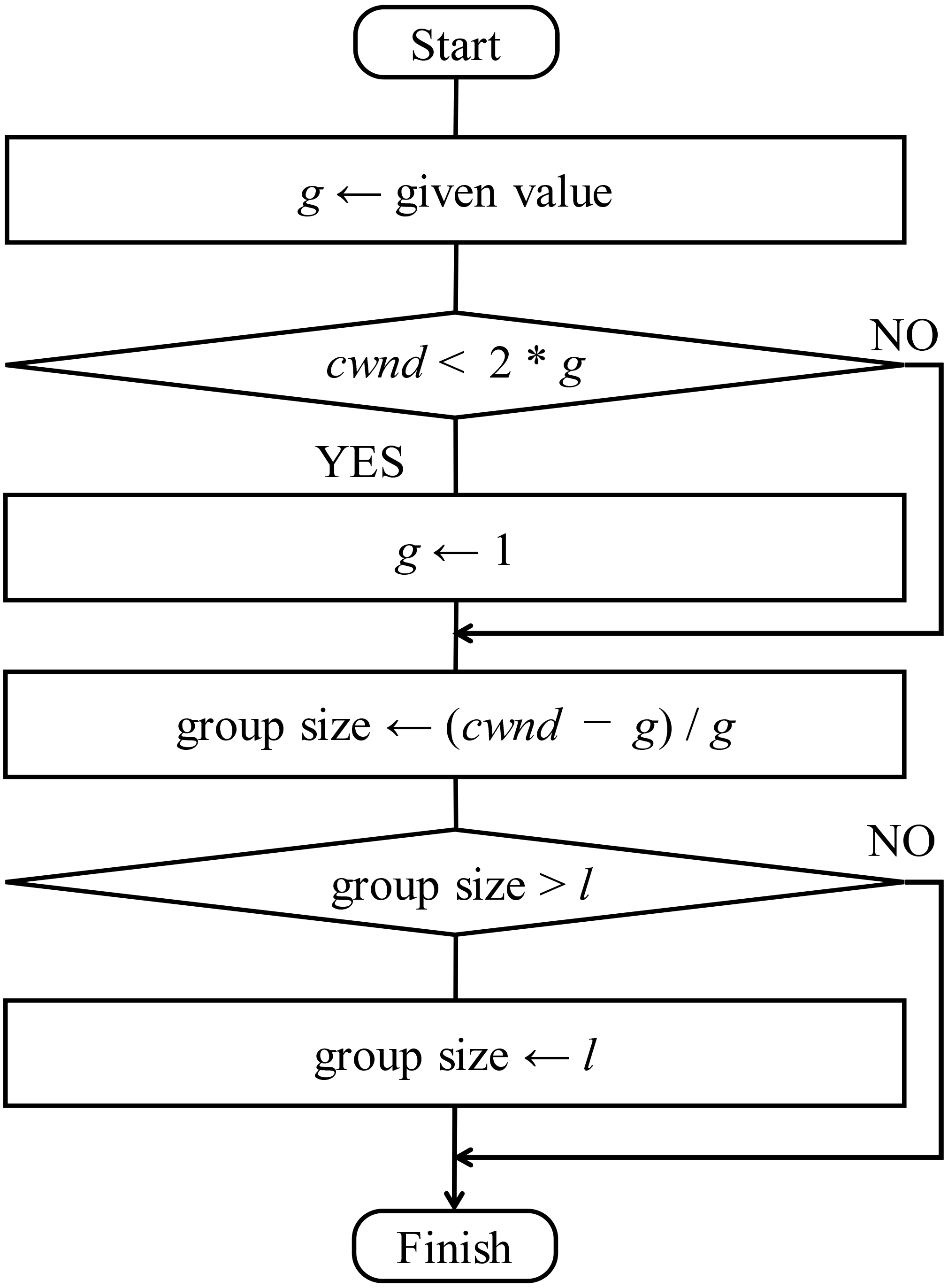


Figure 5.7: Calculation of group size

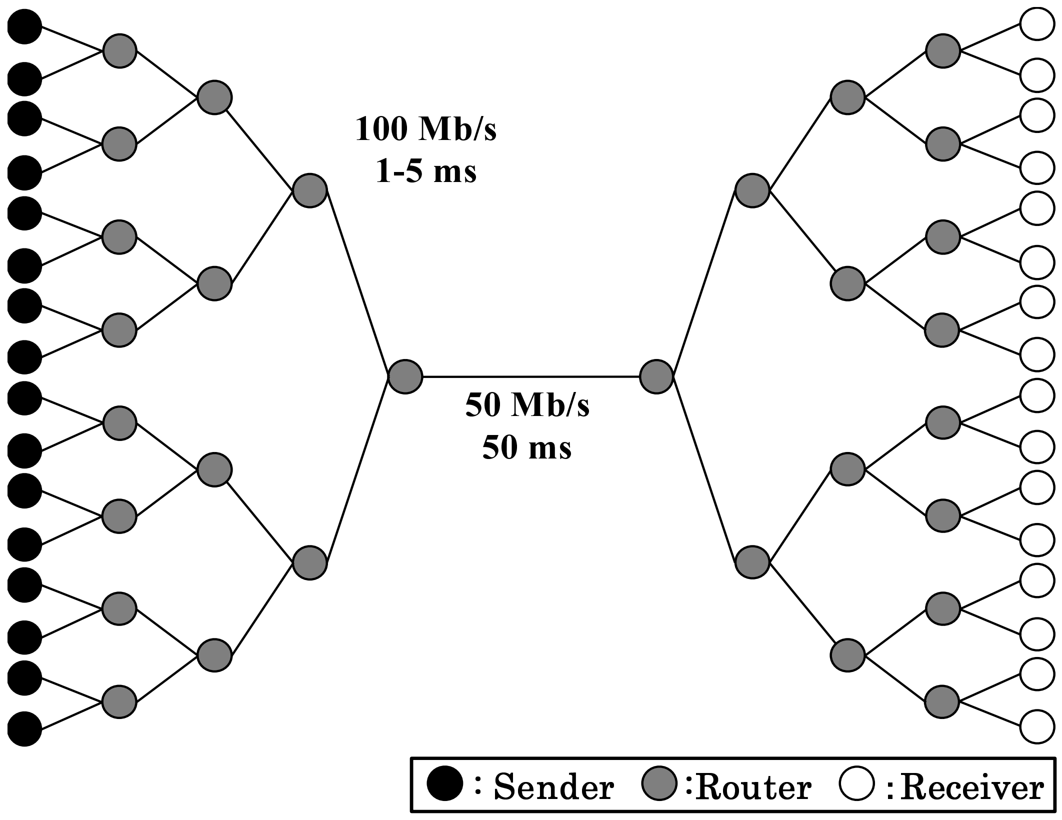


Figure 5.8: Simulation model

burst loss environments were analyzed in terms of number of TCP fast recoveries and timeouts, number of recovery packets, average redundancy rate, and effective recovery rate. Effective recovery rate is defined as the ratio of recovered packets to redundant packets; for example, a rate of 10 % means that 90 % of redundant packets are not utilized effectively. Furthermore, the characteristics of the proposed and conventional schemes in transient and steady states were analyzed. Transient and steady states are defined as a simulation period from 0 to 20 s and that from 20 to 60 s, respectively.

Table 5.1: Simulation parameters

Simulation time	60 [s]
Buffer size on routers	300 [packet]
Buffer size on end nodes	∞
Number of flows	8,16
Segment size	1000 [Byte]
Number of trials	10
Upper limit of group size (l)	0, 10, 20, 30, 40
Reduction factor (r)	0.5, 0.7, 1.0
Number of groups (g)	1, 2, 3, 4, 5
TCP algorithm	TCP NewReno

5.5 Simulation Results

The simulation results are presented in the following, and the effectiveness of TCP-FEC with Interleave compared with the conventional TCP is discussed. The effect of upper limit of group size (l), reduction factor of transmission rate (r), and number of groups (g) were investigated first. The effect of delay time and number of flows were also examined. Next, the characteristics of proposed and conventional schemes were then analyzed.

5.5.1 Preliminary Evaluation

First of all, I evaluate the fundamental characteristics of TCP-FEC with Interleave for burst loss environments compared with the conventional scheme (TCP NewReno and TCP-AFEC). The topology and parameters used in this simulation are respectively summarized in Figure 5.9 and Table 5.2, which is the same

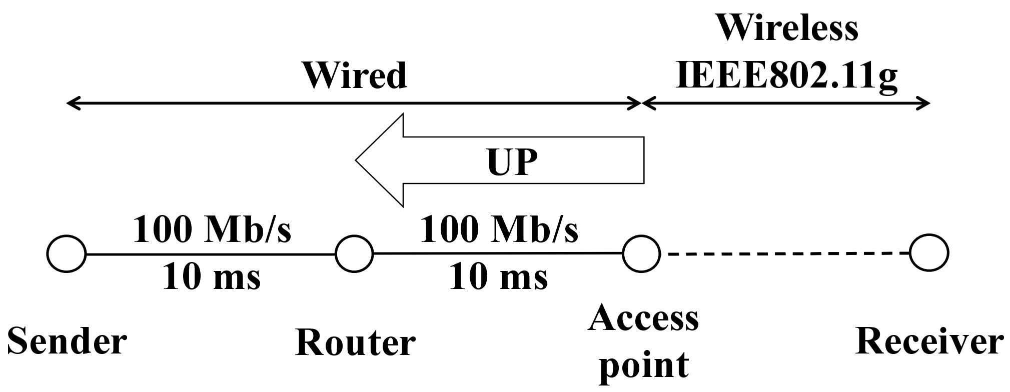


Figure 5.9: Simulation model: Preliminary evaluation

simulation model used in [31]. A sender continuously communicates with the corresponding receiver over a wireless link where random packet losses occur. The average throughput performance of the proposed and conventional schemes are summarized in Table 5.3. TCP-FEC with Interleave for burst loss environments with the reduction factor of 0.9 or 1.0 achieves higher throughput than the conventional schemes.

5.5.2 Throughput Performance

Total throughputs when the proposed and conventional schemes were applied over the whole simulation time are shown in Figures 5.10, 5.11, 5.12, 5.13 and 5.14. According to Figure 5.10, TCP-FEC with Interleave for burst loss environments achieves higher throughput than that of the conventional scheme, namely, "tcp." In the case of TCP-FEC with Interleave for burst loss environments, g is set to 1, 3, and 5, and r is set to 0.7. In particular, TCP-FEC with Interleave for burst loss environments with l of 30–40 attains the highest throughput in this simulation. Small l will cause transmissions of wasteful redundant packets due to excessive redundancy, while large l will cause unsuccessful recovery due to in-

Table 5.2: Simulation parameters: Preliminary evaluation

Simulation time	180 [s]
Buffer size on access point	500 [packet]
Packet loss rate	0.1 [%]
Segment size	1000 [Byte]
Number of trials	30
Upper limit of group size (l)	30
Reduction factor (r)	0.5, 0.7, 0.9, 1.0
Number of groups (g)	3
TCP algorithm	TCP NewReno

Table 5.3: Fundamental characteristics

TCP-NewReno	TCP-AFEC	TCP-FEC with Interleave			
		r=0.5	r=0.7	r=0.9	r=1.0
8 [Mb/s]	19 [Mb/s]	8 [Mb/s]	11 [Mb/s]	23 [Mb/s]	24 [Mb/s]

sufficient redundancy. Namely, an appropriate redundancy should be determined according to network conditions. That issue will be addressed in future work. In the following simulation, l is set to 30. Total throughputs of the proposed and conventional schemes when r was varied from 0.5 to 1.0 and g was varied from 1 to 5 are plotted in Figures 5.11 and 5.12, respectively. These figures indicate that TCP-FEC with Interleave for burst loss environments achieves higher throughput than that achieved by the conventional scheme regardless of the parameters evaluated. In particular, TCP-FEC with Interleave for burst loss environments attains the highest throughput when r and g are 0.7 and 2–3, respectively. From Figure 5.13, TCP-FEC with Interleave for burst loss environments achieves higher

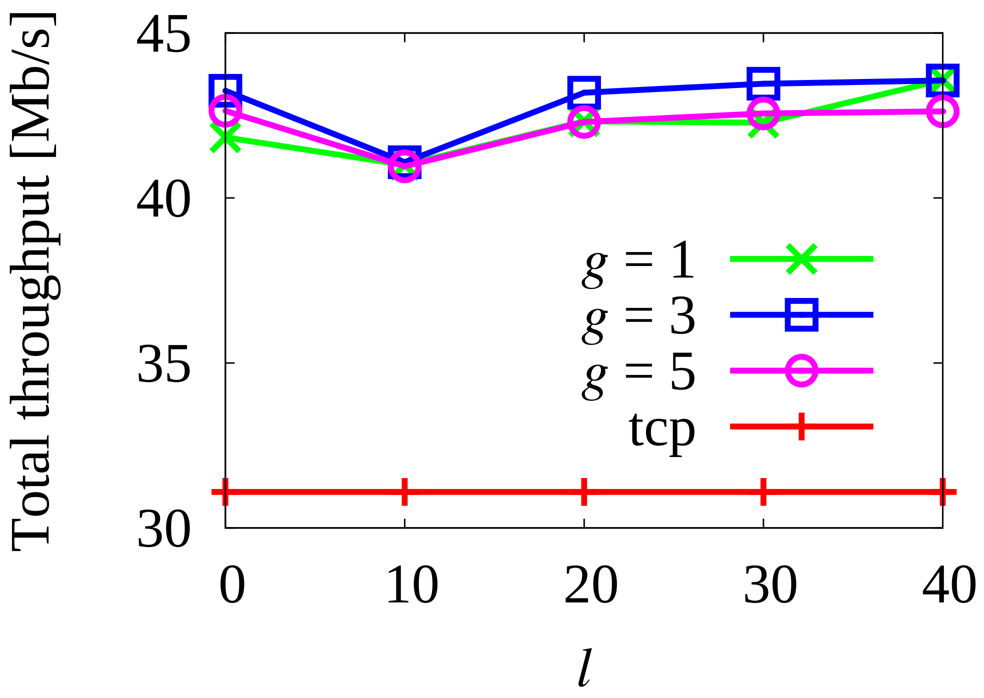


Figure 5.10: Effect of upper limit of group size ($r = 0.7$)

throughput than the conventional scheme. TCP-FEC with Interleave for burst loss environments maintains excellent throughput in a wide range of the delay time although the conventional scheme degrades throughput as the delay time increases. From Figure 5.14, TCP-FEC with Interleave for burst loss environments with large l achieves higher throughput than the conventional scheme in a wide range of the number of flows, i.e., in the case where packet loss rates vary. TCP-FEC with Interleave for burst loss environments with small l also improves throughput at large number of flows, i.e., at high packet loss rates. This result indicates that TCP-FEC with Interleave for burst loss environments will be further improved by setting parameters to appropriate values according to network conditions. Consequently, TCP-FEC with Interleave for burst loss environments

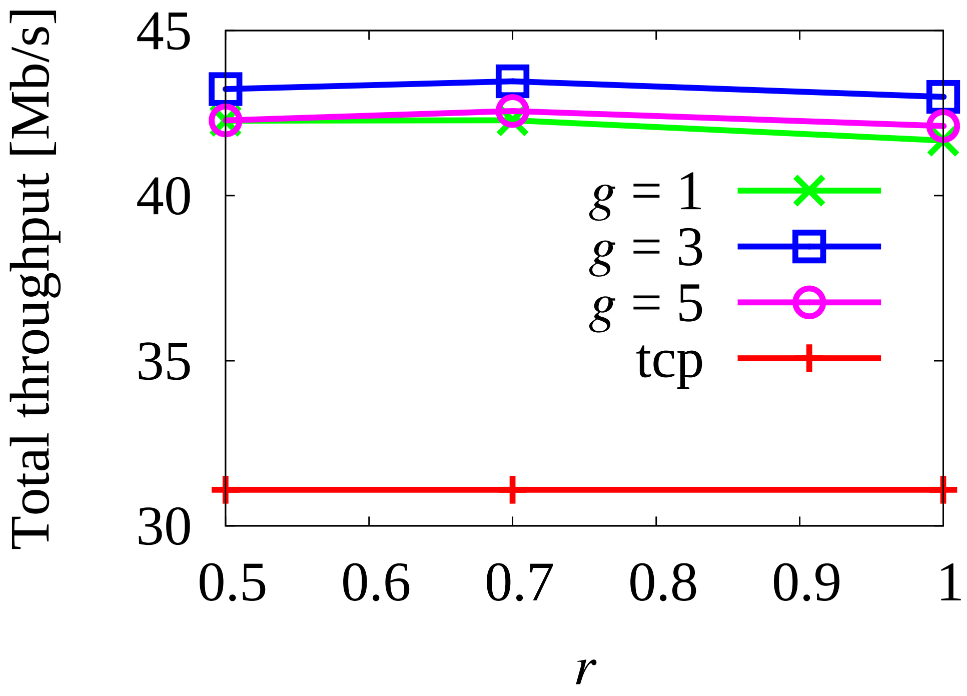


Figure 5.11: Effect of reduction factor ($l = 30$)

can improve TCP throughput performance by applying FEC to TCP operation to recover lost packets effectively. In the following subsections, the characteristics of TCP-FEC with Interleave are discussed in detail.

5.5.3 Analysis of Characteristics of TCP-FEC

The reason that throughput was improved was investigated as described in the following. Number of TCP fast recoveries and timeouts, number of recovery packets, redundancy rate, and effective recovery rate for the proposed and conventional schemes, when l is set to 30, are shown in Figures 5.15, 5.16, 5.17, 5.18 and 5.19. According to Figure 5.15, TCP-FEC with Interleave for burst loss environments significantly reduces the number of TCP fast recoveries, although

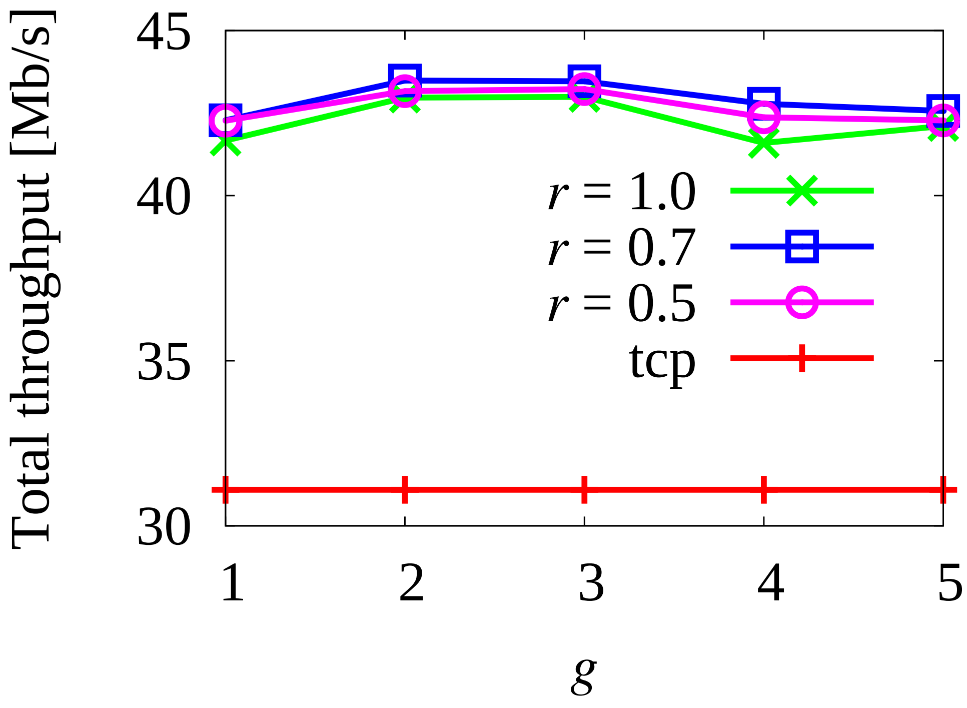


Figure 5.12: Effect of number of groups (l = 30)

the conventional scheme causes a large number of them. In the case of TCP-FEC with Interleave for burst loss environments, the number of fast recoveries decreases as number of groups increases, because a larger number of groups can help to recover lost packets effectively by interleaving. On the other hand, TCP-FEC with Interleave for burst loss environments with the large reduction factor (r = 1.0) causes a large number of fast recoveries. This is because TCP-FEC with Interleave for burst loss environments does not avoid congestion, although it can recover lost packets effectively when the reduction factor is large. Number of TCP timeouts, shown in Figure 5.16, shows a similar trend to that of number of fast recoveries. However, TCP-FEC with Interleave for burst loss environments with r of 1.0 and g of 3–5 increases the number of timeouts. This is because a

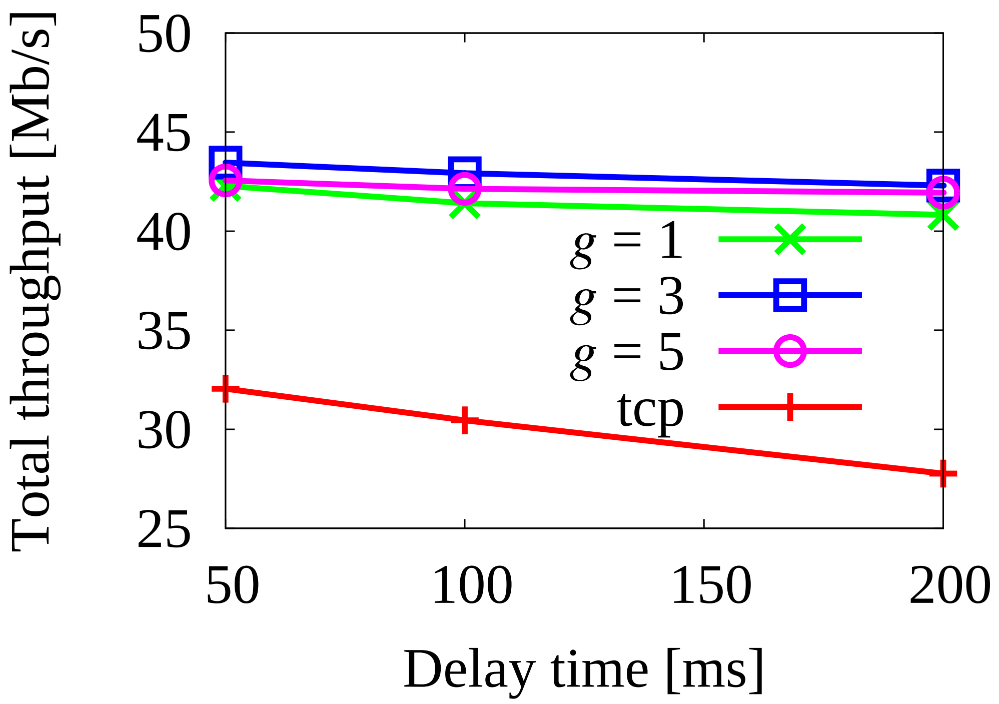


Figure 5.13: Effect of delay time (l=30, r=0.7)

large number of groups increases maximum duplicate ACK suppression time, so timeouts will easily occur in a congested situation ($r = 1.0$).

The effect of FEC on the recovery of lost packets was investigated as follows. As shown in Figure 5.17, number of recovery packets increases as number of groups due to the effect of interleaving. Namely, a large number of groups can help to recover lost packets effectively, while it also causes high redundancy, as shown in Figure 5.18. If redundancy is too high, redundant packets will be wastefully transmitted; consequently, most redundant packets will not be used to recover lost packets. This outcome can be confirmed in terms of effective recovery rate, shown in Figure 5.19. It is clear from the figure that TCP-FEC with Interleave for burst loss environments achieves the most-efficient recovery of lost packets

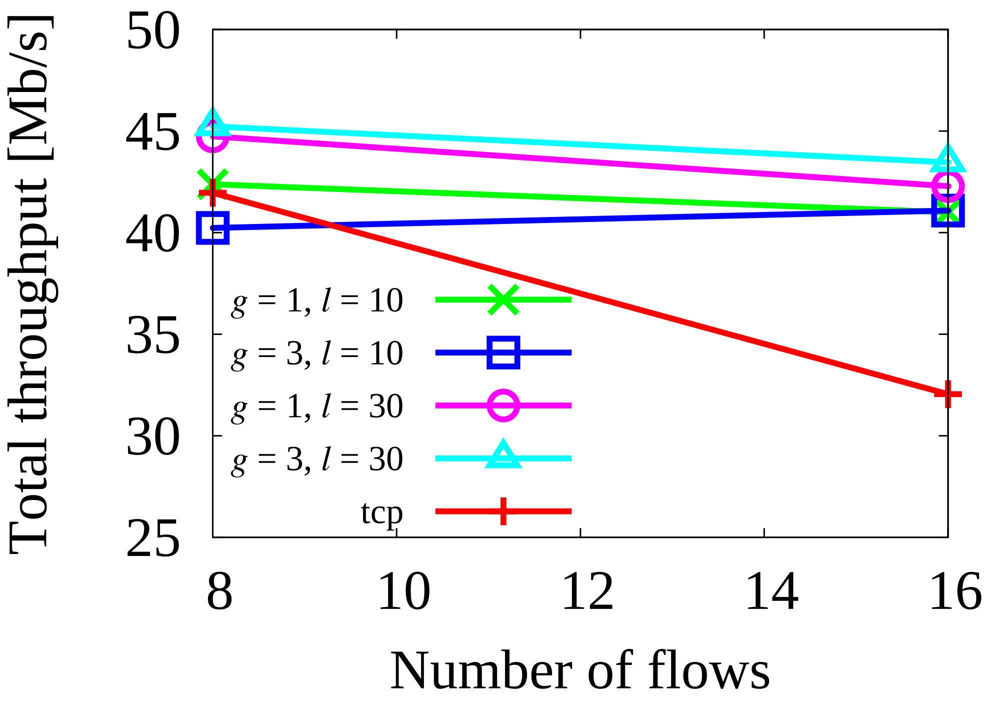


Figure 5.14: Effect of the number of flows(r=0.7)

when number of groups is 3.

The above simulation results demonstrate TCP-FEC with Interleave for burst loss environments achieves the highest throughput performance by recovering lost packets effectively when r and g are set to 0.7 and 3, respectively.

5.5.4 Analysis Characteristics of TCP-FEC in Each State

The characteristics of the proposed and conventional schemes in transient and steady states were investigate as follows. Total throughput, data-packet loss rate, number of TCP timeouts, and effective recovery rate for the proposed and conventional schemes in transient and steady states, are shown in Figures 5.20 and 5.21, respectively. According to Figures 5.20(a) and 5.21(a), TCP-FEC with

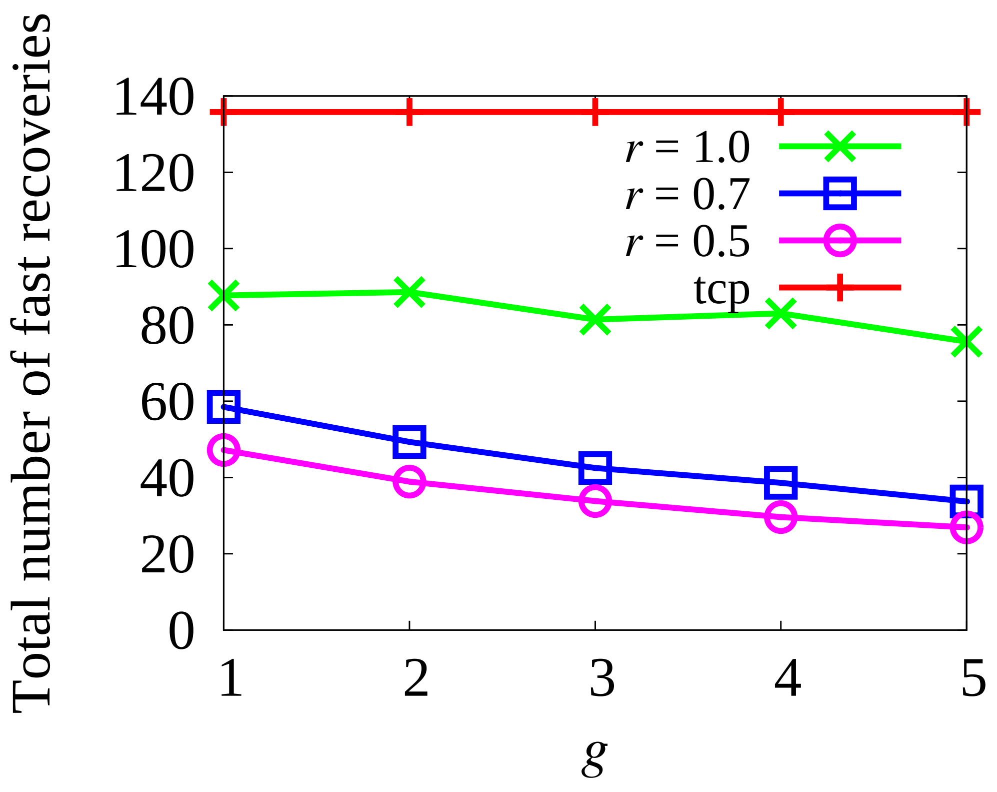


Figure 5.15: Number of fast recoveries

Interleave for burst loss environments achieves higher throughput than the conventional scheme by recovering lost packets effectively in both states. This result can be confirmed by data-packet loss rate shown in Figures 5.20(b) and 5.21(b). TCP-FEC with Interleave for burst loss environments can drastically reduce the packet loss rate in both states. The packet loss rate in transient state is relatively larger than that in steady state. This is because, in transient state, transmission rate significantly changes (due to TCP's slow-start mode) and packet losses can easily occur. In particular, TCP-FEC with Interleave for burst loss environments with large g ($=$ 3–5) severely reduces packet loss rate in transient state for the same reason described above. As shown in Figures 5.20(c) and 5.21(c), TCP-FEC with Interleave for burst loss environments also reduces number of TCP

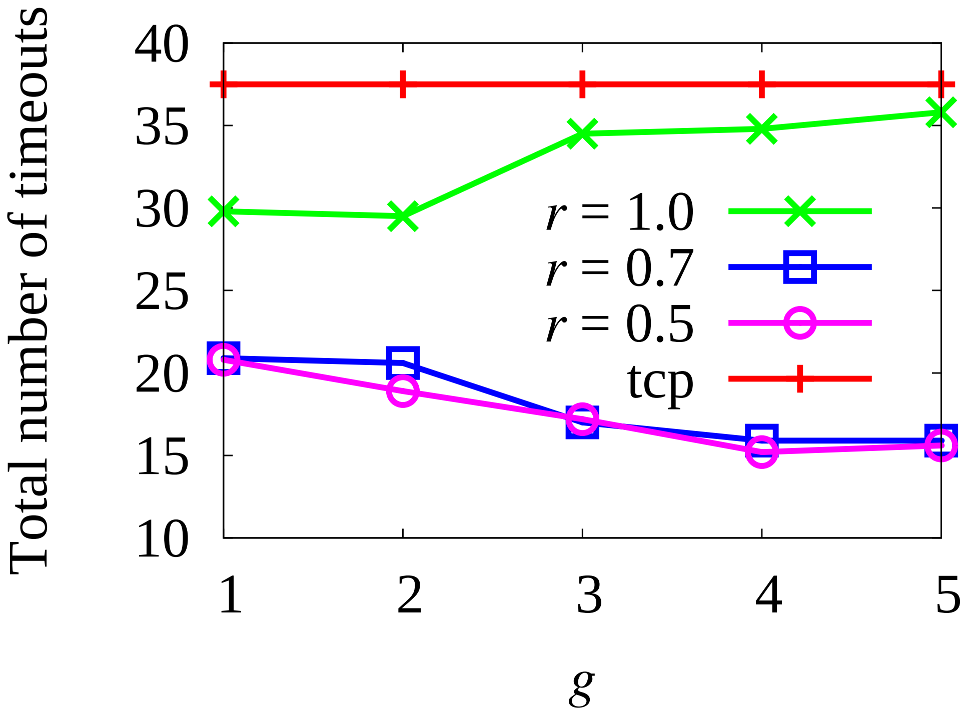


Figure 5.16: Number of timeouts

timeouts in both states. In the case of TCP-FEC with Interleave for burst loss environments, number of timeouts decreases as number of groups increases, in a similar manner to the results for packet loss rate in transient state, because a larger number of groups can help to recover lost packets effectively by interleaving. However, in steady state, it increases when the reduction factor is 1.0 and number of groups is larger than 2. In steady state, since transmission rate, i.e., *cwnd*, is relatively large, maximum duplicate ACK suppression time becomes likely long, and it will easily cause timeouts, particularly in congested situations. In transient and steady states, TCP-FEC with Interleave for burst loss environments with g of 3 and 2, respectively, attains the highest throughput. As shown in Figures 5.20(d) and 5.21(d), this is because the lost packets can be recovered

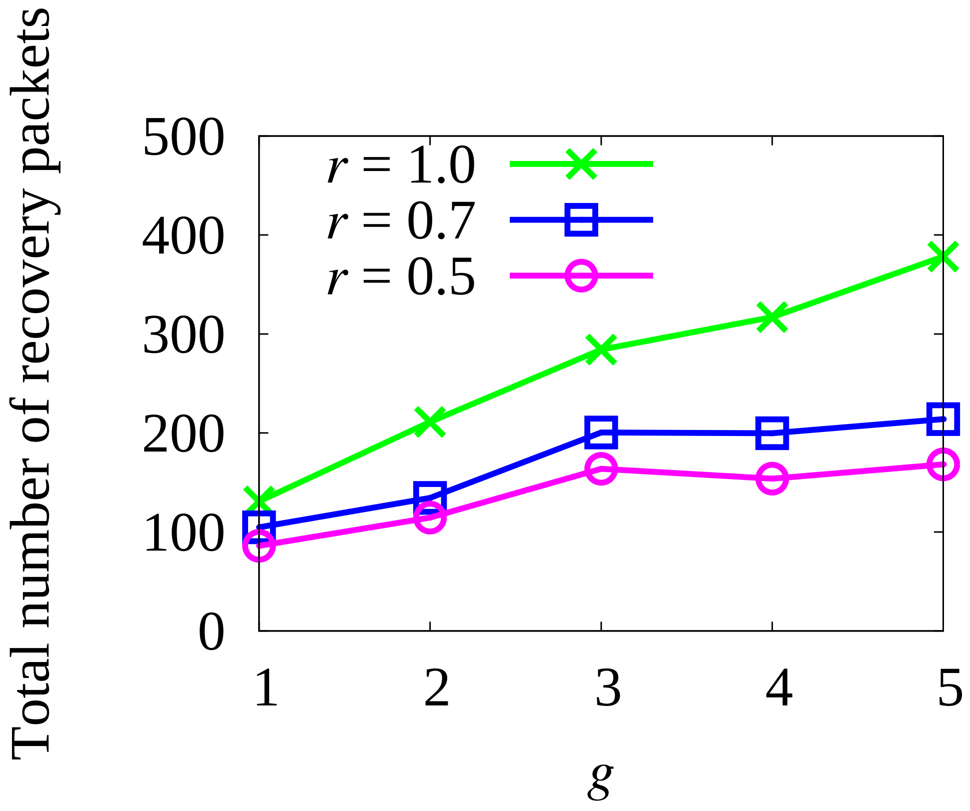


Figure 5.17: Number of recovery packets

most effectively in each state when the number of groups is set to these values. In other words, bursty packet losses can more easily occur, and larger number of groups is needed to effectively interleave in transient state than in steady state.

From the above results, the upper limit of group size and number of groups affect recovery rate of lost packets. A small upper limit of group size achieves high recovery rate but causes high overhead, while a large one achieves low overhead but causes low recovery rate. Similarly, a large number of groups helps to recover bursty packet losses but causes high overhead, while a small one achieves low overhead but causes low recovery rate. The reduction factor has an impact on the throughput and fairness performance. When some flows employing TCP-FEC with Interleave for burst loss environments with the same reduction factor coexist

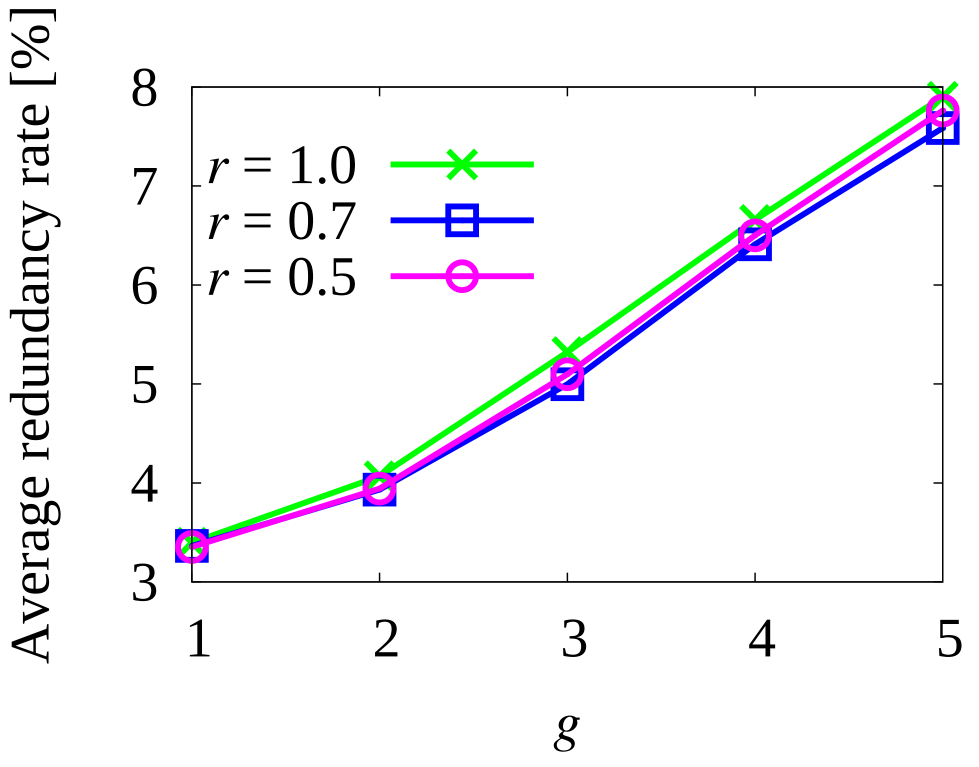


Figure 5.18: Redundancy rate

on the shared link, high fairness will be achieved just like the case of conventional TCP flows. On the other hand, flows with different reduction factors coexist, fairness will be degraded. TCP-FEC with Interleave for burst loss environments can therefore be further improved by dynamically adjusting the parameters according to network conditions such as packet loss rate and competing flows, which will be considered in future work, although it achieves higher throughput by applying FEC with static parameter setting of appropriate values than the conventional scheme.

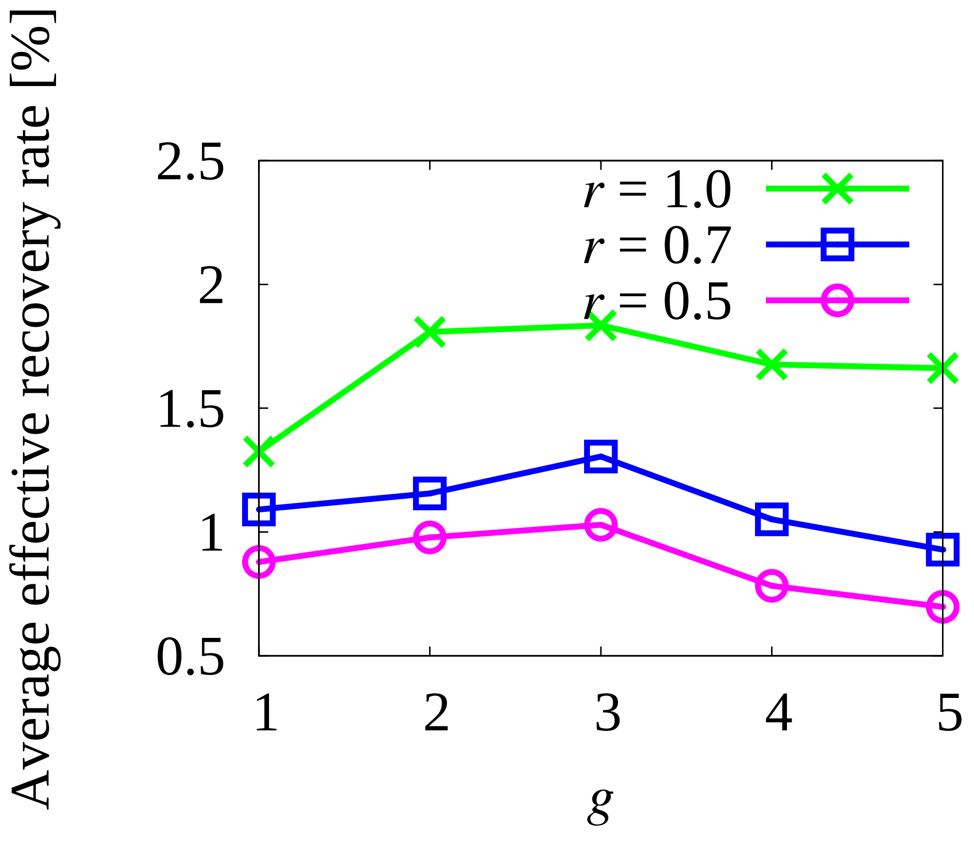
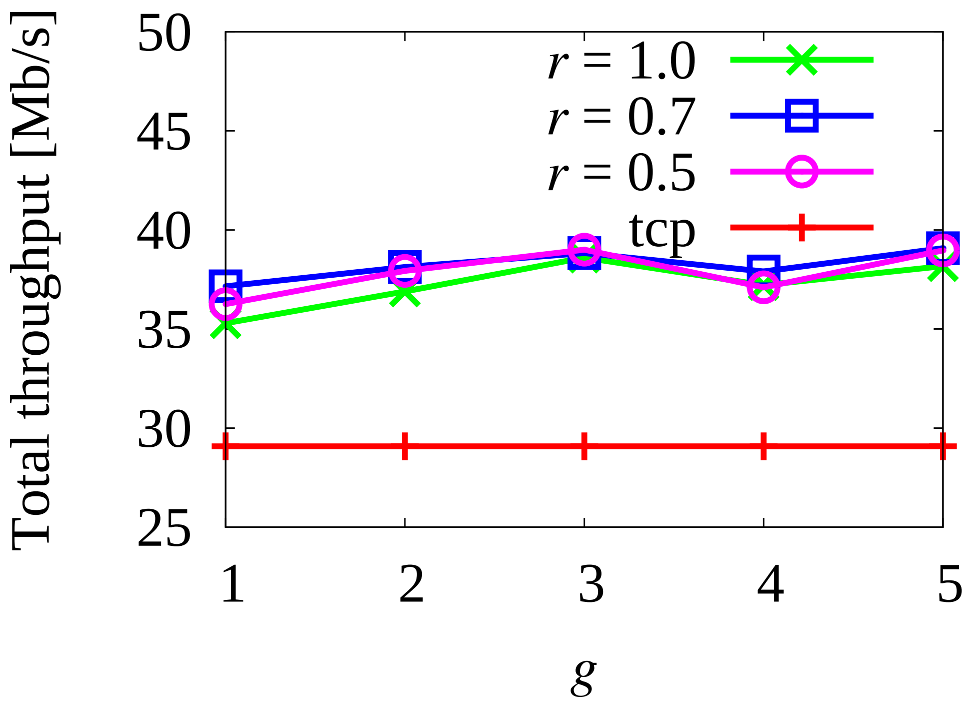


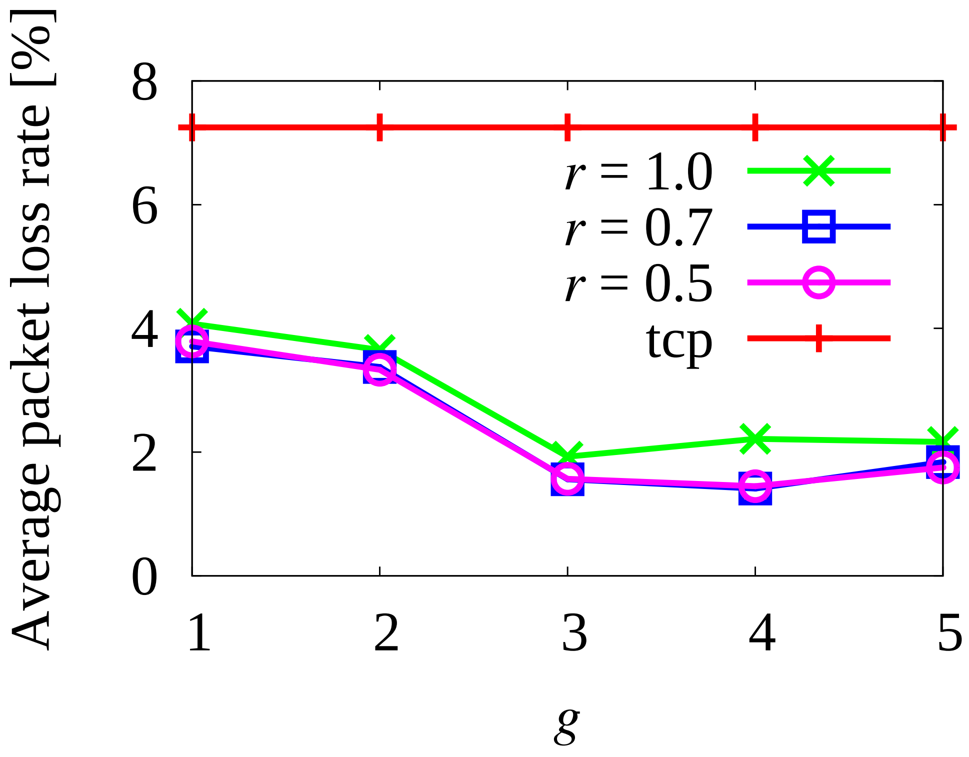
Figure 5.19: Effective recovery rate

5.6 Conclusion

Packet losses significantly degrade TCP performance in burst loss environments. To improve TCP throughput in such networks, a scheme to apply FEC to the entire TCP operation was proposed. This scheme consists of functions for controlling redundancy level and transmission rate, suppressing the return of duplicate ACKs, and interleaving redundant packets. Evaluations of various characteristics by simulation showed that TCP-FEC with Interleave for burst loss environments offers higher TCP throughput performance than the conventional scheme by recovering lost packets effectively. In future work, I aim to devise a scheme to determine the appropriate values of each parameter according to network conditions and to more effectively recover lost packets in order to achieve high throughput and fairness.

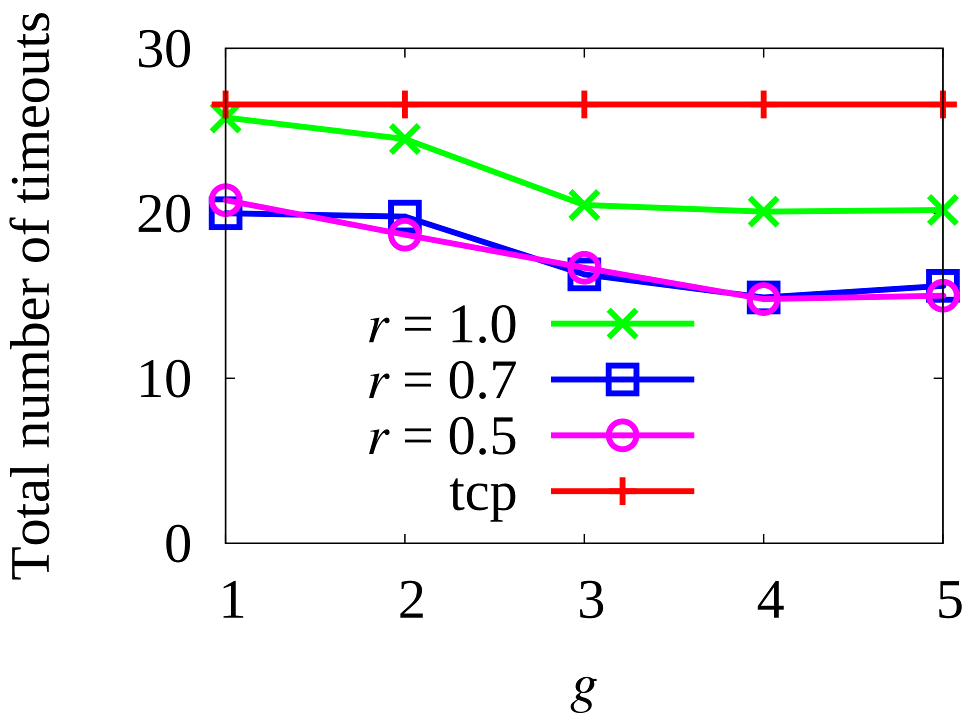


(a) Total throughput

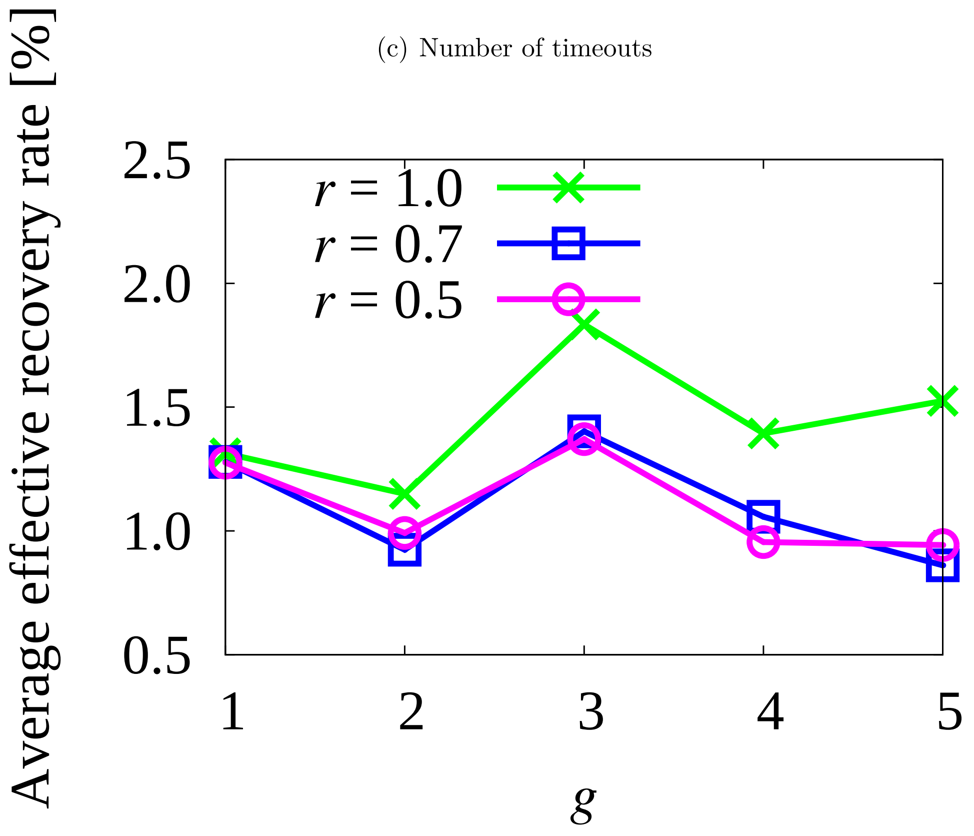


(b) Data-packet loss rate

Figure 5.20: Transient state (0–20 s)

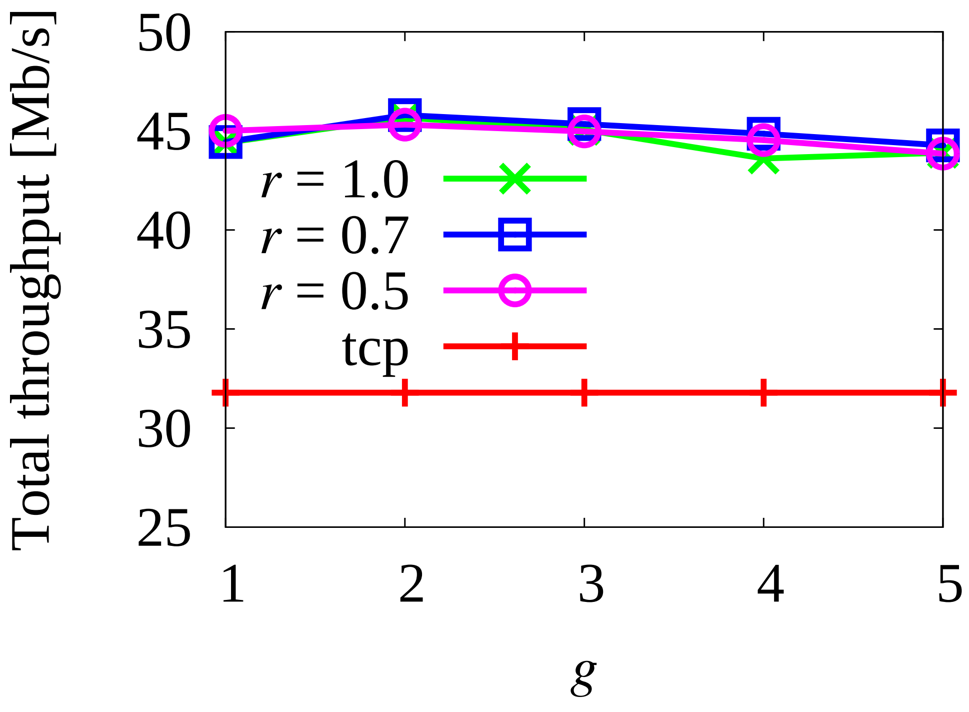


(c) Number of timeouts

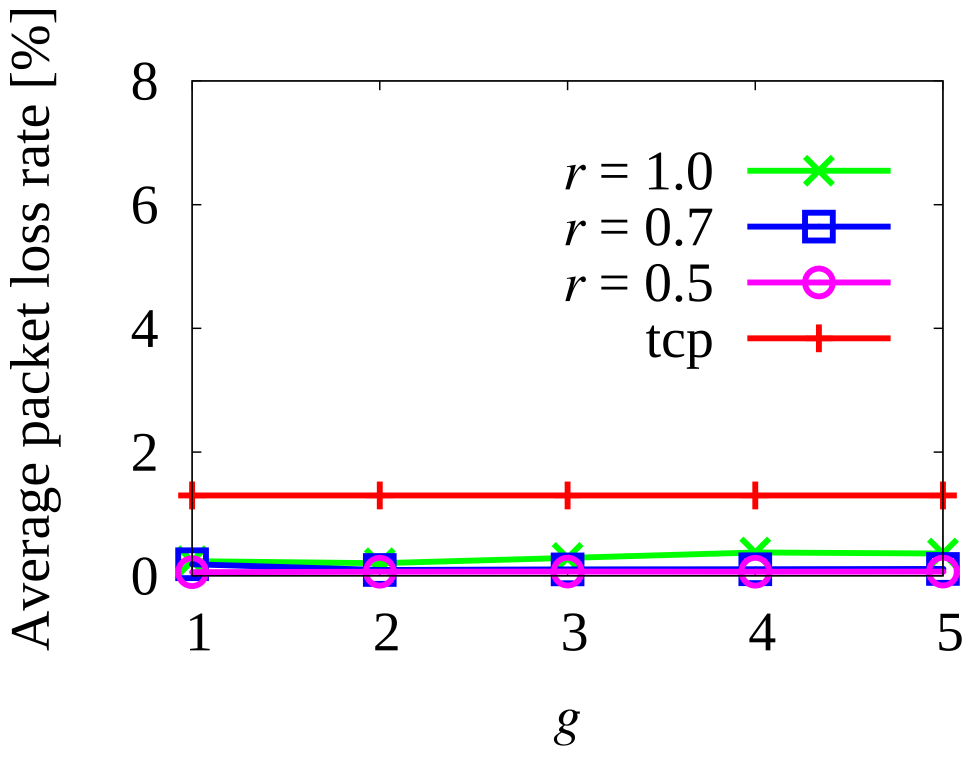


(d) Effective recovery rate

Figure 5.20: Transient state (0–20 s)

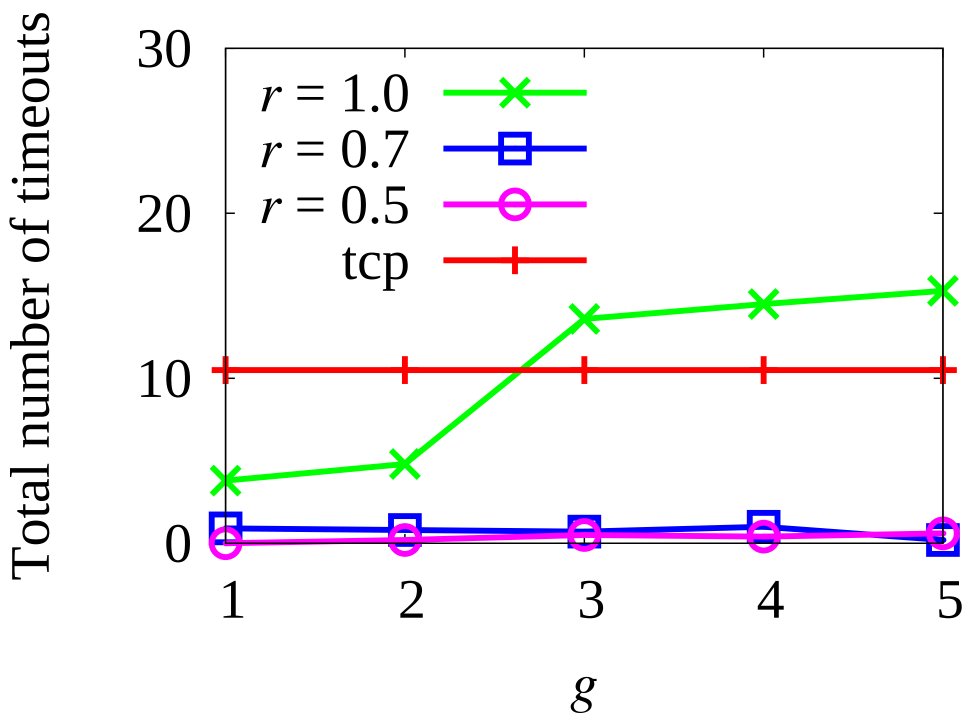


(a) Total throughput

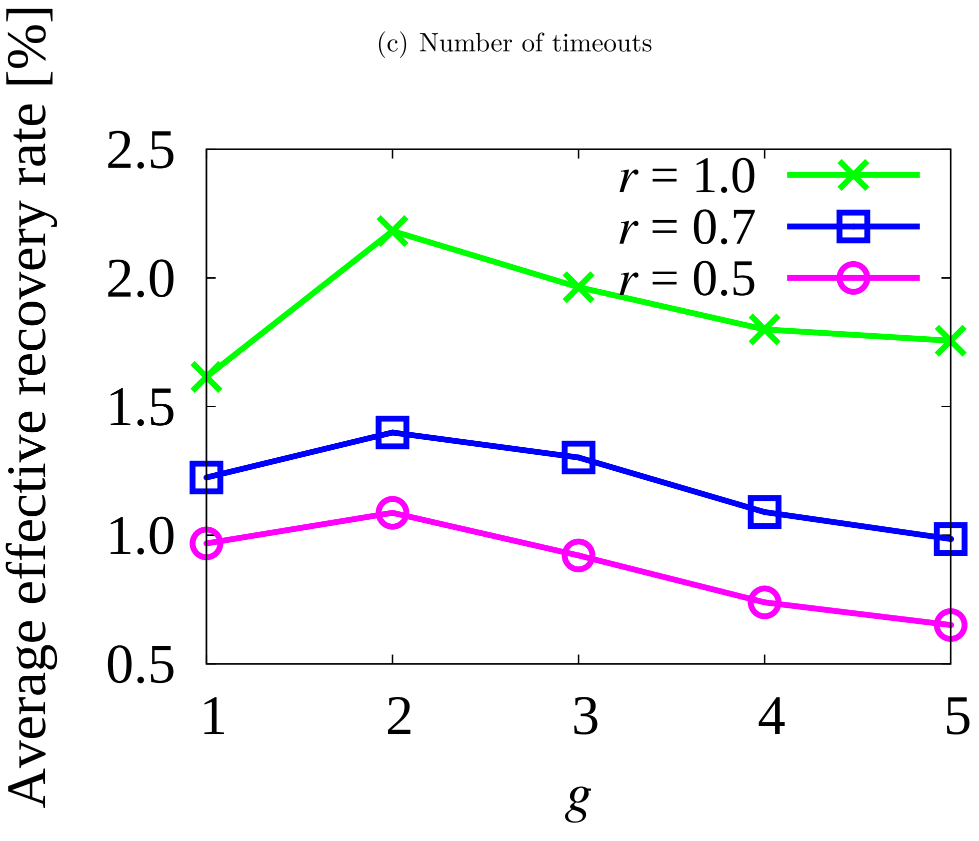


(b) Data-packet loss rate

Figure 5.21: Steady state (20–60 s)



(c) Number of timeouts



(d) Effective recovery rate

Figure 5.21: Steady state (20–60 s)

6 Conclusion

In this chapter, this research is concluded. I describe a summary of this dissertation and future works in the following sections.

6.1 Summary of This Dissertation

The growth of the Internet has led to provide a wide variety of network services such as multimedia services, e-commerce, online banking and trading, social network services, and online games. Our desire in network services has changed significantly as the Internet grows. In particular, there is a great demand for real-time and delay-sensitive services to improve the quality of life. The quality of such services depends especially on transmission delays. It has been improved with the development of broadband network technologies, though the improvement will have limitations due to physical distance between communication ends. Therefore, further improvement should be achieved by improving the transmission efficiency of communication protocols to transmit data between the communication ends.

TCP is still commonly used as reliable data-transmission protocol although network environment for such services has significantly changed as described above. It generally estimates an available bandwidth of networks on the basis of packet losses due to congestion. Lost packets are recovered by retransmissions and the transmission rate is kept low while the lost packets are recovered. Since it needs at least “one round-trip time” to recover lost packets, a long recovery time causes the degradation of service quality. By this “reactive” control to recover lost packets, TCP has an essential problem to provide real-time and delay-sensitive services. To prevent this problem, the number of retransmissions must be kept as low as possible.

One efficient way to prevent packet losses by “proactive” control is to apply a technology called “forward error correction” (FEC). FEC enables a sender to transmit packets with redundant information to recover lost packets by the information at a receiver. The recovery success rate depends on the amount of redundant information; however, redundant information places an additional load on the network. It is typically used for UDP communication, which has a constant transmission rate, while it is difficult to adapt to TCP communication, where transmission rate changes often, because it is harder to select an appropriate redundancy level according to network conditions. For this reason, although there have been few studies on generally applying FEC to TCP operations, there have been several studies on TCP restrictively using it.

This study aimed to improve TCP performance for real-time and delay-sensitive services. To achieve this, I proposed schemes to apply FEC to the entire TCP operation. The effectiveness of the proposed schemes was demonstrated through simulation evaluations.

Chapter 2 introduced related works and basic knowledge which were necessary for understanding this dissertation. For example, TCP’s congestion control and error correction technology were explained.

In Chapter 3, I first considered a scheme to simply apply FEC technology to the entire TCP operation. The proposed scheme dynamically controls redundancy level according to transmission rates. I investigated the fundamental characteristics of the proposed scheme focusing on the redundancy, especially including in a high-latency environment. Simulation evaluations showed that the proposed scheme enabled higher throughput than the conventional schemes, especially in high-latency environments.

In Chapter 4, I examined the characteristics of the proposed scheme in detail. A simple application of FEC to TCP operation might not work effectively. If the redundancy is too low, lost packets might not be recovered effectively. Moreover, unnecessary retransmissions are possibly caused due to the reception of duplicate ACKs even if recovery is successful. Therefore, I extended the proposed scheme to consider the ways to limit minimum redundancy and suppress the return of duplicate ACKs. I investigated the characteristics of the proposed scheme focusing on the introduced functions in an environment where random packet losses occur. Simulation evaluations showed that the proposed scheme improved throughput significantly by suppressing the return of duplicate ACKs and controlling minimum redundancy, especially in high-latency environments, although FEC technology cannot work effectively when simply applied to TCP operation.

In Chapter 5, I considered adapting the proposed scheme to real environments. FEC cannot recover lost packets if both of original and redundant packets are "burstily" lost in a network. In addition, when FEC recovers lost packets, congestion does not be controlled through original TCP operations. Therefore, I further extended the proposed scheme to consider the ways to interleave redundant packets from original packets and control transmission rates when recovery is successful. I investigated the characteristics of the proposed scheme focusing on these functions in a real environment where burst packet losses occur. Evaluations of various characteristics by simulation showed that the proposed scheme offered higher TCP throughput performance than the conventional scheme by recovering lost packets effectively.

6.2 Future Works

In future work, I will devise a scheme to determine the appropriate values of each parameter according to network conditions and to more effectively recover lost packets in order to achieve higher throughput and fairness.

In this research, I introduced a “proactive” control to recover lost packets into TCP operations, which originally recovers lost packet by “reactive” control, i.e., retransmissions. I believe that this approach can be applied to a wide variety of services such as satellite communications and telemedicine. For example, when surgical images are missed or delayed in telesurgery that specialists examine and treat patients over networks, the patient’s life will be risked. I believe that such situations can be prevented by surely protecting communication data from losing and delaying with error correction technologies. In addition, although dedicated channels are generally needed for high-quality delay-sensitive services, this approach will be helpful to provide such services without dedicated channels in the Internet. Therefore, this research can improve the quality of life on network users.

References

[1] Cisco Systems, Inc., "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021," Feb. 2017. Information available at `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-net/working-index-vni/mobile-white-paper-c11-520862.pdf`. [Retrieved: Febrary 2017]

[2] J. Postel, "Transmission control protocol," *IETF Request for Comments 793*, September 1981.

[3] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," *IETF Request for Comments 5681*, September 2009.

[4] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "Forward error correction (FEC) building block," *IETF Reguest for comments 3542*, December 2002.

[5] J. Postel, "User datagram protocol," *IETF Request for Comments 768*, August 1980.

[6] Y. Sato, H. Koga, and T. Ikenaga, "Redundancy control and duplicate ACK suppression methods for TCP with FEC," *Proc. IEEE ICNP2013*, 2 pages, October, 2013. `DOI:10.1109/ICNP.2013.6733623`

[7] Y. Sato, H. Koga, and T. Ikenaga, “Improving TCP throughput using forward error correction,” *IEICE Communications Express*, Vol. 6, No. 1, pp. 28–33, January 2017. DOI:10.1587/COMEX.2016XBL0158

[8] Y. Sato, H. Koga, and T. Ikenaga, “TCP using adaptive FEC to improve throughput performance in high-latency environments,” *IEICE Transaction on Communication*, Vol. E102-B, No. 3, pp. 537–544, March 2019.

[9] K. Fall and S. Floyd, “Simulation-Based Comparisons of Tahoe, Reno and SACK TCP,” *ACM Computer Communication Revier*, Vol. 26, No. 3, pp. 5-21, July 1996.

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP selective acknowledgment options,” *IETF Request for Comments 2018*, October 1996.

[11] K. Ramakrishnan, S. Floyd, and D. Black, “The addition of explicit congestion control notification (ECN) to IP,” *IETF Request for Comments 168*, September 2001.

[12] A. Vitervi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE Transaction on Information Theory*, Vol. 13, No. 2, pp. 260–269, April 1967.

[13] I. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of the Society for Industrial and Applied Mathematics*, Vol. 8, No. 2, pp. 300–304, June 1960.

[14] Y. W. Kwon, H. Chang, and J. Kim, “Adaptive FEC control for reliable high-speed UDP-based media transport,” *Proc. IEEE PCM2004*, pp. 364–372, December 2004. DOI:10.1007/978-3-540-30542-2_45

[15] C. Bouras, N. Kanakis, V. Kokkinos, and A. Papazois, "Application layer forward error correction for multicast streaming over LTE networks," *Wiley InterScience, International Journal of Communication Systems*, Vol. 26, No. 11, pp. 1459–1474, November 2013. `DOI:10.1002/dac.2321`

[16] A. Vishwanath, V. Sivaraman, and M. Thottan, "Enabling a bufferless core optical network using edge-to-edge packet-level FEC," *IEEE Transaction on Communications*, Vol. 61, No. 2, pp. 690–699, March 2013. `DOI:10.1109/TCOMM.2012.022513.110788`

[17] Y. Sohn, J. Hwang, and S. Kang, "Adaptive packet-level FEC algorithm for improving the video quality over IEEE 802.11 networks," *Science and Engineering Reseach Support Society, International Journal of Software Engineering and its Applications*, Vol. 6, No. 3, pp. 27–34, July 2012.

[18] M. Sakakibara, Y. Tanigawa, and H. Tode, "Highly reliable TCP transfer method with error correction technology," *Proc. MITA2009*, pp. 321–322, August 2009.

[19] V. Sharma, K. Ramakrishnan, K. Kar, and S. Kalyanaraman, "Complementing TCP congestion control with forward error correction," *Proc. IFIP Networking2009, Springer*, pp. 378–391, May 2009. `DOI:10.1007/978-3-642-01399-7_30`

[20] L. Baldantoni, H. Lundqvist, and G. Karlsson, "Adaptive end-to-end FEC for improving TCP performance over wireless links," *Proc. IEEE ICC2004*, pp.4023–4027, June 2004.

[21] H. Seferoglu, A. Markopoulou, U. C. Kozat, M. R. Civanlar, and J. Kempf, "Dynamic FEC algorithms for TFRC flows," *IEEE Trans-*

action on Multimedia, Vol. 12, No. 8, pp. 869–885, June 2010. DOI:10.1109/TMM.2010.2053840

[22] T. Tsugawa, N. Fujita, T. Hama, H. Shimonishi, and T. Murase, "TCP-AFEC: An adaptive FEC code control for end-to-end bandwidth guarantee," *Proc. IEEE PV2007*, pp. 294–301, November 2007. DOI:10.1109/PACKET.2007.4397053

[23] Y. Huang, S. Mehrotra, and J. Li, "A hybrid FEC-ARQ protocol for low-delay lossless sequential data streaming," *Proc. IEEE ICME2009*, pp. 718–725, June/July 2009. DOI:10.1109/ICME.2009.5202596

[24] B. Ganguly, B. Holzbauer, and K. Kar, "Loss-tolerant TCP (LT-TCP): Implementation and experimental evaluation," *Proc. IEEE MILCOM2012*, pp. 1–6, October/November 2012. DOI:10.1109/MILCOM.2012.6415694

[25] T. Flach, N. Dukkipati, A. Terzis, B. Taghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Basset, and R. Govindan "Reducing web latency: The virtue of gentle aggression," *Proc. ACM SIGCOMM2013*, pp. 159–170, August 2013. DOI:10.1145/2486001.2486014

[26] S. Ferlin and O. Alay, "TCP with dynamic FEC for high delay and lossy networks," *Proc. ACM CoNEXT Student Workshop*, 2 pages, December 2016. DOI:10.1145/299572.3004861

[27] The ns-2 network simulator, http://www.nsnam.org/.

[28] L. A. Griece and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *ACM Computer Communication Review*, Vol. 34, No. 2, pp. 25–38, April 2004. DOI:10.1145/997150.997155

[29] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-Friendly high-speed TCP variant," *ACM Operating Systems Review*, Vol. 42, No. 5, pp. 64–74, July 2008. DOI:10.1145/1400097.1400105

[30] The ns-3 network simulator, http://www.nsnam.org/.

[31] F. Teshima, H. Obata, R. Hamamoto, and K. Ishida, "TCP-TFEC: TCP congestion control based on redundancy setting method for FEC over Wireless LAN," *IEICE Transactions on Information and Systems*, Vol. E100-D, No. 12, pp. 2818–2827, December 2017. DOI:10.1587/transinf.2017PAP0017