Modeling, Analysis and Enhancement of Transmission Control Protocol

by

Wei Bao

B.E., Beijing University of Posts and Telecommunications, China, 2009

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2011

Abstract

Transmission control protocol (TCP) is one of the core protocols of the Internet protocol (IP) suite, which provides congestion control and reliable end-to-end connections in the Internet. In wireless environment, due to the random packet loss, many previous TCP variants primarily designed for wired networks may not perform well. In this thesis, we first analyze the impact of random packet loss on the throughput performance of TCP CUBIC. Then, by incorporating online network coding, we propose a new TCP variant called TCP Vegas with online network coding (TCP VON), which can be efficiently applied in wireless networks.

In the first part of this thesis, we propose a Markov chain model to determine the steady state throughput of TCP CUBIC in wireless environment. The proposed model considers both congestion loss and random packet loss caused by the wireless environment. We derive the stationary distribution of the Markov chain and obtain the average throughput based on the stationary distribution. Simulations are carried out to validate the analytical model.

In the second part of this thesis, we propose TCP Vegas with online network coding (TCP VON), which incorporates online network coding into TCP. TCP VON includes

two mechanisms, namely congestion control and online network coding control. The congestion control is extended from TCP Vegas. For the online network coding control, the sender transmits redundant coded packets when packet losses happen. Otherwise, it transmits innovative coded packets. As a result, all the packets can be decoded consecutively and the average decoding delay is small. We establish a Markov chain model to compute the analytical delay performance of TCP VON. We also conduct ns-2 simulations to validate the proposed analytical models. Finally, we compare the average delay and throughput performance of TCP VON and automatic repeat request (ARQ) network coding based TCP (TCP ARQNC) for different topologies. Results show that TCP VON outperforms TCP ARQNC.

Preface

A version of Chapter 2 has been published. Wei Bao, Vincent W.S. Wong, and Victor C.M. Leung, "A model for steady state throughput of TCP CUBIC," in *Proc. of IEEE Global Communications Conference (Globecom)*, Miami, Florida, December 2010. I was responsible for designing the analytical model of TCP CUBIC. I also conducted simulations. Prof. Vincent Wong checked the system model and gave important suggestions. The paper was originally prepared by me, and further revised by all the co-authors.

Chapter 3 has been finished under the guidance of Mr. Vahid Shah-Mansouri, Prof. Vincent Wong, and Prof. Victor Leung. I was responsible for algorithm proposal, analytical model establishment and simulations. Mr. Vahid Shah-Mansouri checked the analytical model and the simulation codes. Prof. Vincent Wong gave important suggestions on the presentation of the chapter. The chapter was originally prepared by me, and further revised by Mr. Vahid Shah-Mansouri, Prof. Vincent Wong, and Prof. Victor Leung.

Table of Contents

List of Tables

List of Figures

List of Acronyms

3GPP	3rd Generation Partnership Project
ACK	Acknowledgment
AIMD	Additive Increase Multiplicative Decrease
ARQ	Automatic Repeat-request
CDF	Cumulative Distribution Function
ECN	Explicit Congestion Notification
FTP	File Transfer Protocol
HSTCP	High Speed Transmission Control Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
LTE	Long Term Evolution
MAC	Medium Access Control

MIMD	Multiplicative Increase Multiplicative Decrease
pdf	probability density function
RFC	Request for Comments
RMS	Root Mean Square
RTT	Round Trip Time
SACK	Selective Acknowledgment
STCP	Scalable Transmission Control Protocol
TCP	Transmission Control Protocol
TCP ARQNC	Automatic Repeat Request Network Coding based TCP
TCP VON	TCP Vegas with Online Network Coding
TDD	Time Division Duplexing
WiMAX	Worldwide Interoperability for Microwave Access

Acknowledgements

First, I would like to express my deepest gratitude to my supervisors, Prof. Vincent Wong and Prof. Victor Leung, for their patient guidance, constant encouragement, and excellent advice throughout my graduate study. Without their invaluable help, this work would not be possible.

A special thanks goes to my colleague, Vahid Shah-Mansouri, who provided me with precious assistance during the completion of this thesis. I am also thankful to all my colleagues in Prof. Wong's group: Vahid Shah-Mansouri, Man Hon Cheung, Keivan Ronasi, Pedram Samadi, Binglai Niu, Wenbo Shi, Xiaolei Hao, Jinbiao Xu and Shaobo Mao as well as other friends in the data communications group, for sharing their experiences and knowledge during the time of my study.

Finally, I take this opportunity to express my profound gratitude to my beloved parents for their understanding, support, and endless love during my study in Canada. To them I dedicate this thesis.

This research is supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the British Columbia Innovation Council (BCIC) Natural Resources and Applied Sciences (NRAS) grant.

Chapter 1

Introduction

1.1 Transmission Control Protocol (TCP)

1.1.1 Transmission Control Protocol (TCP) Basics

The transmission control protocol (TCP) is one of the core protocols of the Internet protocol (IP) suite. It is a transport layer protocol and it supports congestion control and reliable end-to-end connections in the Internet [1]. The basic requirement, format and functions of TCP are defined in Internet Engineering Task Force (IETF) Request for Comments (RFCs) [2–4].

TCP connection is established by the three-way handshake process [1, 2]. Once the connection has been setup, the sender and receiver start to communicate with each other. At the sender side, the data is viewed as an ordered stream of bytes. It is divided into segments. Each segment is labeled by the number of the first byte in the segment, which is called the sequence number. Then the segments are placed into the network in order. As the receiver accepts TCP segments, it sends acknowledgement (ACK) segments, indicating which byte it is expecting next in the ACK number field.

Two important mechanisms, namely flow control and congestion control are included in TCP [1, 2, 5]. The purpose of flow control is to properly control the transmission rate of sender in order to avoid the buffer overflow at the receiver. To achieve the flow control, a variable called the receive window ($rwnd$) is maintained, which indicates the free buffer space available at the receiver. Another key component of TCP is the congestion control, which is used to avoid too many segments injected in the network. By congestion control, the sender limits the transmission rate it sends into the network if it perceives that there might be congestion along the path to the receiver. The sender maintains a variable called congestion window ($cwnd$), constraining the transmission rate. In summary, by taking both the flow control and congestion control into consideration, the unacknowledged amount of data at the sender cannot exceed either $cwnd$ or $rwnd$

$$LastByteSent - LastByteAcked \leq \min(cwnd, rwnd), \tag{1.1}$$

where $LastByteSent$ is the sequence number of the last byte sent and $LastByteAcked$ is the sequence number of the last byte ACKed.

In most literature, in order to focus on congestion control mechanism, the buffer at the receiver is assumed to be large enough. Thus, the $rwnd$ constraint can be ignored and the unacknowledged amount of data is only limited by the value $cwnd$. Let RTT denote the average round trip time (RTT). The transmission rate can be roughly computed by $cwnd/RTT$. The sender can adjust the transmission rate by changing the value of $cwnd$, which can be regarded as the core of TCP congestion control mechanism.

We briefly review the congestion control mechanism of TCP Reno, a traditional TCP

version. A finite state machine is used to describe of the congestion control of TCP Reno, as shown in Fig. 1.1 [1]. There are namely three states: *slow start*, *congestion avoidance* and *fast recovery*. When a TCP connection begins, the slow start phase initializes a congestion window to 1 segment. When ACKs are returned, the congestion window increases by one segment for each ACK. The value of *cwnd* roughly doubles every RTT during the slow start phase. A threshold value $Threshold$ is maintained to determine the window size at which slow start ends. The sender leaves slow start and enters congestion avoidance if $cwnd > Threshold$. In the congestion avoidance phase, *cwnd* is increased by 1 segment every RTT if there is no loss event. If there is a loss event, it is regarded as indication of network congestion and the sender will decrease *cwnd*. If the loss event indicated by three duplicate ACKs, the state enters fast recovery and *cwnd* is halved ($dupACKcount$ is the counter to record the number of duplicate ACKs). If there is a loss indicated by timeout, the state will go to the slow start state and $Threshold$ is set to be half of the *cwnd*. Note that in most cases when the networks are not heavily congested, loss events are indicated by three duplicate ACKs rather than timeout. Therefore, *cwnd* is additively increased if there is no congestion and is multiplicatively decreased if the sender detects a loss event. This congestion control algorithm is referred to as the additive increase, multiplicative decrease (AIMD) algorithm.

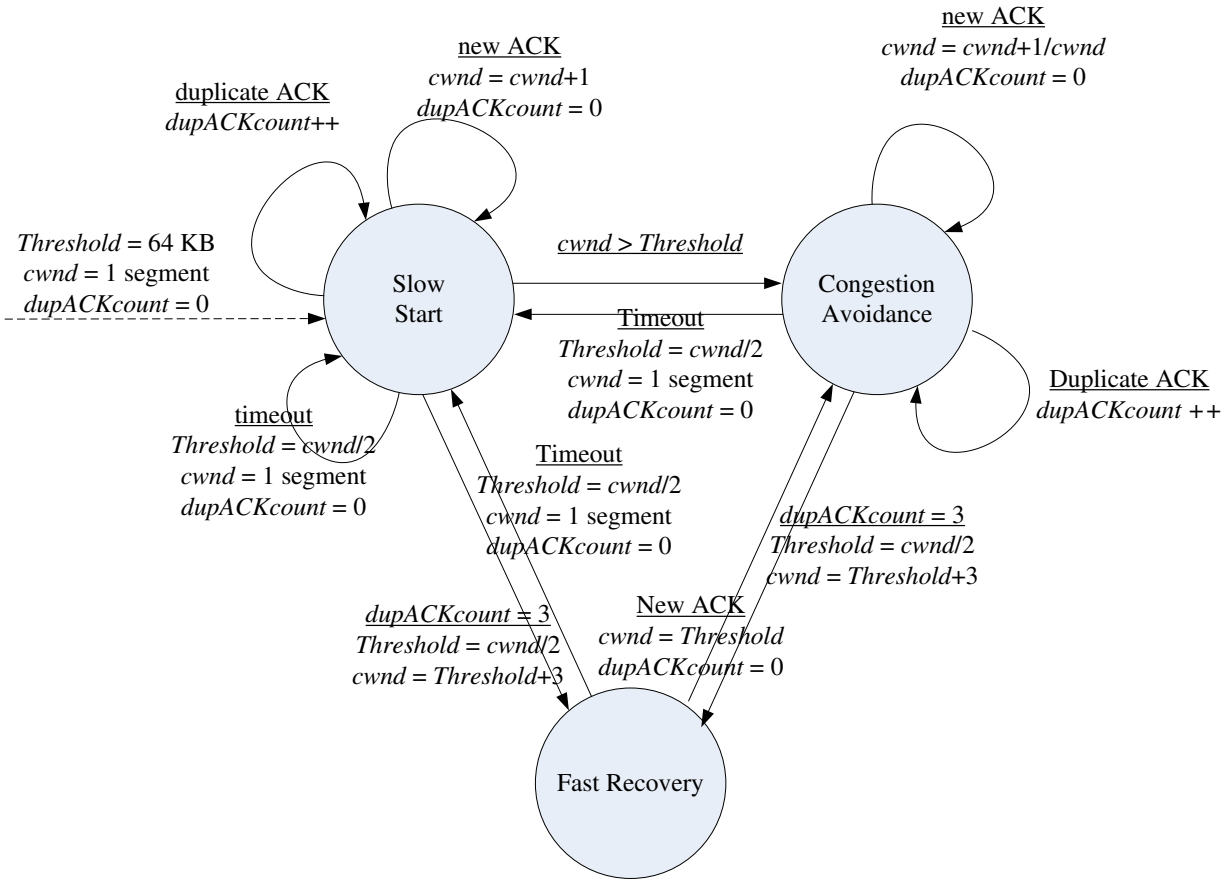


Figure 1.1: Finite state machine of TCP Reno [1].

1.1.2 TCP Variants

Besides TCP Reno, there are many variations of TCP proposed in the literature. The TCP variants which have been deployed in the current Internet include selective acknowledgment (SACK) [6] and New Reno [7].

SACK improves TCP congestion control algorithm in the face of multiple packet losses within a short time. With SACK, the receiver can inform the sender about the sequence numbers of all the lost packets, so that the sender can retransmit only the packets that

have actually been lost.

TCP New Reno made modifications based on TCP Reno. In the situation that SACK is not available, it improves retransmission process during the fast recovery phase so that the multiple packet losses within a short time can be recovered more efficiently.

There are some other important TCP variants, such as Scalable TCP (STCP) [8], TCP Vegas [9], and CUBIC [10]. STCP [8] is designed specifically for long-distance, high-speed links, which employs multiplicative increase multiplicative decrease (MIMD) congestion control algorithm. The congestion window size will grow a constant value for each new ACK, and it will multiplicative decrease when there is a loss event.

TCP Vegas [9] congestion control algorithm is delay-based, which emphasizes packet delay, rather than packet loss, as a signal to determine the rate at which to send packets. The transmission rate is adjusted based on the measured round trip RTT and the base RTT $BaseRTT$. $BaseRTT$ is set to be the minimum of all measured RTTs and it is commonly the RTT of the first packet in the TCP connection. There are also two predetermined threshold values a and b, $a < b$. The default setting is $a = 1$ packet and $b = 3$ packets. The congestion window size $cwnd$ is adjusted as follows. If $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT < a$, then $cwnd$ is linearly increased. If $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT > b$, then $cwnd$ is linearly decreased.

TCP CUBIC [10] is implemented and used by default in Linux kernels 2.6.19 version. It is designed for long-distance, high latency links. It combines additive increase and binary search increase to achieve good scalability as well as RTT fairness. TCP CUBIC

performs well in wired networks with large bandwidth-delay product. In addition, the window growth function of TCP CUBIC is defined in real-time instead of RTT, so that the window growth rate is independent of RTT, which guarantees the RTT fairness. In this thesis, we focus on the performance analysis of TCP CUBIC in wireless networks in Chapter 2.

For TCP CUBIC congestion control algorithm, the congestion window size is a cubic function of time since the last loss event. Let τ denote the elapsed time from the last window reduction. The window size just before the last window reduction is denoted by x. The constant α denotes the window growth factor. A large value of α implies faster window growth rate. The constant β represents the multiplicative decrease factor. The window reduces to βx at the time of the last reduction. Let $w(x, \tau)$ denote the window size as a function of x and τ. The congestion window of CUBIC is determined by [10][1]

$$w(x, \tau) = \alpha \left(\tau - \sqrt[3]{(1-\beta)x/\alpha} \right)^3 + x. \tag{1.2}$$

Despite the fact that TCP CUBIC has been implemented in Linux operating systems (OS), analytical models for the performance of TCP CUBIC are few. The previous study of TCP CUBIC are mainly conducted via simulations and experiments. Leith *et al.* performed experimental evaluation for TCP CUBIC in [11] and Bateman *et al.* presented a simulation based study of BIC, CUBIC, scalable TCP (STCP), and high-speed TCP (HSTCP) in [12]. Moreover, most of the previous analysis of TCP CUBIC

[1]The β in (2.4) corresponds to $(1-\beta)$ of (1) in [10]. As a result $w(x, 0) = \beta x$, meaning that the window reduces to βx at the time of last reduction.

[10–12] focus on wired networks. There are very few related work on the performance of TCP CUBIC for wireless networks. In Chapter 2 of this thesis, we propose an analytical model to analyze the performance of TCP CUBIC in wireless networks.

1.1.3 TCP in Wireless Networks

With the development of wireless communication technology, more and more hosts access the Internet through wireless mode. However, traditional TCP variants were primarily designed for wired networks and they may not be efficient in wireless scenarios. As discussed in literature, in wireless networks, if we directly apply TCP variants designed for wired networks (e.g., TCP Reno, Vegas and STCP), it may result in degraded end-to-end performance [13–16]. Our study in Chapter 2 also shows that random packet loss reduces the normalized average throughput of TCP CUBIC, especially for end-to-end flow with large bandwidth-delay product. In wired networks, it is assumed that random packet loss is negligible and packet losses are mainly caused by congestion (i.e., some buffers in the route of transmission are full and some packets are discarded). In wireless network, packet losses are probably caused by the lossy nature of wireless links (e.g. deep fading and temporary disconnections due to handoff). Traditional TCP congestion control algorithms misinterpret random packet losses as congestion losses, causing degraded performance.

There are many TCP variants proposed to adapt the features of wireless networks. For satellite networks, TCP-Peach [17] was proposed to tackle the problem of long propaga-

tion delay of satellite link. For ad-hoc networks, ATCP [18] was designed as an end-to-end solution to improve TCP throughput. It relies on explicit congestion notification (ECN) to distinguish congestion losses from random packet losses. For mixed wired and wireless networks, split TCP variants such as I-TCP [19] and M-TCP [20] were proposed, they separate the end-to-end connections into two split TCP connections, one in the wired part and the other in the wireless part. Other TCP variants, such as Freeze-TCP [21], TCP Veno [22], TCP Westwood [23] and TCP Jersey [24] are also proposed for wireless networks.

In this thesis, we also propose a new TCP algorithm, TCP VON, to deal with both congestion losses and random packet losses efficiently. It is expected to perform well in mixed wired and wireless networks. Different from the previous designs, we incorporate the online network coding technique into our new TCP algorithm. The online network coding technique will be introduced in Section 1.2 and the TCP VON will be presented in Chapter 3.

1.2 Online Network Coding Based TCP

Network coding was introduced by Ahlswede *et al.* in [25]. It has been shown that network coding can increase the throughput and improve the reliability of the end-to-end communication [25–27]. There are several related works for network coding in a practical setting. Chou *et al.* [28] proposed that the coefficients used for linear combination of the packets be stored in the packet header. Katti *et al.* [29] proposed COPE which uses

opportunistic coding to increase the throughput. Chachulski *et al.* [30] proposed MORE which uses opportunistic routing to further improve the throughput performance.

For the schemes presented in [28–30], the source divides native data packets into *generations* or *blocks*. Network coding is performed within the same generation at the source and intermediate nodes. The source starts to transmit the next generation of the packets only after being acknowledged by the destination that the current generation of the packets have been decoded. It is shown in [31] that for generation based network coding, the max-flow min-cut capacity for the end-to-end communication can be achieved if the generation size is sufficiently large. However, the decoding delay can be large since the destination has to receive enough independent coded packets before decoding a generation. CodeOR was proposed in [32] to overcome the problem of long decoding delay by allowing multiple generations to be transmitted together.

The concept of *online network coding* was proposed by Sundararajan *et al.* in [33] and discussed further in [34–36]. For online network coding, at the source, each native packet is ready to be combined and transmitted as soon as it is created. At the destination, by performing linear operations on the received packets, the packets can be decoded consecutively instead of generation by generation. In [34], an online network coding scheme was proposed to maximize the throughput for a packet erasure broadcast channel. In [35], a new encoding rule was proposed to guarantee small decoding delay by recovering packet losses within reasonable time. In [36], the performance of online random linear network coding approach for time division duplexing channels under Poisson arrivals was

studied. SlideOR [37] is an online network coding scheme, which uses a moving sliding window at the source to determine the set of native packets to be coded.

Some of the previous works on online network coding are limited in different aspects. The work in [34] only analyzed the multicast case with one source and three receivers. The work in [35] assumed zero feedback delay, which is not the case in real networks. In [36], although the scheme is called *online network coding*, it is fundamentally generation based network coding with an improved online feedback mechanism. In [37], the network coding scheme is tailored for wireless mesh networks. For a network with a mixed wired and wireless network nodes, since the broadcast nature of wireless channels is not available in the wired part, the opportunistic routing based scheme has no advantage over routing.

Recently, the joint problem of network coding and transport layer design has received much attention. The work in [38] and [39] showed that the combination of network coding in medium access control (MAC) or network layer and TCP can have significant impact on the throughput, fairness and delay performance of wireless networks. Hassayoun *et al.* [40] analyzed the performance of TCP in coded wireless mesh networks with random packet loss. However, only the butterfly topology was considered and the model cannot be extended for general network topologies. Chen *et al.* [41] proposed a distributed rate control algorithm for network coding based on the utility maximization model. To implement the algorithm in real networks, a complex interface between network coding and TCP is required. It is because the proposed algorithm does not support sliding window and is not compatible with current TCP. In [42], a congestion control algorithm

for unicast flows over coded packet networks using COPE [29] was proposed based on network utility maximization. Although an interface between COPE and transport layer was presented in [42], it can only improve throughput performance by increasing the coding opportunity. Moreover, the decoding delay was not studied in [42]. An automatic repeat request (ARQ) network coding based TCP protocol (which is referred to as TCP ARQNC in this thesis) was proposed in [43] and extended in [44]. This approach gives a new solution for the sliding window based TCP algorithm. However, packets of one generation are decoded when the receiver receives enough independent coded packets. This can potentially lead to a large decoding delay. In [45], an analytical model was established to evaluate the throughput performance of network coded TCP proposed in [43]. However, the delay performance was not studied.

Our goal in Chapter 3 is to design an online network coding based TCP for real time applications (e.g., VoIP, video conferencing) such that the packets can be decoded with a small decoding delay. The proposed protocol should be able to distinguish between random packet loss and congestion loss for wireless links. To settle these issues, our proposed algorithm includes congestion control and online network coding control. For congestion control, we use TCP Vegas [9]. We can distinguish random packet losses from the congestion losses using the difference between the time that a packet is transmitted and the time that the sender is notified the packet is lost. The online network coding control is used to address the effect of random packet loss for wireless links. The sender chooses to transmit either a packet with new information or a redundant packet according

to the feedback information. Meanwhile, packets are decoded consecutively instead of generation by generation so that the average decoding delay is small. Another advantage of our protocol which makes it practical is that there is no need to change the protocol stack at the intermediate nodes. We only need to modify TCP in the transport layer at the sender and receiver. Since we use TCP Vegas as the congestion control algorithm, the TCP friendly feature is also guaranteed.

1.3 Contributions

This thesis covers several aspects of modeling, analysis and enhancement of transmission control protocol. The results are divided into two chapters. In Chapter 2, we propose an analytical model to determine the steady state throughput of TCP CUBIC in wireless environment. In Chapter 3, we propose a new mechanism that incorporates online network coding into TCP.

The contributions in each chapter are as follows:

- In Chapter 2, we formulate an analytical model to analyze the performance of TCP CUBIC in wireless networks. We propose a Markovian model to determine the steady state throughput of TCP CUBIC. The model takes into account both buffer router and fading in the wireless environment by considering both congestion loss and random packet loss. Then, we derive the stationary distribution of the Markov chain and obtain the average TCP throughput. The analytical model is

validated via simulations. Finally, we evaluate the throughput performance of TCP CUBIC. Results show that random packet loss reduces the normalized average throughput more for end-to-end flow with large bandwidth-delay product. We propose an improvement to increase the throughput performance of TCP CUBIC by moderately increasing the window growth factor and the multiplicative decrease factor.

- In Chapter 3, we propose TCP Vegas with online network coding (TCP VON), which is a combination of TCP Vegas and online network coding with low decoding delay. We establish an analytical framework to model the TCP VON and derive the average decoding delay using Markov chain. We also conduct ns-2 simulations to validate the analytical model. We compare the average decoding delay and throughput performance of TCP VON with TCP ARQNC [43] under different topologies. The results show that for a fixed network throughput, TCP ARQNC has a larger average decoding delay than TCP VON. Under the same decoding delay constraint, our proposed protocol provides a higher throughput than the TCP ARQNC.

1.4 List of Publications

The following publications have been completed based on the work in this thesis.

- Wei Bao, Vincent W.S. Wong, and Victor C.M. Leung, "A model for steady state throughput of TCP CUBIC," in *Proc. of IEEE Global Communications Conference*

(Globecom), Miami, Florida, December 2010.

1.5 Structure of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we present the analytical model of the throughput performance of TCP CUBIC in wireless networks. In Chapter 3, we present the algorithm design, analytical model and the simulation results of TCP VON. Finally, conclusions and future work are given in Chapter 4.

Chapter 2

A Model for Steady State Throughput of TCP CUBIC

In the transport layer, CUBIC is a TCP-friendly high-speed variant, in which the window size is a cubic function of time since the last loss event. TCP CUBIC is implemented in Linux operating systems and performs well in wired networks with large bandwidth-delay product. Most of the evaluations of TCP CUBIC are conducted via simulations or experiments. Analytical models for TCP CUBIC are few. In this chapter, we propose a Markovian model to determine the steady state throughput of TCP CUBIC in wireless environment. The proposed model considers both congestion loss and random packet loss due to fading. We derive the stationary distribution of the Markov chain and obtain the average throughput based on the stationary distribution. Then, we conduct simulations to validate our analytical model. Finally, we analyze the throughput performance of TCP CUBIC based on our model. The publication [46] has been completed based on the work in this chapter.

2.1 System Model for TCP CUBIC

In this section, we first present the network model and state the assumptions of the system model. We then describe the window behavior of TCP CUBIC as a Markov chain. After that, we derive the stationary distribution of the Markov chain and obtain the steady state throughput based on the stationary distribution.

2.1.1 Congestion Loss and Random Packet Loss

Consider the network where the *bottleneck link* is the last hop wireless link. This wireless bottleneck link has a capacity of C bits/sec and is smaller than the capacities of other intermediate links between the source and destination pair. This scenario is applicable to the scenario as in 3GPP (Third Generation Partnership Project) LTE (Long Term Evolution) or WiMAX (Worldwide Interoperability for Microwave Access). The source has a large file to send to the destination. We assume that packet losses are caused by two factors: *congestion loss* and *random packet loss*.

Congestion loss happens when the transmission rate attains the maximum capacity C of the bottleneck link. We assume that the average RTT is a constant, which is a common assumption in loss-based TCP analytical modeling (e.g., [47]). Thus, the maximize congestion window size W is

$$W = C \cdot RTT. \tag{2.1}$$

Equivalently, congestion loss happens when window size attains the maximum window

size W.

Random packet loss is caused by fading or interference in the wireless link. We assume that random packet loss experiences a random Poisson process with rate λ. This assumption has also been made in [40]. Given a time instant t_0, the time duration τ_{loss} from time t_0 to the next loss event is a random variable with an exponential distribution. The probability density function (pdf) of τ_{loss} is

$$f(\tau_{loss}) = \lambda \exp(-\lambda \tau_{loss}), \qquad \tau_{loss} > 0. \tag{2.2}$$

Given the time instant t_0, the probability that the next loss event happens within the time interval $(t_0 + T_1, t_0 + T_2]$ is

$$P(T_1 < \tau_{loss} \leq T_2) = \exp(-\lambda T_1) - \exp(-\lambda T_2). \tag{2.3}$$

2.1.2 Congestion Control for TCP CUBIC

As discussed in Section 1.1, for TCP CUBIC, the window size is a cubic function of time since the last loss event.

$$w(x, \tau) = \alpha \left(\tau - \sqrt[3]{(1-\beta)x/\alpha}\right)^3 + x. \tag{2.4}$$

where τ denotes the elapsed time from the last window reduction; the window size just before the last window reduction is denoted by x; the constant α denotes the window growth factor; the constant β represents the multiplicative decrease factor; $w(x, \tau)$ denotes the window size as a function of x and τ.

The congestion window reduction occurs due to either congestion loss or random packet loss event. When window reduction happens, the window size reduces to β times the window size just before the loss event. After that, it grows according to (2.4). Let $D(x, y)$ denote the time duration in which the window size grows from βx to y without encountering another loss event, after the last window reduction happened at the value of x. We have

$$D(x, y) = \sqrt[3]{\frac{y - x}{\alpha}} + \sqrt[3]{\frac{(1 - \beta)x}{\alpha}}. \tag{2.5}$$

2.1.3 Markov Chain Formulation

We now present our proposed Markovian model. The range of congestion window size $(0, W]$ is partitioned into N equal-sized intervals. The ith interval is $((i-1)W/N, iW/N]$. If the congestion window size is in the ith interval, we regard the window size to be the midpoint of the interval, with value $(i - 0.5)W/N$, denoted by a_i. This is similar to *quantization*: we *map* a continuous range of values $(0, W]$ to a finite set of values $\{a_1, a_2, ..., a_N\}$. The number of intervals N can also be regarded as quantization precision: when N increases, the mapping becomes more precise. The quantization is an essential step of the Markov chain formulation: the finite set of values derived through quantization corresponds to the Markov chain with N states.

The Markov chain is observed at each time instant when there is a window reduction (i.e., loss event). The kth time instant, where $k = 1, 2, \ldots$, corresponds to the kth window reduction. For a TCP CUBIC session, when the kth window reduction is just about to

happen, the congestion window size is denoted by x_k. x_k is in one of the N intervals, and is mapped to $\widetilde{x}_k$, $\widetilde{x}_k \in \{a_1, a_2, ..., a_N\}$.

When $\widetilde{x}_k = a_i$, the *state* of the TCP CUBIC session is in the ith state at the time instant k. Let X_k denote the state at the time instant k. We have $X_k = i$ and $\widetilde{x}_k = a_i$ if $x_k \in ((i-1)W/N, iW/N]$. Table 2.1 shows the definition of x_k, $\widetilde{x}_k$ and X_k.

Let random variable τ_k denote the time duration between the time instant k (i.e., the kth window reduction) and the time instant $(k+1)$ (i.e., the $(k+1)$th window reduction). Given x_k, the maximum value of congestion window size is W. The maximum value of τ_k is $D(x_k, W)$.

Consider the *state sequence* $X_1, X_2, \ldots, X_k$. From (2.4), the congestion window size at the next loss event after time instant k is only related to x_k and τ_k. That is

$$\begin{aligned} x_{k+1} &= w(x_k, \tau_k) \\ &= \alpha \left(\tau_k - \sqrt[3]{\frac{(1-\beta)x_k}{\alpha}} \right)^3 + x_k, \qquad \tau_k \leq D(x_k, W). \end{aligned} \tag{2.6}$$

Thus, x_{k+1} is independent of $x_1, x_2, \ldots, x_{k-1}$. We have

$$P(x_{k+1} \mid x_k, x_{k-1}, \ldots, x_1) = P(x_{k+1} \mid x_k). \tag{2.7}$$

Furthermore, the conditional probabilities of $\widetilde{x}_{k+1}$ and X_{k+1} are as follows:

$$P(\widetilde{x}_{k+1} \mid \widetilde{x}_k, \widetilde{x}_{k-1}, \ldots, \widetilde{x}_1) = P(\widetilde{x}_{k+1} \mid \widetilde{x}_k), \tag{2.8}$$

and

$$P(X_{k+1} \mid X_k, X_{k-1}, \ldots, X_1) = P(X_{k+1} \mid X_k). \tag{2.9}$$

Table 2.1: Definitions of x_k, $\widetilde{x}_k$ and X_k.

Symbol	**Definition**
x_k	When the kth window reduction is just about to happen, the congestion window size is x_k, $x_k \in (0, W]$.
$\widetilde{x}_k$	The mapped value of x_k at the time instant k, if $(i-1)W/N < x_k \leq iW/N$, then $\widetilde{x}_k = a_i = (i-0.5)W/N$.
X_k	The state at the time instant k, if $(i-1)W/N < x_k \leq iW/N$, then $X_k = i$, $X_k \in \{1, 2, \ldots, N\}$.

Therefore, the state sequence, $X_1, X_2, \ldots$, is a Markov chain.

2.1.4 State Transition Probability

Fig. 2.1 shows the process when the state transits from the ith state at the time instant k to jth ($j \neq N$) state at the time instant $(k+1)$ (i.e., $X_k = i$ and $X_{k+1} = j$). At time instant k, the loss event occurs, the congestion window size is reduced from a_i to βa_i. After that, the congestion window size grows within the interval of $((j-1)W/N, jW/N]$.

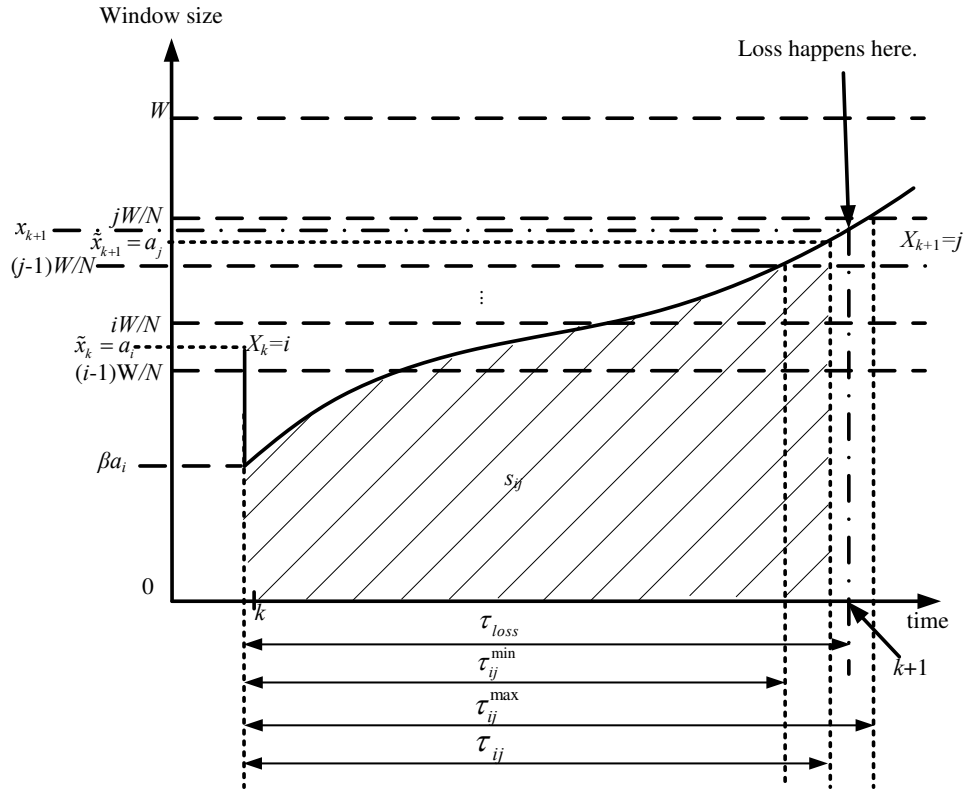


Figure 2.1: Transition probability and average throughput calculation.

At the time instant $(k+1)$, the congestion cannot happen below the value βa_i.Thus,

$$P_{ij} = 0, \qquad \text{if } jW/N < \beta a_i. \tag{2.10}$$

Since $a_i = (i - 0.5)W/N$, when the inequality $jW/N < \beta a_i$ holds, it is equivalent to state $j < \beta(i - 0.5)$.

When the state transits from the ith state to the jth state, it is in the interval $((j-1)W/N, jW/N]$ that a loss event happens. We have

$$\tau_{ij}^{\min} < \tau_{loss} \leq \tau_{ij}^{\max}, \tag{2.11}$$

where $\tau_{ij}^{\min}$ is the minimum time duration that the state transits from the ith state to the jth state; it is the time that the congestion window size grows from βa_i to $(j-1)W/N$

after the window reduction happened at the value of a_i.

$$\begin{aligned} \tau_{ij}^{\min} &= (D(a_i,(j-1)W/N))^+ \\ &= \left(\sqrt[3]{\frac{(j-1)W/N-a_i}{\alpha}}+\sqrt[3]{\frac{(1-\beta)a_i}{\alpha}}\right)^+, \end{aligned} \tag{2.12}$$

where $(a)^+ = \max(a, 0)$.

$\tau_{ij}^{\max}$ is the maximum time duration that the state transits from the ith state to the jth state. It is given by

$$\begin{aligned} \tau_{ij}^{\max} &= D(a_i, jW/N) \\ &= \sqrt[3]{\frac{jW/N-a_i}{\alpha}}+\sqrt[3]{\frac{(1-\beta)a_i}{\alpha}}. \end{aligned} \tag{2.13}$$

Thus, according to (2.3), if $jW/N \geq \beta a_i$ (i.e., $j \geq \beta(i-0.5)$) and $j \neq N$, then the state transition probability is

$$\begin{aligned} P_{ij} &= P\left((D(a_i,(j-1)W/N))^+ < \tau_{loss} \leq D(a_i,jW/N)\right) \\ &= \exp\left(-\lambda\left(D\left(a_i,\frac{(j-1)W}{N}\right)\right)^+\right) \\ &\qquad - \exp\left(-\lambda D\left(a_i,\frac{jW}{N}\right)\right) \\ &= \exp\left(-\lambda\left(\sqrt[3]{\frac{(j-1)W/N-a_i}{\alpha}}+\sqrt[3]{\frac{(1-\beta)a_i}{\alpha}}\right)^+\right) \\ &\qquad - \exp\left(-\lambda\left(\sqrt[3]{\frac{jW/N-a_i}{\alpha}}+\sqrt[3]{\frac{(1-\beta)a_i}{\alpha}}\right)\right). \end{aligned} \tag{2.14}$$

When state j is equal to N (i.e., $j = N$), we have

$$P_{iN} = 1 - \sum_{j=1}^{N-1} P_{ij}. \tag{2.15}$$

Fig. 2.2 shows an example of the Markov chain when $N = 5$ and $\beta = 0.5$. According to (2.10), (2.14) and (2.15), the ith state can transit to the jth state if $j \geq \beta(i-0.5)$.

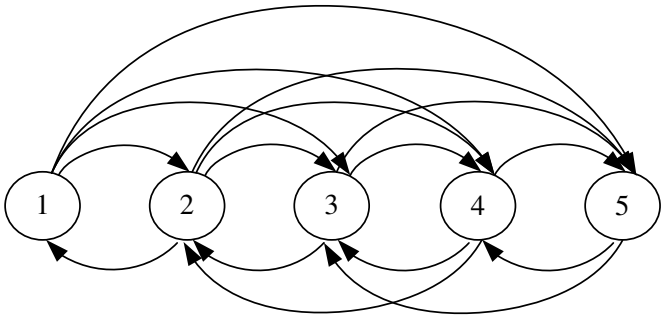


Figure 2.2: Markov chain example.

2.1.5 Stationary Distribution and Throughput

Let $(\pi_1, \pi_2, \ldots, \pi_N)$ denote the stationary distribution of the Markov chain, in which π_i is the stationary probability of the ith state. Given the state transition probabilities from (2.10), (2.14) and (2.15), we can derive the stationary distribution $(\pi_1, \pi_2, \ldots, \pi_N)$ by solving

$$\sum_{i=1}^{N} \pi_i P_{ij} = \pi_j, \qquad j \in \{1, 2, \ldots, N\}, \tag{2.16}$$

and

$$\sum_{i=1}^{N} \pi_i = 1. \tag{2.17}$$

The duration that the state transits from the ith state to the jth state is a random variable in the interval of $\big((D(a_i, (j-1)W/N))^+, D(a_i, jW/N)\big]$. When N is large enough, we can regard

$$\tau_{ij} = (D(a_i, (j - 0.5)W/N))^+, \tag{2.18}$$

as the average time duration that the state transits from the ith state to the jth state.

Let $s_{ij} = \int_0^{\tau_{ij}} w(a_i, t)dt$, which is the shaded area in Fig. 2.1.

$$\begin{aligned} s_{ij} &= \int_0^{\tau_{ij}} w(a_i, t)d\tau \\ &= \int_0^{\tau_{ij}} \left(\alpha \left(t - \sqrt[3]{\frac{(1-\beta)a_i}{\alpha}} \right)^3 + a_i \right) d\tau \\ &= a_i\tau_{ij} + \frac{\alpha}{4}\left((\tau_{ij} - L_C)^4 - L_C^4 \right), \end{aligned} \tag{2.19}$$

where $L_C = \sqrt[3]{\frac{(1-\beta)a_i}{\alpha}}$.

Let r_{ij} denote the total transmission amount while the state transits from the ith state to the jth state. From (2.19), we have

$$\begin{aligned} r_{ij} &= \int_0^{\tau_{ij}} \frac{w(a_i, t)}{RTT} d\tau \\ &= \frac{s_{ij}}{RTT}. \end{aligned} \tag{2.20}$$

Let t denote the total transmission time and $r(t)$ denote the total transmission amount during t. $M_{ij}(t)$ denotes the occurrence times of the ith state transition to the jth state during t. When t is large, by the law of large numbers, the limit $\lim_{t\to\infty} \frac{M_{ij}(t)}{\sum_{i=1}^{N}\sum_{j=1}^{N} M_{ij}(t)}$ is equal to the occurrence probability of the ith state transition to the jth state. Thus, the

average normalized TCP CUBIC throughput is

$$\begin{aligned}
\overline{x} &= \frac{\lim_{t\to\infty}\frac{r(t)}{t}}{C} = \frac{RTT}{W}\lim_{t\to\infty}\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}M_{ij}(t)r_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N}M_{ij}(t)\tau_{ij}} \\
&= \frac{1}{W}\lim_{t\to\infty}\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}M_{ij}(t)(r_{ij}\cdot RTT)}{\sum_{i=1}^{N}\sum_{j=1}^{N}M_{ij}(t)\tau_{ij}} \\
&= \frac{1}{W}\lim_{t\to\infty}\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}M_{ij}(t)s_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N}M_{ij}(t)\tau_{ij}} \\
&= \frac{1}{W}\lim_{t\to\infty}\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\left(M_{ij}(t)/\sum_{p=1}^{N}\sum_{q=1}^{N}M_{pq}(t)\right)s_{ij}\right)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\left(M_{ij}(t)/\sum_{p=1}^{N}\sum_{q=1}^{N}M_{pq}(t)\right)\tau_{ij}\right)} \\
&= \frac{1}{W}\lim_{k\to\infty}\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}P(X_{k+1}=j,X_k=i)s_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N}P(X_{k+1}=j,X_k=i)\tau_{ij}} \\
&= \frac{1}{W}\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}\pi_i P_{ij}s_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N}\pi_i P_{ij}\tau_{ij}}.
\end{aligned} \tag{2.21}$$

2.2 Performance Evaluation

In this section, we first validate our proposed analytical model via simulation. We then present the TCP throughput results under different parameters.

2.2.1 Analytical Model Validation via Simulation

We develop a discrete-event simulator to validate the accuracy of our proposed analytical model. Let M_i, where $i \in \{1, 2, \ldots, N\}$, denote the counter of the ith state, K denote the total simulation time. We first initialize W, N, β, α and set M_i to be equal to 0 at the beginning; x_1 is randomly generated in $(0, W]$. We simulate the behavior of TCP CUBIC starting from the time instant $k = 1$ to $k = K$. The simulation at each time instant is performed as follows: At each time instant k, we first calculate $\widetilde{x}_k$ and X_k according to x_k and update the state counter (i.e., increase M_{X_k} by 1). Then, we simulate τ_k, which is the time duration from the time instant k to the next loss event. The value of τ_k is set to be equal to $\min(\tau_{loss}, D(x_k, W))$, in which τ_{loss} is randomly generated according to its pdf in (2.2). Based on x_k and τ_k, the window size at the next window reduction $x_{k+1} = w(x_k, \tau_k)$ can be calculated. The stationary distribution as well as the average TCP CUBIC throughput can then be determined.

For performance metric, we consider the root mean square (RMS) error, which is defined as

$$RMS = \sqrt{\frac{\sum_{i=1}^{N}(\pi_i - \widetilde{\pi}_i)^2}{N}}, \tag{2.22}$$

where π_i and $\widetilde{\pi}_i$ are obtained via analytical model and simulation, respectively. The RMS error reflects the gap between the analytical results and the simulation results. Fig. 2.3 shows the RMS error under different simulation time. The parameters are as follows: $C = 100$ Mb/s, $RTT = 100$ ms, $\alpha = 1$ Mb/s, $\beta = 0.5$, $\lambda = 1$ s^{-1}. The two curves show that when the N is increased, the RMS error will decrease. This illustrates that an

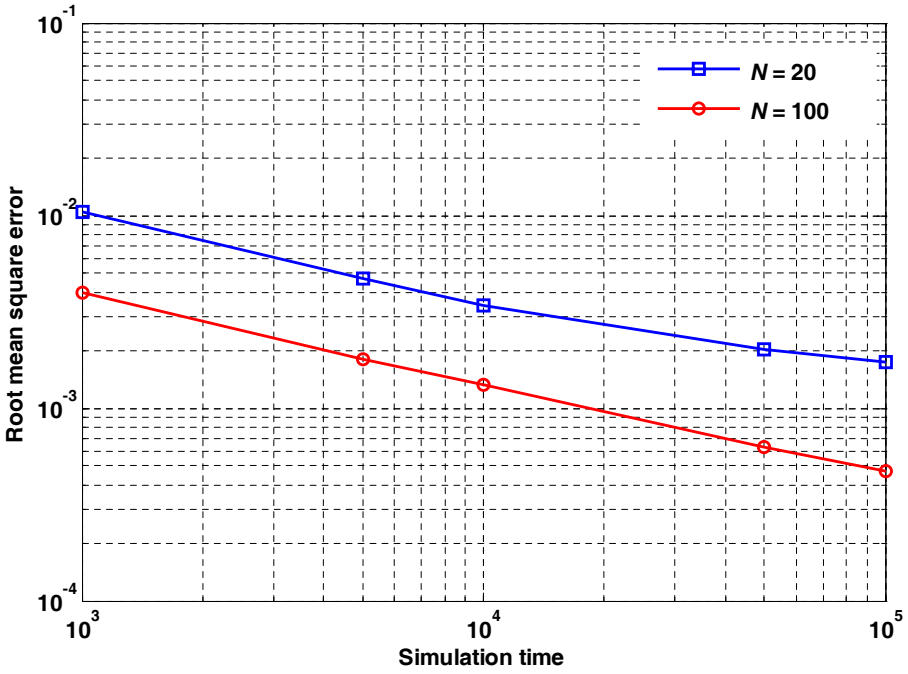


Figure 2.3: Root mean square (RMS) error versus total simulation time.

increase in simulation time leads to the simulation results being closer to the analytical results. In addition, we notice that when the number of intervals N increases, the RMS error becomes smaller. This illustrates that larger N will lead to more accurate analytical results.

Fig. 2.4 shows the analytical and simulated average throughput under different loss rate λ. The parameters chosen are as follows: $C = 100$ Mb/s, $RTT = 100$ ms, $\alpha = 1$ Mb/s, and $\beta = 0.5$. This figure shows that the analytical results and simulation results agree with each other.

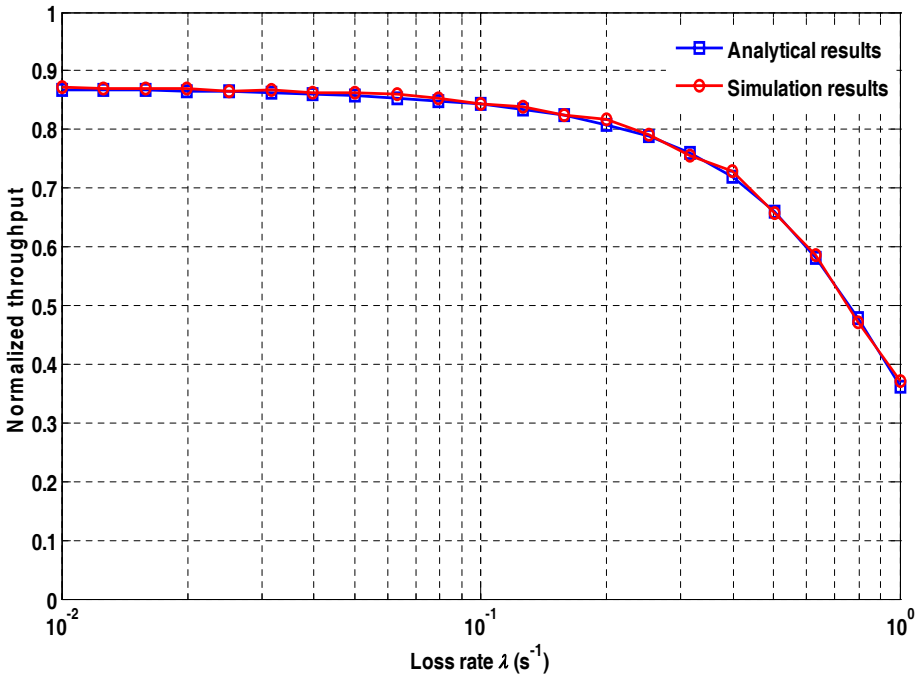


Figure 2.4: Analytical and simulated average normalized throughput under different loss rate λ.

2.2.2 Throughput Performance of TCP CUBIC

We now present the throughput results of TCP CUBIC based on our proposed Markovian model.

Fig. 2.5 shows the throughput performance of TCP CUBIC under different bandwidth-delay product $C \cdot RTT$ and loss rate λ, when $\alpha = 1$ Mb/s and $\beta = 0.5$. The number of intervals N is set to 100. Results show that even when λ is very small, the normalized average throughput will not attain 1. This is because even when there is no random packet loss, there are still congestion losses, causing multiplicative decrease when the window size is equal to W. Fig. 2.5 also shows that if $C \cdot RTT$ increases, the normalized average throughput decreases. Random packet loss reduces the normalized average throughput

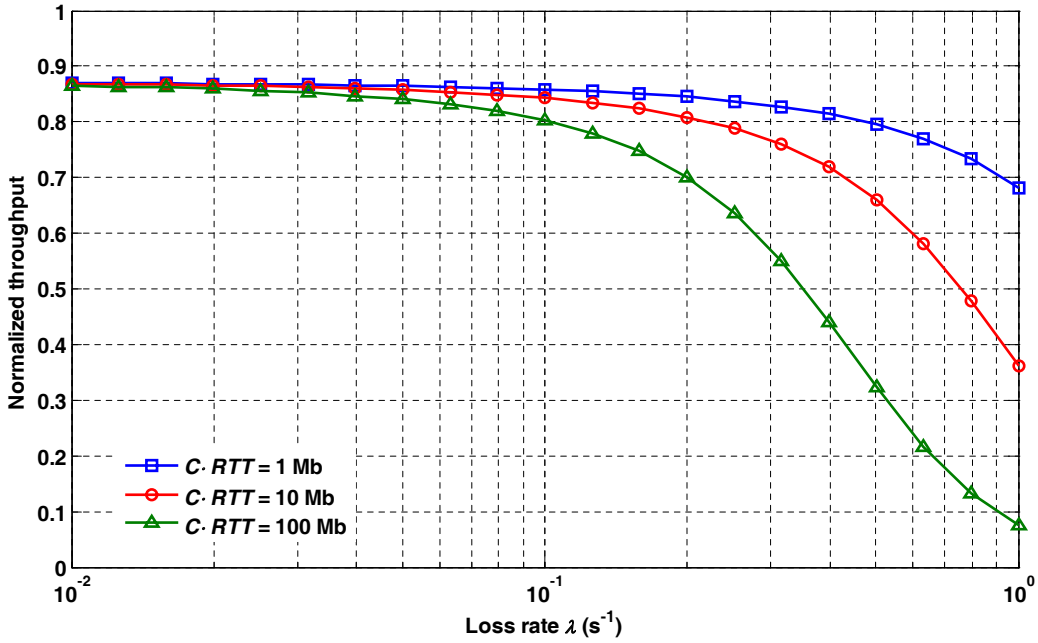


Figure 2.5: The normalized average TCP CUBIC throughput under different bandwidth-delay product $C \cdot RTT$ and loss rate λ.

of large bandwidth-delay product links more.

Our analytical results show that, for $i, j \in \{1, 2, \ldots, N\}$, $P_{ij}, \pi_i, \frac{s_{ij}}{W}$ and τ_{ij} depend on the value of $\frac{C \cdot RTT}{\alpha}$ (i.e., bandwidth-delay product over window growth factor).

Therefore, the normalized throughput depends on the value of $\frac{C \cdot RTT}{\alpha}$. The effect of an increase of bandwidth-delay product is equivalent to a decrease of the window growth factor, leading to a decrease of the normalized throughput. Therefore, in contrast to the wired cases, large bandwidth-delay product will decrease the normalized throughput of TCP CUBIC in wireless scenarios due to the random packet loss.

Fig. 2.6 shows the normalized average throughput under different window growth factor α with $C = 100$ Mb/s, $RTT = 100$ ms, $N = 100$, and $\beta = 0.5$. Results show

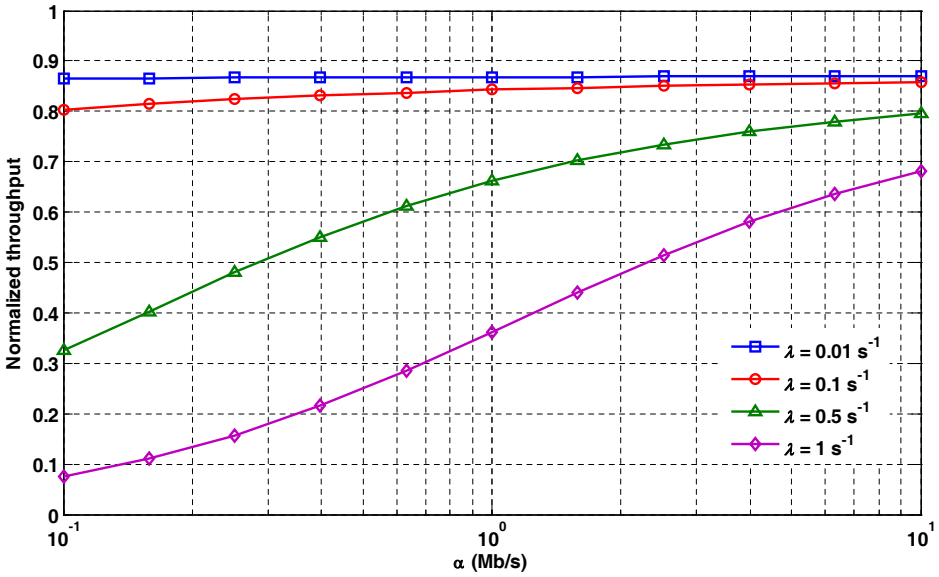


Figure 2.6: The normalized average throughput of TCP CUBIC under different window growth factor α.

that by moderately increasing α, the throughput performance will improve. When $\lambda = 1$ s^{-1}, the average normalized throughput increases by about 806% when α grows from 0.1 Mb/s to 10 Mb/s. When λ is small, increasing α will bring less throughput gain. In the figure, when $\lambda = 0.01$ s^{-1}, the normalized average throughput is not improved much.

Fig. 2.7 shows the normalized average throughput under different β with $C = 100$ Mb/s, $RTT = 100$ ms, $N = 100$, and $\alpha = 1$ Mb/s. Results show that by moderately increasing β, the throughput performance will improve. When $\lambda = 1$ s^{-1}, the average normalized throughput increases by about 137% when β grows from 0.5 to 0.9. When λ is small (e.g., $\lambda = 0.01$ s^{-1}), increasing β will bring less throughput gain.

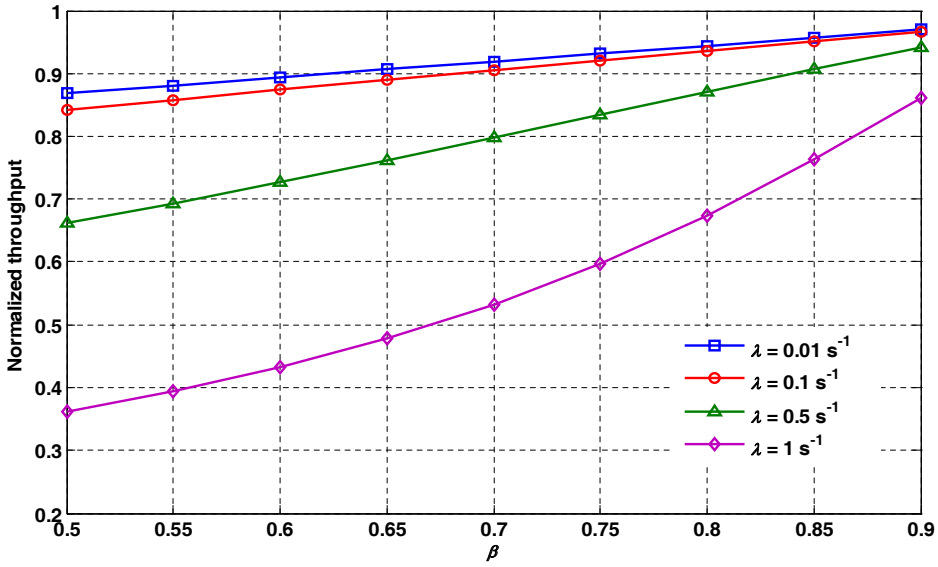


Figure 2.7: The normalized average throughput of TCP CUBIC under different β.

2.3 Summary

In this chapter, we proposed an analytical model to determine the steady state throughput of TCP CUBIC in wireless environment. We considered both congestion loss and random packet loss and established a Markov model. We derived the stationary distribution of the Markov chain and obtained the average throughput based on the stationary distribution. The accuracy of the model was validated via simulation. Based on our proposed model, we evaluated the throughput performance of TCP CUBIC. Results showed that random packet loss reduces the normalized average throughput more for end-to-end flow with large bandwidth-delay product. In wireless environment, we showed that the throughput of TCP CUBIC can be improved by moderately increasing the window growth factor α and the multiplicative decrease factor β.

Chapter 3

TCP VON: Online Network Coding Based TCP

In this chapter, we propose TCP Vegas with online network coding (TCP VON), which incorporates online network coding into TCP and it can be efficiently applied in mixed wired and wireless networks. The scheme requires minor modifications to the current protocol stack at the sender and the receiver. TCP VON includes two mechanisms, namely congestion control and online network coding control. The congestion control is extended from TCP Vegas so that congestion losses become rare and all the packet losses can be regarded as random packet losses. For the online network coding control, the sender transmits redundant coded packets when packet losses happen. Otherwise, it transmits innovative coded packets. As a result, all the packets can be decoded consecutively instead of generation by generation and the average delay from sending a packet at the sender to decoding it at the receiver is small. We establish a Markov chain model to compute the analytical delay performance of TCP VON. We also conduct ns-2 simulations to validate the proposed analytical models. Finally, we compare the delay and throughput performance of TCP VON and TCP ARQNC for different topologies. Results

show that TCP VON outperforms TCP ARQNC.

3.1 Preliminaries and Basic Definitions

In this section, we present the preliminaries and basic definitions of TCP VON. We first define different types of packets and then present the basic idea of how packets are coded in an online feature.

3.1.1 Packet Types

According to the previous works in [28, 29, 32, 43], there are two types of data packets in the network, namely *native packets* and *coded packets*. Native packets are the original packets generated by the sender, which are treated as vectors over a finite field $\mathbb{F}_q$ of size q. Each native packet has a unique index number k corresponding to the order it is generated. Let $\mathbf{p_k}$ denote the kth native packet. In this chapter, the native packets are with the same size (shorter packets are padded with zeros). A coded packet is a linear combination of several native packets. For example, if a coded packet $\mathbf{q}$ is a linear combination of packets $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_m}$, then $\mathbf{q} = \sum_{i=1}^{m} \alpha_i \mathbf{p_i}$, where α_i is a non-zero multiplicative coefficient randomly selected from the field $\mathbb{F}_q$.

The native packets can be divided into different categories. A native packet $\mathbf{p_k}$ is called a *sent* packet when the sender has transmitted a coded packet $\mathbf{q} = \alpha_k \mathbf{p_k} + \sum_{l \neq k} \alpha_l \mathbf{p_l}$. That is, $\mathbf{p_k}$ is combined in a coded packet $\mathbf{q}$, and $\mathbf{q}$ is transmitted by the sender. In this chapter, we use the term *transmit* for coded packets and the term *send*

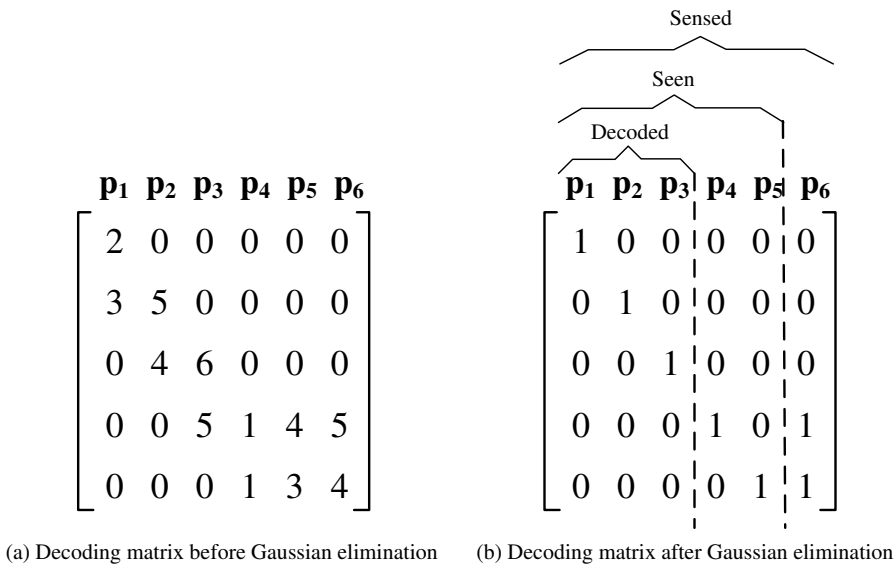
for native packets.

A native packet $\mathbf{p_k}$ is *sensed* when the receiver has received a coded packet $\mathbf{q} = \alpha_k \mathbf{p_k} + \sum_{l \neq k} \alpha_l \mathbf{p_l}$. That is, $\mathbf{p_k}$ is combined in a coded packet $\mathbf{q}$, and $\mathbf{q}$ is received at the receiver. Otherwise, $\mathbf{p_k}$ is *unsensed*. Packet $\mathbf{p_k}$ is *acknowledged as sensed* when the sender receives an ACK indicating that $\mathbf{p_k}$ has been sensed at the receiver.

As discussed in [43], a native packet $\mathbf{p_k}$ is defined as *seen*, when the receiver has enough information to compute a linear combination of the form $(\mathbf{p_k} + \mathbf{r_k})$, where $\mathbf{r_k} = \sum_{i>k} \alpha_i \mathbf{p_i}$ and $\alpha_i \in \mathbb{F}_q$. Otherwise, $\mathbf{p_k}$ is *unseen*. A native packet is *acknowledged as seen* when the sender receives an ACK indicating that $\mathbf{p_k}$ has been seen at the receiver.

A native packet $\mathbf{p_k}$ is called *decoded* when the receiver has enough information to decode $\mathbf{p_k}$. Otherwise, $\mathbf{p_k}$ is *undecoded*. As an example, when the receiver receives coded packets $\mathbf{p_1} + \mathbf{p_2}$, $\mathbf{p_1} + 2\mathbf{p_2}$ and $\mathbf{p_1} + \mathbf{p_2} + \mathbf{p_3} + \mathbf{p_4}$, packets $\mathbf{p_1}$ and $\mathbf{p_2}$ can be decoded. A packet $\mathbf{p_k}$ is *acknowledged as decoded* when the sender receives an acknowledgement (ACK) indicating that $\mathbf{p_k}$ has been decoded at the receiver.

In this chapter, the decoding mechanism at the destination is based on Gaussian elimination, which has also been applied in [37, 43, 44]. If we consider the native packets as unknown variables, then the set of received packets composes a system of linear equations. The decoding process is equivalent to solving a set of linear equations. The receiver can form a *decoding matrix* using the coefficients of coded packets. Gaussian elimination can be applied to convert this matrix to a reduced row echelon form and solve the linear equations. In the reduced row echelon form, the non all zero columns correspond to the



(a) Decoding matrix before Gaussian elimination (b) Decoding matrix after Gaussian elimination

Figure 3.1: Decoding matrix at the receiver.

sensed packets. The rows with leading coefficient 1 correspond to the seen packets. The rows with only one non-zero entry (i.e., the leading coefficient 1) correspond to the decoded packets. The rank of the matrix is equal to the number of seen packets. We notice that if all the native packets are seen, then the matrix has full rank and thus all the native packets can be decoded. Fig. 3.1 shows a sample of a decoding matrix before and after Gaussian elimination. The receiver has received 5 linear combinations of packets $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_6}$. After the Gaussian elimination, packets $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}$ are decoded, packets $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_5}$ are seen, and packets $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_6}$ are sensed. Native packets with index larger than 6 have not been sensed at the receiver.

3.1.2 Preliminaries of Coded Packets

At the sender side, let i_p, i_q and i_r denote the smallest index among the packets that have not been acknowledged as decoded, seen, and sensed, respectively. Let i_s denote the smallest index of the packet that has not been sent. We notice that $i_p \leq i_q \leq i_r \leq i_s$. To transmit a new coded packet, the sender combines native packets with consecutive indices within a *coding window*. Let $i_{\min}$ and $i_{\max}$ denote the minimum and maximum index of the native packets combined in the new coded packet, where the coding window begins at packet $i_{\min}$ and ends at $i_{\max}$.

For a new coded packet to be transmitted from the sender, it is required that $i_{\min} = i_q$ and $i_{\max}$ can be equal to either $i_s - 1$ or i_s. The reason is as follows: First, it is not necessary to have $i_{\min} < i_q$. Even if packet $\mathbf{p_l}$ for $l < i_q$ is combined in the coded packet, since $\mathbf{p_l}$ has already been seen at the receiver, the Gaussian elimination will eliminate the term of $\mathbf{p_l}$. Because $\mathbf{p_{i_q}}$ may not be seen at the receiver, the sender must combine $\mathbf{p_{i_q}}$ in the new coded packet. Therefore, $i_{\min} = i_q$. In this chapter, the packet with index i_q is called the code base[1]. Second, if $i_{\max} > i_s$, the transmitted packet contains at least two new variables (i.e., $\mathbf{p_{i_s}}$ and $\mathbf{p_{i_s+1}}$) for the receiver. However, it adds only one equation to the system of linear equations at the receiver. Therefore, to provide decoding opportunity at the receiver, $i_{\max} \leq i_s$. Because packet $\mathbf{p_{i_s-1}}$ has already been sent, $i_{\max} \geq i_s - 1$.

[1]In conventional TCP, the send base is the first byte which has not been acknowledged, or the first byte in the congestion window. Similarly, the code base is the first packet which has not been acknowledged as seen, or the first packet in the coding window.

Therefore, $i_{\max}$ can be equal to either $i_s - 1$ or i_s.

We use subscript and superscript to distinguish the coded packets transmitted by the sender. The subscript shows the last packet of the coding window. The superscript shows the number of times coded packets with same coding window is transmitted. For the new coded packet transmitted by the sender, if $i_{\max} = i_s$, then the transmitted packet contains information of a new native packet for the receiver and increases the number of variables at the receiver by one. We call such a packet an *innovative packet with index* i_s, which has the form

$$\mathbf{q}_{\mathbf{i_s}}^{(\mathbf{0})} = \sum_{k=i_q}^{i_s} \alpha_{i_s,0,k} \mathbf{p_k}, \tag{3.1}$$

where $\alpha_{i_s,0,k}$ is a randomly generated coefficient of $\mathbf{p_k}$. If $i_{\max} = i_s - 1$, then the coded packet is a linear combination of native packets $\mathbf{p_{i_q}}, \ldots, \mathbf{p_{i_s-1}}$,

$$\mathbf{q}_{\mathbf{i_s-1}}^{(\mathbf{j})} = \sum_{k=i_q}^{i_s-1} \alpha_{i_s-1,j,k} \mathbf{p_k}, \tag{3.2}$$

which is called *the jth redundant packet with index $i_s - 1$*. We use j to denote the number of times coded packets with linear combination of the same set of native packets $\mathbf{p_{i_q}}, \ldots, \mathbf{p_{i_s-1}}$ is generated. $\alpha_{i_s-1,j,k}$ is a randomly generated coefficient of $\mathbf{p_k}$ combined in the jth redundant packet. In this chapter, when the index of a coded packet (either an innovative or a redundant packet) is *not important*, we use $\mathbf{q}$ to denote a coded packet, α_k to denote the randomly generated coefficient of $\mathbf{p_k}$ combined in $\mathbf{q}$. When the index of a coded packet is important, we use the notations $\mathbf{q}_{\mathbf{i_s}}^{(\mathbf{j})}$ and $\alpha_{i_s,j,k}$.

For the case that there is no packet loss, the receiver can decode one native packet each

time it receives an innovative packet. For example, when the sender transmits innovative packets $\mathbf{q_1^{(0)}} = \alpha_{1,0,1}\mathbf{p_1}$, $\mathbf{q_2^{(0)}} = \alpha_{2,0,1}\mathbf{p_1} + \alpha_{2,0,2}\mathbf{p_2}$, and $\mathbf{q_3^{(0)}} = \alpha_{3,0,1}\mathbf{p_1} + \alpha_{3,0,2}\mathbf{p_2} + \alpha_{3,0,3}\mathbf{p_3}$ consecutively, the receiver can decode packet $\mathbf{p_1}$ when $\mathbf{q_1^{(0)}}$ arrives, decode packet $\mathbf{p_2}$ when $\mathbf{q_2^{(0)}}$ arrives, and decode packet $\mathbf{p_3}$ when $\mathbf{q_3^{(0)}}$ arrives.

For the case that there are packet losses in the network, the sender should transmit redundant packets to compensate the effect of packet losses. Postponing the transmission of redundant packets in this case can delay the decoding process. As an example, if the sender transmits M innovative packets back to back and x of them (where $x < M$) are lost, then those M packets can be decoded only if x redundant packets are subsequently received successfully at the receiver.

3.1.3 Acknowledgement (ACK)

Let I_p, I_q and I_r denote the index of the next packet to be decoded, seen and sensed at the receiver, respectively. These indices imply that the packets with indices smaller than I_p are decoded, the packets with indices smaller than I_q are seen, and the packets with indices smaller than I_r are sensed. Each ACK packet includes all of the three indices, in order to inform the sender which packets are decoded, seen and sensed. Note that i_p, i_q and i_r are variables at the sender while I_p, I_q and I_r are variables at the receiver and in the ACK packets. When an ACK packet arrives at the sender, i_p, i_q and i_r are then updated by the values of I_p, I_q and I_r, respectively.

Fig. 3.2 shows an example of packet coding according to an ACK packet received at

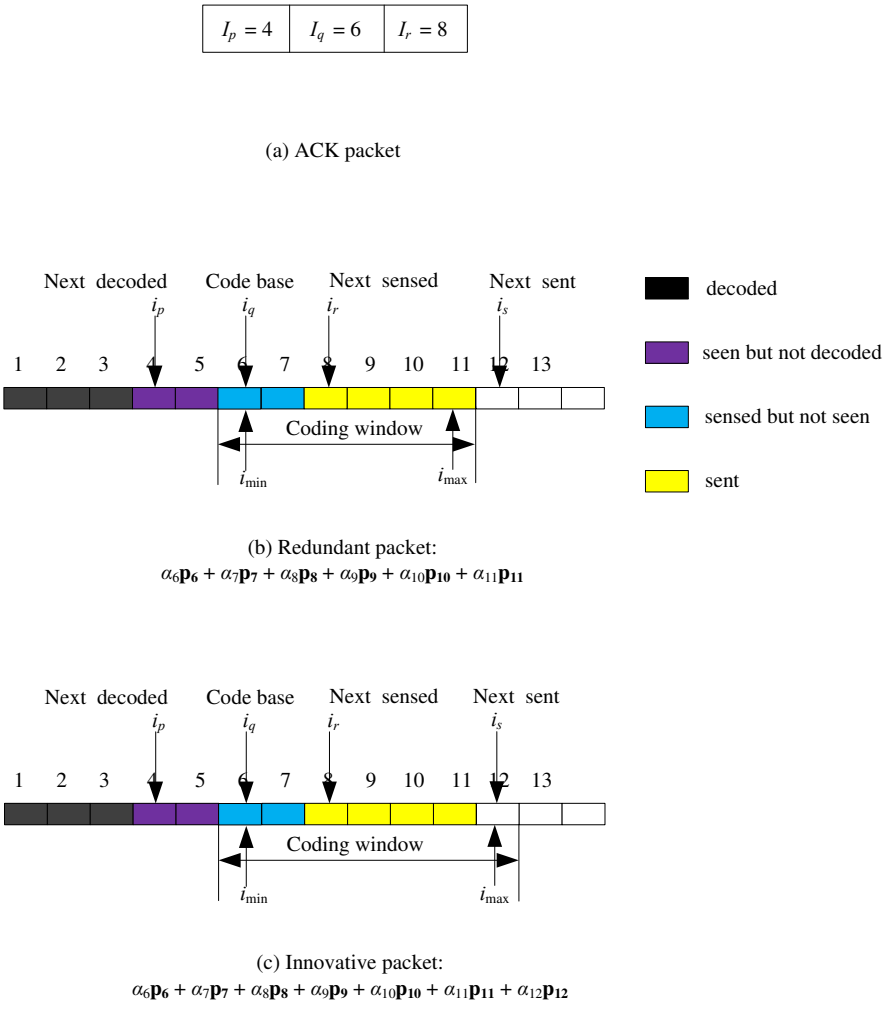


Figure 3.2: The two types of coded packets and ACK.

the sender. At the beginning, native packets $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \ldots, \mathbf{p_{11}}$ are sent back to back by the sender. Then, the sender receives an ACK packet which indicates $I_p = 4$, $I_q = 6$ and $I_r = 8$, as shown in Fig. 3.2 (a). The sender then updates $i_p = 4$, $i_q = 6$ and $i_r = 8$. Packets $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}$ are decoded, packets $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, ..., \mathbf{p_5}$ are seen and packets $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \ldots, \mathbf{p_7}$ are sensed. Then, the sender can transmit either a redundant packet (a linear combination of packets $\mathbf{p_6}, \ldots, \mathbf{p_{11}}$, shown in Fig. 3.2 (b)) or an innovative

packet (a linear combination of packets $\mathbf{p_6}, \ldots, \mathbf{p_{12}}$, shown in Fig. 3.2 (c)). In the next section, we will show how to make the decision to transmit an innovative packet or a redundant packet.

3.2 TCP VON Algorithm

The algorithm of TCP VON includes the sender side operation and the receiver side operation. The sender side operation is presented in Algorithm 1 and the receiver side operation is in Algorithm 2. At the sender, a nonnegative integer R is used as a tunable parameter. We call R the redundant factor. A higher value of R allows the the sender to transmit more redundant packets to compensate the detected packet losses. In networks with high loss probability, it is of interest to select higher values of R to reduce the decoding delay. Nevertheless, a higher value for R decreases the network throughput and there is an inherent tradeoff in choosing the value of R.

In this section, we first present the sender side operation for the case that $R = 0$ along with the receiver side operation. Then, we introduce the sender side operation for $R > 0$.

3.2.1 Sender Side Operation ($R = 0$ Case)

TCP is responsible for reliable end-to-end data transmission. When network coding is merged in the transport layer, this layer needs to perform the coding operations as well as congestion control and flow control. We first explain the congestion and flow control

part. Then, we introduce the online network coding control.

Congestion and Flow Control

For the congestion and flow control, we follow the TCP Vegas algorithm [9]. The sender maintains several variables including congestion window size *cwnd*, measured round trip time RTT, the minimum of all measured round trip times $BaseRTT$, standard deviation of the measured round trip time Δ, and the receive window size *rwnd*. The number of packets on the flight is approximated by the number of sent but unseen packets (i.e., $i_s - i_q$), which is limited by the congestion window size *cwnd* and receive window size *rwnd*

$$i_s - i_q \leq \min\{cwnd, rwnd\}. \tag{3.3}$$

In order to focus on congestion control, we assume that the buffer size at the receiver is large enough so that the effect of the receive window *rwnd* can be ignored [1, 9, 47]. The congestion window size *cwnd* is adjusted according to the algorithm from TCP Vegas [9]. We set two predetermined threshold values a and b, $a < b$. If $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT < a$, then *cwnd* is linearly increased. If $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT > b$, then *cwnd* is linearly decreased. In our algorithm, we select the default values of $a = 1$ packet and $b = 3$ packets.

Table 3.1: Definitions of Variables Used for Online Network Coding Control.

Variable name	Definition	Variable type	Initial value	Update condition
RealGap	The number of native packets sensed but unseen. $RealGap = i_r - i_q$.	Nonnegative integer	0	The sender receives an ACK.
OldRealGap	The previous value of *RealGap*.	Nonnegative integer	0	Just before *RealGap* is being updated.
$RealGapDiff$	The difference between *RealGap* and *OldRealGap*. $RealGapDiff = RealGap - OldRealGap$.	Nonnegative integer	0	Right after *RealGap* has been updated.
ExpGap	The maximum number of lost packets can be tolerated at the sender.	Nonnegative integer	0	A redundant packet is transmitted. *RealGap* is decreased ($RealGapDiff > 0$).
State	A variable to instruct whether the next packet to transmit is redundant or innovative.	Either $REDUNDANT$ or $INNOVATIVE$	$INNOVATIVE$	If $RealGap > ExpGap$, $State = REDUNDANT$, otherwise, $State = INNOVATIVE$.

Online Network Coding Control

The core of our online network coding control is to decide whether to transmit an innovative packet or a redundant packet such that the receiver can decode the packets with small decoding delay. The online network coding control maintains different variables including the real gap *RealGap*, the old real gap *OldRealGap*, the real gap difference $RealGapDiff$, the expected gap *ExpGap*, and the state *State*. Table 3.1 shows the definitions and properties of the variables. The real gap *RealGap* is the number of native

packets sensed but unseen at the sender. That is, $RealGap = i_r - i_q$. The value of *RealGap* is updated whenever the sender receives an ACK packet. *RealGap* shows the number of lost innovative packets. If the *RealGap* is zero, it indicates that all packets have been decoded at the receiver. If the *RealGap* is greater than zero, it implies that there are packet losses in the system. The old real gap *OldRealGap* is the variable to store the previous value of *RealGap*. Whenever an ACK is received, *OldRealGap* is set to be the value of *RealGap* just before *RealGap* is being updated. $RealGapDiff$ is the difference between *RealGap* and *OldRealGap* (i.e., $RealGapDiff = RealGap - OldRealGap$), which shows the decrease in real gap upon receiving an ACK packet. $RealGapDiff$ is calculated right after *RealGap* has been updated. The *ExpGap* is the maximum number of lost packets which can be tolerated at the sender. It is updated whenever a redundant packet is transmitted or *RealGap* is decreased, which will be discussed in details later. *State* is a variable taking either $REDUNDANT$ or $INNOVATIVE$ to instruct whether the next packet to transmit is a redundant packet or an innovative one. If $RealGap > ExpGap$, then the number of lost packets exceeds the maximum number that the sender can tolerate and the *State* will change into $REDUNDANT$. The sender transmits a redundant packet in this case. Otherwise (i.e., $RealGap \leq ExpGap$), the *State* is $INNOVATIVE$ and the sender transmits an innovative packet.

We now explain the use of *ExpGap*. At the beginning, both *RealGap* and *ExpGap* are set to zero. During operation, the *RealGap* is equal to one, which indicates a packet loss. The sender transmits a redundant packet to compensate the packet loss and *ExpGap* is

increased by one. It causes *RealGap* to be equal to *ExpGap*, and the sender does not transmit more redundant packets during the period when *RealGap* is equal to *ExpGap*.

If the transmitted redundant packet gets lost in the network, the sender should be able to detect the loss of the redundant packet. Therefore, as soon as the sender transmits a redundant packet, it starts a timer with value $RTT + 4\Delta$ (where 4Δ is used to provide some time margin). When the seder receives an ACK indicating that the redundant packet is successfully received by the receiver, the *RealGap* is decreased by one before the timer expires. If *RealGap* has not been decreased when the timer expires, it indicates that the redundant packet has got lost. In this case, the sender decreases *ExpGap* by one, which causes *RealGap* to be greater than *ExpGap*, so that the sender transmits another redundant packet. Note that in our algorithm, one timer starts whenever a redundant packet is transmitted, and each redundant packet has its own timer.

If the transmitted redundant packet is received successfully at the receiver, then the sender is informed by an ACK packet indicating the decrease of *RealGap* by one (i.e., $RealGapDiff = 1$). In this case, the sender decreases *ExpGap* accordingly. The timer corresponding to the transmission of the redundant packet is stopped. $RealGapDiff$ can be larger than one, which indicates that more than one redundant packets have been received at the receiver. In this case, *ExpGap* is decreased by $RealGapDiff$, and the $RealGapDiff$ oldest timers are stopped.

Algorithm of Sender Side Operation

Algorithm 1 shows the algorithm of TCP VON at the sender. Line 1 shows the input of the tunable variable R. We first present the $R = 0$ case. The $R > 0$ case will be discussed in Section 3.2.5. Lines 2 to 4 are used to initialize the variables. Let i_m denote the total number of native packets created. As shown in Lines 6 to 9, if new data with X bytes arrives from upper layer, $n_s = \lceil \frac{X}{L} \rceil$ native packets are created, each with length of L bytes (the last packet is padded with 0 if necessary). At the same time, the total number of native packets is increased by n_s. Lines 10 to 37 show the online network coding algorithm. When *RealGap* is greater than *ExpGap*, a redundant packet is transmitted, *ExpGap* is increased by one, and a timer is started. Otherwise, an innovative packet is transmitted. Lines 38 to 53 present the operations when an ACK packet is received. The sender updates $BaseRTT$, Δ, i_p, i_q and i_r (Lines 39 to 40). Then, it calculates the value of *OldRealGap*, *RealGap* and $RealGapDiff$ (Lines 41 to 43). If the $RealGapDiff$ is larger than zero, the sender will decrease *ExpGap* accordingly and $RealGapDiff$ timers will be stopped. Then, the congestion control is performed in Lines 48 to 52. Lines 54 to 56 show the timeout event. The *ExpGap* will be decreased by one when a timeout event occurs.

Algorithm 1 Algorithm of TCP VON at the Sender.

Input: R
Set $i_p := 1$, $i_q := 1$, $i_r := 1$, $i_s := 1$, $i_{\min} := 1$, $i_{\max} := 1$, $i_m := 0$
Set $RealGap := 0$, $OldRealGap := 0$, $RealGapDiff := 0$, $ExpGap := 0$
Set $State := INNOVATIVE$, $cwnd := 1$
while the TCP connection is established
  if data with length X bytes is received from upper layer
    Segment data into $n_s := \lceil \frac{X}{L} \rceil$ packets: $\mathbf{p_{i_m+1}}$, $\mathbf{p_{i_m+2}}$, $\ldots$, $\mathbf{p_{i_m+n_s}}$
    Set $i_m := i_m + n_s$
  end if
  while $i_s - i_q \leq cwnd$ **and** $i_{\max} \leq i_m$
    if $R = 0$
      if $RealGap > ExpGap$
        Set $State := REDUNDANT$
      else
        Set $State := INNOVATIVE$
      end if
    elseif $R > 0$
      if $RealGap + R > ExpGap$ **and** $RealGap > 0$
        Set $State := REDUNDANT$
      else
        Set $State := INNOVATIVE$
      end if
    end if
    if $State = INNOVATIVE$
      Set $i_{\min} := i_q$, $i_{\max} := i_s$, $i_s := i_{\max} + 1$
      Create coded packet $\mathbf{q} = \sum_{j=i_{\min}}^{i_{\max}} \alpha_j \mathbf{p_j}$ using random coding coefficients
      Include $i_{\min}$, $i_{\max}$ and $\alpha_{i_{\min}}, \ldots, \alpha_{i_{\max}}$ in the packet header
      Transmit the coded packet
    elseif $State = REDUNDANT$
      Set $i_{\min} := i_q$, $i_{\max} := i_s - 1$
      Create redundant coded packet $\mathbf{q} = \sum_{j=i_{\min}}^{i_{\max}} \alpha_j \mathbf{p_j}$ using random coefficients
      Include $i_{\min}$, $i_{\max}$ and $\alpha_{i_{\min}}, \ldots, \alpha_{i_{\max}}$ in the packet header
      Transmit the coded packet
      Set $ExpGap := ExpGap + 1$
      Start a new countdown timer with value $RTT + 4\Delta$
    end if
  end while
  if an ACK is received
    Set $i_p := I_p$, $i_q := I_q$, $i_r := I_r$
    Compute $BaseRTT$ and Δ
    Set $OldRealGap := RealGap$
    Set $RealGap := i_r - i_q$
    Set $RealGapDiff := RealGap - OldRealGap$
    if $RealGapDiff > 0$
      Set $ExpGap := ExpGap - RealGapDiff$
      Stop $RealGapDiff$ oldest timers.
    end if
    if $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT > 3$
      Set $cwnd := cwnd - 1$ in the next RTT
    elseif $\left(\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}\right) \times BaseRTT < 1$
      Set $cwnd := cwnd + 1$ in the next RTT
    end if
  end if
  if a timer timeouts
    $ExpGap := ExpGap - 1$
  end if
end while

3.2.2 Receiver Side Operation

Algorithm 2 shows the algorithm of TCP VON at the receiver. I_p, I_q and I_r are initialized with value 1 (Line 1). **D** is the decoding matrix (Line 2). When the receiver receives an uncorrupted coded packet (Line 5), it first checks the packet header and retrieves $i_{\min}$, $i_{\max}$ and the coding coefficients of native packets combined. Then, the coefficients are added as a new row to the decoding matrix and Gaussian elimination is executed (Lines 6 to 8). Then, operations corresponding to the Gaussian elimination are performed on the received packets so far (Lines 9 to 11). Then, the receiver will find the next packet to be decoded using the *while loop* in Lines 14 to 17. The sender increases the value of I_p by one until it finds that the native packet with index I_p is not decoded. Thus, I_p is equal to the index of the first undecoded packet when the *while loop* is terminated. Similarly, the receiver will find the first packet unseen in Lines 18 to 20, and the first packet unsensed in Lines 21 to 23. Finally, an ACK packet indicating the indices I_p, I_q and I_r is generated and sent at the receiver (Line 24).

Algorithm 2 Algorithm of TCP VON at the Receiver.

```
1:  Set I_p := 1, I_q := 1, I_r := 1
2:  Decoding matrix D := [ ]
3:  while TCP connection is established
4:      if a packet is received
5:          if packet is not corrupted
6:              Retrieve i_min and i_max and the coding coefficients α_{i_min}, ..., α_{i_max}
7:              Add coding coefficients as a new row to matrix D
8:              Perform Gaussian elimination on D
9:              if an all-zero row appears
10:                 Discard the packet
11:                 Eliminate the all-zero row in D
12:             else
13:             Perform linear operations to packets corresponding to the Gaussian elimination
14:                 while native packet with index I_p is decoded
15:                     Deliver data in p_{I_p} to upper layer
16:                     I_p := I_p + 1
17:                 end while
18:                 while native packet with index I_q is seen
19:                     I_q := I_q + 1
20:                 end while
21:                 while native packet with index I_r is sensed
22:                     I_r := I_r + 1
23:                 end while
24:                 Send an ACK packet indicating I_p, I_q and I_r
25:             end if
26:         else
27:             Discard the corrupted packet
28:         end if
29:     end if
30: end while
```

3.2.3 Reliable Data Transfer

TCP VON is able to provide reliable data transfer and handle with packet loss, corruption, duplication and reordering. If a coded packet is either lost or corrupted, redundant packets will be transmitted to compensate the loss or corruption. If a coded packet is duplicated, an all-zero row will appear after the Gaussian elimination. The receiver will find that it is not linearly independent and discard the redundant packet. If packet reordering occurs, it is corresponding to row exchange in the decoding matrix and the

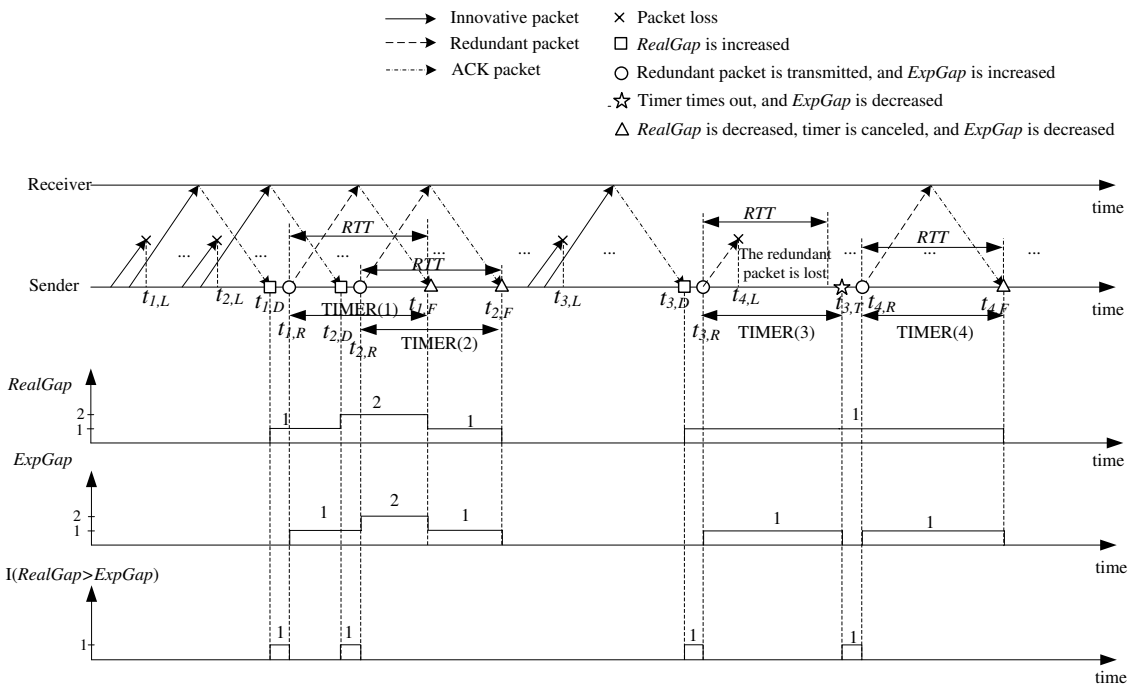


Figure 3.3: Example of real gap and expected gap.

decoding results do not change.

3.2.4 Example of the Algorithm of TCP VON with $R = 0$

Fig. 3.3 provides an example for the algorithm when $R = 0$. $I(RealGap > ExpGap)$ is an indicator function, having the value 1 if $RealGap > ExpGap$, and 0 otherwise. At the beginning, both $RealGap$ and $ExpGap$ are equal to 0. At time $t_{1,L}$, one packet is lost. The packet following the lost packet is received by the receiver and an ACK indicating that $I_r - I_q = 1$ is sent by the receiver. At time $t_{1,D}$, the sender receives the ACK and updates i_r and i_q by the values of I_r and I_q, respectively. Then, the $RealGap = i_r - i_q = 1$ is calculated. Since $RealGap$ is greater than $ExpGap$, at time $t_{1,R}$, a redundant packet

is transmitted, *ExpGap* is set to 1, and timer $TIMER(1)$ is set. Similarly, the second packet is lost at time $t_{2,L}$, and the sender receives an ACK packet indicating *RealGap* to be equal to 2 at time $t_{2,D}$. At time $t_{2,R}$, another redundant packet is transmitted, *ExpGap* is set to 2, and timer $TIMER(2)$ is set. At time $t_{1,F}$, the sender is informed by an ACK that the first redundant packet is received successfully at the receiver and *RealGap* becomes 1. $TIMER(1)$ is stopped and *ExpGap* is reduced to 1. Similarly, at time $t_{2,F}$, the sender is informed by an ACK that the second redundant packet is received successfully and *RealGap* becomes 0. $TIMER(2)$ is stopped and *ExpGap* is set to 0.

At time $t_{3,L}$, a third packet is lost. At time $t_{3,D}$, *RealGap* becomes 1. At time $t_{3,R}$, a redundant packet is transmitted, *ExpGap* becomes 1, and timer $TIMER(3)$ is set. However, the redundant packet transmitted is also lost at time $t_{4,L}$. Therefore, the *RealGap* does not decrease until time $t_{3,T}$ when $TIMER(3)$ times out and *ExpGap* becomes 0. Then *RealGap* is greater than *ExpGap* again and the fourth redundant packet is transmitted at time $t_{4,R}$. Finally, the fourth redundant packet is received successfully. At time $t_{4,F}$, *RealGap* becomes 0, timer $TIMER(4)$ is stopped, and *ExpGap* is set to 0.

3.2.5 Tunable Parameter R in the Sender Side Operation

We now present the use of the tunable parameter (redundant factor) R in the sender side operation. For networks with high loss probability, we may set $R > 0$ to allow the the sender to transmit more redundant packets to compensate the detected packet losses.

Consider the scenario that *RealGap* is increased from 0 to 1 when *ExpGap* is equal

to 0, due to the high loss probability, it is possible that there are more than one packet losses and only the first one is indicated by the ACK packet. In this case, the receiver cannot decode the packets even if it receives one redundant packet, and the decoding delay may become quite large. Thus, sending only one redundant packet may not be enough when *RealGap* is increased from 0 to 1 and *ExpGap* is equal to 0. With the redundant factor R greater than 0, the sender transmits $R+1$ redundant packets in this situation. As shown in Algorithm 1, if $R > 0$, the sender decides whether to transmit an innovative packet or a redundant packet according to Lines 18 to 22. It transmits redundant packets if $RealGap + R > ExpGap$ and $RealGap > 0$, otherwise, it transmits innovative packets. The reasons are as follows: First, when the sender detects the first packet loss (i.e., *RealGap* is increased from 0 to 1 when $ExpGap = 0$), the sender transmits $R+1$ redundant packets until $ExpGap = R+1$ and $RealGap + R$ is no larger than *ExpGap*. Second, if the sender detects x more packet losses (i.e., $RealGap = x+1$), the sender transmits x more redundant packet until $ExpGap = R+x+1$ and $RealGap+R$ is no larger than *ExpGap*. Third, when $RealGap = 0$, which means that all the packets can be decoded at the receiver, the sender does not transmit redundant packets.

After transmitting $R+1$ redundant packets, even if there are up to R more packet losses following the first packet loss, the receiver will still be able to decode all the packets when it receives the $R+1$ redundant packets. As a result, the decoding delay is expected to be smaller compared to the case when R is equal to 0.

Higher values of R can provide better decoding delay performance for networks with

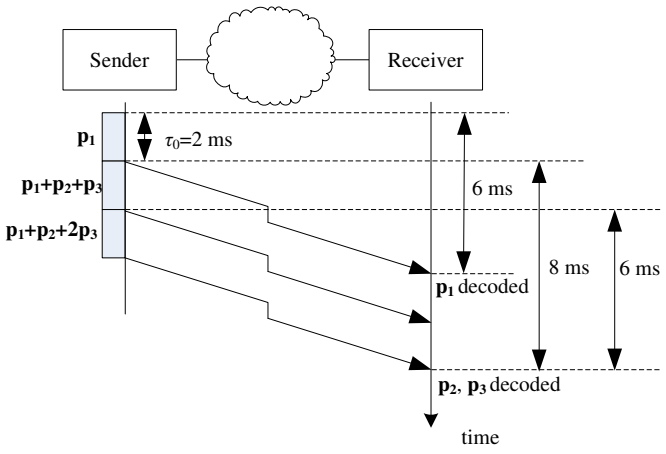


Figure 3.4: Example of decoding delay.

high loss probability, while it may deteriorate the throughput efficiency of the end-to-end communication for networks with low loss probability. We recommend to set R to 0 for wired networks and set R to 2 for wireless networks. Through the analytical methods in Section 3.3 and simulations in Section 3.4, we further study the delay-throughput tradeoff in choosing the tunable parameter R.

3.3 Decoding Delay Analysis of TCP VON

In this section, we establish a system model to analyze the decoding delay of TCP VON. First, we present a Markov chain model for the case $R = 0$. Then, we develop a method to estimate the performance when R is greater than zero.

The *decoding delay* is defined as the difference between the time that a packet is transmitted by the sender and the time at which that packet is decoded at the receiver. In this section, we determine the *average decoding delay* of all the packets to evaluate

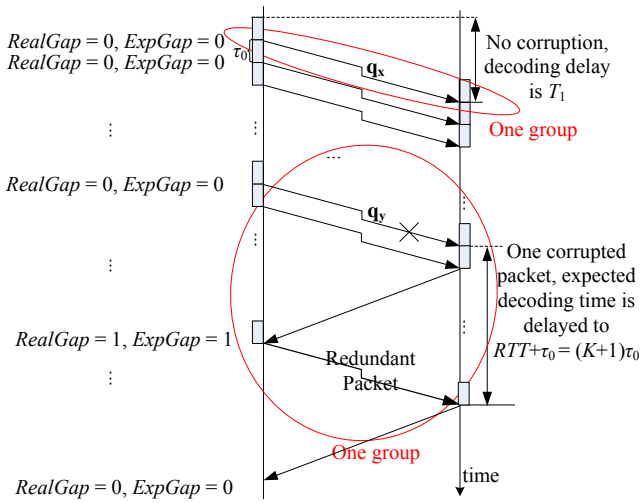
the decoding delay performance. For example, as shown in Fig. 3.4, the end-to-end delay from the sender to the receiver is 6 ms, the time duration between the two consecutive packets τ_0 is 2 ms. The decoding delay of $\mathbf{q_1}$, $\mathbf{q_2}$ and $\mathbf{q_3}$ are 6 ms, 8 ms and 6 ms, respectively. Thus, the average decoding delay is 20/3 ms.

For the delay analysis, we consider a single TCP session running in the network, where the last hop is a wireless link. This wireless link has a capacity of C bits/sec which is smaller than the capacity of all other intermediate links between the sender and the receiver. Therefore, this wireless link is the *bottleneck link* for the end-to-end communication. We assume that the sender has a large file to send. The packet size is L bits.

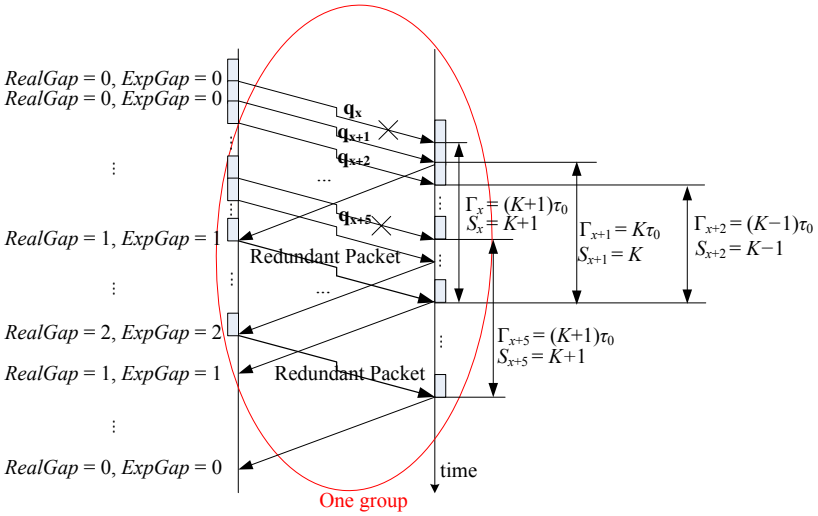
Due to the congestion control mechanism of TCP Vegas, we assume that the transmission rate at the sender attains the capacity of the bottleneck link C, and there is no congestion loss at the network. The propagation delay from the sender to the receiver is T_0, which is a given value. The time to transmit a packet τ_0 is equal to L/C. Due to the property of TCP Vegas, there are on average two packets in the buffers along the path from the sender to the receiver and the average queuing delay is $2L/C$ [9]. Thus, the end-to-end delay from the sender to the receiver is $T_1 = T_0 + 3L/C$. Because the length of each ACK packet is small, the end-to-end delay from the receiver to the sender T_2 is approximately equal to the propagation delay $T_2 = T_0$. Finally, the round trip time RTT is equal to $T_1 + T_2 = 2T_0 + 3L/C$. Because there is no cross traffic, we assume that T_1, T_2 and RTT are constants and Δ is 0. The transmission rate at the sender is C/L

packets/sec.

Each packet experiences random loss at the wireless link independently with probability p. In the model, we can equivalently assume that each packet is corrupted independently with probability p. Since the receiver discards corrupted packets and keeps the correctly received packets, the case that a packet is lost is equivalent to the case that a packet is corrupted. Let $\mathbf{q_k}$ denote the kth coded packet transmitted by the sender, or equivalently, the kth coded packet received by the receiver (either innovative or redundant, either correct or corrupted). The time duration between the arrivals of the two consecutive packets (i.e., $\mathbf{q_k}$ and $\mathbf{q_{k+1}}$ for any $k = 1, 2, \ldots$) at the receiver is also τ_0.



(a) Zero or one packet corruption in one group.



(b) Multiple packet corruptions in one group.

Figure 3.5: Examples of groups of packets with different number of corrupted packets.

3.3.1 System Model for $R = 0$ Case

A *group* of packets is referred to as a set of packets which can be decoded together. Figs. 3.5 (a) and (b) show some samples of group of packets. If there are no corrupted

packet arrivals at the receiver, each packet can be decoded immediately when it arrives at the receiver, with decoding delay T_1. There will be only one packet in each group in this case. If a corrupted packet arrives at the receiver, the consecutive innovative packets cannot be decoded until a redundant packet arrives at the receiver. In this case, there will be several packets in one group.

We define the *expected decoding time* of a packet $\mathbf{q_x}$ as the difference between the arrival time of $\mathbf{q_x}$ and the time packet $\mathbf{q_x}$ is expected to be decoded if no further packet corruptions happen. Let Γ_x denote the expected decoding time of packet $\mathbf{q_x}$. As shown in Fig. 3.5 (a), if there are no corrupted packet arrivals before $\mathbf{q_x}$ and $\mathbf{q_x}$ is correctly received, Γ_x is equal to 0 because $\mathbf{q_x}$ can be decoded immediately when it arrives at the receiver. If $\mathbf{q_y}$ is corrupted, the receiver receives $\mathbf{q_{y+1}}$ τ_0 seconds later, then it sends back an ACK indicating that $RealGap = 1$, which will arrive at the sender in T_2 seconds. Then, the sender will transmit an redundant packet and set $ExpGap = 1$. Finally, the redundant packet will arrive at the receiver in T_1 seconds. In summary, the receiver can decode $\mathbf{q_y}$, until it receives the redundant packet which is expected to arrive $\tau_0 + T_2 + T_1$ (i.e., $RTT + \tau_0$) seconds after the arrival of $\mathbf{q_y}$. Thus, the expected decoding time of $\mathbf{q_y}$ (Γ_y) is delayed up to $RTT + \tau_0$. We use K to denote the ratio of RTT over τ_0. We approximate this value with $\lceil RTT/\tau_0 \rceil$. Since RTT is much larger than τ_0, the error of approximation is negligible. After the receiver receives the redundant packet, it sends back another ACK, to set the *RealGap* and *ExpGap* to be 0 at the sender.

As another example, consider Fig. 3.5 (b). When a corrupted packet $\mathbf{q_x}$ arrives

at the receiver, *RealGap* is set to be 1 after $\tau_0 + T_2$ seconds, and redundant packet is transmitted and $ExpGap = 1$. The redundant packet is expected to arrive at the receiver $RTT + \tau_0 = (K+1)\tau_0$ seconds after $\mathbf{q_x}$ arrives at the receiver and the expected decoding time of $\mathbf{q_x}$ is $RTT + \tau_0 = (K+1)\tau_0$. When a correctly received packet (e.g., packet $\mathbf{q_{x+1}}$ or $\mathbf{q_{x+2}}$) arrives, the expected decoding time will be decreased by τ_0. If there is another corrupted packet $\mathbf{q_{x+5}}$, $RealGap = 2$ and $ExpGap = 2$ at the sender and another redundant packet is transmitted. Thus, the expected decoding time of $\mathbf{q_{x+5}}$ becomes $(K+1)\tau_0$ again. For any k, the expected decoding time $\Gamma_k = 0, \tau_0, 2\tau_0, \ldots, (K+1)\tau_0$.

For the sequence of the packets $\mathbf{q_1}, \ldots, \mathbf{q_{k+1}}$, the sequence of expected decoding times is $\Gamma_1, \ldots, \Gamma_{k+1}$. The expected decoding time Γ_{k+1} is determined based on Γ_k and whether $\mathbf{q_{k+1}}$ is corrupted or correct. Thus Γ_{k+1} is independent of $\Gamma_{k-1}, \ldots, \Gamma_1$

$$P(\Gamma_{k+1} \mid \Gamma_k, \Gamma_{k-1}, \ldots, \Gamma_1) = P(\Gamma_{k+1} \mid \Gamma_k), \qquad \Gamma_k = 0, \tau_0, 2\tau_0, \ldots, (K+1)\tau_0. \quad (3.4)$$

We use S_k to denote the ratio of Γ_k to τ_0 (i.e., $S_k = \frac{\Gamma_k}{\tau_0}$). Packet $\mathbf{q_k}$ is expected to be decoded right after S_k following packets arrives at the receiver, if none of the S_k packets is corrupted. We also have

$$P(S_{k+1} \mid S_k, S_{k-1}, \ldots, S_1) = P(S_{k+1} \mid S_k), \qquad S_k = 0, 1, \ldots, K+1. \quad (3.5)$$

The sequence $S_1, S_2, \ldots$ constructs a discrete Markov chain. If a corrupted packet arrives, the expected decoding time becomes $(K+1)\tau_0$ and the state becomes $K+1$. If a packet is correctly received, the expected decoding time is decreased by τ_0 and the state decreases by 1. Fig. 3.6 shows the state transition of the Markov chain. Let $\mathbf{M}$ denote the transition

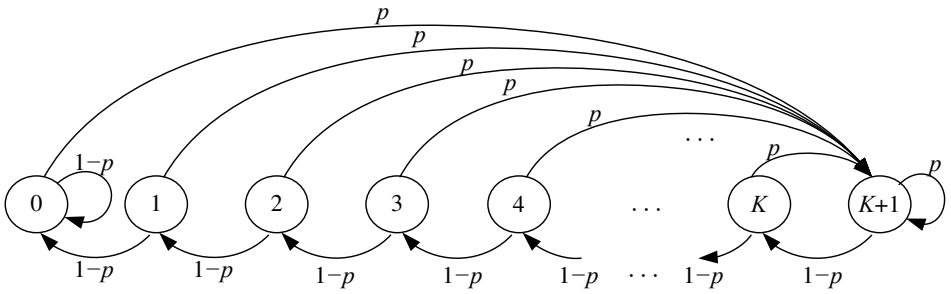


Figure 3.6: Markov state transition for redundant factor $R = 0$.

probability matrix of the Markov chain,

$$\mathbf{M} = \begin{pmatrix} m_{00} & m_{01} & \dots & m_{0(K+1)} \\ m_{10} & m_{11} & \dots & m_{1(K+1)} \\ \vdots & \vdots & \ddots & \vdots \\ m_{(K+1)0} & m_{(K+1)1} & \dots & m_{(K+1)(K+1)} \end{pmatrix}$$

$$= \begin{pmatrix} 1-p & 0 & \dots & 0 & p \\ 1-p & 0 & \dots & 0 & p \\ 0 & 1-p & \dots & 0 & p \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1-p & p \end{pmatrix}, \tag{3.6}$$

where $m_{j(K+1)} = p$ $(j = 0, 1, \dots, K+1)$, $m_{(j+1)j} = 1-p$ $(j = 0, 1, \dots, K)$, $m_{00} = 1-p$ and all the other entries of $\mathbf{M}$ are 0. When the Markov chain enters state 0, all the packets can be decoded. If there are i packets in one group, there will be i transitions between two consecutive visits of state 0 in the Markov chain accordingly. Let P_i denote the probability that there are i packets in one group (or equivalently, the probability that there are i transitions between two consecutive visits of state 0 in the Markov chain).

We determine P_i as follows:

$\mathbf{M}$ can be divided into four submatrices

$$\mathbf{M} = \left(\begin{array}{c|c} \mathbf{M}_{00} & \mathbf{M}_{01} \\ \hline \mathbf{M}_{10} & \mathbf{M}_{11} \end{array}\right) \tag{3.7}$$

$$= \left(\begin{array}{c|ccc} m_{00} & m_{01} & \dots & m_{0(K+1)} \\ \hline m_{10} & m_{11} & \dots & m_{1(K+1)} \\ \vdots & \vdots & \ddots & \vdots \\ m_{(K+1)0} & m_{(K+1)1} & \dots & m_{(K+1)(K+1)} \end{array}\right).$$

We have $P_1 = m_{00}$ and $P_2 = \sum_{j=1}^{K+1} m_{0j}m_{j0} = \mathbf{M}_{01}\mathbf{M}_{10}$, respectively. In general, we have

$$\begin{aligned} P_i &= \sum_{j_1=1}^{K+1}\sum_{j_2=1}^{K+1}\dots\sum_{j_{i-1}=1}^{K+1} m_{0j_1}m_{j_1j_2}\dots m_{j_{i-2}j_{i-1}}m_{j_{i-1}0} \\ &= \mathbf{M}_{01}\mathbf{M}_{11}^{i-2}\mathbf{M}_{10}, \qquad i \geq 3. \end{aligned} \tag{3.8}$$

The decoding delay of the ith (last) packet in the group with i packets is equal to the propagation delay T_1. The decoding delay of the $(i-1)$th packet in the group is $T_1 + \tau_0$. The decoding delay of the $(i-j)$th $(j = 0, 1, \dots, i-1)$ the packet in the group is $T_1 + j\tau_0$. The average decoding delay of a group with i packets is

$$\tau_i = \frac{\sum_{j=0}^{i-1}(T_1 + j\tau_0)}{i} = T_1 + (i-1)\tau_0/2, \qquad i \geq 1. \tag{3.9}$$

Let t denote the total transmission time, $\tau_T(t)$ denote the total decoding delay of all the packets during time t and $N(t)$ denote the total number of packets transmitted

during time t. The average decoding delay can be computed as

$$\overline{\tau} = \lim_{t\to\infty} \frac{\tau_T(t)}{N(t)}. \tag{3.10}$$

We use $n_i(t)$ to denote the number of groups with i packets in total during time t. We have $N(t) = \sum_{i=1}^{\infty} i n_i(t)$ and $\tau_T(t) = \sum_{i=1}^{\infty} i n_i(t)\tau_i$. Thus,

$$\overline{\tau} = \lim_{t\to\infty} \frac{\sum_{i=1}^{\infty} i n_i(t)\tau_i}{\sum_{i=1}^{\infty} i n_i(t)}. \tag{3.11}$$

The nominator and denominator of (3.11) can be divided by $\sum_{j=1}^{\infty} n_j(t)$ at the same time,

$$\begin{aligned}\overline{\tau} &= \lim_{t\to\infty} \frac{\sum_{i=1}^{\infty} i \left(n_i(t) / \left(\sum_{j=1}^{\infty} n_j(t) \right) \right) \tau_i}{\sum_{i=1}^{\infty} i \left(n_i(t) / \left(\sum_{j=1}^{\infty} n_j(t) \right) \right)} \\ &= \frac{\sum_{i=1}^{\infty} i \left(\lim_{t\to\infty} n_i(t) / \left(\sum_{j=1}^{\infty} n_j(t) \right) \right) \tau_i}{\sum_{i=1}^{\infty} i \left(\lim_{t\to\infty} n_i(t) / \left(\sum_{j=1}^{\infty} n_j(t) \right) \right)}. \end{aligned} \tag{3.12}$$

For large values of t, by the law of large numbers, the limit of $n_i(t)/\sum_{j=1}^{\infty} n_j(t)$ approaches P_i, the probability that there are i packets in one group. Thus,

$$\overline{\tau} = \frac{\sum_{i=1}^{\infty} i P_i \tau_i}{\sum_{i=1}^{\infty} i P_i}. \tag{3.13}$$

Therefore, the average decoding delay can be computed.

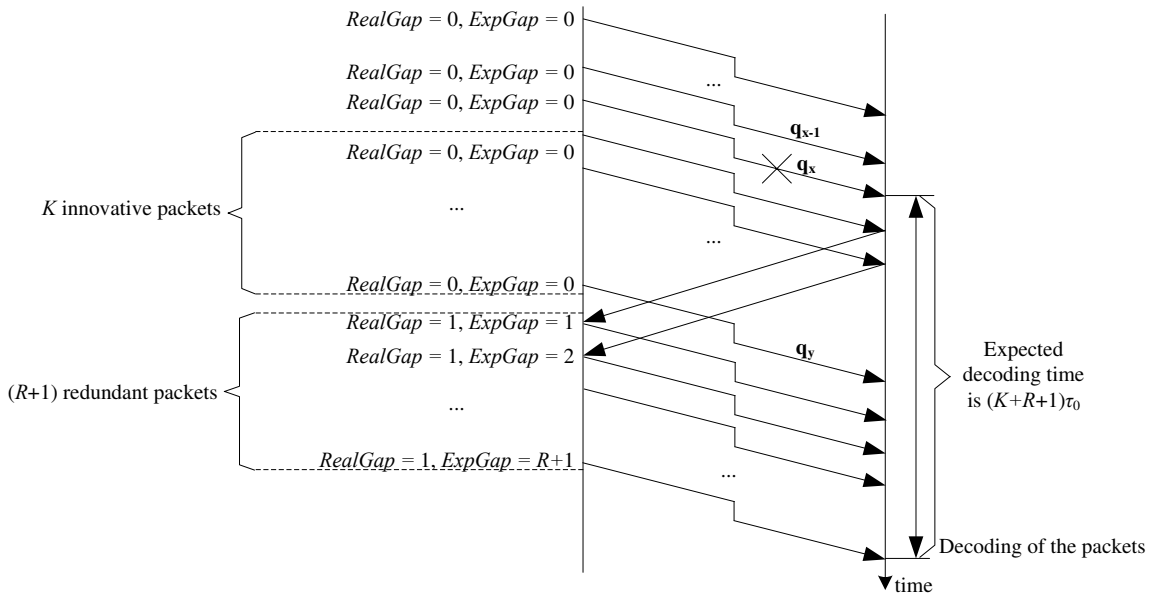


Figure 3.7: Example of the expected decoding time with redundant factor $R > 0$.

3.3.2 System Model for the $R > 0$ Case

When the redundant factor R is greater than zero, the source transmits $R+1$ redundant packets if a packet loss occurs and the real gap is zero. In this case, whether a packet can be decoded depends on many factors including the number of redundant packets transmitted before the transmission of the packet and which packets have been lost before the transmission of the packet by the sender. It is complicated to establish an accurate model. We develop a method to estimate the delay performance for $R > 0$ case in this subsection.

Fig. 3.7 shows an example of $R > 0$ case. At the beginning, $RealGap = 0$, $ExpGap = 0$, and no packets are corrupted. $\mathbf{q_{x-1}}$ can be decoded as soon as it arrives at the receiver, the expected decoding time of $\mathbf{q_{x-1}}$ (Γ_{x-1}) is 0. If the receiver receives one corrupted

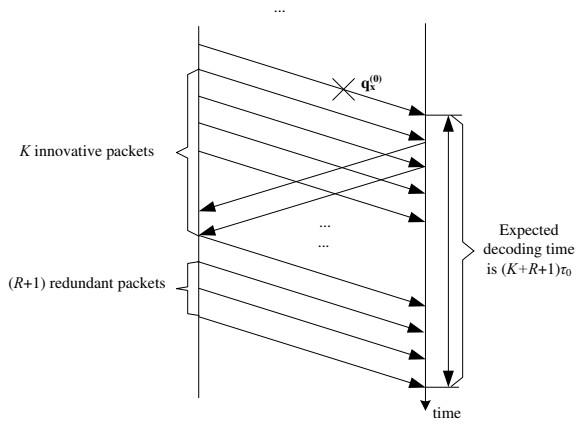


Figure 3.8: Estimated Markov state transition for redundant factor $R > 0$.

packet $\mathbf{q_x}$, it sends an ACK indicating $RealGap = 1$. Then, the sender is expected to transmit $R+1$ redundant packets, until $RealGap = 1$ and $ExpGap = R+1$. The packets are expected to be decoded when the redundant packets arrive at the receiver. Thus, the expected decoding time of $\mathbf{q_x}$ (Γ_x) is $(K + R + 1)\tau_0$. After that, the expected decoding time will be decreased by τ_0 each time a coded packet is correctly received by the receiver. In the case that $R > 0$, Γ_{k+1} is determined based on Γ_k and whether $\mathbf{q_{k+1}}$ is corrupted or not in the following conditions:

(a) If $\Gamma_k = 0$ and $\mathbf{q_{k+1}}$ is corrupted, $\Gamma_{k+1} = (K + R + 1)\tau_0$;

(b) If $\Gamma_k = 0$ and $\mathbf{q_{k+1}}$ is not corrupted, $\Gamma_{k+1} = 0$;

(c) If $\Gamma_k > 0$ and $\mathbf{q_{k+1}}$ is not corrupted, $\Gamma_{k+1} = \Gamma_k - \tau_0$.

Γ_{k+1} is not only determined based on Γ_k and whether $\mathbf{q_{k+1}}$ is corrupted or not in the following condition:

(d) If $\Gamma_k > 0$ and $\mathbf{q_{k+1}}$ is corrupted.

The occurrence of conditions (a) (b) and (c) is much more than that of condi-

tion (d) when p is not large (e.g., $p \leq 0.1$). We *approximately* assume that Γ_{k+1} is determined based on Γ_k and whether $\mathbf{q_{k+1}}$ is corrupted or not. Therefore, we have $P(\Gamma_{k+1} \mid \Gamma_k, \Gamma_{k-1}, \ldots, \Gamma_1) = P(\Gamma_{k+1} \mid \Gamma_k)$ and consequently $P(S_{k+1} \mid S_k, S_{k-1}, \ldots, S_1) = P(S_{k+1} \mid S_k)$.

As discussed in Section 3.3.1, the state is the expected decoding time divided by τ_0. Therefore, in the estimated Markov chain model, the state space is $\{0, 1, 2, \ldots, K+R+1\}$. The state 0 corresponds to the packets decoded immediately when they arrive at the receiver. The state $K + R + 1$ corresponds to the corrupted packet causing the expected decoding time delayed up to $(K + R + 1)\tau_0$. The transition probability from state 0 to state $K + R + 1$ is equal to p.

Let p_e denote the transition probability from any state $j = 1, 2, \ldots, K+R+1$ to state $K + R + 1$. p_e is a value smaller than p. The reason is as follows: Due to the redundancy introduced by R, if a packet is corrupted, the expected decoding time may not become $(K+R+1)\tau_0$ and the state may not transit into state $K+R+1$. For example, in Fig. 3.7, if $\mathbf{q_y}$ is also corrupted, $\mathbf{q_y}$ can be decoded when the receiver receives the redundant packets, and the expected decoding time of $\mathbf{q_y}$ does not become $(K + R + 1)\tau_0$. Thus, transition probability from state j $(j = 1, 2, \ldots, K + R + 1)$ to state $K + R + 1$ is smaller than p.

Let η denote the probability that there are $K + R + 2$ packets in one group. It is also the probability that there are $K + R + 2$ transitions between two consecutive visits of state 0. As shown in the example of Fig. 3.7, the first packet $\mathbf{q_x}$ of the group is corrupted. Then, after receiving packet $\mathbf{q_x}$ at the receiver, $K+R+1$ packets, including K innovative

packets and $R+1$ redundant packets, will arrive. If up to R of the $K+R+1$ packets are corrupted, these packets can still be decoded when the last redundant packet arrives $(K+R+1)\tau_0$ seconds after the arrival of $\mathbf{q_x}$. In this situation, there will be $K+R+2$ packets in the group. The probability that $\mathbf{q_x}$ is corrupted is p and the probability that up to R of the $K+R+1$ following packets are corrupted is $\sum_{i=0}^{R}\binom{K+R+1}{i}(1-p)^{K+R+1-i}p^i$. Thus, the probability that there are $K+R+2$ packets in one group can be computed as

$$\eta = p\left(\sum_{i=0}^{R}\binom{K+R+1}{i}(1-p)^{K+R+1-i}p^i\right). \tag{3.14}$$

When there are $K+R+2$ transitions between two consecutive visits of state 0, the only state transition pattern is $0 \to K+R+1 \to K+R \to \ldots \to 1 \to 0$. Thus, η can also be computed as

$$\eta = p(1-p_e)^{K+R+1}. \tag{3.15}$$

We can compute the value of p_e using equations (3.14) and (3.15) as

$$p_e = 1 - \left(\sum_{i=0}^{R}\binom{K+R+1}{i}(1-p)^{K+R+1-i}p^i\right)^{\frac{1}{K+R+1}}. \tag{3.16}$$

We now show that $p_e < p$ as follows

$$\begin{aligned} p_e &= 1 - \left(\sum_{i=0}^{R}\binom{K+R+1}{i}(1-p)^{K+R+1-i}p^i\right)^{\frac{1}{K+R+1}} \\ &= 1 - \left((1-p) + \sum_{i=1}^{R}\binom{K+R+1}{i}(1-p)^{K+R+1-i}p^i\right)^{\frac{1}{K+R+1}} \\ &< 1 - (1-p)^{\frac{1}{K+R+1}} < 1 - (1-p) = p. \end{aligned} \tag{3.17}$$

Finally, given the value of p_e, we can write the transition probability matrix $\mathbf{M}$ based

on Fig. 3.8. Following the steps from equations (3.7) - (3.13), the estimated decoding delay can be computed similarly.

3.4 Performance Analysis

In this section, we analyze the performance of TCP VON via ns-2 simulations [48]. We first validate our analytical models by comparing the analytical and simulation results. Then, we compare the throughput and delay performance of TCP VON and TCP AR-QNC [43, 44]. Unless stated otherwise, the packet size L is set to 1250 bytes. The results are average on 50 simulation runs and each simulation lasts 100 seconds.

3.4.1 Performance Validation

We first consider a single-hop wireless network with one sender and one receiver. C is the capacity of the wireless link. Fig. 3.9 shows the comparison between the analytical and simulation results for $R = 0$ under different loss probabilities p. The propagation delay from the sender to the receiver T_0 is 5 ms. The packet loss probability varies from 0 to 0.1 with step size 0.01. The results show that when the loss probability increases, the average decoding delay increases. In addition, when C increases, the average decoding delay increases as well. This is because when C increases, $\tau_0 = L/C$ decreases and the number of packets transmitted within one RTT is increased. Thus, the number of transmitted but unacknowledged packets becomes larger. In this case, it is more likely that there are packet losses among the transmitted but unacknowledged packets. This

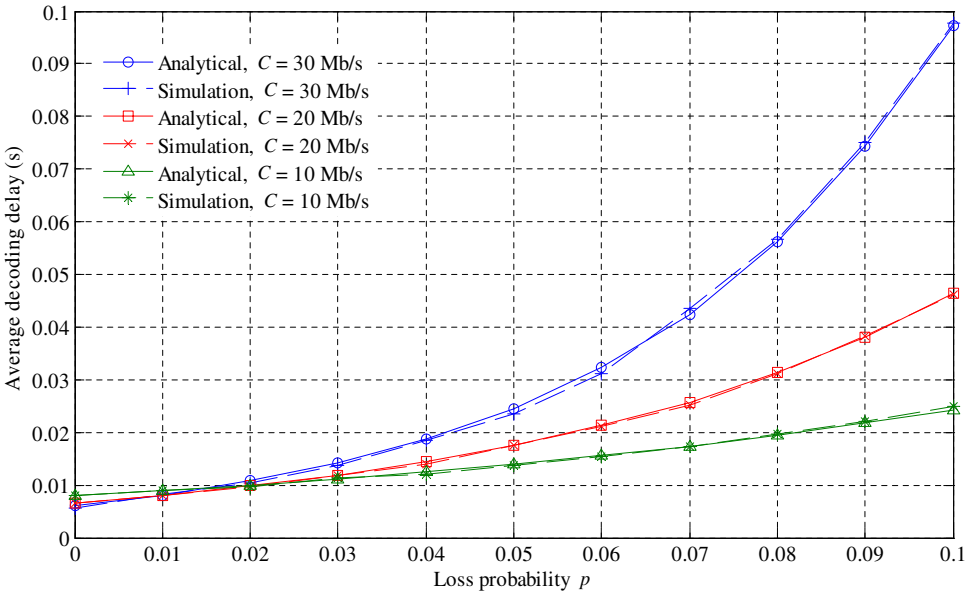


Figure 3.9: Analytical and simulation results of average decoding delay for different values of loss probability p for $R = 0$, $T_0 = 5$ ms, and $L = 1250$ bytes.

accordingly increases the decoding delay. Fig. 3.10 shows the comparison between the analytical and simulation results under different values of R where T_0 is 5 ms and p is 0.1. The results show that a higher value of R decreases the decoding delay.

In the analytical model, we assume that there is no cross traffic, RTT is a constant, and $\Delta = 0$. In the simulation, the topology is single-hop topology and there is only one TCP session. Thus, the round trip time does not change much and Δ is small in the simulation, which is quite close to the assumptions in the analytical model. Therefore, the simulation results match the analytical results in Figs. 3.9 and 3.10.

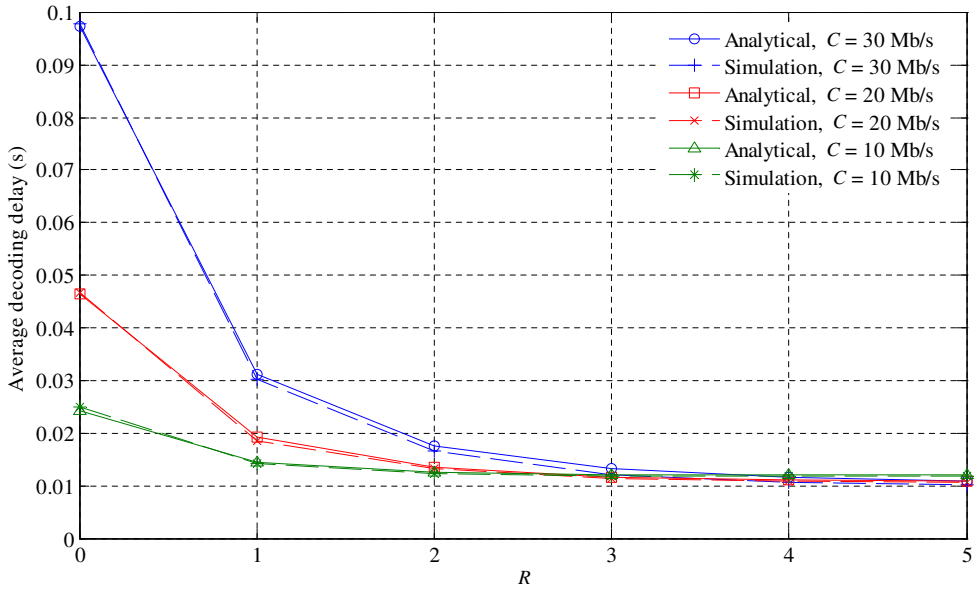


Figure 3.10: Analytical and simulation results of average decoding delay for different values of redundant factor R for $p = 0.1$, $T_0 = 5$ ms, and $L = 1250$ bytes.

3.4.2 Performance Comparison

Next, we compare the performance of TCP ARQNC [43, 44] and TCP VON. For TCP ARQNC, the coded packets are transmitted in a generation by generation fashion. There are N_0 native packets in one generation and each coded packet is a linear combination of the N_0 native packets. For each generation, the sender first transmits $N_G = N_0 + R_0$ coded packets, where R_0 is the redundant factor. The receiver can decode all the native packets if N_0 or more coded packets are successfully received at the receiver. If more than R_0 packets are lost, the sender has to transmit more linear combinations until the receiver receives enough independent coded packets. We use two different network topologies for performance comparison described as follows.

The multihop tandem topology is shown in Fig. 3.11. The first two hops are wired

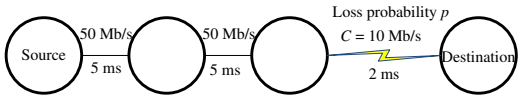


Figure 3.11: Simulation topology 1: Multihop tandem topology.

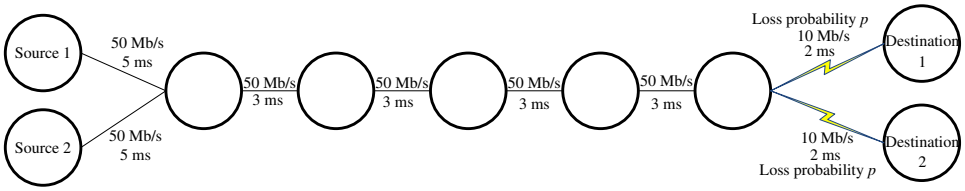


Figure 3.12: Simulation topology 2: Cross traffic topology.

links with capacity of 50 Mb/s, and propagation delay of 5 ms. The last hop is wireless link with capacity of 10 Mb/s and the propagation delay of 2 ms. One TCP session is established from Source to Destination.

The cross traffic topology is shown in Fig. 3.12. All the links are wired links except the last link to the destinations which are wireless links. Each wired link has a capacity of 50 Mb/s and propagation delay of 3 ms. The wireless links have a capacity of 10 Mb/s and propagation delay of 2 ms. We assume that both of the links have random packet loss probability p. Two TCP sessions are established; one from Source 1 to Destination 1 and the other one from Source 2 to Destination 2.

We investigate the joint throughput and delay performance of different TCP schemes using ns-2 simulation. Figs. 3.13 and 3.14 show the throughput and delay performance of TCP VON and TCP ARQNC when p changes from 0 to 0.1 with step 0.01 in multihop tandem topology and cross traffic topology, respectively. For TCP ARQNC, When N_0 and R_0 are small, although the delay is low, the throughput performance is poor. When N_0 and R_0 are increased, the throughput is improved but the delay performance degrades.

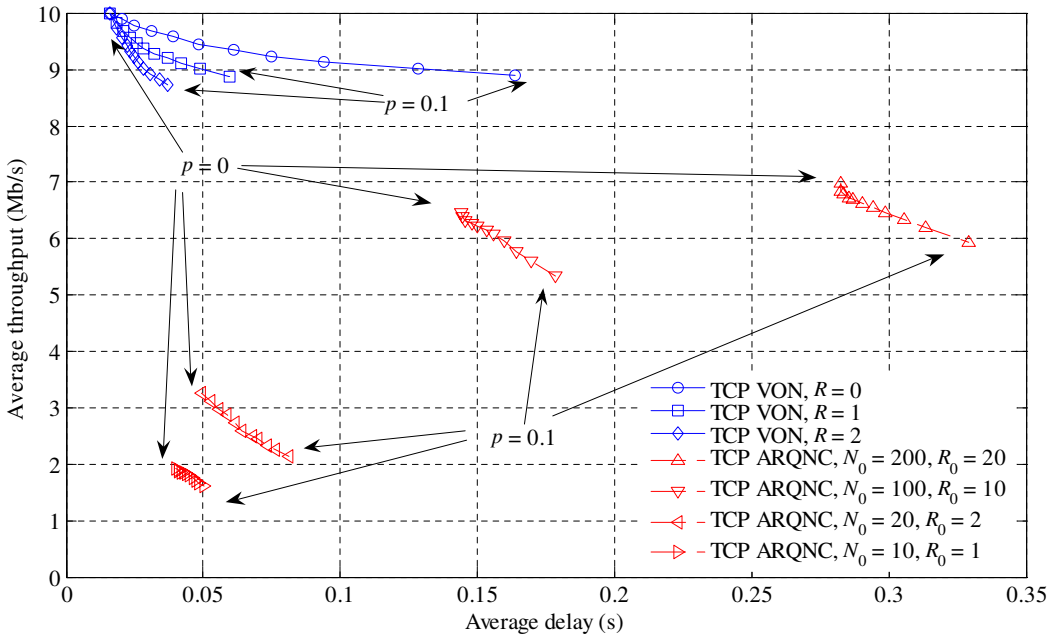


Figure 3.13: Performance comparison for multihop tandem topology under different values of loss probability p.

This is because the sender starts to transmit the next generation of the packets only after being acknowledged that the current generation of the packets have been decoded. For each generation, the sender is idle when it has transmitted $N_0 + R_0$ packets but the ACK indicating the decoding of the generation of packets has not been received. Smaller generation size (i.e., N_0 and R_0 are small) leads to more frequent idle of the sender, causing the waste of bandwidth and throughput. When the generation size is large (i.e., N_0 and R_0 are large), the receiver needs to wait for a long time to decode a generation of the packets, causing large decoding delay. For TCP VON, its throughput outperforms that of TCP ARQNC. When p is large ($p > 0.05$), we can moderately increase the redundant factor R to decrease the decoding delay while maintaining the throughput performance.

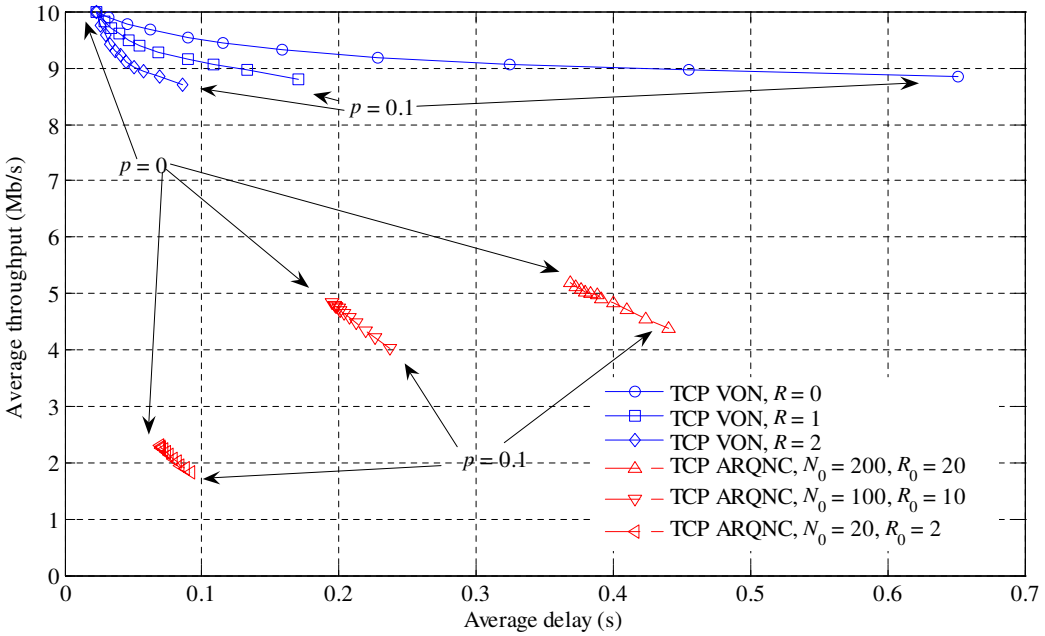


Figure 3.14: Performance comparison for cross traffic topology under different values of loss probability p.

Fig. 3.15 shows the throughput and delay performance of TCP VON and TCP ARQNC when $p = 0.05$. For TCP VON, the redundant factor R varies from 0 to 5 with step 1. For TCP ARQNC, we choose different values of N_0 and R_0. For N_0 equals to 5000, 500, and 50, we vary R_0 from 0 to 1000 with step 500, from 0 to 100 with step 50, and from 0 to 10 with step 5, respectively. The results show that for TCP VON, we can moderately increase R to decrease the decoding delay. However, if we choose a high value of R (e.g., $R \geq 4$), the throughput performance degrades without giving acceptable delay improvement. This is because the decoding delay is approaching the minimum value (the end-to-end delay) and it cannot be further improved even we continuous to increase R. If we choose a proper value of R (e.g., $R = 2$), both the throughput and delay performance of TCP VON are better than that of TCP ARQNC.

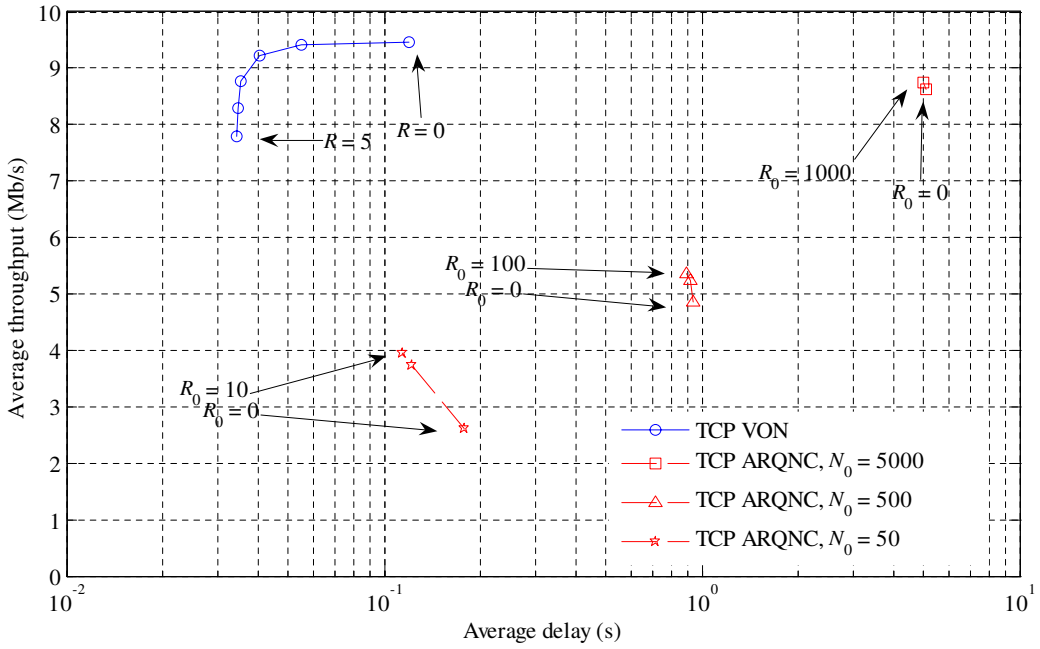


Figure 3.15: Performance comparison for cross traffic topology under different values of redundant factor R.

3.5 Summary

In this chapter, we proposed TCP VON, a new mechanism that incorporates online network coding into TCP, which can be efficiently applied in mixed wired and wireless networks. The scheme introduces minor changes to the current protocol stack only at the sender and the receiver. TCP VON mainly includes two mechanisms, congestion control and online network coding control. Congestion control is extended from TCP Vegas so that congestion loss becomes rare and all the packet losses can be regarded as random packet loss. In the online network coding control, the sender transmits redundant coded packets when packet losses are indicated from ACK packets, otherwise it transmits innovative packets. As a result, packets can be decoded consecutively instead of generation

by generation and the delay from the sending a packet at the sender to decoding it at the receiver receiver is small, without reducing the throughput performance significantly. In networks with high loss probability, we introduce a redundant factor $R > 0$ so that the sender transmits more redundant packets when packet losses are detected. Then, we established a precise Markov chain model to compute the analytical delay performance of TCP VON for $R = 0$ case. This model has also been extended to a closely approximated model for $R > 0$ case. We also conducted ns-2 simulations to validate the correctness of our analytical models for both $R = 0$ case and $R > 0$ case. Finally, we compared the delay and throughput performance of TCP VON and TCP ARQNC in multihop tandem topology and cross traffic topology. Results showed that TCP VON outperforms TCP ARQNC.

Chapter 4

Conclusions and Future Work

In this chapter, we conclude the thesis by summarizing our contributions. We also suggest several topics for further research.

4.1 Conclusions

This thesis covered several aspects of modeling, analysis and enhancement of TCP. The main results were divided into two chapters. In Chapter 2, we proposed a analytical model to determine the steady state throughput of TCP CUBIC in wireless environment. In Chapter 3, we studied a new mechanism that incorporates online network coding into TCP.

In Chapter 2, we proposed an analytical model to determine the steady state throughput of TCP CUBIC in wireless environment. We considered both congestion loss and random packet loss and established a Markov model. We derived the stationary distribution of the Markov chain and obtained the average throughput based on the stationary distribution. The accuracy of the model was validated via simulation. Based on our proposed model, we evaluated the throughput performance of TCP CUBIC. Results showed

that random packet loss reduces the normalized average throughput more for end-to-end flow with large bandwidth-delay product. In wireless environment, we showed that the throughput of TCP CUBIC can be improved by moderately increasing the window growth factor and the multiplicative decrease factor.

In Chapter 3, we proposed TCP VON, a new mechanism that incorporates online network coding into TCP, which can be efficiently applied in mixed wired and wireless networks. The scheme introduces minor changes to the current protocol stack only at the sender and the receiver. TCP VON mainly includes two mechanisms, congestion control and online network coding control. Congestion control is extended from TCP Vegas so that congestion loss becomes rare and all the packet losses can be regarded as random packet loss. In the online network coding control, the sender transmits redundant coded packets when packet losses are indicated from ACK packets, otherwise it transmits innovative packets. As a result, packets can be decoded consecutively instead of generation by generation and the delay from the sending a packet at the sender to decoding it at the receiver receiver is small, without reducing the throughput performance significantly. In networks with high loss probability, we introduce a redundant factor $R > 0$ so that the sender transmits more redundant packets when packet losses are detected. Then, we established a precise Markov chain model to compute the analytical delay performance of TCP VON for $R = 0$ case. This model has also been extended to a closely approximated model for $R > 0$ case. We also conducted ns-2 simulations to validate the correctness of our analytical models for both $R = 0$ case and $R > 0$ case. Finally, we compared the

delay and throughput performance of TCP VON and TCP ARQNC in multihop tandem topology and cross traffic topology. Results showed that TCP VON outperforms TCP ARQNC.

4.2 Future Work

For future work, we can consider several potential extensions of the current work.

Empirical study of TCP variants and online network coding based TCP in real networks. We consider carrying out empirical study of different TCP variants and network coding based TCP in real networks. By running different applications (e.g., FTP, VoIP, video conferencing), we can measure the throughput and delay performance of different TCP variants, as well as evaluate whether these TCP variants can efficiently support for these applications.

Establishment of congestion control models for network coding based on network utility maximization (NUM). We also consider establishing NUM models for network coding based networks. Suitable utility functions will be chosen to formulate the objective function and constraints related to the characteristics of network topology and the attributes of network will be listed. Then, some transformations of the objective function and constraints will be made in order to convert the problem to a convex optimization problem. Finally, the convex optimization problem will be solved in a distributed manner, which can then be interpreted as a network management schemes, such as congestion control, network price feedback and queue management schemes.

Decoding delay and throughput tradeoff for online network coding. For online network coding, higher values of the redundant factor R will provide better decoding delay performance, while it may deteriorate the throughput efficiency of the end-to-end communication. We are interested in establishing mathematical models to further study the delay throughput tradeoff.

Bibliography

[1] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-down Approach*, 5th ed. Addison-Wesley, 2010.

[2] J. Postel, "Transmission control protocol," *IETF RFC 793*, Sept. 1981.

[3] R. Braden, "Requirements for Internet hosts - communication layers," *RFC 1122*, Oct. 1989.

[4] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *IETF RFC 1323*, May 1992.

[5] TCP congestion control. [Online]. Available: http://condor.depaul.edu/jkristof/technotes/congestion.pdf

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," *IETF RFC 2018*, Oct. 1996.

[7] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," *IETF RFC 3782*, Apr. 2004.

[8] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 83–91, Apr. 2003.

[9] L. Brakmo and L. Peterson, "TCP Vegas: End-to-end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communication*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[10] I. Rhee and L. Xu, "A new TCP-friendly high-speed TCP variant," in *Proc. PFLDNet'05*, Lyon, France, Feb. 2005.

[11] G. D. J. Leith and R. N. Shorten, "Experimental evaluation of Cubic-TCP," in *Proc. PFLDNet'07*, Los Angeles, CA, Feb. 2007.

[12] M. Bateman, S. Bhatti, G. Bigwood, D. Rehunathan, C. Allison, and T. Henderson, "A comparison of TCP behaviour at high speeds using ns-2 and Linux," in *Proc. of Communications and Networking Simulation*, Ottawa, Canada, Apr. 2008.

[13] A. DeSimone, M. C. Chuah, and O.-C. Yue, "Throughput performance of transport-layer protocols over wireless LANs," in *Proc. of IEEE GLOBECOM*, Houston, TX, Dec. 1993.

[14] G. Xylomenos, G. Polyzos, P. Mahonen, and M. Saaranen, "TCP performance issues over wireless links," *IEEE Communications Magazine*, vol. 31, no. 4, pp. 52–58, Apr. 2001.

[15] S. Xu and T. Saadawi, "Performance evaluation of TCP algorithms in multi-hop wireless packet networks," *Wireless Communications and Mobile Computing*, vol. 2, no. 1, pp. 85–100, Feb. 2002.

[16] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Trans. on Networking*, vol. 5, no. 3, pp. 336–350, Jun. 1997.

[17] I. F. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: A new congestion control scheme for satellite IP networks," *IEEE/ACM Trans. on Networking*, vol. 9, no. 3, pp. 307–321, June 2001.

[18] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 7, pp. 1300–1315, July 2002.

[19] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. ICDCS'95*, Vancouver, Canada, May 1995.

[20] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 5, pp. 19–43, Oct. 1997.

[21] H. Balakrishnan, S. Seshan, and R. H. Katz, "Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments," in *Proc. of IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000.

[22] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 216–228, Feb. 2003.

[23] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. of ACM Mobicom*, Rome, Italy, July 2001.

[24] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 4, pp. 747–756, May 2004.

[25] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[26] S. Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. on Information Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[27] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. on Networking*, vol. 11, no. 5, pp. 371–381, Oct. 2003.

[28] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. of 41st Allerton Annual Conference on Communication, Control, and Computing*, Urbana-Champaign, IL, Oct. 2003.

[29] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in the air: Practical wireless network coding," *IEEE/ACM Trans. on Networking*, vol. 16, no. 3, pp. 497–510, June 2008.

[30] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *Proc. of ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[31] D. S. Lun, M. Médard, R. Koetter, and M. Effros, "On coding for reliable communication over packet networks," *Physical Communication*, vol. 1, no. 1, pp. 3–20, Mar. 2008.

[32] Y. Lin, B. Li, and B. Liang, "CodeOR: Opportunistic routing in wireless mesh networks with segmented network coding," in *Proc. of IEEE International Conference on Network Protocols (ICNP)*, Orlando, FL, Oct. 2008.

[33] J. K. Sundararajan, D. Shah, and M. Médard, "ARQ for network coding," in *Proc. of IEEE International Symposium on Information Theory*, Toronto, Canada, Jul. 2008.

[34] ——, "Online network coding for optimal throughput and delay - The three-receiver case," in *Proc. of International Symposium on Information Theory and Its Applications (ISITA)*, Auckland, New Zealand, Dec. 2008.

[35] J. Barros, R. A. Costa, D. Munaretto, and J. Widmer, "Effective delay control in online network coding," in *Proc. of IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009.

[36] D. E. Lucani, M. Médard, and M. Stojanovic, "Online network coding for time-division duplexing," in *Proc. of IEEE GLOBECOM*, Miami, FL, Dec. 2010.

[37] Y. Lin, B. Liang, and B. Li, "SlideOR: Online opportunistic network coding in wireless mesh networks," in *Proc. of IEEE INFOCOM*, San Diego, CA, Mar. 2010.

[38] P. S. David and A. Kumar, "Network coding for TCP throughput enhancement over a multi-hop wireless network," in *Proc. of International Conference on Communication Systems Software and Middleware and Workshops*, Bangalore, India, Jan. 2008.

[39] Z. Liu and S. Jin, "Diagnosing the limitations of network coding at transport layer," in *Proc. of IEEE International Symposium on Wireless Pervasive Computing (ISWPC)*, Modena, Italy, May 2010.

[40] S. Hassayoun, P. Maille, and D. Ros, "On the impact of random losses on TCP performance in coded wireless mesh networks," in *Proc. of IEEE INFOCOM*, San Diego, CA, Mar. 2010.

[41] L. Chen, T. Ho, S. H. Low, M. Chiang, and J. C. Doyle, "Optimization based rate control for multicast with network coding," in *Proc. of IEEE INFOCOM*, Anchorage, AK, May 2007.

[42] H. Seferoglu and A. Markopoulou, "Network coding-aware queue management for unicast flows over coded wireless networks," in *Proc. of IEEE International Symposium on Network Coding (NetCod)*, Toronto, Canada, June 2010.

[43] J. K. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in *Proc. of IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009.

[44] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: Theory and implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, Mar. 2011.

[45] M. Kim, M. Médard, and J. Barros, "Modeling network coded TCP throughput: A simple model and its validation," in *Proc. of International Conference on Performance Evaluation Methodologies and Tools*, Cachan, France, May 2011.

[46] W. Bao and V. Wong, "A model for steady state throughput of TCP CUBIC," in *Proc. of IEEE GLOBECOM*, Miami, FL, Dec. 2010.

[47] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. of ACM SIGCOMM*, Vancouver, Canada, Sept. 1998.

[48] The network simulator - ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/