Congestion Control for Next-Generation Global Internets

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Yuan Gao, M.S.

* * * * *

The Ohio State University

2002

Dissertation Committee:

Professor Jennifer C. Hou, Adviser

Professor Hitay Özbay, Co-adviser

Professor Yuan F. Zheng

Approved by

_____

Co-Adviser

_____

Co-Adviser

Department of Electrical Engineering

ABSTRACT

As the size and application domains of the Internet grow explosively during recent years, several new phenomena have been observed and new research issues have emerged in Internet congestion control.

First, as most Internet continuous media based applications do not support end-to-end resource and congestion control, wide deployment of these applications can have a severe negative impact on self-controlled TCP flows (which constitute the majority of the Internet traffic). Before these applications can be fully deployed on the Internet, effective congestion control mechanisms must be devised to ensure them respond to network congestion in a TCP-friendly manner so as to coexist with TCP flows.

Second, with the proliferation of HTTP applications for document transport, there often exist at a busy server ('hot spot') multiple, concurrent, TCP or UDP connections destined for the same destination host or destination subnet. This distinct scenario gives rise to several issues that the current TCP does not adequately deal with. First, most TCP connections (especially those initiated by HTTP) are short-lived and seldom have enough time to probe, and fully utilize, the available network bandwidth. Second, without coordination, multiple, concurrent connections may compete blindly for network resource with each other, leading to increased packet losses and variation in the aggregate traffic throughput. These issues call for an effective endhost congestion management scheme to coordinate competing connections.

Another phenomenon that has been observed in recent measurement and research of Internet traffic is that network traffic exhibits self-similar and long-range dependent behaviors. In spite of the abundant correlation structure across multiple time scales, little work has been carried out to judiciously exploit the structure (and hence the predictability) of network traffic to better manage network resources for congestion control. Extracting and exploiting the correlation structure at multiple time scales will enable congestion controllers, e.g. router queues that employ active queue management (AQM) to better respond to network dynamics.

Finally, significant research efforts have been made to study and improve the performance of AQM based on feedback control theory. Several analytical models have been proposed to approximate the dynamic additive increase and multiplicative decrease (AIMD) behaviors of TCP in conjunction with AQM. There are, however, several protocol effects that are not considered in these models. As a result, they can not fully characterize the dynamics of TCP and its interaction with AQM. Models that take into account of these effects are needed to devise better AQM schemes based on control theory.

In this dissertation, we address the above congestion control issues for next-generation Internets with the focus on both the transport and the IP layers. Specifically, we address the following problems:

(I) Congestion control of multicast for continuous media applications with the objectives of (weighted) fairness, TCP-friendliness, and scalability.

(II) End-host-based, coordinated congestion control of TCP/UDP traffic to enable connections that traverse the same backbone link to share congestion information and to coordinate among them all the congestion avoidance/control activities.

(III) Exploitation of the correlation structure across multiple time scales (and hence the predictability) of Internet traffic for better AQM scheme and TCP congestion control.

(IV) Incorporation of protocol effects ignored by previous TCP models in an enhanced TCP model. Design of an AQM controller to stabilize the queue at a router based on the enhanced model.

The dissertation is a combination of two synergistic components: design of algorithms/protocols in an analytical framework and their validation with detailed *ns-2* simulation and software system building and experiments in *FreeBSD* on a network testbed.

To My Parents

Jinshui Gao and Yicang Qin

To My Brother

Shang Gao

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my adviser, Professor Jennifer C. Hou, for her guidance and support throughout the course of my research. I owe a great deal to her for the knowledge that I have gained in the last five years in computer communications as well as for the presentation skills that have been proved useful in my career pursuit.

I would also like to thank Professors Hitay Özbay and Professor Yuan F. Zheng for both their help and intellectually stimulating discussion on my research.

A special thanks goes to all the members of DRCL originally at Ohio State University and now at University of Illinois at Urbana-Champaign. Interaction with them both academically and socially has been enriching and enlightening my life. Many intellectually intriguing discussions have broadened my spectrum of knowledge and been very beneficial.

Last but not least, I wish to thank my family for their constant love, support and encouragement, without which this dissertation would not be possible.

VITA

April 7, 1971	Born - P. R. China
1994	B.S. Tsinghua University, Beijing, P. R. China
1997	M.S. Tsinghua University, Beijing, P. R. China
1997-present	Graduate Research Associate, The Ohio State University

PUBLICATIONS

Research Publications

Yuan Gao and Jennifer C. Hou, Stablizing Queue on AQM Routers for TCP Flows Supporting ECN. *Proceedings of IEEE INFOCOM 2003*, San Francisco, USA, April 2003.

Guanghui He, Yuan Gao, and Jennifer C. Hou, A Case for Exploiting Self-Similarity of Network Traffic in TCP Congestion Control. *Proceedings of IEEE ICNP 2002*, Paris, France, November 2002.

Yuan Gao, Guanghui He, and Jennifer C. Hou, On the Exploiting Traffic Predictability in Active Queue Management. *Proceedings of IEEE INFOCOM 2002*, New York City, USA, June 2002.

Charles D. Cranor, Yuan Gao, Theodore Johnson, and Oliver Spatscheck, Gigascope: High Performance Network Monitoring with an SQL Interface. *ACM SIGMOD/PODS 2002*, Madison, Wisconsin, USA, June 2002.

Yuan Gao and Jennifer C. Hou, RACCOOM: A Rate-Based Congestion Control Scheme for Multicasts. *IEEE Transactions on Computers*, to appear.

Yuan Gao and Jennifer C. Hou, RACCOOM: A Rate-Based Congestion Control Scheme for Multicasts. *SPIE Conference on Scalability and Traffic Control*, Denver, Colorado, USA, August 2001.

Yuan Gao, Ye Ge, and Jennifer C. Hou, Reliable Multicasts for Core-based Multicast Routing. *Proceedings of IEEE ICNP 2000*, Osaka, Japan, October 2000.

FIELDS OF STUDY

Major Field: Electrical Engineering

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER 1

Introduction

As the size and application domains of the Internet grow explosively during recent years, several new phenomena have been observed and new research issues have emerged in Internet congestion control.

First, we have observed an explosive growth in transporting multimedia applications on the Internet. Examples include continuous media servers, digital libraries, video conferencing, WWW traffic, and distance learning. As most Internet continuous media based applications do not support end-to-end resource and congestion control, wide deployment of these applications can have a severe negative impact, ranging from starvation of self-controlled TCP flows (which constitute the majority of the Internet traffic) to the potential for congestion collapse. Effective congestion control mechanisms must be devised to ensure these applications respond to network congestion in a TCP-friendly manner so as to coexist with TCP flows.

Second, with the proliferation of HTTP applications for document transport, there often exist at a busy server ('hot spot') multiple, concurrent, TCP or UDP connections destined for the same destination host or destination subnet. This distinct scenario gives rise to several issues that the current TCP does not adequately deal with. First, most TCP connections (especially those initiated by HTTP) are short-lived and seldom have enough time to probe,

and fully utilize, the network bandwidth available under the current TCP congestion control mechanism. This may lead to long response times. Second, without coordination, multiple, concurrent connections may compete blindly for network resource with each other, leading to increased packet loss and variation in the aggregate traffic throughput. These issues call for an effective endhost congestion management scheme to coordinate competing connections.

Another phenomenon that is recently observed as a result of transporting WWW traffic is that network traffic in local and wide area networks exhibits self-similar and long-range dependent behaviors, i.e., the measured time series is bursty across several time scales [96]. Scale-invariant burstiness introduces new complexities into resource control and QoS provisioning. In particular, it implies the existence of concentrated periods of high activity and low activity at a wide range of time scales, which adversely affects traffic control and induces extended periods of either over-utilization or underutilization. On the other hand, the abundant correlation structure across multiple time scales in long range dependent traffic can be judiciously exploited to better better manage network resources for the purpose of congestion control. For example, extracting and exploiting the correlation structure at multiple time scales can enable AQM to better respond to network dynamics.

Finally, significant research efforts have been made to study and improve the performance of AQM. The common approach envisions a network that consists of TCP connections and AQM routers as a dynamic feedback control system, in which AQM routers act as controllers and TCP traffic sources act as plants [47, 64]. Several analytical models have been proposed to approximate the dynamic additive increase and multiplicative decrease (AIMD) behaviors of TCP in conjunction with AQM [47, 64]. Control theory is then applied to analyze and design AQM controllers. In these models, two effects are not

considered. First, the congestion window is not *gradually* decreased at the rate of $\frac{w^2 p}{2}$, but *suddenly* halved upon receipt of congestion indication. Second, the congestion window is halved at most once during one round trip time (RTT). A model that takes into account of these effects is needed to better characterize the protocol interaction so that better AQM schemes based on control theory can be devised.

1.1 Problem Definition and Motivation

The main goal of this thesis is to investigate congestion control issues for next generation Internets, in both the transport layer and the IP layer. Specifically, we address, and develop solution approaches for, the following problems:

(I) Congestion control of multicast for continuous media applications with the objectives of (weighted) fairness, TCP-friendliness, and scalability.

(II) End-host-based, coordinated congestion control of TCP/UDP traffic to enable connections that traverse the same backbone link to share congestion information and to coordinate among them all the congestion avoidance/control activities.

(III) Exploitation of the correlation structure across multiple time scales and the predictability of Internet traffic for better AQM design.

(IV) Incorporation of protocol effects ignored in previous TCP models in an enhanced TCP model. Design of an AQM controller to stabilize the queue at a router based on this enhanced model.

1.2 Contributions of the Dissertation

The contributions of this thesis include the following four parts:

Multicast congestion control We design and evaluate a rate-based congestion control scheme for multicasts for continuous media multicast applications, with the objectives of TCP-friendliness, scalability, and adaptability to membership and traffic change. In particular, we study how to achieve weighted fairness among competing multicast connections based on results obtained from the feedback control theory.

End-host congestion control We investigate the issue of endpoint, coordinated congestion management and propose a coordinated congestion management scheme for busy Internet servers. The basic idea is to identify and group connections that may traverse the same backbone link, to enable them to share congestion information, and to coordinate among them all the congestion avoidance/control activities. The size of a group can be dynamically adjusted so as to magnify the benefits of congestion management. In addition, we propose to take into account non-responsive UDP connections and "bundles" them into a virtual connection that is subject to TCP-like congestion control. We implement the proposed approach in *FreeBSD*, with the minimal impact to the current protocol stack, and empirically evaluate its performance against other existing proposals.

Exploiting traffic predictability in active queue management We investigate how to take advantage of the predictability of Internet traffic to better design AQM controllers at routers. We propose to exploit the correlation structure across multiple time scales on-line and predict the future traffic to better stabilize the queue length at a router (which in turns will lead to less variable and more predictable end-to-end packet delays). This is achieved by figuring in in the calculation of packet dropping probability the prediction results.

An AQM scheme to stabilize queue for TCP flows supporting ECN We present an analytical TCP model that takes into account of several protocol issues that were ignored in the other existing models (such as those in [47, 64]), i.e., (i) the congestion window is not gradually decreased at the rate of $\frac{w^2p}{2}$, but suddenly halved upon receipt of congestion indication and (ii) the congestion window is halved at most once during one RTT. We also include the delayed ACK option in the model. We show that this enhanced model characterizes the TCP dynamics and its interaction with AQM more faithfully. With the use of state feedback control theory, we then design an AQM controller to stabilize the queue length at routers based on this enhanced model. The performance of the new controller is shown, via *ns-2* simulation, to outperform several other schemes under a variety of network scenarios and traffic loads, in terms of fluctuation in the queue length, link utilization, and packet loss ratio.

1.3 Structure of the Dissertation

The structure of the thesis is as follows. In Chapter 2, we present background material that pertains to the issues addressed in the dissertation. In Chapter 3 and 4, we focus on congestion control issues in the transport layer: Chapter 3 introduces a rate-based multicast congestion control scheme for continuous media applications, while Chapter 4 presents an end-host congestion control scheme for busy Internet servers. Chapter 5 and 6 cover congestion control issues in the IP layer. In Chapter 5, we describe a prediction-based AQM scheme that exploits the long range dependent characteristics of Internet traffic. In Chapter 6, we present an enhanced TCP model based on which we devise an AQM scheme for TCP flows with ECN support. All the schemes proposed in the thesis are

validate/evaluated by simulations, and some of them by empirical implementation and experiment in *FreeBSD/Linux* kernels. The thesis concludes in Chapter 7 with a summary of our contributions and a list of research avenues for future work.

CHAPTER 2

Background Material

In this chapter, we provide background material that pertains to the problems addressed in the thesis.

2.1 Network Model

The network model considered is a packet-switched network that provides unreliable, best-effort service [77]. The peer IP entities in the network layer fragments data into packets that can fit into datagrams, encapsulates each of them with a protocol header, and sends them to the network. The protocol header of each packet contains, among other control information, the IP address of the destination host. When a data packet arrives at a router, the IP entity at the router retrieves the header information, and determines on which outgoing interface the packet should be forwarded. The decision is made based on the destination address (and sometimes the source address) retrieved from the header and the routing table (which is updated periodically and maintained by an underlying routing protocol). Each data packet is forwarded by routers on a hop-by-hop basis toward the destination.

2.2 Routing in Internet

In a packet-switched network, the primary function of routing is to select the best route to deliver data packets from a source to a destination, with the objectives of maximizing the throughput and minimizing the packet delay. To this end, each router must know the (partial or complete) network topology and the availability and status of resources (link bandwidth and buffer at router) in the network. Periodic exchange of the topology information is governed by the routing protocols.

2.2.1 Uniticast Routing

The well-known unicast routing protocols include distance vector routing protocol, e.g. routing information protocol (RIP) [61], and link state routing protocol, e.g. open shortest path first routing protocol (OSPF) [66]. They are designed to periodically exchange the topology information among routers in an autonomous system. With the topology information, each router then calculates the shortest path to each subnet by either the Dijkstra or the Bellman-Ford algorithms, and updates its routing table (usually indexed by possible destination hosts). Upon receipt of a data packet, a router looks up the routing table to determine an appropriate outgoing interface on which the packet is forwarded.

2.2.2 Multicast Routing

The conventional data transport paradigm on the Internet is unicast, i.e. one-to-one communication. Since early 90's, multicast has been proposed as an efficient way of disseminating data from a source to multiple members in a multicast group. Instead of sending a separate copy of the data to each individual group member, the source simply sends a single copy to all the members. An underlying *multicast routing* protocol determines, with

respect to certain optimization objectives, a *multicast tree* connecting the source(s) and the group members. Data generated by the source(s) flows through the multicast tree, traversing each tree edge exactly once. When group members join or leave a multicast group, the multicast tree is dynamically reconfigured.

The approaches for constructing multicast trees can be classified into two categories: (i) the source-based multicast tree approach, e.g., Distance-Vector Multicast Routing Protocol (DVMRP) [78], Protocol Independent Multicast Dense Mode (PIM-DM) [25], and Multicast extensions to Open Shortest Path First Protocol (MOSPF) [66]; and (ii) the core-based multicast tree approach, e.g., the Core Based Tree (CBT) protocol [10] and the Protocol Independent Multicast Sparse Mode (PIM-SM) [25, 97]. In the former (source-based) approach, a tree rooted at a source node is constructed and connected to every member in the multicast group. Data packets originating from the source node are sent to all the destination nodes via the links of a multicast tree. In the latter (core-based) approach, one node for each group is selected as the core [10] (or termed in [25, 97] as a rendezvous point) for the group. A tree rooted at the core is then constructed to span all the group members. Data packets flow from any source to its parent and children. A node forwards packets to its parent and children except the one from which data packets arrive.

From the viewpoint of network management, it is more desirable for a multicast group to maintain only one multicast tree. This reduces the number of states each on-tree node has to keep, improves scalability, and makes handling of dynamic group membership changes more tractable. The price core-based multicast routing has to pay is, however, that the resulting multicast tree may be sub-optimal with respect to some source(s).

2.3 Reliable Transport

In the current Internet, packets are delivered on a best-effort basis, and may get delayed, discarded, duplicated, or delivered out of order. To deal with packet retransmission and reordering and to present reliable byte streams to applications, an end-host transport layer is laid on top of the network layer (IP layer) and is responsible for reliable and in-sequence transport of data.

2.3.1 Reliable Transport in Unicast

TCP [90] is an example transport layer that provides reliable transport for unicast. To ensure reliable in-order data transmission, TCP uses the sliding-window algorithm [88] for error control. The sliding window algorithm works as follows. The sender assigns a sequence number to each data byte to be sent. When a packet is transmitted, the sequence number of the first byte in the packet and the packet length are recorded in the packet header. The receiver can then determine the sequence number of each data byte in a packet received. A sender maintains three variables: (i) the sending window size (*SWS*, which is the maximum number of outstanding (unacknowledged) data bytes that the sender can transmit at any time without acknowledgment); (ii) the sequence number of the last acknowledgment received (*LAR*); and (iii) the sequence number of the last byte sent (*LFS*). The sender ensures that the condition $(LFS - LAR) \leq SWS$ holds all the times.

When a receiver receives a data packet, it sends back to the sender an acknowledgment (ACK) packet that contains the lowest sequence number of the data that has not been received yet. For example, suppose a receiver receives data bytes 1, 2, 3, 4 and 6. On receipt of data byte 6, it returns an ACK that contains 5 to (i) acknowledge receipt of data bytes 1 through 4, and (ii) request transmission of data byte 5.

When an ACK arrives, if the sequence number contained in the ACK is greater than *LAR*, the sender updates *LAR* to be the ACK sequence number thereby allowing the sender to transmit more packets.

After the sender sends a packet to the receiver, it sets a retransmission timer. If the timer times out before an ACK returns, the sender retransmits the packet, and sets its timer adaptively, according to the measured round trip time (RTT). In practice, TCP measures this time with a granularity of 100 milliseconds, and uses a retransmission timer with a granularity of half second. If successive retransmissions of the same packet fail, TCP doubles the retransmission timer after each timeout. Note that the sender has to buffer up to *SWS* bytes of data for possible retransmission, until they are acknowledged. The value of *SWS* is selected according to the network congestion status.

2.3.2 Reliable Transport in Multicast

Reliable multicast is a challenging problem that differs in many aspects from reliable unicast in packet-switched network. In particular, the error control method of unicast is either ACK or NAK-based, and can not be directly deployed in multicast because of the following reasons: first, simultaneous NAKs from a large number of receivers can lead to sender and network overload, causing the famous NAK implosion problem [37, 53, 76]. Second, if only a small number of receivers experience data loss, it is inefficient to have the sender multicast the lost data to the entire multicast group. Instead, recovery should be isolated to members that experience data loss. To this end, it has been proposed that the task of processing NAKs and retransmitting lost packets be shared among group members. Several mechanisms have been devised to suppress duplicate NAKs and replies and to designate certain group members to handle NAKs, so as to mitigate the NAK implosion

problem and to reduce recovery latency. Moreover, all the objectives stated above should still be fulfilled when group members join/leave dynamically.

Existing reliable multicast schemes can be roughly classified into two categories: (i) reactive repair-based approaches and (ii) pro-active forward error correction (FEC)-based approaches. In repair-based approaches, the sender or designated router/host sends lost data packets reactively in response to NAKs. They can be further classified into those which are non-TCP-based, such as SRM [37], PGM [30], AIM [54], hierarchical schemes (best represented by LBRRM [40], TMTP [102], and RMTP [57]), *Turning Point* (TP) [76], *Search Party* [19], and ARM [53], and those which extend TCP to accommodate multicast, such as IRMA [51] and MTCP [81].

In a radical departure from repair-based approaches, FEC-based approaches, such as those reported in [13, 68, 83], combine FEC with automatic repeat request (ARQ), and send repair packets proactively before they are required. Specifically, data is transmitted in blocks, each of which consists of k data packets and $n - k$ repair packets. Each block is so encoded (by, for example, Reed-Solomon code) that as long as a receiver receives k out of n packets in that block, it will be able to recover all n packets. Initially the sender transmits k data packets and additional i repair packets, where $0 \leq i \leq n - k$. Only when a receiver fails to decode the entire block will it unicast a NAK (requesting additional repair packets) to the sender. The sender responds to the NAKs by sending an appropriate number, ℓ, of repair packets, where ℓ is determined so as to satisfy all the receivers' requests.

2.4 Congestion Control

As mentioned above, data packets are forwarded by routers from their sources to their destinations on a hop-by-hop basis. An IP router has a number of interfaces each of which

is connected to a transmission link and is equipped with a finite amount of buffers to hold packets that cannot be immediately forwarded. If the packet arrival rate is continuously larger than the packet transmission rate on an interface, a queue of packets will be built up in the buffer and congestion occurs at this router. When congestion occurs (or is about to occur), the common practice is for the router to drop/tag packets according to a queue management policy to signal congestion to the source, in the hope that the source will take appropriate control actions in response to congestion. In essence, the queue management policy deployed at a router and the congestion control algorithm employed at an end-host interact with each other to handle congestion.

2.4.1 Active Queue Management

Among all the queue management policies, drop tail, which drops packets only when its buffer is used up, is the most widely deployed and simplest one. However, it has several drawbacks. First, since a drop tail queue drops packets and conveys congestion signals only at the time of congestion, a significant amount of time may have elapsed between the instant when congestion occurs and the instant when end hosts reduce their sending rates. During this time period, packets may be sent and eventually dropped. Second, the queue length of a drop-tail queue is usually large [24], due to the fact that the drop-tail policy drops packets only when the queue is full. This is against the common belief that the steady-state queue size should be kept small in order to reduce the delay packets experience at the router. Third, as reported in [85], the drop tail policy may result in global synchronization [1], which usually results in a sustained period of low link utilization.

[1]Global synchronization of TCP occurs when multiple TCP connections reduce their transmission rates in response to packet dropping and then increase their rates at the same time when the congestion reduces. It occurs because packets from multiple TCP connections are dropped at routers all at once.

To remedy the aforementioned drawbacks, several *active queue management* (AQM) algorithms have been introduced. By "active," we mean that a router detects congestion and notifies end hosts of (incipient) congestion before congestion actually occurs so that the latter can adapt their sending rates. These algorithms differ in (i) the parameter used as an indicator of traffic load (and congestion); (ii) the policy used to detect congestion (or the likelihood of congestion); and (iii) the policy used to adjust the packet dropping probability in response to (an increased likelihood of) congestion.

The most well-known AQM algorithm is perhaps random early drop (RED) [24, 35]. RED operates by calculating upon packet arrival the average queue length, *avg_queue*, using an exponentially weighted moving average. The parameter *avg_queue* is used to measure traffic load. The policy used to detect the likelihood of congestion is characterized by two thresholds, *minth* and *maxth*. If *avg_queue* is less than *minth*, the router is considered congestion free and no arriving packet is dropped. When *avg_queue* is greater than *maxth*, the router is likely to incur congestion, and all arriving packets are dropped. When *avg_queue* is between the two thresholds, the probability of dropping a packet linearly increases with *avg_queue* from 0 to p_{max}, where p_{max} is the maximum packet dropping probability. RED has been shown to prevent global synchronization, accommodate bursty traffic, incur little overheads, and interact well with TCP under serious congestion conditions. The performance of RED, however, heavily depends on whether or not the two thresholds are properly selected.

2.4.2 TCP Window-Based Congestion Control

The TCP congestion control mechanism maintains, for each connection, a sliding congestion window to bound the number of packets that can be outstanding (sent but not yet

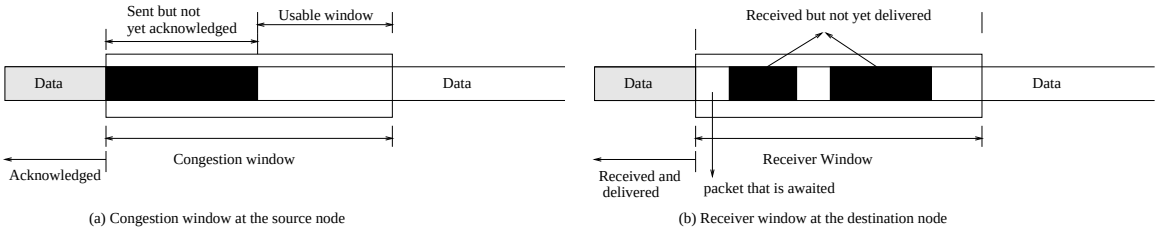


Figure 2.1: Management of TCP congestion window.

acknowledged) for the connection (Fig. 2.1). The size, *cwnd*, of the congestion window is governed by the slow start, congestion avoidance, and fast retransmit/fast recovery algorithms [43]. In the slow start phase, upon receipt of an non-duplicated ACK that acknowledges receipt of a new data packet, the sender increases its *cwnd* by one packet size (i.e., the value of *cwnd* is doubled every RTT) and slides congestion window to the right to allow transmission of new data (Section 2.3.1). When *cwnd* exceeds the *ssthresh* threshold, the congestion avoidance phase commences. In the congestion avoidance phase, upon receipt of a non-duplicated ACK, the sender linearly increases its *cwnd* and slides the congestion window to the right. On the other hand, upon receipt of three duplicate ACKs (which is taken as an indication of packet loss), the sender reduces its *cwnd* by half, in addition to retransmitting the lost packet. This is the well-known *additive increase and multiplicative decrease* (AIMD) algorithm [17]. In the case of retransmission timeouts, which is taken as an indication of severe congestion, the sender reduces *cwnd* to 1.

2.4.3 Rate-based Congestion Control

In contrast to window-based congestion control, in rate-based congestion control a sender adjusts its sending rate according to the network congestion status. To co-exist with TCP window-based congestion control, it is required that rate-based congestion control

should not deprive self-controlled TCP connections of their fair share of network bandwidth. That is, a connection that employs rate-based congestion control should approximately receive, under the same delay and packet loss conditions, the same share of bandwidth as a TCP connection, which shares bottleneck link with it [60]. Approaches in this category can be further classified into two sub-categories: TCP-emulation methods and formula-based methods.

In the TCP-emulation methods (e.g. [80]), the sender emulates the TCP congestion control algorithms (*slow start* and *congestion avoidance*) in the following manner: it reduces the packet transmission rate by half when a packet loss is detected and increases the transmission rate by a small amount when no packet loss is detected in one RTT.

In the formula-based methods [72], the TCP-friendly sending rate is determined using the TCP throughput formula derived in [71]. As the throughput of a TCP connection is expressed in [71] as a function of the RTT and the packet loss probability, a sender estimates its packet loss rate and RTT periodically and then calculates and adjusts the TCP-friendly sending rate accordingly.

2.4.4 End-Host Congestion Control

End-host congestion control aims to develop a unified congestion control mechanism for active unicast connections at a host. The most notable work is the *Congestion Manager* (CM) proposed by Balakrishnan *et al.* [9]. In CM, unicast connections destined for the same destination are bundled together into a single (logical) flow for the purpose of congestion control. All the flows are subject to a single congestion window governed by the AIMD congestion control algorithm. A CM sender also employs a rate-based traffic shaper and a scheduler to regulate packet transmission and to apportion available bandwidth among

flows. In addition, CM uses a loss-resilient protocol to elicit feedback information periodically from receivers about losses and network status.

End-host congestion control applies to both TCP and UDP-based transport. UDP-based connections are controlled in a TCP-friendly manner and respond to indication of network congestion (e.g., packet loss or explicit congestion notification).

CHAPTER 3

RACCOOM: A Rate-Based Congestion Control Approach for Multicasts

In this chapter, we elaborate on how we design and evaluate a rate-based congestion control scheme for multicasts, called *RACCOOM*. We first discuss the issues that must be considered in designing such a scheme. Then we provide an overview of *RACCOOM* (Section 3.1), followed by detailed descriptions of each component mechanism (Section 3.2-Section 3.5). Following that, we present an analysis on how *RACCOOM* achieves weighted fairness and/or TCP-friendliness (Section 3.6-Section 3.7). Finally, we categorize and summarize related work (Section 3.8) and present the simulation results (Section 3.9).

Issues: There are several important issues we consider in designing a rate-based congestion control scheme for multicast. The first issue is support of the notion of TCP-friendliness as described in Section 2.4.3. Since multicast applications may co-exist with TCP-based applications, TCP-friendly congestion control prevents multicast connections from depriving self-controlled TCP connections of their fair share of the network bandwidth.

The second challenge is scalability. Most, if not all, congestion control schemes are ACK-based, NAK-based, or a combination thereof. In the context of multicasts, the ACKs

and NAKs (which we collectively call acknowledgments) received by a sender from multiple receivers reflect diverse congestion conditions in various parts of the network, and have to be appropriately combined when making a single rate control decision. This leads to the problem of what information is carried in an ACK to best represent the congestion status and how ACKs are aggregated for scalability.

The third issue concerns judicious diagnosis of the congestion condition based on the ACK received so as to isolate the effect of *independent* losses of the same packet in a multicast tree. A packet may be lost on one or more tree branches in a multicast tree, and as a result, several receivers may independently report loss of the same packet. If a sender reduces its sending rate for every such loss report, its sending rate will be severely throttled. This is termed as the *loss path multiplicity* problem in [12].

The next challenge is how to interpret and use RTT in congestion detection. Usually a prolonged RTT in unicasts is a good indication of congestion at some intermediate router (which in turn serves as a preamble to packet loss). However, as there are multiple on-tree paths in a multicast tree, one for each receiver, it is neither practical nor scalable to keep track of RTTs along all on-tree paths. This leads to the problems of (i) along which on-tree path(s) the RTT(s) are kept track of and estimated, (ii) how the estimated RTT(s) are used in congestion detection, and (iii) what action to take if the RTT(s) on certain on-tree path(s) change because of dynamic traffic or membership change.

The last but not the least important issue is the mechanism used to adjust the sending rate. Two most widely used objectives in designing the rate adjustment mechanism are TCP-friendliness and fairness (among competing connections that use the same method). No matter which objective is targeted, a good rate adjustment mechanism should be devised

in an analytically provable manner. Also, in order to be widely deployed over the Internet, the mechanism should require as little router support as possible.

3.1 Overview

We design a new rate-based multicast congestion control approach, called *RACCOOM*, that addresses all the above issues. In *RACCOOM*, each acknowledgment message contains, among other things, the sequence number, *LostSeq*, of the packet (if any) most recently detected to be lost, and the sequence number, *AckSeq*, of the packet most recently received. Acknowledgment messages are sent along the reverse path on which data packets are sent and are appropriately aggregated either by intermediate routers, or in the case of no router support, by non-leaf, designated receivers.

In the absence of packet loss, a *RACCOOM* session keeps track of the congestion status of the on-tree path with the largest RTT (called the *target path*), and adjusts its sending rate in a TCP Vegas-like manner [2, 16]. Upon detection of packet loss anywhere in the multicast tree, *RACCOOM* then responds by reducing its sending rate in a TCP Reno manner. By judiciously using parameters in the aggregated acknowledgment and the local states maintained at the sender, a *RACCOOM* session is able to identify independent losses of the same packet, to diagnose the reason of RTT changes (network congestion or dynamic network traffic/membership changes), and to react correspondingly.

To achieve TCP-friendliness, we devise a simple method in *RACCOOM* to emulate how a TCP connection would behave under the same packet loss and delay characteristics. The results thus derived are used by *RACCOOM* to on-line adjust the parameters of its rate adjustment method. Alternatively, we can achieve (weighted) fairness (in terms of bandwidth sharing) among competing *RACCOOM* connections by tuning its parameters

using results obtained from the feedback control theory. We have implemented *RACCOOM* on *ns-2* [50] and conducted simulation to test the TCP-friendliness, fairness, and scalability properties of *RACCOOM* and to compare *RACCOOM* against other existing approaches. The encouraging simulation results, coupled with the fact that all the operations can be performed at end hosts, suggest that *RACCOOM* is a practical yet effective congestion control solution for continuous media multicast applications.

3.2 Acknowledgment Aggregation for Scalability

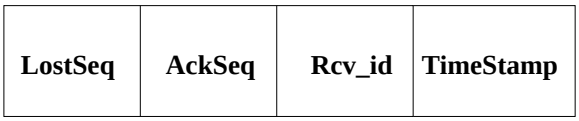


Figure 3.1: Four fields in an acknowledgment message.

Every packet (except retransmitted ones) sent by the source is associated with a unique sequence number and a time stamp (the time instant the packet is sent). A receiver detects packet loss by observing a gap in the sequence number. When a receiver receives a packet, it sends an acknowledgment message to the sender. An acknowledgment message contains the following fields (Fig. 3.1):

1. *LostSeq*: The sequence number of the packet most recently detected to be lost (initially it is set to 0);
2. *AckSeq*: The sequence number of the packet received most recently (initially it is set to 0);
3. *Rcv_id*: The id of the receiver that reports *AckSeq*;

4. *TimeStamp*: the time at which the data packet to be acknowledged was sent. This information is copied from the data packet and used to estimate every packet's RTT.[2]

Acknowledgment messages are sent, and appropriately aggregated, along the reverse path on which data packets are forwarded, to prevent ACK implosion at the sender. We present two versions of acknowledgment aggregation mechanisms: one relies on router support, and the other relies on the help of *designated receivers*.[3] For clarity of presentation, we first present the version that relies on intermediate routers to aggregate acknowledgment messages, and then discuss how the first version can be modified to eliminate router support.

Router-assisted acknowledgment aggregation: In this version, every on-tree router aggregates the acknowledgment messages received on all of its downstream interfaces, and sends upstream a single acknowledgment message. Note that *RACCOOM* does not keep track of downstream interfaces as receivers join/leave, but relies on the underlying multicast routing protocol to maintain such information. A *RACCOOM*-aware router simply obtains such information in the associated (source, multicast group) entry of the forwarding cache. By the same token, the robustness of the acknowledgment aggregation mechanism is ensured by the soft state signaling approach commonly used in underlying multicast routing protocols. For example, if a receiver does not leave gracefully, the downstream interface that leads to this receiver will be deleted by the underlying soft-state based multicast routing protocol as a result of lack of refresh messages arriving on that interface, and will hence not participate in acknowledgment aggregation.

[2]Note that no information is kept at the sender regarding when packets are sent.

[3]Note that designated receivers are only used for acknowledgment aggregation in *RACCOOM*. They should not be confused with designated receivers used for reliable multicasts (the latter usually serve for multiple purposes).

The rules used to aggregate acknowledgment messages are as follows. An on-tree router keeps, for each downstream interface, the latest acknowledgment message received on that interface. Initially the router keeps for each downstream interface a null message with *LostSeq* $= 0$, *AckSeq* $= 0$, *Rcv_id* $=$ *null*, and *TimeStamp* $= 0$. It also keeps the latest acknowledgment message sent upstream. Whenever a new acknowledgment message arrives on one of its downstream interfaces, the router (i) replaces the old acknowledgment message kept for that interface with the new one; (ii) composes an aggregated acknowledgment by setting

1. *LostSeq* to be the maximum value of *LostSeq*s received on all the downstream interfaces.
2. *AckSeq* to be the minimum value of *AckSeq*s received on all the downstream interfaces.
3. *Rcv_id* to be the id of the receiver that gives the minimum value of *AckSeq*.
4. *TimeStamp* to be the *TimeStamp* value in the acknowledgment message with the minimum value of *AckSeq*.

If a new downstream interface is included (as a result of member join) when a multicast is in session, the router initializes the message for that interface as *LostSeq* $= 0$, *AckSeq* $=$ the value of the *AckSeq* forwarded upstream, *Rcv_id* $=$ *null*, and *TimeStamp* $= 0$.

Finally, the following rules are used to determine whether or not an acknowledgment message should be forwarded upstream: When the value of *Rcv_id* in the newly composed acknowledgment message differs from that in the last acknowledgment message sent upstream, the acknowledgment message is forwarded upstream. Otherwise, the acknowledgment message is forwarded only when *LostSeq* or *AckSeq* is greater than the corresponding

value in the last acknowledgment message sent upstream. This acknowledgment aggregation mechanism is in essence similar to that used in IRMA [51] (but with much less number of acknowledgment fields) and requires router support.

Example 3.1 *Fig. 3.2 gives an example that shows how acknowledgment messages are aggregated.* S *is the sender,* R1 *and* R2 *are the receivers, and* G *is an on-tree router. The RTT from* S *to* R1 *(*R2*) is approximately equal to the time to transport 8 packets (10 packets). In the snapshot of Fig. 3.2,* AckSeq *in the aggregated acknowledgment message at router G is set to that in the acknowledgment message forwarded by* R2 *(*$\min\{3, 5\} = 3$*).* AckSeq *received by* S *is 1. Shown below are the* AckSeq *values received from* R1 *and* R2 *at router* G*, the* AckSeq *value forwarded upstream by router* G*, and the* AckSeq *value received at* S *at the kth packet time, $k \geq 0$.*

	Time k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	...
G	AckSeq *received from* R1							*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	...
	AckSeq *received from* R2									*1*	*2*	*3*	*4*	*5*	*6*	...
	AckSeq *forwarded*									*1*	*2*	*3*	*4*	*5*	*6*	...
S	AckSeq *received*											*1*	*2*	*3*	*4*	...

Designated receiver-assisted acknowledgment aggregation: The above router-assisted mechanism can be modified to eliminate the router support as follows: the *RACCOOM* module that handles acknowledgment aggregation now runs at some hosts, called *designated receivers* (DRs). For each multicast group *(sender, multicast group)*, some routers are selected using, for example, the designated receiver selection approaches reported in [46, 55, 56] and the DRs on the directly attached subnets of these selected routers are "forced" to join the multicast group. In this manner, the multicast tree developed by the

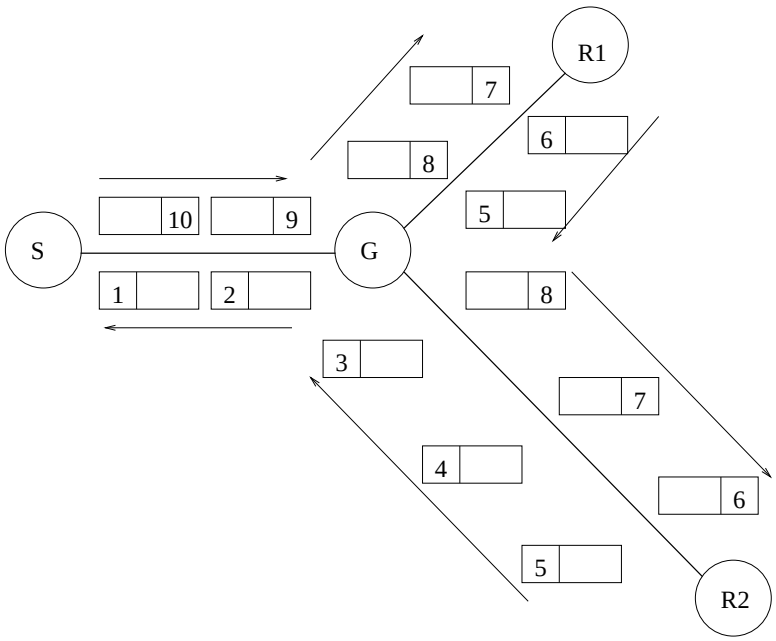


Figure 3.2: An example of acknowledgment aggregation. The boxes above and below a link indicate, respectively, the data packets and the acknowledgment messages that are in transit on the link in the snapshot. The number inside a box indicates either the sequence number (of a data packet) or the *AckSeq* value (of an acknowledgment).

underlying multicast routing protocol contains a subset of the selected routers as intermediate, relay routers and the associated DRs as non-leaf members. Use of designated receivers to handle certain router functions has been reported in RMTP [57] and MTCP [81].

Acknowledgment messages are sent along the reverse path as in the router-assisted approach, but only aggregated at the non-leaf DRs. The router to which a DR is attached intercepts acknowledgment messages and forwards to the DR. The router also passes to the DR the number of the downstream receivers and DRs from which acknowledgment messages will be received. This information can be derived by the underlying multicast routing protocol that employs an *EXPRESS*-like method[4] [39]. A DR then keeps, for each downstream (pure or designated) receiver from which acknowledge messages are received, the latest acknowledgment message received, and aggregates/forwards acknowledgment messages in the same manner as an intermediate router does in the router-assisted mechanism.

Use of aggregated acknowledgment messages: As demonstrated in Example 3.1, because of the way in which acknowledgment messages are aggregated, the *AckSeq*, *Rcv_id*, and *TimeStamp* values received by the sender are always those in the acknowledgment message sent along the on-tree path with the largest RTT. As indicated in [12], a path with a large RTT usually includes at least one link with long queues[5], and hence the congestion status along this path is usually more severe than along other on-tree paths. Let this path be denoted as the *target path*. A *RACCOOM* session keeps track of the congestion status (implied in *AckSeq*, *Rcv_id*, and *TimeStamp* values) on the target path before packet loss takes place. On the other hand, when packet loss is detected anywhere in the multicast

[4]Succinctly, each *EXPRESS*-aware router sends back the number of downstream receivers in its state refresh messages sent upstream. Each *EXPRESS*-aware router will then be able to sum over all downstream interfaces the number of downstream receivers, and provide this information to applications that need it. Note that the EXPRESS mechanism is part of the multicast routing protocol but not part of *RACCOOM*.

[5]Here we exclude the case of satellite links.

tree, it is reported in the *LostSeq* field, and by virtue of the way in which acknowledgment messages are aggregated (i.e., *LostSeq*, if non-zero, contains the sequence number of the most recently lost packet), it is delivered to the sender. This allows the *RACCOOM* sender to detect as early as possible any packet loss along *any* on-tree path.

The use of the *Rcv_id* field will be elaborated on when we discuss how *RACCOOM* handles dynamic traffic and/or membership changes. Finally, the *TimeStamp* field is used to estimate the RTT along the target path. When a sender receives an acknowledgment message at time t, it calculates the current RTT estimate using the difference $t -$ *TimeStamp*, and will only keep the *minimum* value of RTT values estimated.[6]

3.3 Rate Adjustment

Fig. 3.3 outlines the procedure taken by the sender to adjust its sending rate before packet loss is detected. The sender keeps the minimum value, RTT_{min}, of RTTs that it has measured so far along the target path. In other words, RTT_{min} is used to keep track of the round trip *propagation* time along the target path under the condition that all the routers along the target path are queue free and all the packets on the path are in transit. We will discuss in Section 3.5 how *RACCOOM* deals with the situation in which congestion persists and RTT_{min} is overestimated.

Let *SndMax* denote the maximum sequence number that the sender has sent. $N_{real} \stackrel{\triangle}{=} SndMax - AckSeq$ is the total number of data packets outstanding in the network. On the other hand, let R denote the sending rate and S the packet size, then $N_{exp} \stackrel{\triangle}{=} \frac{R \times RTT_{min}}{S}$ is the expected number of outstanding packets under the queue free condition.

[6]The reason we do not estimate RTT using a low-pass filter with an exponential weighted moving average is because we are only interested in the *minimum* value (rather than the average value) of RTTs along the target path.

If no queue is building up along the target path, the difference between N_{real} and N_{exp} should not be significant. On the other hand, when data or acknowledgment messages are queuing up at congested routers along the target path, the difference between N_{real} and N_{exp} increases. Note that RTT_{min} remains the same when queues develop at intermediate routers along the target path, because we keep track of the minimum value, RTT_{min}, of RTTs and hence RTT_{min} will not updated when the RTT increases.

Based on the above observation, we use $\textit{diff} = N_{real} - N_{exp}$ as an index of congestion status, under the case of no packet loss. Specifically, the sender adjusts its sending rate as follows:

$$R_a \leftarrow \begin{cases} R_a + \frac{k_2 \cdot |\textit{diff}| \cdot S}{RTT_{min}} & \textit{diff} < \alpha, \\ R_a - \frac{k_2 \cdot |\textit{diff}| \cdot S}{RTT_{min}} & \textit{diff} > \beta, \\ R_a, & \text{otherwise,} \end{cases} \tag{3.1}$$

and

$$R_{snd} \leftarrow R_a - k_1 \cdot \frac{\textit{diff} \times S}{RTT_{min}}, \tag{3.2}$$

where R_a is the *accumulated* rate change, R_{snd} is the sending rate, and k_1, k_2, α, and β are all positive and tunable parameters. Intuitively Eqs. (3.1)–(3.2) intend to keep the number, N_{real}, of outstanding packets along the target path in the interval $[N_{exp} + \alpha, N_{exp} + \beta]$, or, equivalently, the number of packets that are *queued* at some intermediate routers (rather than in transit) along the target path in the interval $[\alpha, \beta]$. The rate adjustment method used here is similar to the window adjustment method used in TCP Vegas [2, 16], except for the $k_1 \cdot \frac{diff \times S}{RTT_{min}}$ term. The reason for including this term in R_{snd} is for system stability and will be elaborated on in Section 3.6.

By control theory analysis, we can show that the sending rate of a *RACCOOM* session is irrelevant to the values of k_1 and k_2 (as long as they are appropriately selected to drive the system into equilibrium), but is greatly affected by the values of α and β. We will discuss

in Section 3.6 these issues and how to fine tune α and β to achieve either weighted fairness or TCP-friendliness.

```
/* Upon receipt of an acknowledgment message msg */
1.  RTT_est ← RTT_estimate();
2.  if (RTT_est < RTT_min) {
3.      RTT_min ← RTT_est;
4.      Rcv_id ← msg.Rcv_id;
5.  }
6.  N_real ← SndMax − AckSeq;
7.  N_exp ← (R_snd × RTT_min) / S;
8.  diff ← N_real − N_exp;
9.  if (diff < α)
10.     R_a ← R_a + (k2 · |diff| · S) / RTT_min;
11. else if (diff > β)
12.     R_a ← R_a + (k2 · |diff| · S) / RTT_min;
13. else R_a ← R_a;
14. R_snd ← R_a − k1 · (diff × S) / RTT_min;
```

Figure 3.3: Procedure taken to adjust the sending rate when packet loss is not detected.

Similar to TCP, the congestion avoidance phase commences when packet loss is detected (anywhere in the multicast tree) or upon acknowledgment timeout. Lines 1–6 in Fig. 3.4 give the procedure taken by the sender to adjust its sending rate in the congestion avoidance phase. In particular, to handle the *lost path multiplicity* problem [12], the sender keeps a parameter *LastCut* (initialized to zero) to keep track of the value of *SndMax* at the time when the sending rate is reduced by half in response to packet loss or acknowledgment timeout. The sender reduces its sending rate only if *LostSeq* contained in the received acknowledgment message is non-zero and is greater than *LastCut*. That is, the sender only reduces its sending rate once during the interval of transmitting a window worth of packets. Only when congestion persists beyond that interval will the sending rate be further reduced.

```
/* Upon session initialization */
1. LastCut ← 0; Rcv_id ← null;

2. if ((LostSeq ≠ 0 && LostSeq > LastCut) || acknowledgment timeout) {
        /* congestion avoidance phase */
3.      R_a ← R_a/2;
4.      R_snd ← R_snd/2;
5.      LastCut ← SndMax;
6. }
7. else {
        /* procedure taken to adjust the sending rate when packet loss is not detected.
           Figure 3.3 goes here */
8. } /* else */
```

Figure 3.4: Procedure taken by *RACCOOM* to adjust the sending rate.

3.4 Handling of Persistent Congestion

If at the time when a *RACCOOM* session is set up, queues already develop at the routers on the target path, the RTTs thus estimated will be considerably larger than the actual propagation delay of the path, leading to an overestimate of RTT_{min} (and hence N_{exp}). As a result, a *RACCOOM* session may set its sending rate to a value such that it believes its expected number of packets queued along the target path lies between α and β, when in fact it has sent more backlogged packets. This in turn worsens the congestion situation, and is termed as the *persistent congestion* problem that has been reported to exist in TCP Vegas [65].

We deal with this problem by switching, upon packet loss, from the TCP Vegas-like rate adjustment method to the TCP Reno-like rate reduction method. When a *RACCOOM* overestimates RTT_{min} and sends more packets than it should, packet loss in the multicast tree occurs. A *RACCOOM* session then reduces its sending rate by half, and if the congestion persists, the *RACCOOM* session will further reduce its sending rate.

3.5 Capability to Handle Membership or Network Traffic Change

Because members may dynamically join and leave a multicast group, the on-tree path with the largest RTT (i.e., the target path) may change as a result of member join/leave. Similarly, because other traffic being carried over the network may change, the target path may also change as a result of traffic load change. In both cases, the value of RTT_{min} maintained by the sender should be updated so as for the sender to adequately adjust its sending rate. We discuss below how *RACCOOM* deals with each of the cases.

Handling of dynamic traffic changes: Recall that the *Rcv_id* field in an acknowledgment message is used to notify the sender of which receiver lies on the target path. When an on-tree router or a designated receiver aggregates all the acknowledgment messages arrived on downstream interfaces, it sets *Rcv_id* to be the identifier of the receiver that gives the minimum value of *AckSeq* (and hence the most "distant" receiver). In the case of target path change, the *Rcv_id* field in the aggregated acknowledgment will change as well.

To detect whether or not the target path has changed, a *RACCOOM* sender hence keeps a local state variable, *Target_rcv_id*, which keeps track of the receiver on the target path. If the *Rcv_id* contained in an acknowledgment message differs from *Target_rcv_id*, it implies the target path has changed, and the sender should update both *Target_rcv_id* and RTT_{min} to reflect this fact. Specifically, the condition in which RTT_{min} is updated (line 2 in Fig. 3.3) is changed from "if ($RTT_{est} < RTT_{min}$)" to

$$\text{if } ((RTT_{est} < RTT_{min}) \,||\, (\textit{Target\_rcv\_id} \neq \textit{msg.Rcv\_id})). \tag{3.3}$$

Handling of dynamic membership changes: As discussed in Section 3.2, when a member joins the multicast tree at an on-tree router, the underlying multicast routing protocol updates the associated (source, multicast group) entry of the forwarding cache. A *RACCOOM*-aware router or a designated receiver can obtain this information from the forwarding cache and will be able to adequately aggregate acknowledgment messages. Because of the way in which acknowledgment messages are aggregated, the *AckSeq*, *Rcv_id*, *TimeStamp* values received by the sender are those in the acknowledgment message sent along the on-tree path with the largest RTT. If the on-tree path from the sender to the new member is not the one with the largest RTT, then this member join changes neither the *Rcv_id* field of the aggregated message nor RTT_{min}. On the other hand, if the on-tree path leading to the new member does become the one with the largest RTT, then the second test condition in Eq. (3.3) holds true (because the *Rcv_id* field of the aggregated acknowledgment contains the id of the new member). The sender will then update RTT_{min} and *Target_rcv_id* accordingly.

Similarly, the fact that a downstream member leaves (whether gracefully or not) is reflected in the associated entry of the forwarding cache kept by the underlying multicast routing protocol. (If a receiver does not leave gracefully, the downstream interface that leads to this receiver will be deleted upon state refresh timeout under the soft state approach most multicast routing protocols employ.) In the case that the leaving member is the most "distant" one (i.e., the leaving member is the leaf member to which the target path leads) the acknowledgment message the sender receives will be routed along a new target path (and contains a new *Rcv_id*); the sender can update RTT_{min} and *Target_rcv_id* accordingly. In the case that a non-leaf member or a leaf member that does not lie on the target path leaves, the target path does not change and hence RTT_{min} will not be affected.

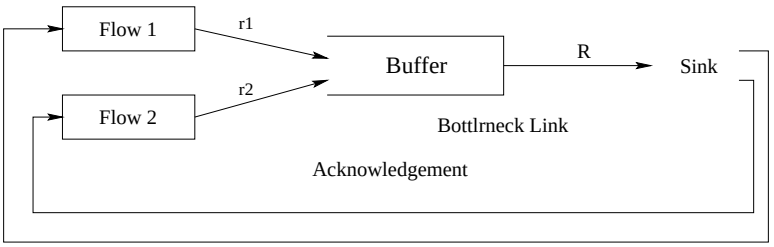


Figure 3.5: Analysis model.

3.6 Analysis

In this section, we analyze, with the use of feedback control theory, the behavior of *RACCOOM* before packet loss occurs. We show that if the parameters α and β in the rate adjustment method are appropriately selected, *RACCOOM* can achieve weighted fairness among competing *RACCOOM* sessions. Apart from leading to a better understanding of the control feedback capability of *RACCOOM*, the analysis also offers guidelines for setting α and β for achieving (weighted) fairness among *RACCOOM* sessions.

Fig. 3.5 depicts the system under consideration. We consider two flows, flow 1 and flow 2 that share the bottleneck link of capacity R on the target path. A flow may be a unicast or multicast session. In the latter case, the flow refers to the data stream (of the multicast session) that traverses the target path. For ease of analysis, we also assume that packets are infinitely divisible, i.e., we use the fluid model. Let r_1 and r_2 denote, respectively, the rate at which flow 1 and flow 2 send their data packets. Both flows adjust, based on the acknowledgments fed back from the receiver (which the target path leads to), their sending rates r_1 and r_2 using the *RACCOOM* rate adjustment mechanism.

Let $q(t)$ denote the queue length at the bottleneck link at time t. Since the rate at which data arrives at the bottleneck link is $r_1(t) + r_2(t)$ and the link capacity is R, $q(t)$ can be

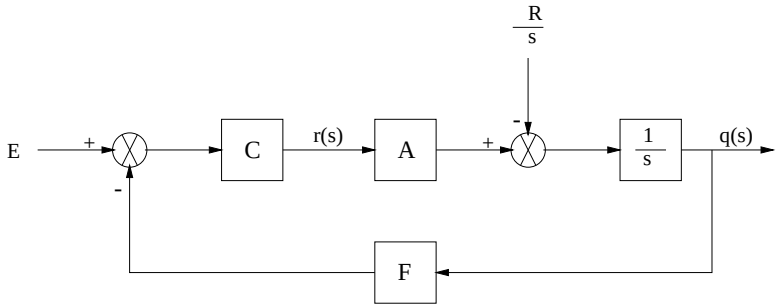


Figure 3.6: Block diagram of the system under consideration.

calculated as

$$q(t) = \int_{-\infty}^{t} [r_1(s) + r_2(s) - R]^+ \, ds, \tag{3.4}$$

where $[a]^+ = \max(a, 0)$. When the sender of flow i receives an acknowledgment message from the receiver, it estimates the queue length, $q_i(t)$, of its data at the bottleneck. Let $q_i(t) \stackrel{\triangle}{=} f_i \cdot q(t)$, where f_i is the fraction of flow-i data packets in the buffer queue. Because of the propagation and queuing delays incurred, the queue length estimated at time t is in fact $f_i \cdot q(t-d)$, where d is the round trip delay.

We analyze the rate adjustment method expressed in Eqs. (3.1)–(3.2) under the assumption that $\alpha_i \stackrel{\sim}{=} \beta_i$. The rate adjustment method can now be described as follows: it first calculates the difference between the expected queue length, E_i ($= N_{exp}$ in Section 3.3), and the actual queue length, $q_i(t-d)$, at the bottleneck link and then adjusts its rate by

$$\begin{aligned} diff &= E_i - f_i \cdot q(t-d), \\ r_i(t) &= k_1 \cdot diff + \int_{-\infty}^{t} k_2 \cdot diff \; dt, \end{aligned}$$

where k_1 and k_2 are the rate gains used in the rate adjustment method.

After Laplace transform the system can be represented in Fig. 3.6 with the following parameters:

1. $\mathbf{E} = \begin{bmatrix} \frac{E_1}{s} \\ \frac{E_2}{s} \end{bmatrix}$: the column vector of the expected queue lengths of flows at the bottleneck link, where E_i ($i = 1, 2$) is the queue length of flow i. Because *RACCOOM* operates by keeping the number of packets *queued* at the bottleneck link along the target path in the dead zone $[\alpha_i, \beta_i]$, without loss of generality, we set $E_i = \frac{\alpha_i + \beta_i}{2}$, $\alpha = E_i - \triangle$ and $\beta = E_i + \triangle$, respectively, where $\triangle$ is a small value. (In the simulation study, we found that the value of $\triangle$ does not affect the performance as long as it is set to a reasonably small value.)

2. $\mathbf{F} = \begin{bmatrix} f_1 e^{-d_1 s} \\ f_2 e^{-d_2 s} \end{bmatrix}$: the column vector of the feedback from the receiver, where f_i ($i = 1, 2$) is the fraction of data packets in the queue that are from flow i.

3. $\mathbf{C} = \begin{bmatrix} k_1 + \frac{k_2}{s} & 0 \\ 0 & k_1 + \frac{k_2}{s} \end{bmatrix}$: the rate controllers of flows.

4. $\mathbf{r}(s) = \begin{bmatrix} r_1(s) \\ r_2(s) \end{bmatrix}$: the column vector of the flow sending rates.

5. $\mathbf{A} = \begin{bmatrix} 1 & 1 \end{bmatrix}$: the adder that performs $r_1(s) + r_2(s)$.

6. R: the link capacity of the bottleneck link.

Now the rates of flows can be calculated as follow:

$$\mathbf{r}(s) = C(E - \frac{F}{s}(Ar(s) - \frac{R}{s})) \; = \; \frac{1}{s}CF[\frac{R}{s} - Ar(s)] + CE.$$

By plugging into the expressions for $\mathbf{C}$, $\mathbf{F}$, $\mathbf{A}$, and $\mathbf{E}$ and solving for $\mathbf{r}(s)$, we have

$$r(s) = \frac{R \begin{bmatrix} (k_1 s + k_2) f_1 e^{-d_1 s} \\ (k_1 s + k_2) f_2 e^{-d_2 s} \end{bmatrix}}{s^3 + (k_1 s + k_2) f_1 s e^{-d_1 s} + (k_1 s + k_2) f_2 s e^{-d_2 s}} + \frac{R \begin{bmatrix} (k_1 s + k_2) E_1 \\ (k_1 s + k_2) E_2 \end{bmatrix}}{s^2 + (k_1 s + k_2) f_1 e^{-d_1 s} + (k_1 s + k_2) f_2 e^{-d_2 s}}. \tag{3.5}$$

Note that the term $k_1 \cdot$ *diff* is introduced to ensure the system will reach equilibrium. By properly selecting k_1, we can shorten the time needed for the system to reach equilibrium. The flow sending rates after the system reaches equilibrium can be obtained by letting $t \to \infty$:

$$\mathbf{r}(\infty) = \lim_{s \to 0} s\mathbf{r}(s) = \begin{bmatrix} \frac{R \cdot f_1}{f_1 + f_2} \\ \frac{R \cdot f_2}{f_1 + f_2} \end{bmatrix} \tag{3.6}$$

Eq. (3.6) indicates that the sending rate of a *RACCOOM* flow i is proportional to f_i. This is a nice feature that we can exploit to realize different levels of fairness. For example, to enforce (weighted) fairness between two flows that share the bottleneck link, i.e. $r_1(t) = \ell \cdot r_2(t)$, we set $f_1 = \ell \cdot f_2$. Since the control system operates by keeping the number of flow-i data packets queued at the bottleneck link as close to E_i as possible, we have $f_i = E_i/q(t)$. Hence, we can achieve (weighted) fairness by setting $E_1 = \ell E_2$.

In the case that the traffic sources are of variable bit rate (VBR), $r_i(t)$ (and hence α_i and β_i) can be dynamically adjusted (on a coarse time scale) to match the data generation rate.

3.7 Setting Parameters to Achieve TCP-friendliness

To achieve TCP-friendliness, a *RACCOOM* flow should adjust its desired queue length (i.e., the parameters α and β) to attain the throughput that a TCP connection would have received under the same packet loss and delay characteristics. When a *RACCOOM* flow detects packet loss, it adjusts its sending rate according to Eqs. (3.1)–(3.2). In addition, it also adjusts E (or equivalently α and β) as discussed below.

Fig. 3.7 depicts an instance of the window adjustment process of a TCP flow in the steady-state congestion avoidance phase (i.e., the slow start phase in the initial stage of a

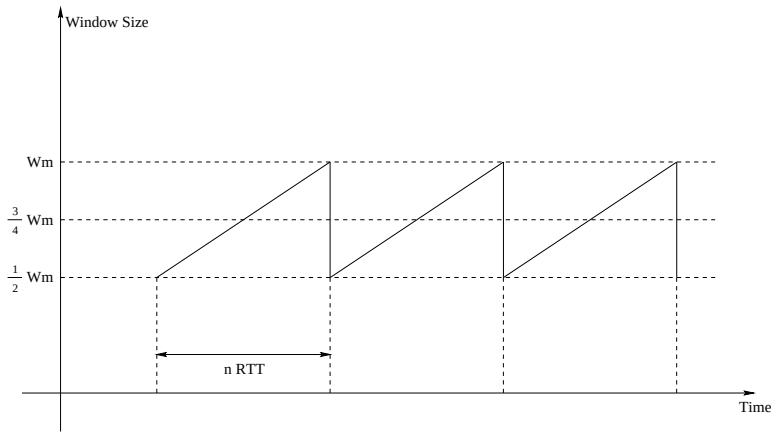


Figure 3.7: Congestion window adjustment of TCP.

TCP flow and the effect of timeouts are not considered). In this phase, a TCP flow increases its congestion window linearly until it encounters packet loss, at which point it reduces its congestion window by half. The average TCP throughput is thus $\frac{3}{4}\frac{W_m}{RTT}$, where W_m is the maximum TCP congestion window size. In order to attain approximately the same average throughput as a TCP flow, a *RACCOOM* flow should adjust its sending rate so that it sends $\frac{3}{4} \cdot W_m$ packets per RTT. That is,

$$N_{real} \simeq \frac{3W_m}{4}. \tag{3.7}$$

Let the time interval between two consecutive packet losses be denoted as a period. From Fig. 3.7 we know that a period is of length n RTTs, where

$$n = \frac{W_m}{2}. \tag{3.8}$$

Combining Eq. (3.7) and Eq. (3.8), we have $W_m = N_{real} + \frac{n}{2}$. Let $E^{(k)}$ denote the desired queue length in the kth period and n_k the length (in units of RTTs) of the kth period. Then,

when a *RACCOOM* flow reaches equilibrium and attains the same TCP throughput,

$$\begin{aligned} E^{(k)} &= N_{real} - N_{exp} = \frac{3W_m}{4} - N_{exp} \\ &= \frac{3}{4}(N_{real} + \frac{n_k}{2}) - N_{exp} \end{aligned} \tag{3.9}$$

Since in the previous period (i.e., the $(k-1)$th period), $N_{real} = N_{exp} + E^{(k-1)}$, we have

$$E^{(k)} = \frac{3E^{(k-1)}}{4} + \frac{3n_k}{8} - \frac{N_{exp}}{4}. \tag{3.10}$$

By keeping track of the time interval (in units of RTTs) between the $(k-1)$th packet loss and the kth packet loss (i.e., n_k), we can use Eq. (3.10) to adjust $E^{(k)}$'s (and hence the parameters α and β) upon every packet loss.

Note that fairness among competing *RACCOOM* sessions and TCP friendliness with co-existent TCP connections can be simultaneously achieved by having each *RACCOOM* session on-line adjust its parameters using Eq. (3.10). This is because under the above iterative approach, the on-line adjusted values of α and β will be approximately the same among all *RACCOOM* sessions. However, *weighted* fairness among *RACCOOM* sessions *with different weights* and TCP-friendliness cannot be simultaneously achieved.

3.8 Related Work

There are two approaches used in TCP-friendly congestion control: *window-based* and *rate-based* approaches as described in Section 2.4.2. Among several proposed TCP-friendly congestion control schemes, the approaches in [42,51,81,82,91] are window-based and those in [44, 63, 72, 80, 94, 99] are rate-based.

Window based approaches: All existing window-based approaches [42, 51, 81, 82, 91] use the TCP congestion control and avoidance method to adjust their window sizes, whenever an ACK message is received or packet loss is detected. In what follows, we highlight their major differences. The approach proposed in [42] does not consider packet retransmission. Upon packet loss, only the window size is reduced by half, but the packet is not retransmitted. SCE [91] is layered between TCP and IP. All the receivers acknowledge the data packets they have received. When the number of ACK messages for a data packet exceeds a pre-determined threshold, a single ACK is generated and sent to the TCP layer. The major drawback of SCE is that it is not scalable, as a sender has to keep the states for all the receivers.

IRMA [51] requires that multicast routers be modified so as to (i) keep track of numerous states (e.g., the identities of downstream routers and upstream router, and for each downstream router, the sequence number, the number of duplicate ACK for the last sequence number) for ACK aggregation, to (ii) estimate RTTs, and to (iii) support TCP semantics at end hosts. In MTCP [81], all receivers are organized into a multicast tree at the transport layer and data are reliably transmitted among them. The drawback is, however, that the resulting tree at the transport layer may not be optimal.

PGMcc [82] is based on the router functions and packet formats defined in PGM, but is, in general, applicable to all router-assisted protocols. PGMcc emulates the evolution of the TCP congestion window when ACK or NAK from a selected representative receiver (called *Acker*) is received. The receivers report to the sender their estimated packet loss rates in their NAKs. The sender estimates RTTs to all the receivers and uses the TCP throughput

characterization equation derived in [71] to calculate the throughput that a TCP connection would attain under the same condition (i.e., with the same RTT, packet loss rate, and so on). The sender then selects, among all the receivers, the one that attains the minimum throughput as the *Acker*. Since a sender is responsible for *Acker* selection, it has to handle all the NAKs from the receivers, and hence is subject to the NAK implosion problem.

Rate-based Approaches: The rate-based approaches can be further classified into two categories: those that adjust their sending rate in an additive increase/multiplicative decrease fashion as in TCP [80, 95] and those [72, 98, 99] that adjust the sending rate in compliance with the TCP throughput characterization equations derived in [69, 71].

The rate adaptation protocol (RAP) [80] falls in the first category. In RAP, the packet transmission rate is reduced by half when packet loss is detected and is increased by a small amount in the absence of packet loss in one RTT. Also included in RAP are (i) a method to use the ratio of long-term and short-term averages of RTT to fine tune the sending rate on a per-packet basis; and (ii) a mechanism (similar to TCP-Sack) to identify a cluster of packet losses that are potentially caused by the same congestion. Sisalem *et al.* [87] proposed to use a modified RTP/RTCP report transmitted between the receiver and the sender to estimate the RTT, the packet loss rate, and the bottleneck link bandwidth. Based on these estimated parameters, the scheme then adjusts the sending rate using an additive increase and multiplicative decrease method. Although simple and feasible, this scheme contains several tunable parameters that must be determined by the user. Both of the above

schemes focus on unicast, and it is not clear whether or not, and how, they can be extended to multicast.

In [44, 63, 95], congestion control is realized by transmitting data in a layered manner. Video data is encoded into a number of layers that can be incrementally combined to provide progressive refinement. Lower layers encode coarse information while high layers encode details. The receivers of each layer constitute a multicast group. The number of layers that a receiver subscribes to depends on its perceived packet loss rate and outcomes of its join experiments. The work in [95] achieves TCP-like additive increase and multiplicative decrease congestion control by using strict time limits on when a receiver may join/leave a group. Although the above layered multicast approaches [63, 95] in general perform well, they have been shown in [52] to exhibit significant instability and several pathological behaviors, such as slow convergence, high loss rate, conservative or aggressive behaviors in some cases as compared to TCP. In [44], the authors argue that join/leave coordination is difficult to achieve in [63,95] if the tree topology is not known. Hence, they develop an algorithm that uses the tree topology and the loss information to determine the optimal subscription bandwidth for each receiver. The major drawbacks of this approach are that it is centralized and that the multicast tree topology and the bandwidth available on each link may not be readily available in reality.

In the second category of rate-based approaches, the sending rate is adjusted with respect to the measured loss rate and RTTs, using the TCP characterization equation derived in [62, 69, 71]. In particular, the protocol proposed in [94] is based on the TCP throughput characterization that does not take into account of timeouts (reported in [62, 69]), while

those in [72, 98] are based on the TCPs throughput characterization that does (reported in [71]). The scheme in [94] relies on the notion of layering. A receiver estimates its RTT and packet loss rate, and calculates the TCP-friendly rate. Based on the calculated rate, each receiver then determines dynamically whether or not to join/leave certain layers. Because the TCP characterization in [69] does not take into account of TCP timeouts and is not accurate when the packet loss rate is higher than 5%, the scheme may not be TCP-friendly when the loss rate exceeds 5%.

Padhye *et al.* [72] proposed the CMTCP protocol in which a more accurate TCP characterization equation reported in [71] is used to adjust the sending rate. Although CMTCP has been validated by simulation and real-world experimentation to be TCP-friendly, authors of CMTCP also observed the following difficulties: (i) measuring packet loss rates accurately is not an easy task; (ii) the re-computation interval, M, over which on-line estimated parameters are updated has an impact on the performance and has to be fine tuned; (iii) CMTCP behaves more aggressively than TCP under the small RTT case.

TFMCC [99] extends the TCP-friendly TFRC [34] protocol from unicast to multicast. In TFMCC, receivers estimate their packet loss rates and RTTs between them and the sender. Using the equation derived in [71], receivers calculate TCP-friendly transmission rates. The rates are fed back to the sender. The sender continuously receives feedbacks from the receivers. If a receiver sends back a rate, which is lower than the sender's current rate, the sender will immediately reduce its sending rate to that rate. Feedback suppression is achieved by using a random feedback timer method. The major pitfall of TFMCC is

that it is difficult to set timers to effectively suppress feedbacks while responding to congestion promptly. This is especially true when TFMCC is deployed in a wide-area and heterogeneous environment.

3.9 Simulation Results

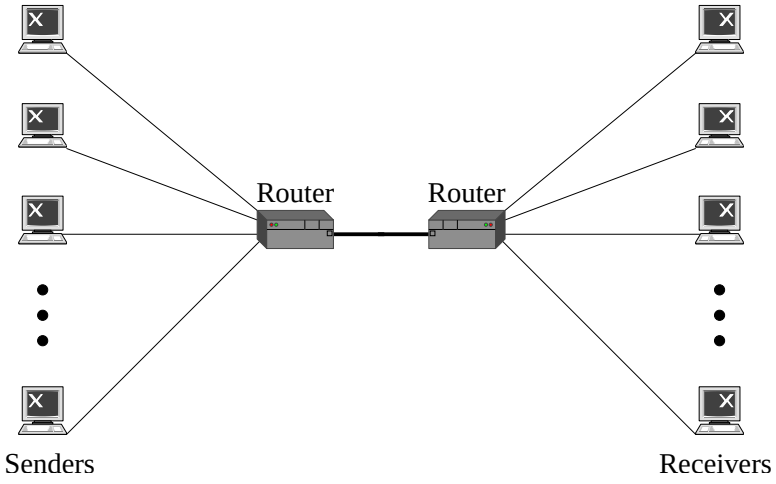


Figure 3.8: A network topology with a simple bottleneck link used in the simulation.

We have implemented *RACCOOM* on *ns-2* [50], and conducted a simulation study to (i) validate the proposed design in terms of fairness, TCP-friendliness, capability to deal with persistent congestion and membership change; to (ii) measure the overhead for ACK aggregation; and to (ii) compare *RACCOOM* against RAP [80] and other formula-based connections (such as CMTCP [72]). All algorithms used in the simulation, except *RACCOOM*, were part of the standard *ns-2* distribution. In particular, the codes for implementing RAP and formula-based approaches are directly downloaded from the *ns-2* website.

We consider network topologies with a single bottleneck link (Fig. 3.8) and multiple bottleneck links (e.g., Fig. 3.9), and arbitrary network topologies (e.g., Figs. 3.10– 3.11) generated by GT-ITG. To test the performance and scalability of *RACCOOM*, most of the experiments are made on topologies that are composed of hundreds of receivers. Also, we use an assortment of traffic sources (mainly infinite-duration TCP, finite-duration TCP, and on-off UDP sources). Due to the space limitation, we report below only results for the cases of infinite-duration RACCOOM, TCP and UDP sources (unless otherwise stated).

In spite of quite a number of system parameters (topology, link capacity, buffer size, and packet size) and algorithm parameters (α, β, k_1, k_2, location of designated receivers) involved, the results are found to be quite robust in the sense that the conclusion drawn from the performance curves for a representative set of parameters is valid over a wide range of parameter values (unless otherwise stated). In particular, the performance of RACCOOM is rather insensitive to the values of k_1 and k_2 (as long as they are appropriately chosen to drive the system into equilibrium). In all the experiments reported below, k_1 and k_2 are fixed at 4 and 0.25, respectively. The initial sending rate of a *RACCOOM* session is set to 5K bytes/second. Packets are of length 500 bytes. The values of α and β are on-line adjusted (using Eq. (3.10)) in experiments in which TCP-friendliness is the performance measure. Designated receivers are selected using the algorithm reported in [46].

The performance metrics of interest are the "friendliness ratio," F, defined as

$$F = \frac{r_{RACCOOM}}{r_{TCP}},$$

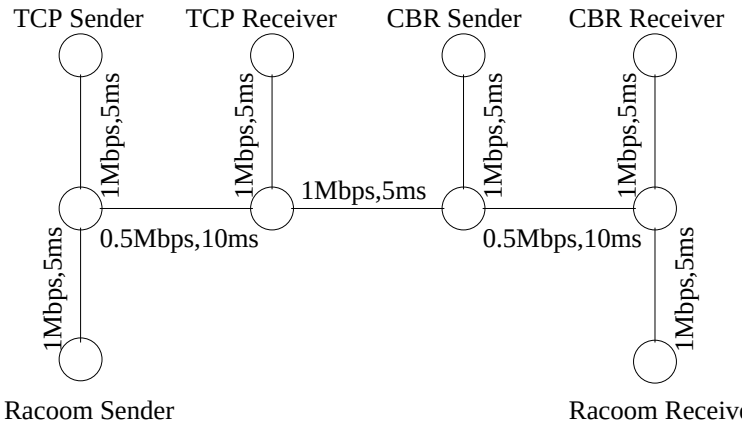


Figure 3.9: A network topology with multiple bottleneck links used in the simulation

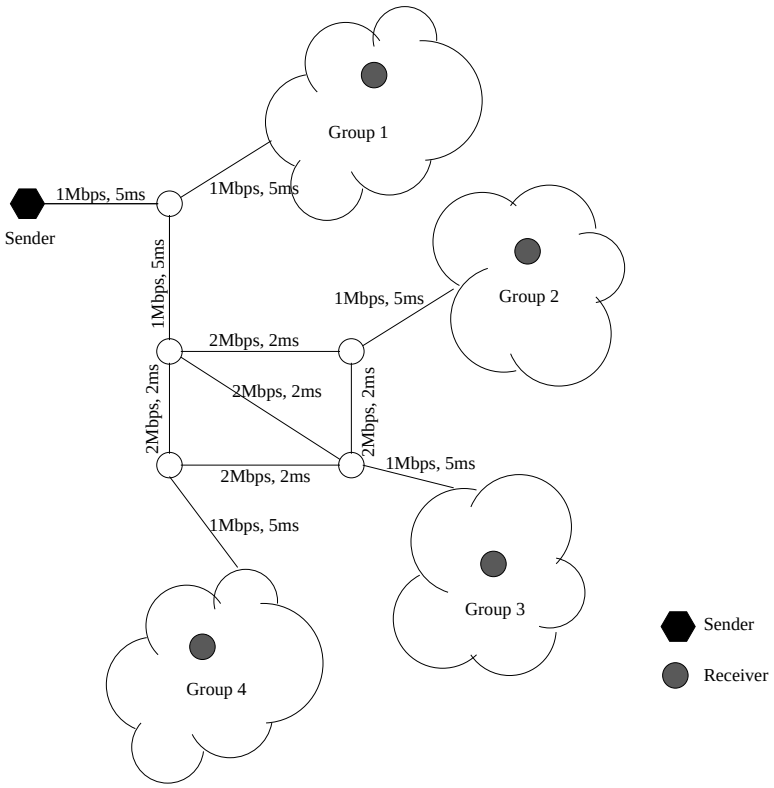


Figure 3.10: An arbitrary network topology used in the simulation.

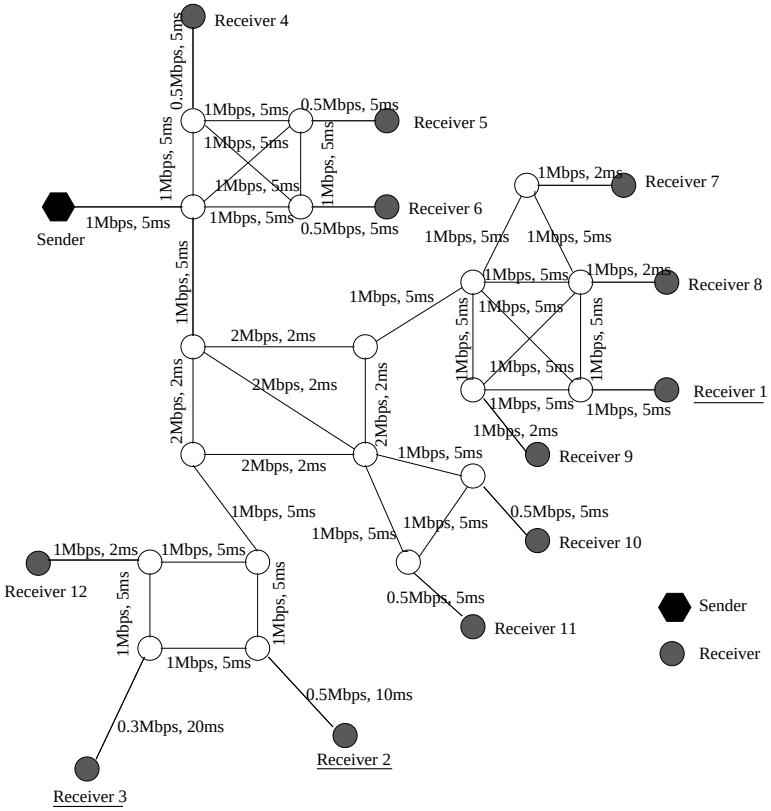


Figure 3.11: Another arbitrary network topology used in the simulation.

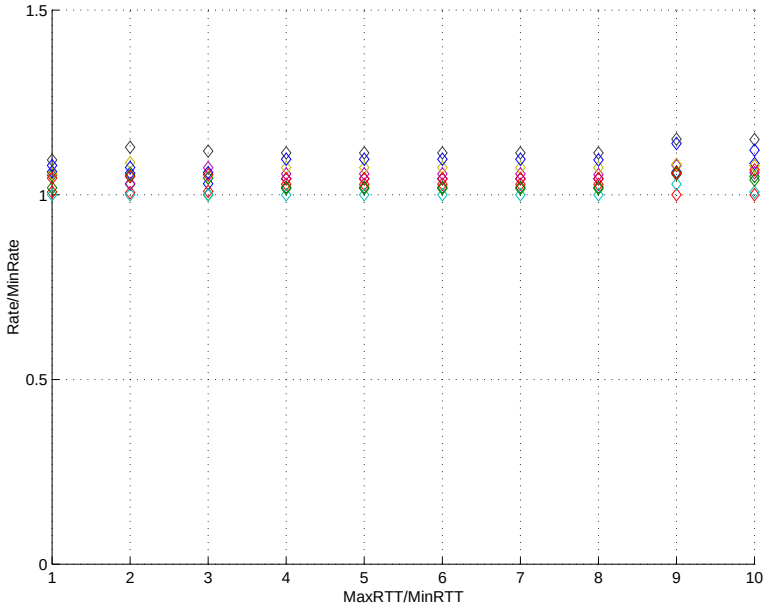
where $r_{RACCOOM}$ and r_{TCP} are the attainable throughput of a *RACCOOM* session and a TCP connection under competition; and the fairness ratio, F_r, of a *RACCOOM* connection to the other competing *RACCOOM* connections, defined as

$$F_r = \frac{r_i}{\min_j r_j},$$

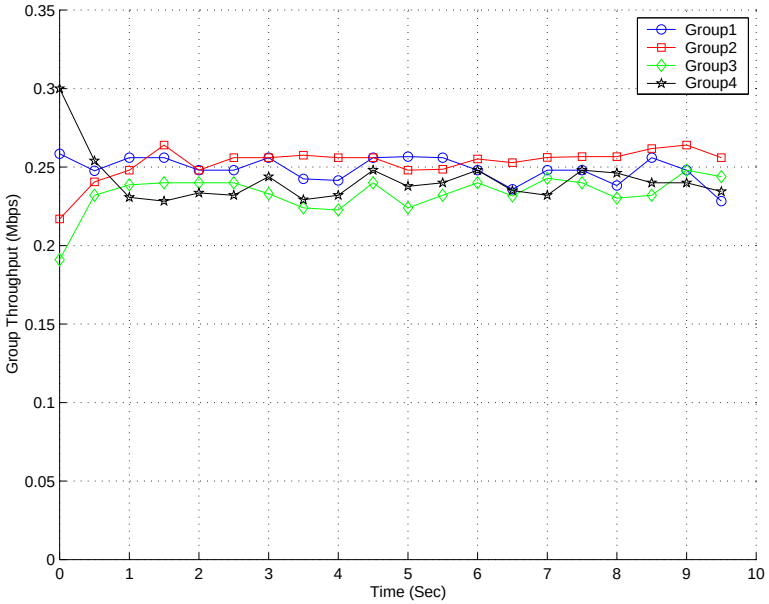
where r_i is the sending rate of the *RACCOOM* connection under consideration and $\min_j r_j$ is the minimum value of the sending rates of the other competing *RACCOOM* connections.

3.9.1 Validation of *RACCOOM* Properties

Fairness amongst *RACCOOM* connections: In the first and second experiments, we study fairness amongst *RACCOOM* connections with different RTTs. In the first experiment, the single-bottleneck network topology is used. The bottleneck link is shared among 10 *RACCOOM* connections and has a capacity of 0.5 Mbps, a delay of 20 ms, and a buffer of size 10 Kb. The links between a sender and the left router and between a receiver and the right router have a capacity of 1 Mbps and a delay that ranges from 5 to 140 ms in different simulation runs. A total of 10 simulation runs are conducted. In the ith ($1 \leq i \leq 10$) simulation run, the delays between senders/receivers and their attached routers are evenly distributed in $[5, 15 * i - 10]$ ms. Thus, the maximum value, *MaxRTT*, of the RTTs among the 10 connections is $\frac{2(15*i-10)+20}{2*5+20} = i$ times of the minimum value, *MinRTT*, of RTTs. The values of α and β are set to 2 and 6, respectively, for all 10 connections in each of the 10 simulation runs. Fig. 3.12 (a) gives the simulation results in terms of F_r with respect to $\frac{\text{MaxRTT}}{\text{MinRTT}}$. As shown in Fig. 3.12 (a), the values of F_r's ranges from 1.0 to 1.15 in all



(a) single bottleneck topology



(b) arbitrary topology

Figure 3.12: The performance of *RACCOOM* in terms of F_r.

the simulation runs, i.e., the bottleneck link bandwidth is shared, independently of RTT, among the 10 *RACCOOM* connections in a fair manner.

The second experiment is conducted in the arbitrary network topology given in Fig. 3.10: there are four multicast groups, each with 100 receivers. Four *RACCOOM* multicast sessions are established between the sender and the receivers. Four groups share the bottleneck link of 1 Mbps. Receivers are randomly located inside their associated groups. The values of α and β are set to 2 and 6, respectively, for all 4 sessions. Fig. 3.12 (b) gives the sending rate attained by the four sessions in a period of 10 seconds. Consistent with the analysis in Section 3.6, after an initial transient time, each of the *RACCOOM* sessions approximately attains a throughput of 0.25 Mbps. Both the first and second experiments validate the fairness analysis in Section 3.6.

Weighted fair share amongst *RACCOOM* connections: In the third experiment, we evaluate *RACCOOM* in terms of its weighted fairness capability. The single-bottleneck network topology is used. The bottleneck link has the same attributes as in the first experiment, and is shared by 31 *RACCOOM* connections. One *RACCOOM* connection is used as the reference connection (with $\alpha = 2$, $\beta = 6$, and $E = \frac{\alpha+\beta}{2} = 4$), and the other 30 *RACCOOM* connections are evenly grouped into 3 groups. The connections in the first, second, and third group are assigned a weight of 2, 3, 4, respectively, and the (α, β, E) values used for the connections in the first, second, third group are $(6, 10, 8)$, $(10, 14, 12)$, and $(14, 18, 16)$, respectively. The links between a sender and the left router and between a receiver and the right router have a capacity of 1 Mbps. The delays of those links for

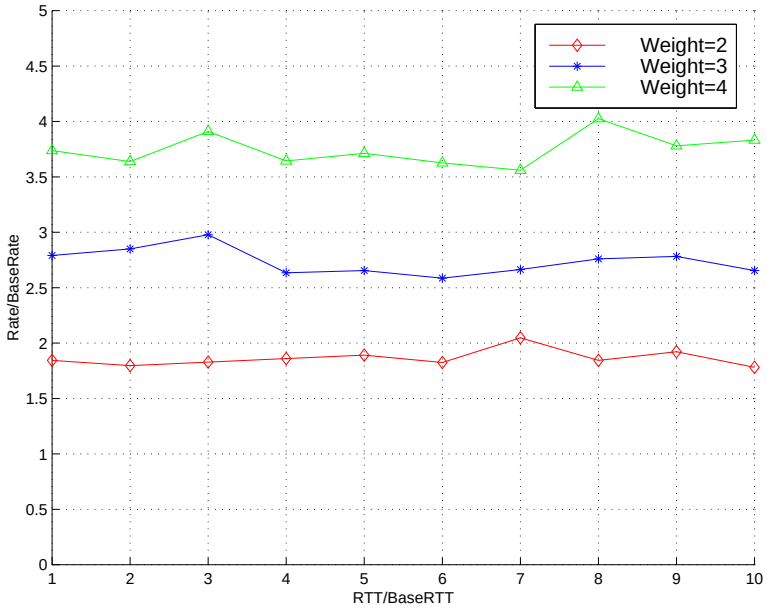
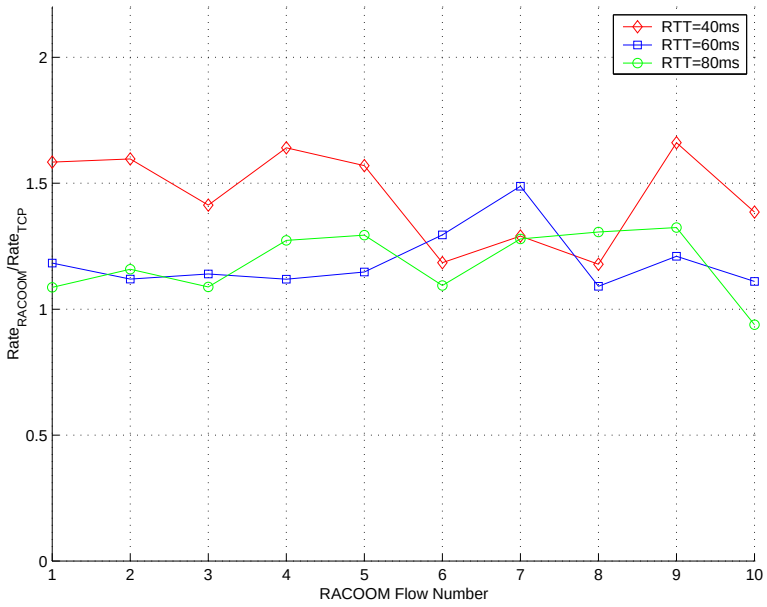


Figure 3.13: The performance of *RACCOOM* in terms of weighted fairness.

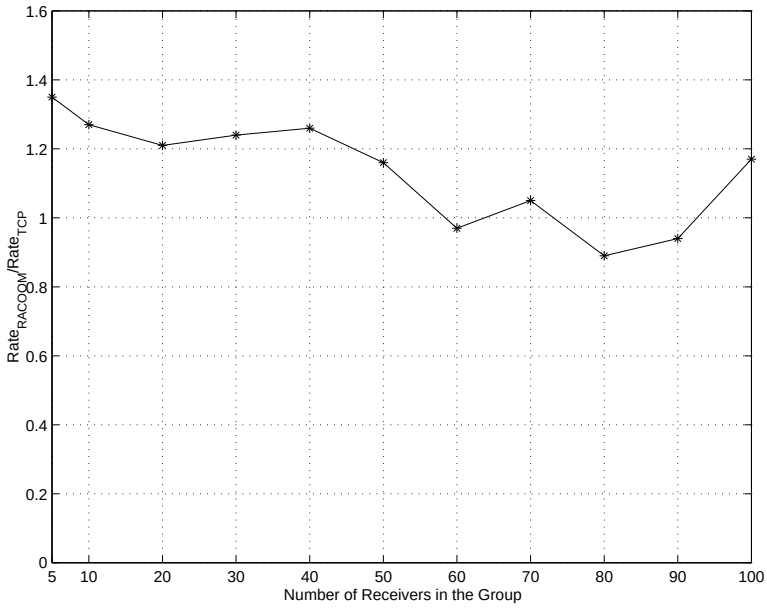
reference connection are 5ms. For other connections, they vary from 5 to 140 ms. Within each group, the values of the RTTs used for the 10 connections then vary from 1 to 10 times of that used for the reference connection.

As shown in Fig. 3.13, the ratio of the sending rate of a group-i connection to that of the reference connection is approximately equal to the weight assigned to the connection, and is rather insensitive to RTT changes. This validates the analysis (Section 3.6) that with the expected queue length along the target path appropriately assigned, *RACCOOM* can achieve weighted fairness among competing connections.

TCP-friendliness: In the fourth and fifth experiments, we evaluate *RACCOOM* in terms of TCP friendliness. In the fourth experiment, the single-bottleneck network topology is



(a) single bottleneck topology



(b) arbitrary topology

Figure 3.14: The performance of *RACCOOM* in terms of TCP-friendliness.

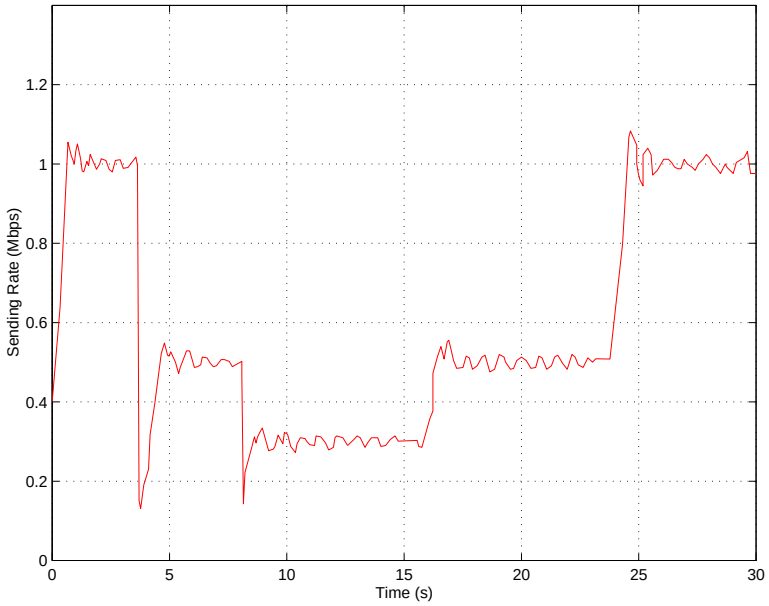
used. The bottleneck link has the same attributes as in the first experiment, and is shared by 1 *RACCOOM* connection (with $k_1 = 2$, and $k_2 = 0.25$) and 10 TCP connections. The values of α and β used in *RACCOOM* are on-line adjusted according to Eq. (3.10). Three simulation runs are conducted. The values of RTTs are the same for all the connections and are set to 40, 60, and 80 ms in the three simulation runs, respectively.[7] Fig. 3.14 (a) gives the ratio of the bandwidth attainable by the *RACCOOM* connection to that by each TCP connection. The ratio is very close to 1 for all the TCP connections in all three simulation runs. This shows that *RACCOOM* achieves TCP-friendliness, for a wide variety of RTT values.

To verify whether or not *RACCOOM* sessions still exhibit TCP-friendliness in arbitrary network topologies, we conduct simulation in an arbitrary network topology given in Fig. 3.10 in the fifth experiment: one *RACCOOM* multicast session is established between the sender and each multicast group. The group size of each multicast group varies from 5 to 100. In addition, one TCP connection is established between the sender and the receiver that lies on the target path in each multicast group. All the connections share the bottleneck link of 1 Mbps. Fig. 3.14 (b) gives the ratio of the sending rate of a *RACCOOM* session to that of the corresponding TCP connection of group 1. The ratios ranges from 0.8 to 1.5, which indicates that *RACCOOM* can achieve TCP-friendliness in an arbitrary topology with numerous receivers.

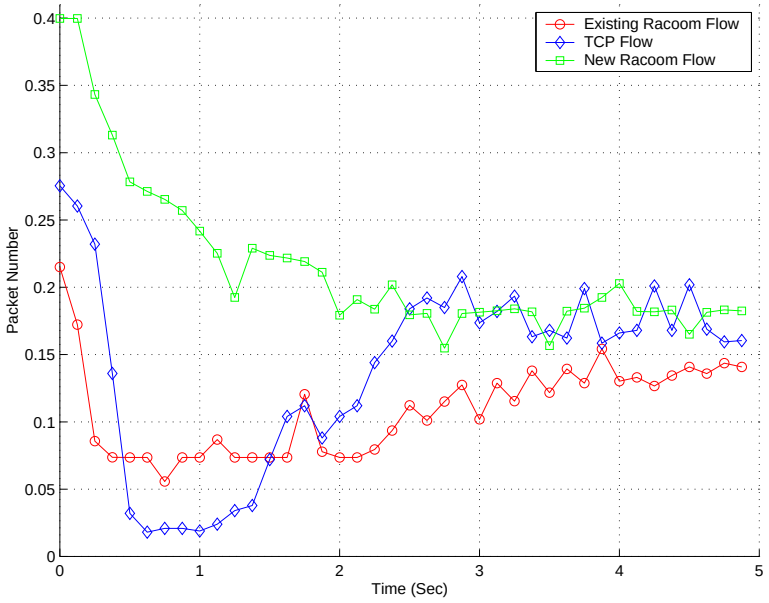
[7] As the RTT value from a host at OSU to a host at UCSB is approximately 60 ms, we consider the range of $[40, 80]$ in which RTT varies adequate.

Performance in the case of membership change: In the sixth experiment, we study how *RACCOOM* responds to membership change. We use the arbitrary network topology given in Fig. 3.11 (with the link delay and the link bandwidth labeled in the figure). There is one *RACCOOM* multicast session (with $\alpha = 2$ and $\beta = 6$), with one sender and 400 hundred receivers. Among the receivers, 12 representative receivers that are on potential target paths are labeled in the figure. The RTT values and the bandwidth of the bottleneck links between the sender and the receivers are listed in Table. 3.1. Initially only receivers 1, and 4–12 are in the multicast group, and the on-tree path from the sender to receiver 1 incurs the largest RTT value and is identified as the target path. Hence, the throughput the *RACCOOM* session can attain is constrained by the bandwidth of the bottleneck link (1 Mbps) on the target path. At time instant 3, receiver 2 joins the group and introduces a new target path with RTT$= 64$ ms and the bottleneck link bandwidth of 0.5 Mbps. At time instant 8, receiver 3 joins the group and introduces a new target path with RTT$= 94$ ms and the bottleneck link bandwidth of 0.3 Mbps. Finally, at time instant 16 and 24, receiver 2 and 3 leave respectively.

As shown in Fig. 3.15 (a), the sending rate of the sender is initially 1 Mbps, reduced to 0.5 Mbps when receiver 2 joins the multicast group (at time instant 3), further reduced to 0.3 Mbps when receiver 3 joins the group (at time instant 8), increased to 0.5 Mbps when receiver 3 leaves (at time instant 16), and finally increased to 1.0 Mbps when receiver 2 leaves (at time instant 24). This demonstrates the capability of *RACCOOM* in keeping track of the correct target path in the case of membership change.



(a)



(b)

Figure 3.15: The performance of *RACCOOM* in terms of (a) the capability to deal with membership change in the arbitrary topology and (b) the capability to deal with persistent congestion.

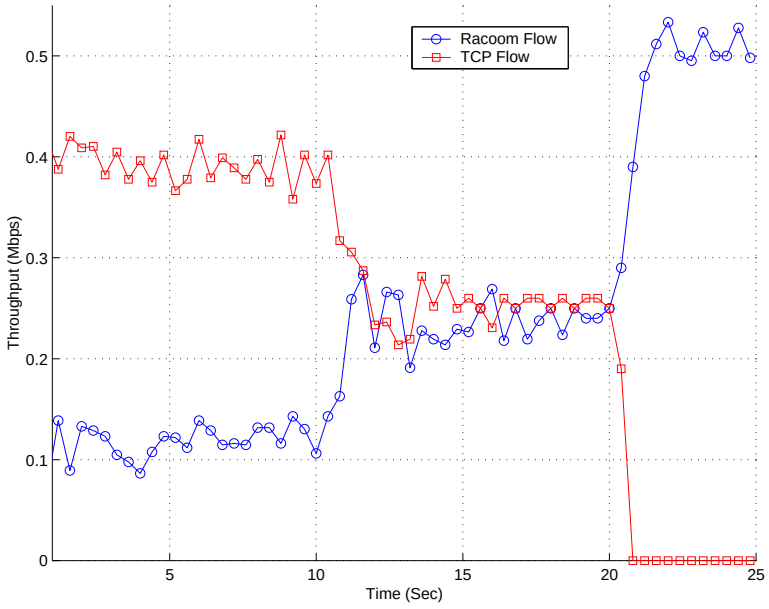
Receiver	BW(Mbps)	RTT(ms)
1	1	54
2	0.5	64
3	0.3	94
4,5,6	0.5	30
7,8,9,12	1	48
10,11	0.5	44

Table 3.1: The RTT values and the bandwidth of bottleneck links between the sender and the receivers.

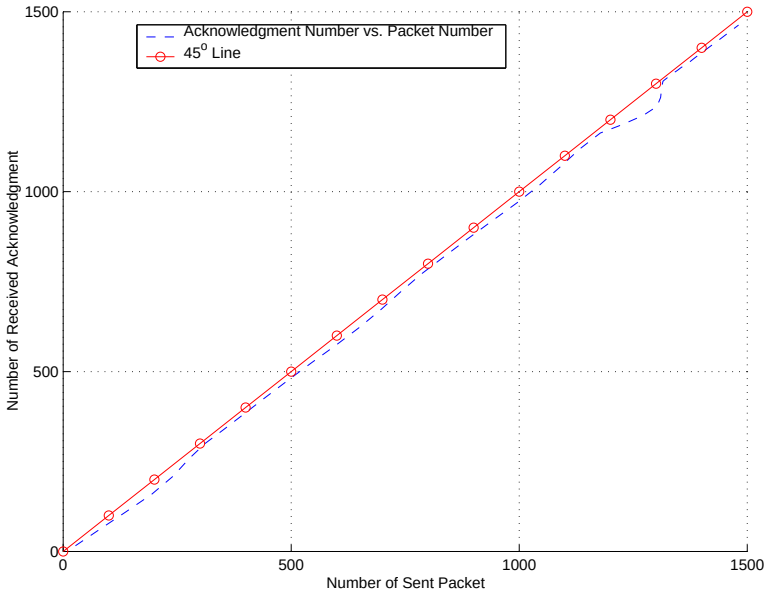
Performance in the case of persistent congestion: In the seventh experiment, we study the effect of persistent congestion on the performance of *RACCOOM*. The single-bottleneck network topology is used. The bottleneck link has a capacity of 0.5 Mbps, a latency of 30 ms, and a buffer of 10 Kb. The links between a sender and the left router and between a receiver and the right router have a capacity of 1 Mbps and a delay of 10 ms. Initially the bottleneck link is shared, and fully utilized, by a TCP connection and a *RACCOOM* connection (with the values of α and β on-line adjusted). At time instant 0, a new *RACCOOM* connection is established. Since the bottleneck link is fully utilized by existing connections, the new *RACCOOM* connection will over-estimate the value of RTT_{min} under the persistent congestion condition. Fig. 3.15 (b) depicts the throughput attainable for each of the three connections for a time period of 5 seconds. Initially the new *RACCOOM* connection starts with a high sending rate because of the over-estimated value of RTT_{min}. As a result, congestion occurs and packets are dropped at the bottleneck link. When both the existing and new *RACCOOM* senders detect packet loss, they reduce their sending rates by half, but only once every RTT. In contrast, the TCP connection repeatedly reduces its

congestion window by half, and at certain point, even shuts down the congestion window. After a transient period of approximately 2.8 seconds, the sending rates of the three connections stabilize at approximately 0.16 Mbps and the bandwidth of the bottleneck link is evenly shared by the three connections.

Performance in the existence of multiple bottleneck links: In the eighth experiment, we analyze how the throughput attainable by a *RACCOOM* connection is affected when the connection traverses more than one bottleneck link. The multiple-bottleneck network topology shown in Fig. 3.9 is used. Both bottleneck links have a capacity of 0.5 Mbps, a latency of 10 ms, and a buffer of 10 Kb. The capacity and latency of other links are 1 Mbps and 5 ms, respectively. One bottleneck link is shared by a *RACCOOM* connection and a TCP connection (that lasts for 20 seconds), and the other is shared by the same *RACCOOM* connection and a UDP CBR connection (that sends at 0.4 Mpbs and lasts for 10 seconds). The values of α and β of the *RACCOOM* connection are on-line adjusted. As shown in Fig. 3.16 (a), because of the existence of the non-responsive UDP CBR connection in the time interval of [0,10 sec], the *RACCOOM* connection only attains the throughput of 0.1 Mbps (which is the bandwidth left on the second bottleneck link). The TCP connection that traverses the other bottleneck link thus attains the throughput of 0.4 Mbps. When the UDP CBR connection terminates at time = 10 seconds, there is only one bottleneck link on which the *RACCOOM* connection shares the bandwidth with the TCP connection in a TCP-friendly manner (each attains a throughput of 0.25 Mbps). When the medium-duration



(a)



(b)

Figure 3.16: The performance of *RACCOOM* (a) in the existence of multiple bottleneck links and (b) in terms of ACK aggregation.

TCP connection terminates at time = 20 seconds, the *RACCOOM* connection responds by capturing all the bandwidth on the bottleneck link.

Ack aggregation for scalability: In the ninth experiment, we measure how many ACK messages arrive at a *RACCOOM* sender. The arbitrary network topology given in Fig. 3.11 is used. The simulation setup is the same as that in the sixth experiment, except that both the router-assisted and designated receiver-assisted ACK aggregation approaches are used in two simulation runs. Fig. 3.16 (b) gives the number of ACK messages received versus the number of data packets sent. (The results obtained under both aggregation approaches are indistinguishable from each other and only one curve is depicted.) Ideally (one ACK for each data packet sent), the result should be a 45 degree straight line. As shown in Fig. 3.16 (b), the result even falls below the 45-degree line, due to the round trip delay between sending of a data packet and receipt of its corresponding acknowledgment. This also shows that the ACK aggregation mechanism used in *RACCOOM* indeed aggregates ACK messages effectively and prevents the occurrence of ACK implosion.

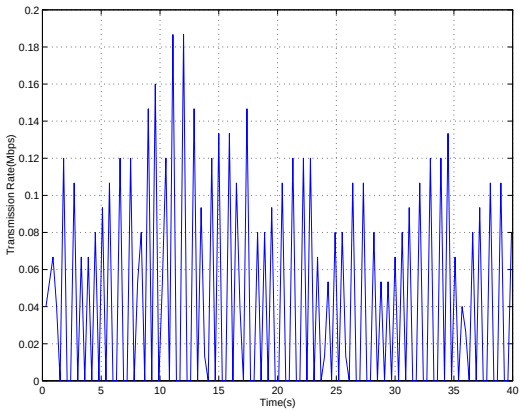
3.9.2 Comparison with Other Schemes

In this subsection, we compare the performance of TCP, RAP, *RACCOOM*, and CMTCP in terms of smoothness in the transmission rate and TCP friendliness. As TCP, RAP, and CMTCP are all targeted for unicast connections, the comparison is only made among unicast connections.

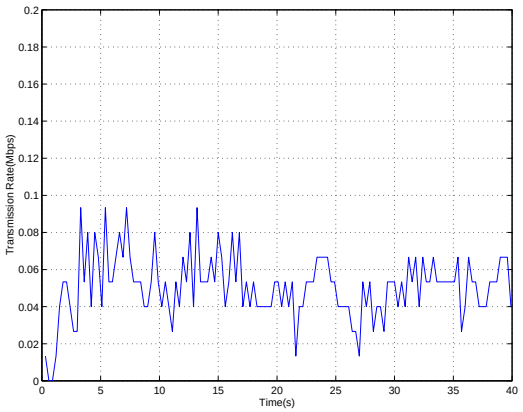
Smoothness in the transmission rate: The single-bottleneck network topology is used. The simulation setup is the same as that in the first experiment, except that the bottleneck link is shared by 10 identical connections that employ a specific congestion control scheme. The links between a sender and the left router and between a receiver and the right router have a capacity of 1 Mbps and a latency of 30 ms. The parameters used in RAP and CMTCP are tuned (according to the guideline that comes with the *ns-2* distribution) to achieve the best performance.

Fig. 3.17 shows the instantaneous goodput of a connection under TCP, RAP, *RACCOOM*, and CMTCP, respectively. The TCP connection incurs the largest variation in the transmission rate. The other approaches are all rated-based, and hence incur smaller variations as compared to TCP. Among the three rate-based approaches, RAP adopts TCP's additive increase and multiplicative decrease method to adjust its sending rate, and hence incurs the largest variation. CMTCP, on the other hand, incurs the smallest rate variation.

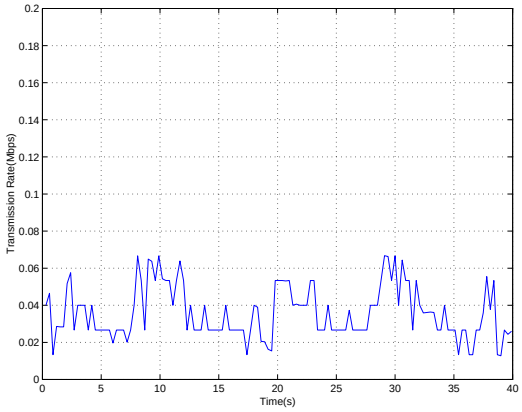
TCP friendliness: In this experiment, the single bottleneck topology is use. In each simulation run (i.e., each data point in Fig. 3.18), the bottleneck link is shared by ℓ TCP connections and ℓ connections that employ a specific scheme (e.g., RAP, *RACCOOM*, or CMTCP), where ℓ varies from 1 to 50. The bottleneck link has a bandwidth of 50Kbps $\times 2\ell$ and a delay of 20 ms. The links between a sender and the router and between a receiver and the router have a capacity of 1 Mbps and a delay of 3 ms. The size of data and ACK packets is 500 bytes and 40 bytes, respectively.



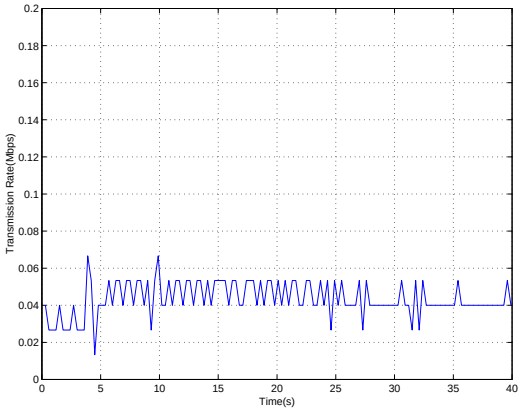
(a) Transmission rate of a TCP connection



(b) Transmission rate of a RAP connection



(c) Transmission rate of a *RACCOOM* connection



(d) Transmission rate of a CMTCP connection

Figure 3.17: Performance comparison (in terms of the variation in the transmission rate) among TCP, RAP, *RACCOOM* and CMTCP.

Fig. 3.18 gives the performance comparison (in terms of TCP friendliness ratio F) among RAP, *RACCOOM*, and CMTCP. All the approaches are not perfectly TCP friendly when the number of connections is small. In particular, RAP and CMTCP sustain a TCP friendliness ratio of $F = 2 – 2.5$. The reason why formula-based approaches do not perform well is because when the number of connections is small, the packet loss probability estimated by these approaches may not be accurate, as dropped packets may not be evenly distributed among connections. When the number of connections is large, all the approaches achieve reasonable TCP friendliness, among which *RACCOOM* performs best.

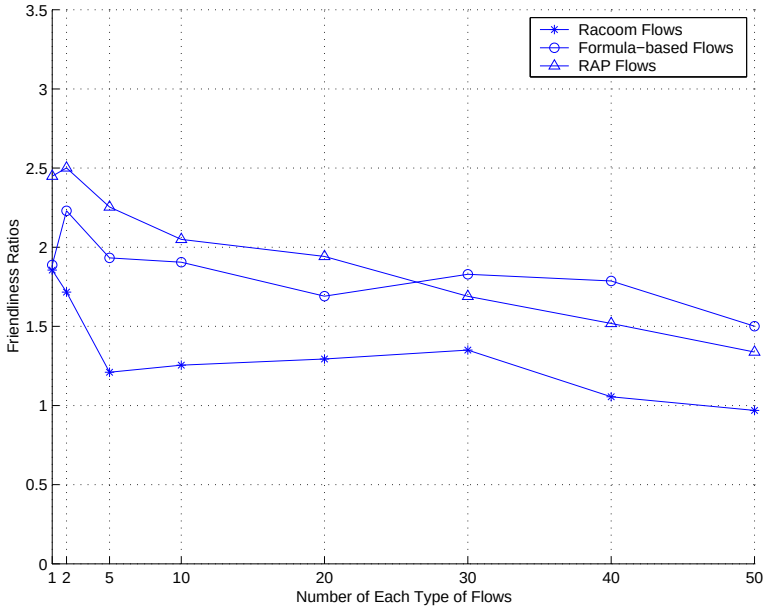


Figure 3.18: The performance comparison (in terms of TCP friendliness) among RAP, *RACCOOM*, and formula-based approaches.

3.10 Conclusion

We have presented in this chapter a rate-based congestion control scheme, *RACCOOM*. In the absence of packet loss, a *RACCOOM* sender adjusts its sending rate in a TCP-Vegas fashion, based on the congestion status of the on-tree path with the largest RTT (called the *target* path). In case of packet loss, a *RACCOOM* sender then responds by reducing its sending rate by half. An ACK aggregation method is judiciously devised to prevent ACK implosion and yet to provide the sender with a simple but comprehensive view of congestion conditions in the multicast tree. *RACCOOM* also achieves (weighted) fairness among competing connections by exploiting feedback control theory and appropriately selecting the parameters used in the rate adjustment mechanism. On the other hand, if TCP friendliness is the performance criterion, then a simple iterative approach can be used to on-line adjust the parameters α and β so as for a *RACCOOM* session to exhibit TCP-friendliness.

Simulation experiments indicate that *RACCOOM* connections can achieve, irrespectively of the RTTs of individual connections, TCP-friendliness, can handle membership/network traffic changes, can deal with persistent congestion, and can achieve (weighted) fairness among competing connections with different RTTs. In terms of the level of TCP friendliness, *RACCOOM* also outperforms RAP and other formula-based approaches (e.g., CMTCP).

CHAPTER 4

COCOON: An Alternative Scheme for End-Point Congestion Management

In this chapter, we present an endpoint congestion management (ECM) scheme, called COordinated COngestion cONtrol (*COCOON*). The basic idea is to identify and group connections that are destined for the same destination host/subnet and coordinate among them all the congestion avoidance/control operations. We first give the technical motivation and design objectives and present an overview of *COCOON* (Section 4.1). Following that, we delve into the detailed description of each component mechanism (Section 4.2-Section 4.5). In particular, we elaborate on how groups are dynamically merged or split, and to which group a new connection should join. Then we summarize related work (Section 4.6) and present a performance study that involves both *ns-2* simulation and Internet experiments (Section 4.7).

Motivation: As reported in [92], TCP traffic constitutes the majority — as high as 95% — of the Internet traffic, and World Wide Web (WWW) constitutes approximately 65–75%

of the TCP traffic today. This is because web servers and clients communicate with each other using the Hypertext Transfer Protocol(HTTP/1.0 or HTTP/1.1) [27,29], which in turn uses TCP as the underlying reliable transport protocol [28]. Consequently, the Web traffic has become the dominant component of the Internet backbone traffic. This observation is corroborated by the statement in [5] that the Web traffic constitutes 50% of the bytes traversing busy backbone links.

The rapid growth of WWW traffic has caused significant characteristic changes in the Internet data traffic. This is because the characteristics of WWW traffic are significantly different from those of other traditional TCP applications (such as Telnet and FTP) in the following three aspects. First, a web page typically contains several embedded components (e.g. images). In order to reduce the latency perceived by users, a Web browser often launches multiple, simultaneous TCP connections to a given server to obtain the embedded objects of a web page. Second, the average size of a web page and its embedded components is usually small, and hence the TCP connections initiated to retrieve a web page and its embedded components are usually short-lived. Third, Web surfing is inherently an interactive activity, and from the user's perspective, the response time is perhaps the most important performance criterion.

The above distinct characteristics give rise to several issues, which the current TCP does not adequately deal with. First, most TCP connections initiated by HTTP are short-lived and seldom have enough time to probe, and fully utilize, the available network bandwidth. Hence, they may experience unnecessarily long response times. This is because each TCP connection commences the *slow-start* phase with a small window size. For a long-lived

connection, e.g., a *FTP* connection that transfers a large file, this *slow-start* period is short as compared to the entire duration of the connection, and the average window size over the entire duration is reasonably large. It is, however, not the case for a short-lived connection, as the window size may not reach the congestion avoidance threshold (*ssthresh*) before the connection completes its transmission. Second, use of multiple, concurrent connections for web page retrieval may lead to blind competition among these connections (and hence increased traffic burstiness).

HTTP/1.1 [27] alleviates the above problems by enabling a web client/server to transfer multiple components sequentially over one persistent TCP connection. The persistent connection approach avoids blind competition among concurrent connections, and allows a new connection to start with a larger congestion window (and hence the small window size in the *slow-start* phase does not impact the response time so dramatically). However, as reported in [6], only 21% of all (1.35 billion) requests to FIFA'98 websites used HTTP/1.1. A similar report is also found in [3] for requests made to the NASA's Glenn Research Center website over a one and half year time period. Moreover, use of HTTP/1.1 only alleviates the above problems in the case that a single client requests multiple components from the same server. Connections from a server destined for the same destination client host/subnet (e.g., employees from the same corporate may connect to the same website to quote/trade stocks simultaneously) and/or connections established by applications other than WWW may still compete for network resources with one another. This is corroborated by the research work on WWW clients [20], which shows that clients on the same subnet usually

share the same interest, and that computers in a lab (on the same subnet) access some popular websites in a much higher frequency than other websites. Hence it is highly likely that more than one clients from the same subnet visit a popular website at the same time. What is needed is an ECM scheme that is not application-specific.

Design Objectives: As discussed in Section 2.4.4, a good ECM scheme should not be application-specific and should and possess the following properties:

1. Multiple, concurrent connections that traverse the same backbone bottleneck link should not compete blindly, but rather cooperate with one another.
2. A new, short-lived connection should be allowed to start with a congestion window that is reasonably large so as to reduce latency, while not inducing congestion.
3. Connections should be treated fairly in the sense that no connection should starve, or attain more throughput than others. Also, connections that employ the ECM scheme should remain TCP-friendly.
4. The scheme should be easy to implement, should not require router support (so that it can be directly deployed on the Internet), and should scale well.

4.1 Overview

As mentioned above, the basic idea of *COCOON* is to identify and group connections that are destined for the same destination host/subnet and coordinate among them all the congestion avoidance/control operations (Fig. 4.1). The granularity of groups can be dynamically adjusted so as to magnify the advantage of control information sharing

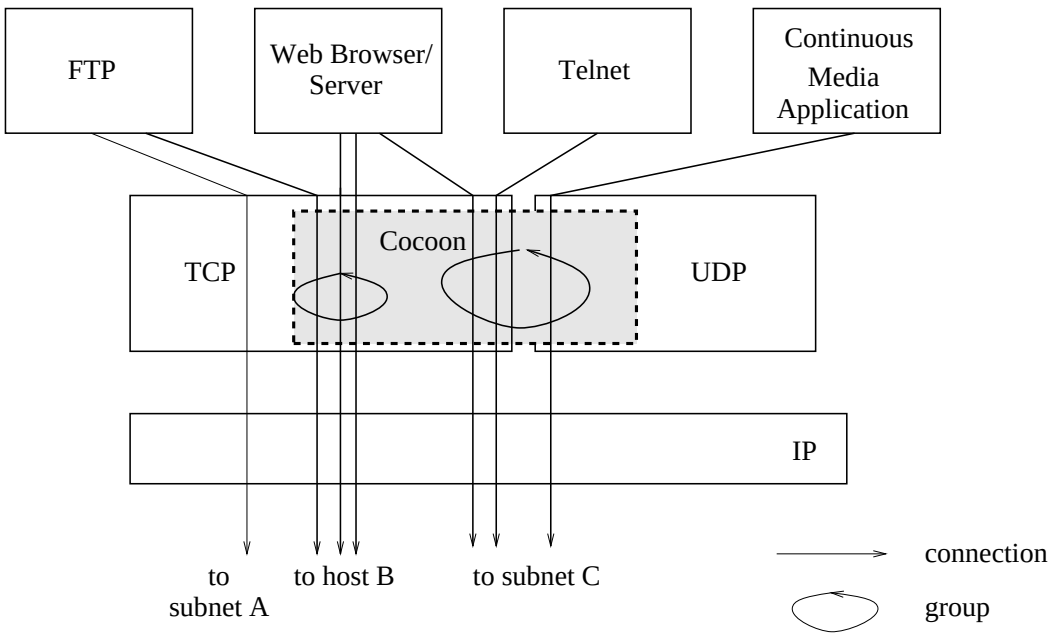


Figure 4.1: *COCOON* in the network protocol stack.

and congestion control coordination. Co-existing with TCP/UDP in the Internet protocol stack, *COCOON* also regulates non-responsive UDP traffic and bundles UDP connections destined for the same destination host/subnet into a virtual connection that is subject to TCP-like congestion control. *COCOON* requires only minor modification at server hosts, is totally transparent to client hosts, and hence can be incrementally deployed over the Internet. Both our *ns-2* simulation and *FreeBSD* empirical studies indicate that as compared to *TCP-Reno*, *TCP-Int*, *Ensemble-TCP*, CM HTTP/1.0, and HTTP/1.1, *COCOON* significantly reduces the packet loss rate, while sustaining the throughput comparable to the best scheme.

COCOON resides at each sender host, and identifies and groups concurrent TCP and UDP connections that are destined to the same destination host or subnet. The rationale behind this design is that these connections are likely to be routed along some common backbone links, are subject to similar congestion characteristics and RTTs, and hence should coordinate on their congestion control activities. *COCOON* enables these connections to share the congestion information so that (i) they can, rather than competing blindly, cooperate with one another in congestion control; (ii) new connections may leverage the shared congestion information and commence with a reasonably large congestion window; and (iii) UDP connections can be regulated so as not to deprive well-behaved TCP connections of their fair resource share and exhibit TCP-friendly behavior.

The granularity of the group of connections that are subject to the same "fate" can be dynamically adjusted. The finest grain, called an *atom*, is the bundle of all TCP/UDP connections destined for the same destination host. Connections whose destination hosts have the same network prefix (e.g., the IP addresses of destination hosts AND'ed with a subnet mask) can also form a group, if they are "observed" to share certain backbone links. Under this case, a group may consist of several atoms.

4.2 Congestion Information Shared in a *COCOON* Group

As all the connections in a group are likely to be routed along the backbone links and hence are subject to the same RTT (except for the minor variation among different subnets), they should share the information of RTT and RTT variance. The way in which a

group estimates its group RTT and group RTT variance is the same as that in which a TCP connection estimates these parameters [90]. That is, when a connection in a group receives a non-duplicate acknowledgment, the group estimates its group RTT (RTT variance) as the weighted average of the previous estimate and the new measured value. Should connections that traverse the same backbone links be group into a group, the parameter thus estimated is more accurate, because approximately n times more samples are used, where n is the number of connections in the group.

The group RTT and RTT variance are used to set up retransmission timers for all the connections in the group, in the same way in which a TCP connection sets up its retransmission timer. Note that all the information shared is the information TCP typically uses for congestion control, but not the user information, and hence security/privacy is not compromised.

4.3 Coordinated Congestion Control for TCP Connections in a *COOON* Group

In addition to sharing congestion information, all the connections in a group coordinate on congestion control activities as follows:

When a connection detects congestion: When a connection C_i receives 3 duplicate acknowledgments (which is taken as the signal for packet loss), the congestion window of C_i (*cwnd*$_i$) is reduced to half of its outstanding data size, *FlightSize*$_i$, as described in [26] (i.e.

/* Upon receipt of 3 duplicate ACKs for the packet with sequence number k,
connection C_i performs the following operation */
If ($k >$ *watermark*) {
    $cwnd_i \leftarrow \frac{1}{2}$ *FlightSize*$_i$;
    inform connection C_j in the same group;
}

/* Upon being notified by connection C_i in the same group of its packet loss,
connection C_j performs the following operation */
If *FlightSize*$_j \geq$ *FlightSize*$_i$ {
    $cwnd_j \leftarrow \max(\frac{1}{2}$*FlightSize*$_j$, *FlightSize*$_i)$;
    ssthresh$_j \leftarrow cwnd_j$; /* congestion avoidance phase commences */
}
else if *FlightSize*$_j \geq \frac{1}{2}$ *FlightSize*$_i$ {
    $cwnd_j \leftarrow$ *FlightSize*$_j$;
    ssthresh$_j \leftarrow \frac{1}{2}$*FlightSize*$_i$; /* congestion avoidance phase commences */
}
else
    ssthresh$_j \leftarrow \min(\frac{1}{2}$*FlightSize*$_i$, *ssthresh*$_j)$;
/* update watermark to avoid repeated reduction of the window size within
one RTT. */
watermark $\leftarrow$ the largest sequence number of C_j's current outstanding data;

Figure 4.2: The window adjustment procedure in the case that a connection incurs packet loss.

cwnd$_i \leftarrow \frac{1}{2}$ *FlightSize*$_i$). Meanwhile, since packet loss may be a result of other concurrent connections aggressively probing available bandwidth, the other connections in the same group should also adjust their congestion windows. In *COCOON*, the congestion windows of connections in a group are adjusted as follows (Fig. 4.2):

(1) If the outstanding data size, *FlightSize*$_j$, of a connection C_j is greater than or equal to that of connection C_i, C_j may potentially be responsible for the congestion, and hence its congestion window size, *cwnd*$_j$, is reduced to the maximum value of $\frac{1}{2}$ *FlightSize*$_j$ and *FlightSize*$_i$. That is, if *FlightSize*$_j \geq 2\times$ *FlightSize*$_i$, the *cwnd*$_j$ is adjusted as if C_j experienced packet loss; otherwise, *cwnd*$_j$ is set to *FlightSize*$_i$. In the latter case, the reduction in the congestion window is not as aggressive as *cwnd*$_j \leftarrow \frac{1}{2}$ *FlightSize*$_j$ so that C_j does not experience significant throughput fluctuation. The slow start threshold, *ssthresh*$_j$, is also set to the same value of *cwnd*$_j$ so as to enforce C_j to enter the congestion avoidance phase.

(2) If *FlightSize*$_j$, of a connection C_j falls in $[\frac{1}{2}$ *FlightSize*$_i$, *FlightSize*$_i)$, then *cwnd*$_j$ is set to *FlightSize*$_j$, implying that C_j is not allowed to send any data until receipt of a new acknowledgment. Moreover, *ssthresh*$_j$ is set to $\frac{1}{2}$ *FlightSize*$_i$. Since *cwnd*$_j \geq$ *ssthresh*$_j$, the congestion avoidance phase commences.

(3) If *FlightSize*$_j$ of a connection C_j is less than $\frac{1}{2}$ *FlightSize*$_i$, then *cwnd*$_j$ is not adjusted because C_j does not grasp the bandwidth as aggressively as C_i and should not be penalized. However, *ssthresh*$_j$ is set the to minimum value of *ssthresh*$_j$ and $\frac{1}{2}$

FlightSize$_i$, so that C_j will commence the congestion avoidance phase (and behaves less aggressively) when its congestion window reaches $\frac{1}{2}$ *FlightSize*$_i$.

When a connection incurs timeout: When a congestion C_i incurs timeout, it commences the slow-start phase. The connections in the same group whose outstanding data size is larger than or equal to $FlightSize_i$ reduce their congestion windows to half of their outstanding data size, while those whose outstanding data size is less than $FlightSize_i$ do not reduce their congestion windows, but instead set *ssthresh*$_j \leftarrow \frac{1}{2}FlightSize_j$. Also, if the first unacknowledged packet of a connection was transmitted before the packet that incurs the timeout, the connection immediately retransmits its first unacknowledged packet, with the hope of avoiding a timeout.

When multiple packet losses or timeouts occur within one RTT: To prevent the congestion window from being reduced multiple times within one RTT, each TCP connection keeps a watermark which is set to the largest sequence number of its outstanding data at the last time the congestion window was adjusted. When a connection incurs packet loss or timeout, it compares the sequence number of the lost packet or the packet that incurs the timeout against its watermark. Only when the former is larger than the latter will the connection, along with the other connections in the same group, adjusts its congestion window.

4.4 Window Set Up for New Connections in a *COCOON* Group

To allow a new connection to start with a congestion window that is large enough to reduce latency while not inducing congestion, we propose the following window setup procedure in *COCOON:* When a new TCP connection, C_k, is established and added to a group, it chooses as its *reference connection* the TCP connection, C_r, with the smallest congestion window among all the *active* TCP connections in the group. (By "active", we mean connections with data to send.) C_k then sets its congestion window to $cwnd_k \leftarrow \frac{1}{2}cwnd_r$ and its slow start threshold to $ssthresh_k \leftarrow \min\{cwnd_r, ssthresh_r\}$. In the case that there is no active TCP connection in the group which C_k joins, then C_k follows the TCP slow start procedure and starts with $cwnd_k = 1$.

Note that by setting $cwnd_k \leftarrow \frac{1}{2}cwnd_r$, the new connection will catch up with the reference connection in one round trip time and will not suffer from the effect of the small window size in the slow start phase. On the other hand, by setting $ssthresh_k \leftarrow \min\{cwnd_r, ssthresh_r\}$, the new connection always starts in the slow start phase, but if the reference connection is itself in the congestion avoidance phase ($cwnd_r \geq ssthresh_r$), the new connection will enter the congestion avoidance phase in one RTT, thus not behaving too aggressively.

4.5 Coordinated Congestion Control for UDP Connections in a *COCOON* Group

To prevent non-responsive UDP connections from depriving TCP connections of their fair share of network resources, *COCOON* treats all the UDP connections in a group as a *virtual* connection. The virtual connection is then regulated by a *virtual* congestion window, $cwnd_v$. The rules for adjusting $cwnd_v$ and *FlightSize*$_v$ of the virtual connection and for regulating UDP connections are as follows:

(1) The congestion window, $cwnd_v$, of the virtual connection is set to the average congestion window (averaged over all the TCP connections in the group) times the number of UDP connections. If at the time when the virtual connection is established, no TCP connection exists in the group, $cwnd_v$ is set to a default value. The outstanding data size, *FlightSize*$_v$, is initialized to 0.

(2) When a virtual connection is established (i.e., some UDP connection(s) are established in a group), it is given the congestion window $cwnd_v$ calculated above. UDP connection(s) in a group are allowed to send packets in a round-robin fashion (for the purpose of fairness) among active UDP connections only if *FlightSize*$_v$ is less than or equal to $cwnd_v$. When a UDP packet is sent, *FlightSize*$_v$ is increased by the amount of data bytes sent. When a non-duplicate acknowledgment is received by a TCP connection in the group, $cwnd_v$ is updated (after the TCP connection's $cwnd$ is updated), and *FlightSize*$_v$ is reduced by the amount of TCP data bytes acknowledged.

This, in some sense, regulates the aggressiveness of a UDP connection to that of TCP connections.

(3) When a TCP connection incurs packet loss and reduces its congestion window, the virtual connection updates its $cwnd_v$ accordingly. If after the update, $\textit{FlightSize}_v \geq cwnd_v$, the virtual connection freezes its transmission until enough non-duplicate ACKs are received by TCP connections in the group and $cwnd_v > \textit{FlightSize}_v$.

(4) When a TCP connection times out, the virtual connection immediately freezes its transmission, no matter if $\textit{FlightSize}_v$ is less than or equal to $cwnd_v$. The virtual connection only resumes transmission (with a updated $\textit{cwnd}_v$), when a new acknowledgment is received by the TCP connection that incurs timeout.

(5) If all TCP connections in the group are either inactive (i.e., idle without sending packets) or closed, no acknowledgments will be received within the group. Under this case, $cwnd_v$ and $FlightSize_v$ remain unchanged and the virtual connection is allowed to send at the rate $\textit{cwnd}_v/\textit{group\_RTT}$, until one of the TCP connections becomes active again or a new TCP connection is established. During the period with no TCP acknowledgment, UDP connections may not respond to network dynamics, due to the lack of up-to-date control messages. [8]

[8] *COCOON* may deliberately avoid the situation by artificially creating a TCP connection with the same destination as the UDP connection. The TCP connection periodically sends one-byte date packets to probe the network status on the behalf of the UDP connection. As this paper is more focused on how *COCOON* prevents UDP connections from depriving TCP connections of their bandwidth shares, we do not discuss this issue.

4.6 Related Work

Several approaches have been proposed in the realm of ECM, among which T/TCP [14, 15, 89], *TCP-Int* [8], *Ensemble-TCP* [21], *TCP Fast Start* [73, 74], and most recently *Congestion Manager* (*CM*) [9] may have received the most attention.

The TCP extension for transaction (*T/TCP* [14, 15, 89]) was proposed to cache RTT, maximum segment size (MSS), congestion window size, and connection counter to expedite the catching up (of the sending rate) of new, repeated connections in the slow start phase. Touch [93] further proposed to share congestion-related information, such as the congestion window size and the estimate of RTT, among TCP connections. They also differentiated information sharing between existing TCP connections from caching of temporal information from previous connections.

TCP-Int was proposed by Balakrishnan *et al.* [8] in which all concurrent TCP connections destined for the same destination use the same congestion window. The congestion window increases when an acknowledgment is received by any connection and is halved whenever packet loss is detected by any connection. *Ensemble-TCP* proposed by Eggert *et al.* [21] leveraged this information sharing concept, and in addition, used caching of temporal information to avoid the three-way handshake operation and the slow-start phase for repeated connections.

Both *TCP-Int* and *ensemble-TCP* require the use of schedulers to schedule packet transmission so that TCP connections can share in some fair fashion the "quota" dictated by the congestion window. Another drawback of the single congestion window approach is that

if one connection incurs packet loss, the congestion window is halved. This may be too conservative, as it is equivalent to reducing the congestion windows of all the connections by half. In contrast, *COCOON* enables multiple, concurrent connections to have their individual congestion windows, but coordinates their congestion control activities. This eliminates the need for a packet scheduler. Also, when a single connection incurs packet loss, only those that may potentially be responsible for the congestion adjust their congestion windows. Finally, *COCOON* considers how to regulate, based on the control information carried in TCP traffic, non-responsive UDP connections in a TCP-friendly way.

TCP Fast Start [73, 74] used the cached information, such as the TCP congestion window and RTT, to expedite the start up of new connections. More packets can be sent in the slow start phase, and are marked with a higher dropping preference. In case of congestion, these packets will be dropped by intermediate *TCP-Fast-Start*-aware routers. Router support is thus required for the best performance.

In TCP trunking [48], a TCP trunk is first established and all the connections destined for the same host are multiplexed in the truck. The major drawbacks of TCP trunking are that a TCP trunk has to be established with the exchange of several, additional control messages and that packets have to be scheduled in order to share among connections the bandwidth of a trunk in some fair manner.

The work that comes closest to ours is the *CM* [9] work by Balakrishnan *et al*. *CM* maintains congestion parameters, performs congestion control/avoidance operations, and schedules packet transmission, for all the connections destined for the same destination. A *CM* sender ensembles concurrent unicast flows with the same destination host into a single

flow. All the flows are subject to a single congestion window governed by a window-based AIMD congestion control algorithm. Congestion parameters are shared among the flows. In particular, packet loss from any of the flows leads to multiplicative decrease of the congestion window. A *CM* sender also employs a rate-based traffic shaper and a scheduler to regulate packet transmission and to apportion available bandwidth among flows. In addition, *CM* uses a loss-resilient protocol to elicit feedback information periodically from receivers about losses and network status.

The major differences between *CM* and *COCOON* lie in (i) *COCOON* supports information sharing and coordinates congestion management not only among connections destined for the same host but also among connections destined for the same subnet; (ii) all concurrent connections are *not* subject to the same congestion window, but are instead governed by their individual congestion windows. What is being shared is the congestion information. This eliminates the need for traffic shapers or schedulers (to regulate/schedule packet transmission among flows). Moreover, a single packet loss will not significantly impact the instantaneous throughput attainable by the flows as in *CM*. (iii) *COCOON* does not introduce additional control protocol/messages to elicit feedback information; and (iv) *COCOON* requires modification only at senders (which are usually the servers) and is transparent to receivers.

4.7 Performance Evaluation

We have implemented *COCOON* both in *ns-2* and in *FreeBSD* 2.2.8, and conducted performance studies to (i) validate the proposed design in terms of packet loss rate, frequency of retransmission timeouts, attainable throughput, and response time, and (ii) compare *COCOON* against *TCP-Reno*, *TCP-Int*, *CM*, HTTP/1.0, and HTTP/1.1.

4.7.1 Simulation Study

In the simulation study, we compare *COCOON* against *TCP-Reno* and *TCP-Int* with respect to packet loss rate and response time. *TCP-Reno* is used as the base scheme, while the reason for selecting *TCP-Int* for comparison is because in the *ns-2* distribution, *TCP Fast Start* has been included in the *TCP-Int* implementation as part of the mechanism to expedite the start up of connections. The combination of *TCP-Int* and *TCP Fast Start* contains essentially all the features of *Ensemble-TCP*.

We consider network topologies with a single bottleneck link (e.g., Fig. 4.3), multiple bottleneck links (e.g., Fig. 4.4), and arbitrary network topologies. We use an assortment of traffic sources (mainly infinite-duration TCP, finite-duration TCP, and on-off UDP sources). Due to the space limitation, we only report on a small set of simulations that we believe is most representative. In spite of quite a number of system parameters (topology, link capacity, buffer size, object size, distribution of destination hosts) and algorithm parameters (M, N, and c) involved, the results are found to be quite robust in the sense that the conclusion

drawn from the performance curves for a representative set of parameters is valid over a wide range of parameter values (unless otherwise stated).

In the single-bottleneck network topology shown in Fig. 4.3, there are one sender (server) and n subnets (each of which contains a client), where n varies from 10 to 100. Totally n concurrent TCP connections are established for file transfer. The link bandwidth and delay between the sender and R_1 are 10Mbps and 2ms, respectively, while those between R_1 and R_2 are 1Mbps and 20ms, respectively. The link bandwidth and delay between R_2 and the first half of receivers are 10Mbps and 2ms, respectively, and those between R_2 and the other half of receivers are 10Mbps and (2+δ) ms, respectively, where δ is set to 0, 2, 4, 6, 8, 10 or 12 ms to assess the capability of *COCOON* group management. Each router employs a FCFS, drop-tail buffer management policy and is equipped with buffers of size 20 packets. The packet size is set to 500 bytes. Two file sizes are used: 30Kbytes (which is the average size of a webpage [59]) and 1M bytes (which represents large file transfer). Transfer of 30-Kbyte files mimics the scenario in which multiple clients request web objects from the same server. Each connection starts at a random time that is uniformly distributed in $[0, 30]$ ms.

In the simulation runs for *TCP-Reno* and *COCOON*, one TCP connection is established between the sender and each receiver. On the other hand, to magnify the benefits of using *TCP-Int*, two TCP connections are established between the sender and receiver $2i$ in the simulation runs for *TCP-Int*, for $1 \leq i \leq \frac{n}{2}$, resulting in totally $\frac{n}{2}$ flows.

In the multiple-bottleneck network topology shown in Fig. 4.4, in addition to the n TCP connections mentioned above, there are other five infinite-duration TCP connections

(labeled in the figure) traversing the bottleneck links between R_1 and R_2 and between R_3 and R_4, respectively. Finally, to demonstrate the capability of *COCOON* in regulating non-responsive UDP connections, in one of the experiments reported below, we replace 2 of the n TCP connections with non-responsive UDP connections with a total sending rate of 0.40 Mbps.

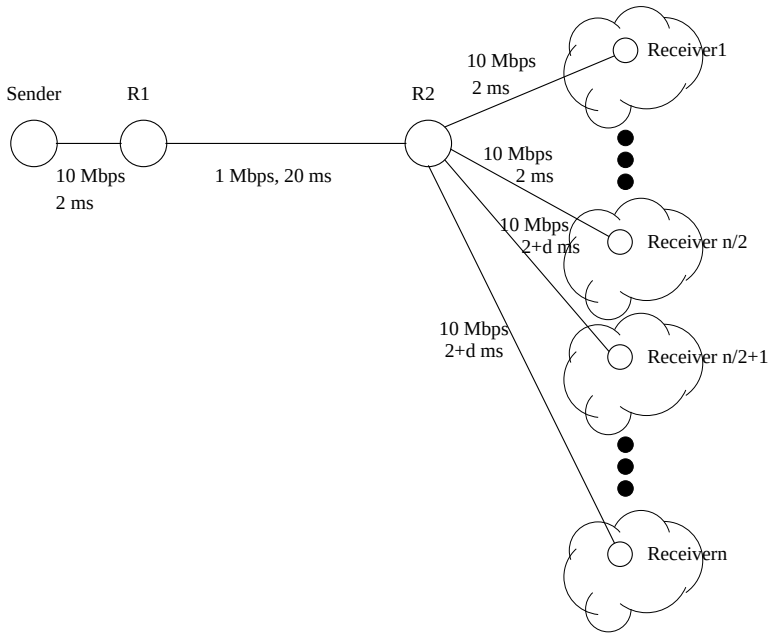


Figure 4.3: The single bottleneck network topology used in the simulation.

Performance in terms of packet loss rate Figs. 4.5–4.6 give the performance of *TCP-Reno*, *TCP-Int* and *COCOON* with respect to packet loss rate in both the single and multiple bottleneck topologies ($\delta = 4$ ms). *COCOON* groups the n TCP connections into two

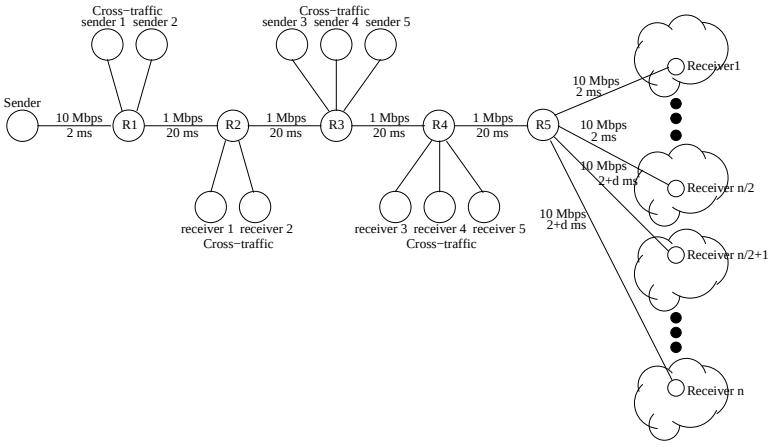
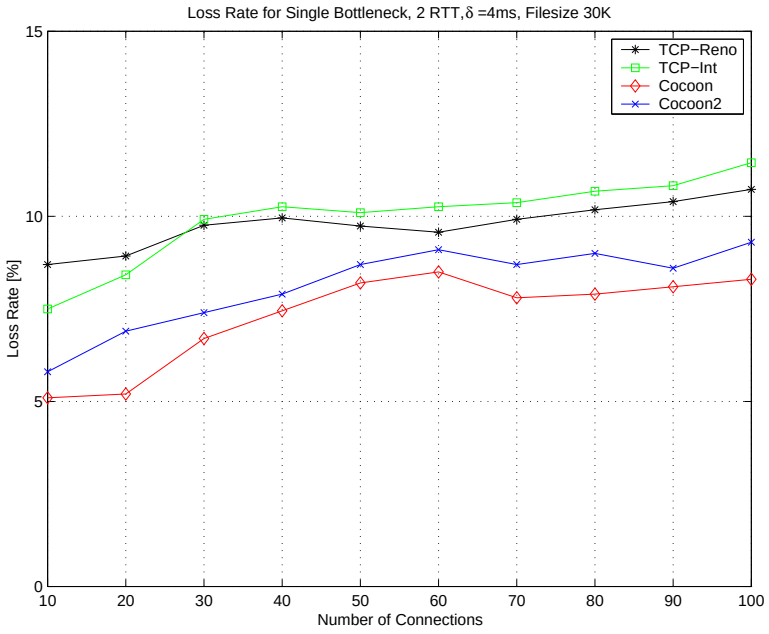


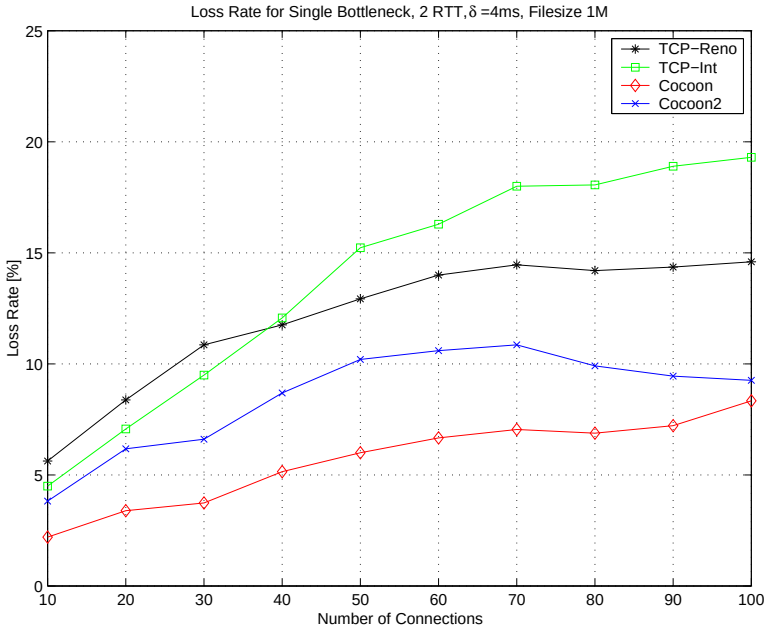
Figure 4.4: The multiple bottleneck network topology used in the simulation.

groups, with the first $n/2$ connections in the first group and the rest in the second group. The two curves labeled as *COCOON* and *COCOON2* in Figs. 4.5–4.6 give the loss rates experienced by the two groups. Under all cases, *COCOON* performs best, and the performance discrepancy becomes more significant as the number of concurrent connections increases or as the size of files transferred increases. The fact that *COCOON* outperforms *TCP-Reno* demonstrates the benefit of coordination among potentially competing connections. The fact that *COCOON* outperforms *TCP-Int*, on the other hand, demonstrates that the scope of coordination should not be restricted to connections destined for the same destination.

Performance in terms of response time Recall that a *COCOON* connection may reduce its congestion window and/or enter the congestion avoidance phase when some other peer connection incurs packet loss. That is, *COCOON* is pro-active in congestion control and

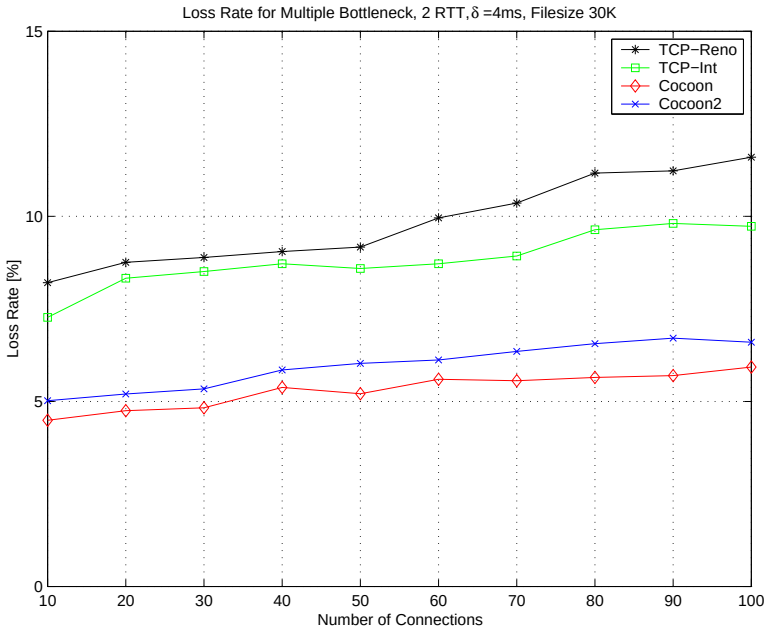


(a) 30K bytes

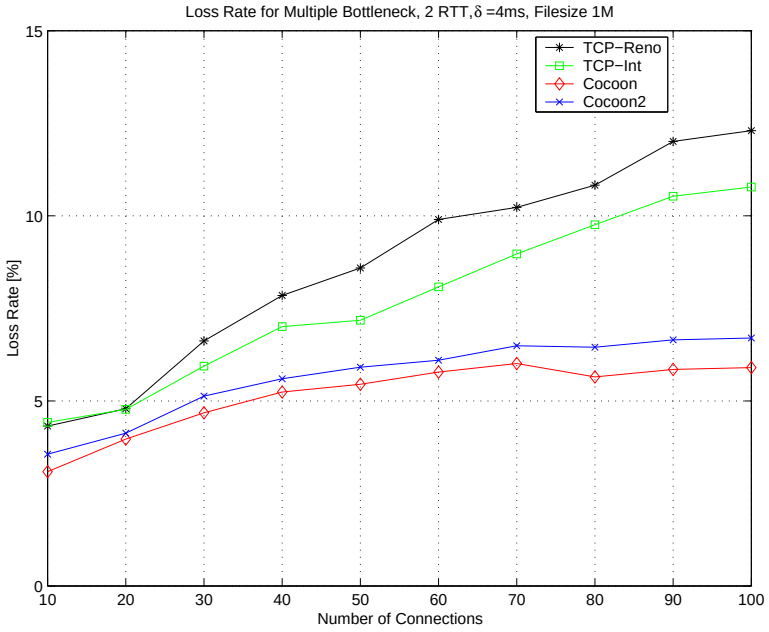


(b) 1M bytes

Figure 4.5: Loss rate vs. the number of concurrent connections in the single bottleneck topology.

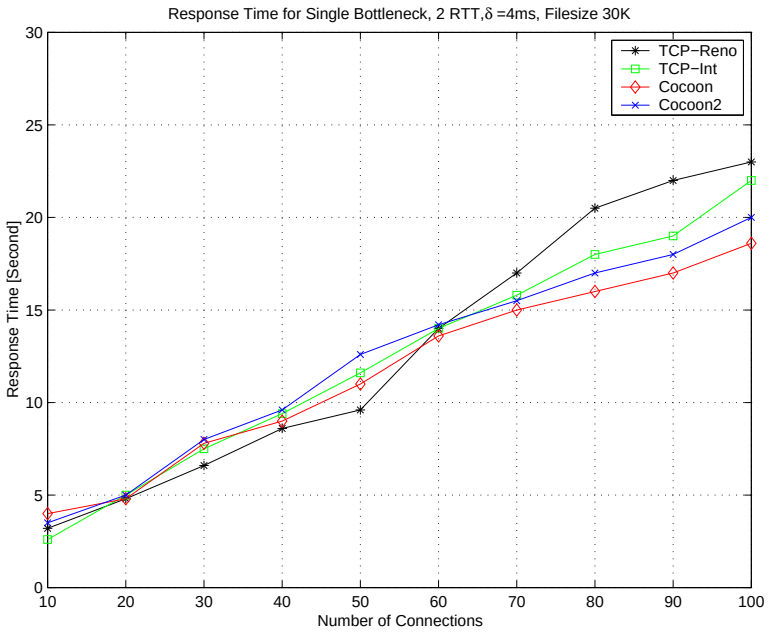


(a) 30K bytes

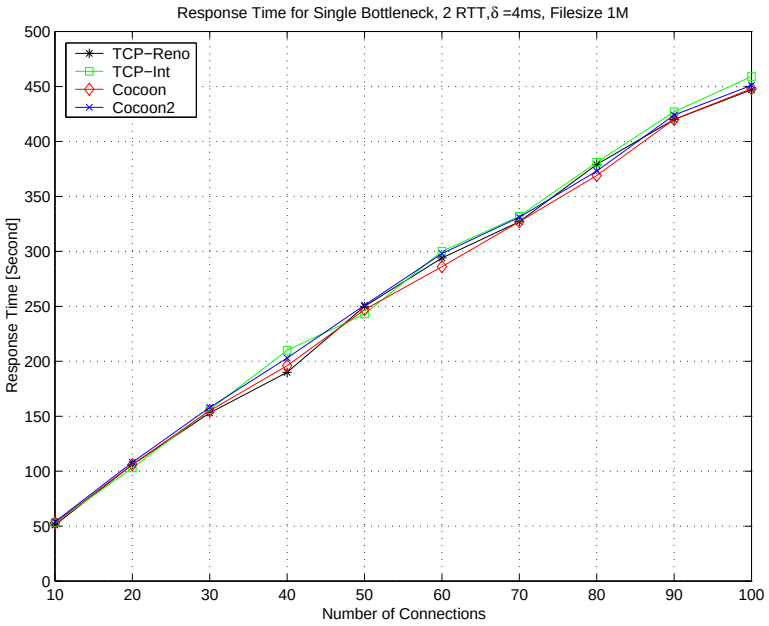


(b) 1M bytes

Figure 4.6: Loss rate vs. the number of concurrent connections in the multiple bottleneck topology.

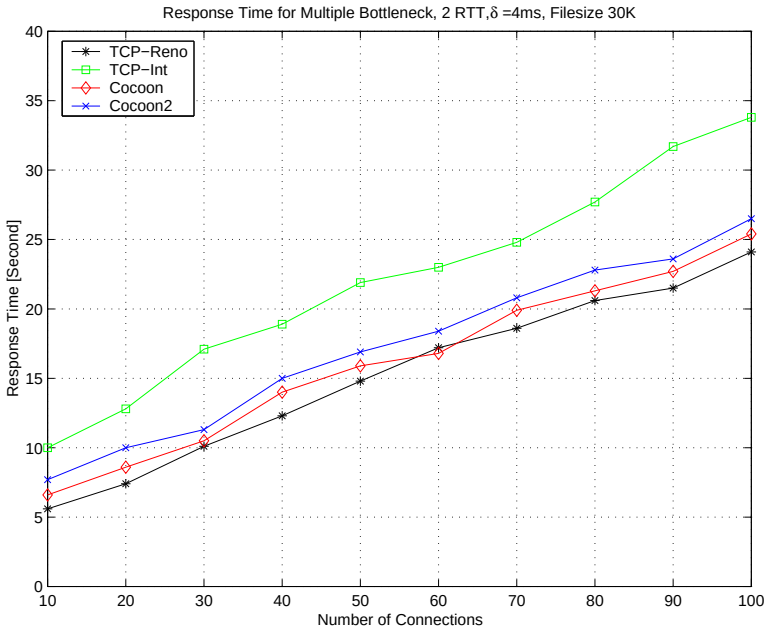


(a) 30K bytes

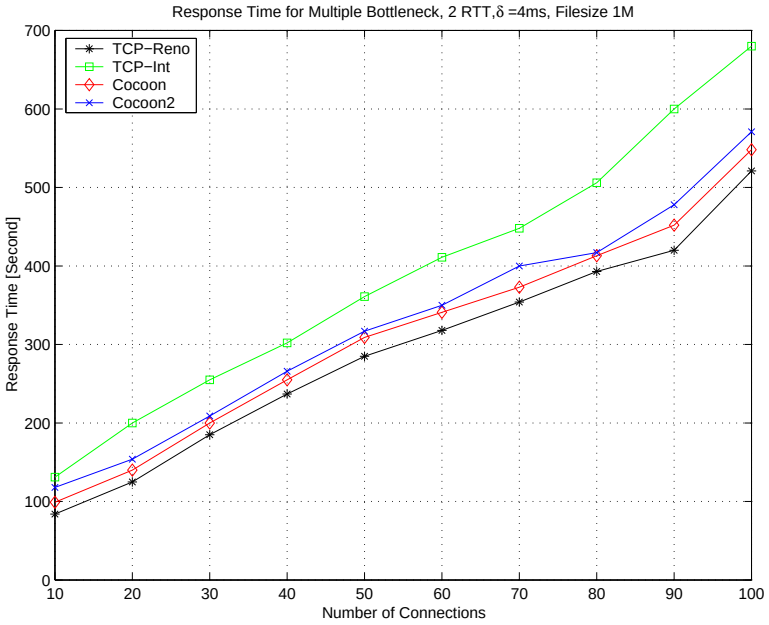


(b) 1M bytes

Figure 4.7: Response time vs. # concurrent connections in the single bottleneck topology.



(a) 30K bytes



(b) 1M bytes

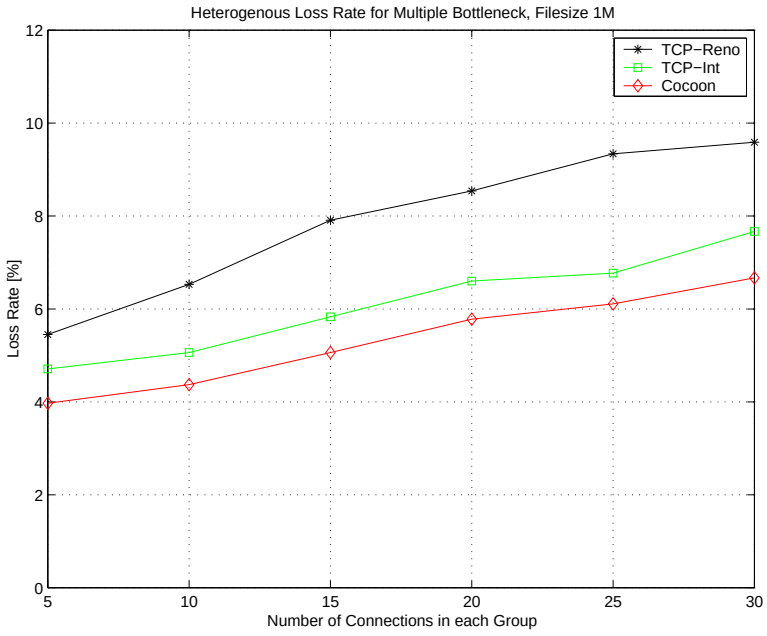
Figure 4.8: Response time vs. # concurrent connections in the multiple bottleneck topology.

is not as aggressive as TCP-Reno in utilizing available bandwidth. On the other hand, by coordinating congestion control among connections, *COCOON* incurs less packet losses (and hence less packet retransmission and timeouts, as illustrated in Figs. 4.5–4.6). This, coupled with the fact that a new *COCOON* connection may begin with a larger congestion window, helps to sustain more throughput. To study which factors dominate, we measure the time it takes to transfer files of 30K bytes and 1M bytes using *TCP-Reno*, *TCP-Int* and *COCOON*. Figs. 4.7-4.8 give the results in both the single bottleneck and multiple bottleneck topologies, respectively. Again, the two curves labeled as *COCOON* and *COCOON2* represent the response time experienced by the two *COCOON* groups.

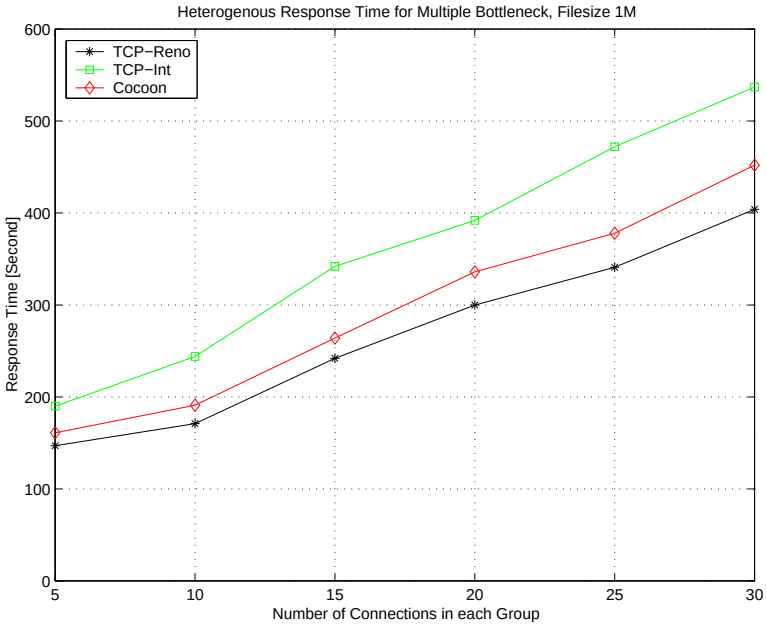
As shown in Fig. 4.7, the curves are very close to one another in the single bottleneck case. However, in the existence of interfering traffic (Fig. 4.8), *TCP-Reno* and *COCOON* outperform *TCP-Int*. This is because in *TCP-Int* all connections destined for the same destination host are treated as single flow that is subject to a single congestion window, so in the existence of interfering traffic, the total throughput sustained by *TCP-Int* flows is less than that in the other two schemes. In contrast, the n connections are still treated as n flows in *COCOON*, each of which is subject to its own congestion window (but shares the congestion information with the other connections in the same group). When a packet loss occurs, only connections (in the same group) that may potentially contribute to the congestion reduces their congestion windows. As a result, the total throughput sustained by *COCOON* flows is comparable to that by *TCP-Reno* flows, and a *TCP-Reno* and a *COCOON* client do not perceive much difference in the response time. From the perspective

of network management, however, the bandwidth is better utilized under *COCOON* as a result of reduced packet loss rate.

Performance in the co-existence of *TCP-Reno*, *COCOON*, and *TCP-Int* servers To evaluate the performance when *TCP-Reno*, *COCOON*, and *TCP-Int* connections co-exist and compete against each other, we set $\delta = 0$ (i.e., all n TCP connections are subject to the same RTTs), and divide the n TCP connections into three groups, each of which running *TCP-Reno*, *COCOON*, and *TCP-Int*, respectively. Under *TCP-Int*, two TCP connections are established between the sender and receiver $6i$, for $1 \leq i \leq \frac{n}{6}$, resulting in totally $\frac{n}{6}$ flows. Fig. 4.9 gives the packet loss rate and response time incurred in sending files of 1M bytes in the multiple bottleneck topology (in the presence of cross traffic). First, *COCOON* performs best in terms of packet loss rate, because of its better coordination among concurrent connections, while *TCP-Reno* performs worst due to the blind competition among concurrent TCP connections. Second, *TCP-Reno* performs best in terms of response time, because each individual *TCP-Reno* connection probes and grabs the available bandwidth on its own. *TCP-Int*, on the other hand, gives the worst performance, because there are effectively $\frac{n}{6}$ *TCP-Int* flows competing against other connections (as well as against cross traffic) for the available bandwidth. The performance of *COCOON* lies in between, because similar to *TCP-Reno*, the $\frac{n}{3}$ connections are still treated as $\frac{n}{3}$ individual flows, but when a connection incurs packet loss, the other connections in the same group may have to backoff and hence do not grab the available bandwidth as aggressively as *TCP-Reno*.



(a) Loss rate



(b) Response time

Figure 4.9: Performance in the co-existence of *TCP-Reno*, *TCP-Int* and *COCOON* connections.

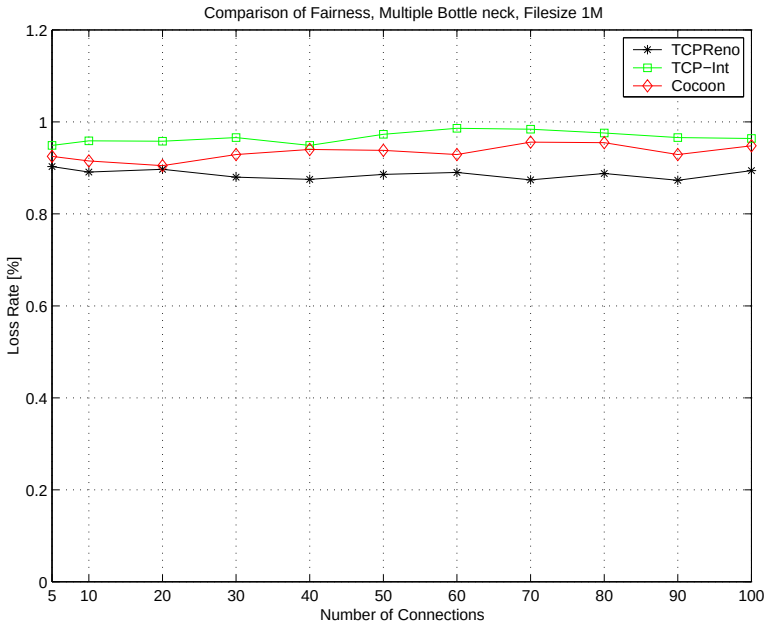


Figure 4.10: The fairness index vs. the number of connections for transfer of files of size 1M bytes.

Fairness amongst Cocoon connections To study the impact of congestion coordination in *COCOON* on the fairness among connections in the same group, we repeat the experiments in Figs. 4.5–4.8 with $\delta = 0$, and calculate the fair index defined in [17]

$$F(x) = \frac{(\sum_{i=1}^{n} x_i)^2}{n(\sum_{i=1}^{n} x_i^2)}, \tag{4.1}$$

where x_i is the throughput attained by the ith connection in a group. Fig. 4.10 depicts the fair index under *TCP-Reno*, *TCP-Int*, and *COCOON*, respectively. (As the curves representing results obtained under the single multiple bottleneck topologies are indistinguishable, only the one for the multiple bottleneck topology is shown.) The fairness index for *COCOON* is inferior to that for *TCP-Int*, but is superior to that for *TCP-Reno*. This difference results from the fact that *TCP-Int* uses a scheduler to fairly regulate packet transmission among connections. *COCOON*, on the other hand, coordinates congestion control among connections in a group by requiring connections with large congestion windows to back off more in the case of packet losses.

Performance in the existence of non-responsive UDP connections We run the same set of experiments as in Fig. 4.10, except that 2 of the n TCP connections are replaced with non-responsive UDP connections with a total sending rate of 0.40 Mbps. Fig. 4.11 gives the TCP performance of *TCP-Reno*, *TCP-Int* and *COCOON* with respect to packet loss rate and response time incurred in transferring files of 1M bytes in the existence of UDP connections and other interfering connections in the multiple bottleneck topology.

Client host name	Client OS	Avg RTT to server
alpha.ece.ucsb.edu	SunOS 5.7	66.699 ms
rodan.ics.uci.edu	SunOS 5.7	67.101 ms
dsp7.eng.umd.edu	Digital UNIX	26.298 ms
hertz.ece.wisc.edu	SunOS 5.6	23.688 ms

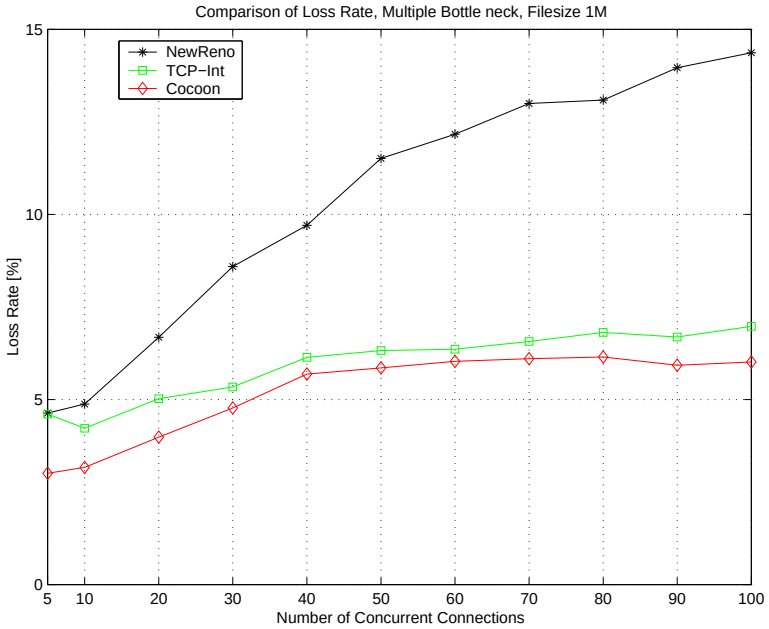
Table 4.1: The host and network characteristics for experiments run on the Internet.

COCOON achieves the best performance in both measures because it regulates UDP connections in a TCP-compliant manner and prevents them from depriving TCP connections of their fair bandwidth share. *TCP-Reno* incurs the worst packet loss rate, due to the lack of coordination among the n TCP connections. On the other hand, *TCP-Int* incurs the largest response time, because the number ($\frac{n}{2}$) of concurrent TCP connections in *TCP-Int* is less than that in the other schemes, and hence the total throughput attained in the presence of other UDP/interfering connections is also the least.

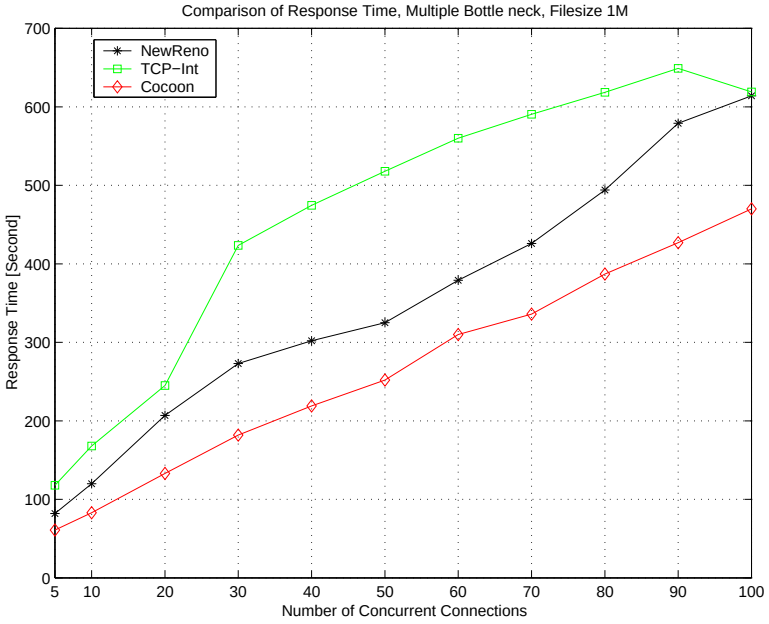
4.7.2 Empirical Study

We have implemented *COCOON* in *FreeBSD* 2.2.8. We then carried out experiments over the Internet, with clients located at UCSB (alpha.ece.ucsb.edu), UCI (rodan.ics.uci.edu), UMD (dsp7.eng.umd.edu), and UW-Madison (hertz.ece.wisc.edu) and the server located at OSU (eepc121.eng.ohio-state.edu). The host and network characteristics are listed in Table 4.1.

In the empirical study, we compare *COCOON* against *CM*, HTTP/1.0, and HTTP/1.1 with respect to attainable throughput, frequency of retransmission timeouts, and response

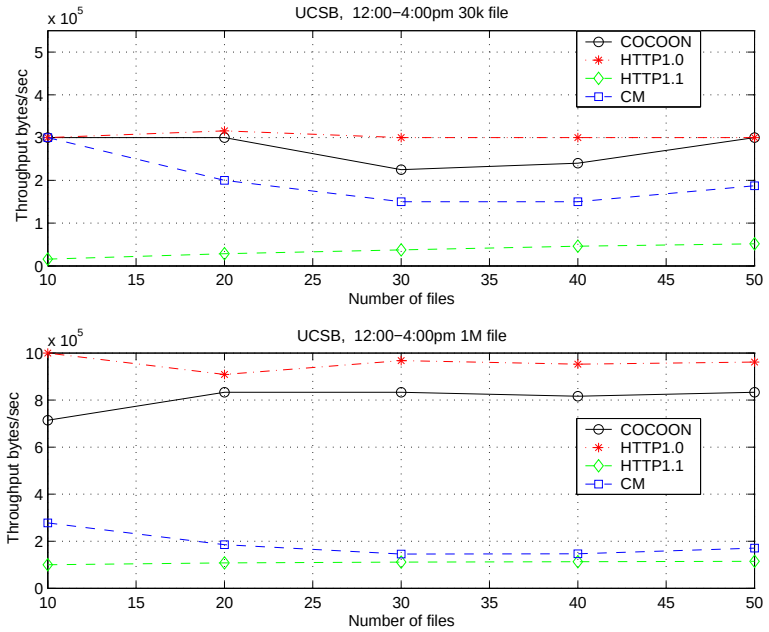


(a) Packet loss rate (1M bytes)

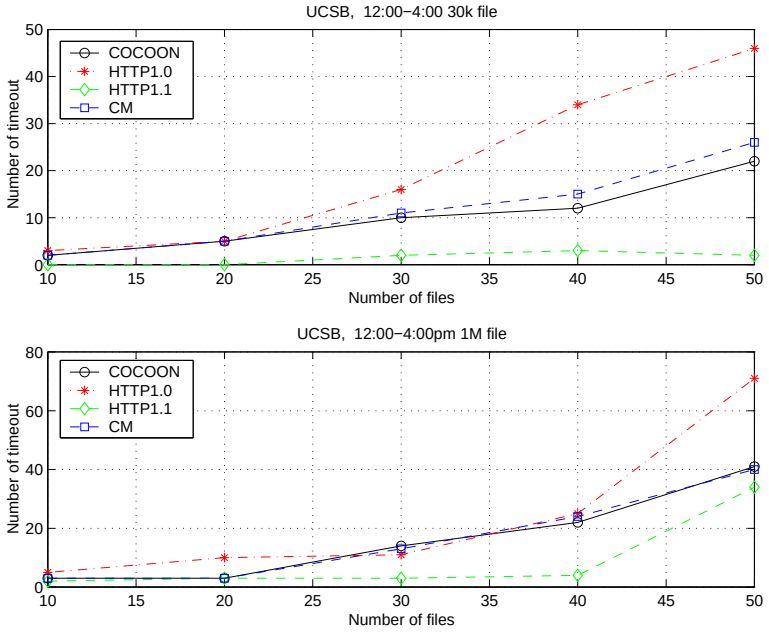


(b) Response time (1M bytes)

Figure 4.11: Performance in the existence of non-responsive UDP connections.

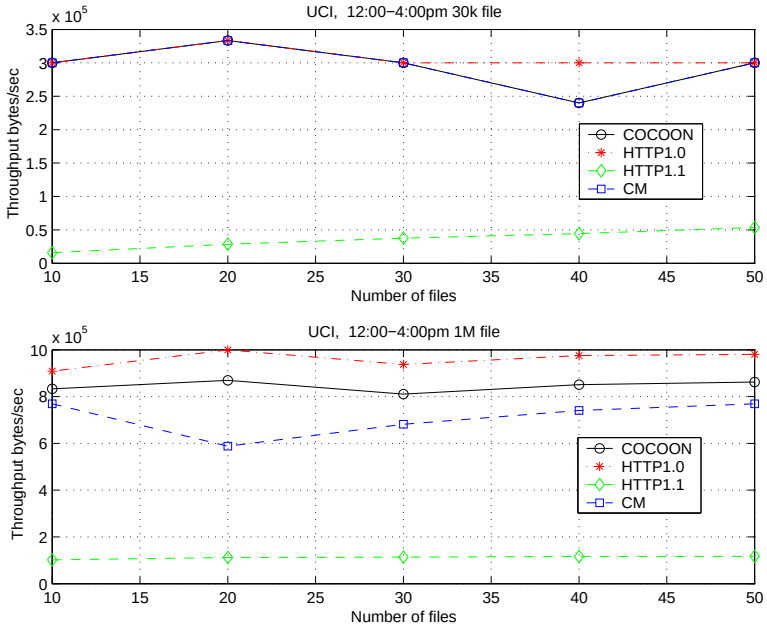


(a) Throughput

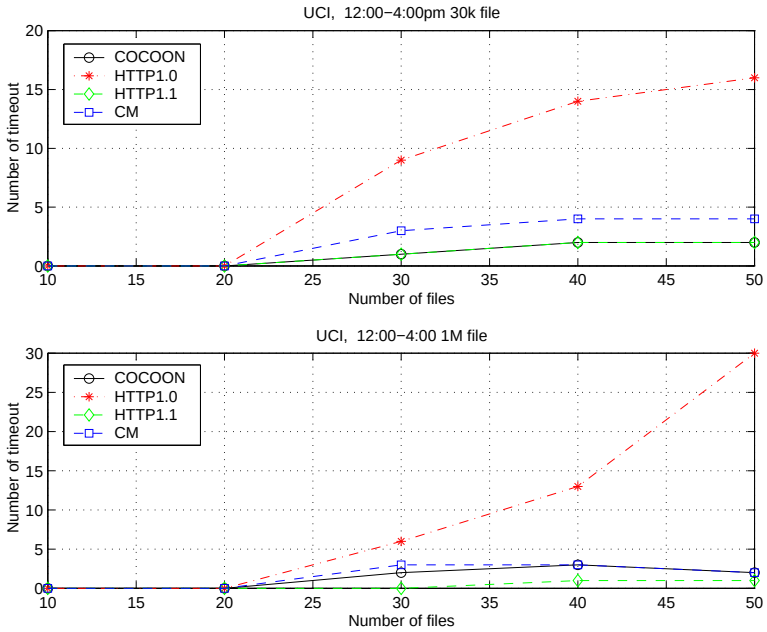


(b) # Retransmission timeouts

Figure 4.12: Empirical results for HTTP requests initiated at UCSB.



(a) Throughput



(b) # Retransmission timeouts

Figure 4.13: Empirical results for HTTP requests initiated at UCI.

time. The experiments were carried out in 3 different time intervals (morning 8:00-11:00am, afternoon 1:00-4:00pm, and night 9:00-12:00pm) on a daily basis for a period of 2 weeks. In each set of experiments, four methods are used:

HTTP 1.0 n HTTP requests of the form "GET file_name HTTP/1.0" are made from a client to the HTTP 1.0 web server, where n varies from 5 to 100. In this case n parallel connections are established. No modification is made in the kernel at the server host.

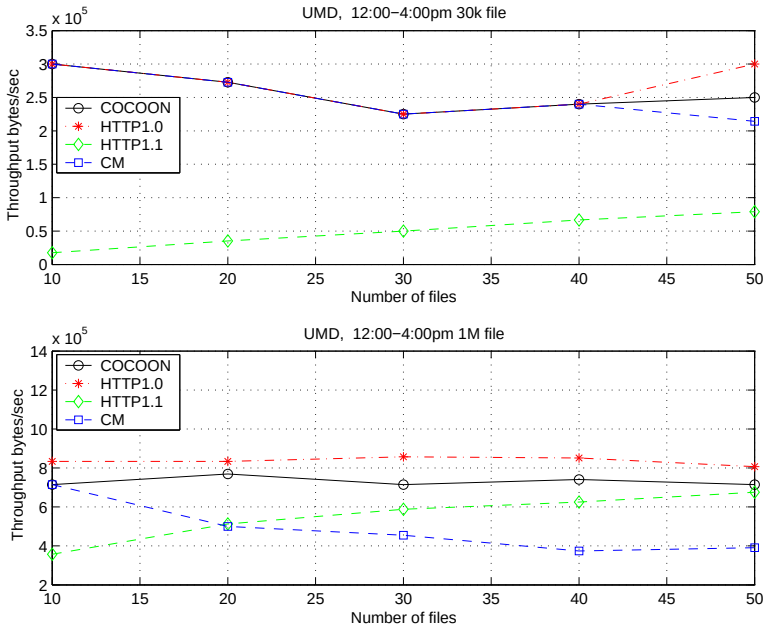
HTTP 1.1 HTTP requests of the form "GET file_name_1 HTTP/1.1; ... ; GET file_name_n HTTP/1.1" are made from a client. In this case, one persistent connection is established. No modification is made in the kernel at the server host.

CM n HTTP 1.0 requests of the form "GET file_name HTTP/1.0" are made from a client to the web server. An implementation of *CM* alpha release is used at the server host.

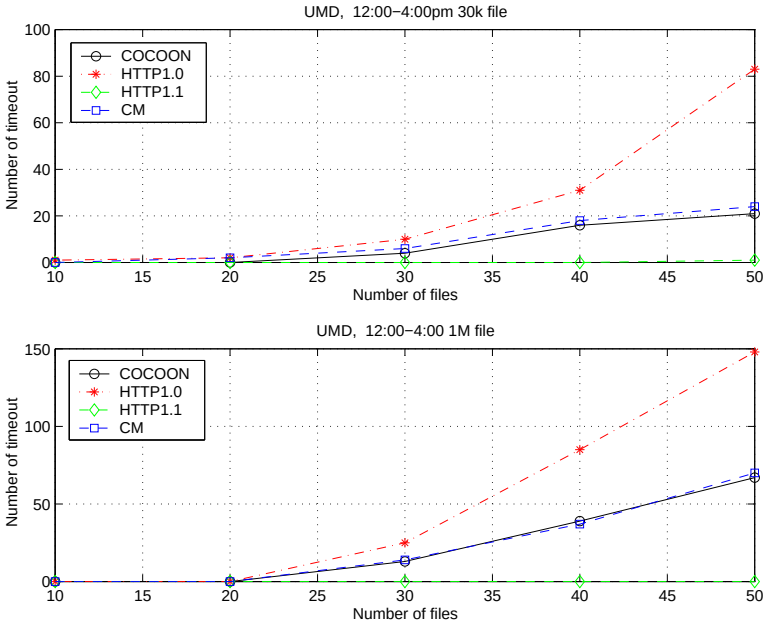
COCOON n HTTP 1.0 requests are made from a client to the web server. An implementation of *COCOON* is used at the server host.

All the web servers run on an Intel Pentium II 400MHz machine with 128M byte memory. In each run, the throughput attainable by a client and the total number of retransmission timeouts were recorded. Five experiments were carried out for each combination of (# HTTP requests, method) in each time interval (i.e., each data point reported below is the average value over $5 \times 7 \times 2 = 70$ runs).

Figs. 4.12–4.15 give the empirical results for the experiments carried out in the afternoon time intervals, with the client being the host at UCSB, UCI, UMD, and UW-Madison,

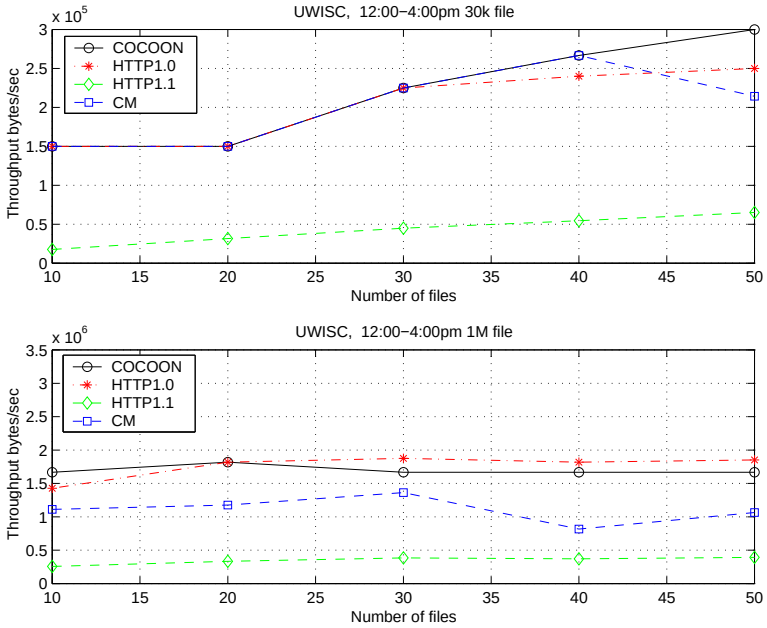


(a) Throughput

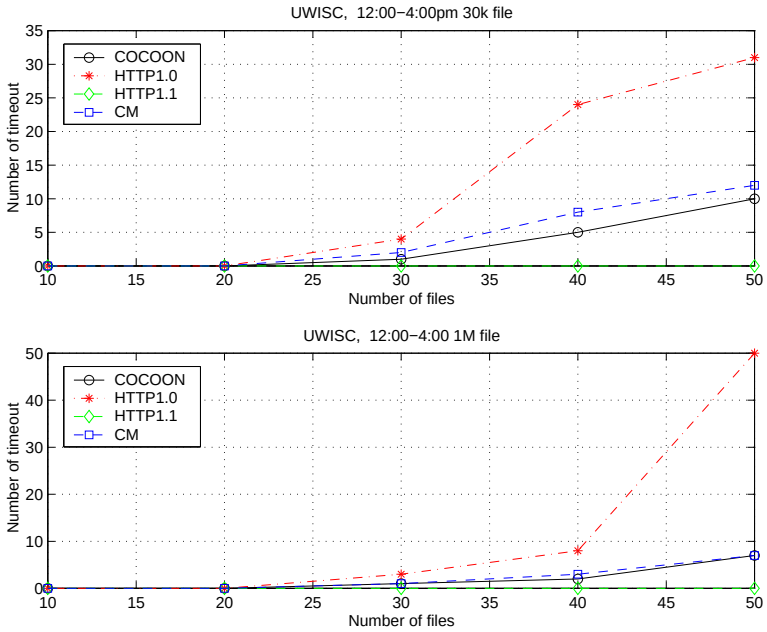


(b) # Retransmission timeouts

Figure 4.14: Empirical results for HTTP requests initiated at UMD.



(a) Throughput



(b) # Retransmission timeouts

Figure 4.15: Empirical results for HTTP requests initiated at Univ. of Wisconsin.

respectively. Experiments carried out in the morning/night intervals give similar, but less pronounced results, and hence are not shown. We believe this is because the Internet carries more traffic in the afternoons and hence different congestion control methods can be better differentiated in this interval.

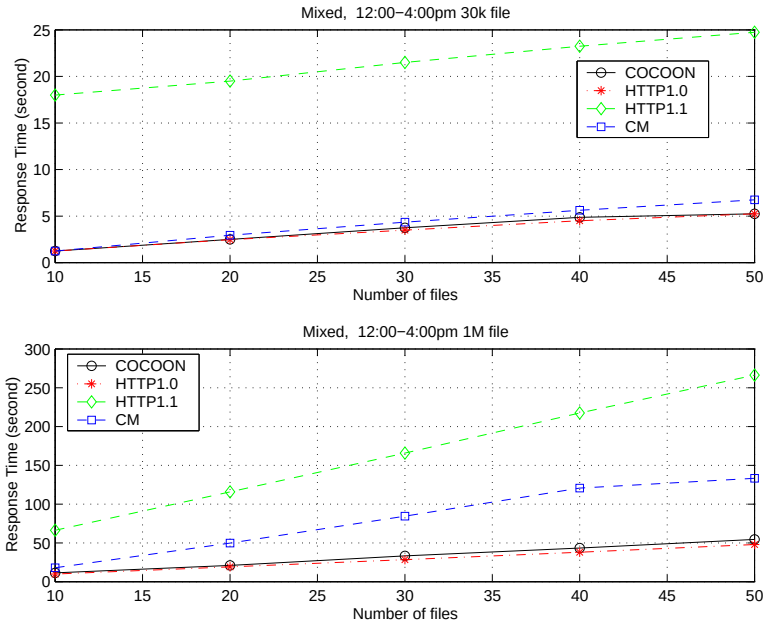
Several observations are in order: first, HTTP 1.1 and HTTP 1.0 achieves, respectively, the best and worst performance in terms of retransmission timeouts in all experiments. This is because with the persistent connection approach, HTTP 1.1 establishes one connection to sequentially transfer the n files. Timeouts occur only as a result of competition with other Internet traffic. HTTP 1.0, on the other hand, establishes n independent connections that compete blindly with one another as well as other Internet traffic, thus yielding the worst performance. Both *COCOON* and *CM* lie in between, with *COCOON* performing slightly better than *CM*.

Second, in the case of transferring large files, the performance in terms of attainable throughput (from best to worst) is HTTP 1.0, *COCOON*, *CM*, and HTTP 1.1. This is attributed to (i) both *CM* and HTTP 1.1 establish only one connection, which has to compete with other Internet traffic. *COCOON* and HTTP 1.0, on the other hand, establish n individual TCP connections and hence are capable of grabbing more bandwidth shares; (ii) *COCOON* coordinates congestion control among group members, and hence is not as aggressive as HTTP 1.0 in grabbing bandwidth (but at the same time incurs less retransmission timeouts); and (iii) in case of packet loss in a group, *CM* reduces the composite congestion window by half (which is equivalent to reducing the congestion windows of all

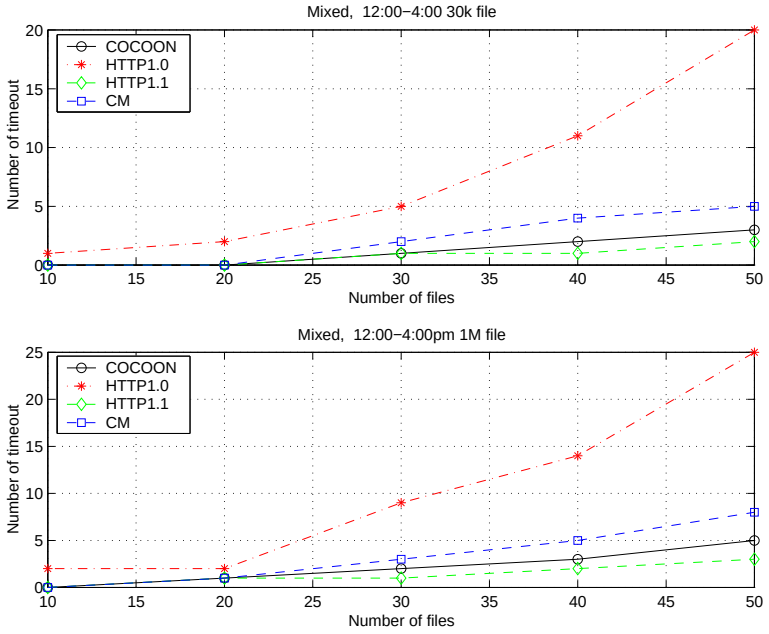
individual connections by half), while *COCOON selectively* reduces the congestion windows of those which may be potentially responsible for the congestion.

Third, in the case of transferring small files, HTTP 1.0, *COCOON*, and *CM* achieve similar performance in terms of attainable throughput (and all of them outperform HTTP 1.1). This may be due to the fact that in the case of small file sizes, the time taken to transfer files is relatively small as compared to the time taken to perform three-way handshaking operations. As a result, the difference in the attainable throughput (which is amortized by the connection setup time) becomes insignificant. Furthermore, during the small file transfer period, not many packet losses or retransmissions occur, so the benefit of congestion management via *COCOON* or *CM* cannot be magnified.

To demonstrate the grouping capability of *COCOON*, we also carry out the following set of experiments (again in three time intervals on a daily basis over a period of two weeks): four clients from UCSB, UCI, UMD, and UW simultaneously make HTTP requests to the web server to transfer n files, where n varies from 10 to 50. In the case of HTTP 1.0 (HTTP 1.1), $4n$ (4) parallel connections are established. In the case of *COCOON*, $4n$ connections are established and grouped into 4 groups, and in the case of *CM*, connections are ensembled into 4 flows. Fig.4.16 gives the empirical results for experiments carried out in the afternoon intervals. *COCOON* and HTTP 1.0 achieve comparable performance and outperform HTTP 1.1 and *CM* in terms of response time. Also, *COCOON* better coordinates congestion management among n connections in a group, as the number of retransmission timeouts is smaller than that incurred in HTTP 1.0 and *CM*.



(a) Avg. response time



(b) # Retransmission timeouts

Figure 4.16: Empirical results for HTTP requests simultaneously initiated at UCSB, UCI, UMD, and UM-Madison.

4.8 Conclusion

In this chapter, we present an alternate endpoint congestion management scheme, called coordinate congestion control (*COCOON*). *COCOON* groups concurrent TCP and UDP connections destined for the same destination host or subnet, enables them to share congestion information (but not the congestion window, and coordinates congestion control activities among them. The size of a *COCOON* group can be dynamically adjusted, and the overhead of group management has been analytically shown to be reasonably small (no more than 2.5%) under an extreme wide spectrum of subnet distribution and network loads. *COCOON* also expedites the start up of a new connection by allowing it to commerce with a congestion window that is large enough to catch up with other connections but not to induce congestion. Finally, *COCOON* takes into account non-responsive UDP connections in a group and "bundles" them into a virtual connection, which is subject to TCP-like congestion control.

COCOON requires modification only at the server hosts and can be readily deployed over the Internet. We have implemented *COCOON* in *ns-2* and in *FreeBSD* and performed simulation/empirical studies with respect to various network topologies and traffic loads. We observe that as compared to *TCP-Reno*, *TCP-Int*, *CM*, HTTP 1.0, and HTTP 1.1, *COCOON* indeed reduces packet loss rate of concurrent connections, while sustaining throughput comparable to the best scheme. In particular, in the empirical study conducted over the Internet, we observe that there exists a tradeoff between maximizing the attainable bandwidth and minimizing the packet loss. Approaches that treat all connections destined for the

same destination as one flow and subject them to a single congestion window (e.g., HTTP 1.1 and *CM*) avoids blind competition but attains less throughput in the existence of other Internet traffic. Approaches without coordination of congestion management (e.g., HTTP 1.0) attains the most throughput, but also suffers from blind competition among individual connections (which is evidenced by the higher frequency of retransmission timeouts). The best approach is to treat connections as individual flows but coordinate congestion management activities among them.

CHAPTER 5

Exploiting Traffic Predictability in Active Queue Management

In this chapter, we elaborate on how the predictability of Internet traffic can be exploited to facilitate design of AQM. We first give the technical motivation and present an overview of AQM with traffic prediction (Section 5.1). Then we delve into the detailed descriptions of PAQM. In particular, we propose an LMMSE-based predictor (Section 5.2), and present the design of the controller and derive the expression of packet dropping probability to be used in the next time interval (Section 5.3). Following that, we give taxonomy, and a detailed summary, of existing AQM schemes (Section 5.4), and present the simulation results (Section 5.5).

Motivation: A number of recent empirical studies of network traffic measurements from a variety of working packet networks have convincingly demonstrated that network traffic is self-similar or long-range dependent (LRD) in nature [22, 96, 100, 101]. This implies the existence of concentrated periods of high activity and low activity (i.e., burstiness) at a wide range of time scales. Scale-invariant burstiness introduces new complexities into resource control and QoS provisioning. On the one hand, burstiness at coarser time

scales induces extended periods of either over-utilization or under-utilization. Packets that arrive in the periods of over-utilization experience long delays and are even dropped due to buffer overflow, while packets that arrive in the periods of under-utilization experience the opposite. The large variation in the end-to-end delay packets experience has an adverse impact on transport of QoS-sensitive traffic such as multimedia traffic. On the other hand, if resource reservation is deployed for QoS provisioning, resources have to be reserved with respect to the *peak* rates over a *wide* range of time scales. This adversely affects resource control and degrades the overall performance.

While the LRD characteristic of network traffic introduces difficulty and complexity into traffic and resource management, it also opens up a new direction for research — the existence of nontrivial correlation structure at larger time scales can be judiciously exploited for better congestion and resource control. How to exploit the abundant correlation structure to improve the performance of AQM is the subject matter of this chapter.

Central to the notion of AQM with traffic prediction is prediction of the future traffic based on recent traffic measurements and use of the prediction results to modulate the magnitude of the packet dropping probability at a router. We first design a traffic predictor based on the *linear minimum mean square error* (*LMMSE*) estimation. Through *ns-2* simulation and analytical reasoning, we show that the LMMSE predictor performs as accurately as the other fractional-model based predictors (e.g., Fractional Brownian Motion (*FBM*) model and the Fractal ARIMA (*FARIMA*) model), and yet can be practically implemented in hardware/software with readily available fast algorithms. Then, we design a simple controller that takes the prediction result (i.e., the amount of traffic to arrive in the next a few

measurement intervals) as input to determine the packet dropping probability. The objective function used here is to stabilize the queue length at a desirable level. We choose this objective function for two reasons: if the queue length can be controlled (in anticipation of future traffic) stable at a certain small value (or an pre-determined trajectory), (i) the delay jitter experienced by packets that traverse the router remain approximately constant (which is an important criterion for transporting QoS-sensitive multimedia applications); and (ii) the capacity of the link can be fully utilized even under pro-active packet dropping/marking. The resulting AQM scheme is termed as *predictive AQM* (*PAQM*).

The controller output of PAQM indicates that RED can be viewed as a special case of PAQM, with the predicted future traffic always fixed at certain value. In other words, PAQM augments the set of congestion indices with a new dimension — the amount of traffic to arrive in the next a few measurement intervals (or equivalently the future arrival rate). This also avoids the use of the queue length as both the congestion and performance indices — which is commonly believed to be the cause that queue-length-based AQM schemes (such as RED) cannot simultaneously achieve high link utilization and low packet loss ratio [7]. By stabilizing, with consideration of future traffic, the queue at a desirable level, PAQM enables the link capacity to be fully utilized, while not incurring excessive packet loss ratio. Through *ns-2* simulation, we show that PAQM can indeed simultaneously achieve both.

5.1 Overview

RED was shown in [36] to prevent global synchronization,[9] accommodate bursty traffic, incur little overheads, and coordinate well with TCP under serious congestion conditions. The performance of RED, however, heavily depends on whether or not the two thresholds are properly tuned. Also, RED was shown to be unfair to individual flows [58], be unable to achieve high link utilization and low packet loss ratio simultaneously [7,33,49], and exhibit short-term fluctuation in the queue length [70]. In particular, it was shown in [7, 33, 41, 49] that queue length should not be the only parameter to be observed and controlled. Instead, other control variables and control policies should be deployed. To this end, we propose a novel AQM scheme that takes a new angle and manages the queue in anticipation of future incoming traffic.

The block diagram of PAQM is shown in Fig. 5.1. Let $f(i)$ denote a time series representing the amount of data (in bytes) in a packet that arrives at a router. The predictor keeps track of the aggregate series samples, $f^m(1), f^m(2), \ldots, f^m(n)$, measured in the past n measurement intervals, where $f^m(k)$, $1 \leq k \leq n$, is the aggregate series sample taken in the $(n+1-k)$th most recent interval, i.e.,

$$f^m(k) = \frac{1}{l} \sum_{i \in interval(k)} f(i), \tag{5.1}$$

where l is the number of packets that arrive in a measurement interval (and may vary from an interval to another). Based on these aggregate series samples, the predictor predicts the amount of traffic, $\widehat{f}(n+1) \stackrel{\triangle}{=} f^m(n+1)$, and $\widehat{f}(n+2) \stackrel{\triangle}{=} f^m(n+2)$, in the next

[9] Global synchronization results from signaling all TCP connections to reduce their congestion windows at the same time, and is usually followed by a sustained period of low link utilization.

two (or more) intervals. The controller then uses the predicted results to modulate the packet dropping probability, p, to be used in the next time interval, with the objective of stabilizing the instantaneous queue length at a desirable and stable level. As mentioned before, the fact that the amount of future traffic is figured in in the calculation of packet dropping probability is equivalent to augmenting the set of congestion indices with a new dimension.

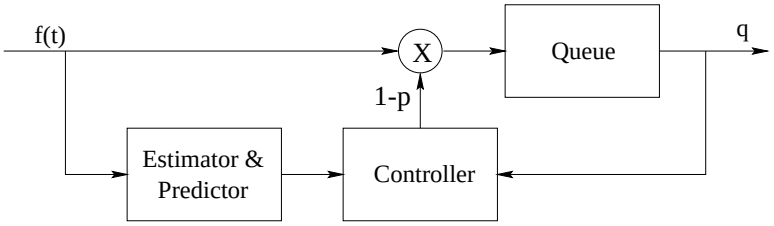


Figure 5.1: AQM with traffic prediction.

5.2 Design of the Traffic Predictor

Several issues have to be addressed in designing a good traffic predictor: First, we have to determine the traffic quantity to measure and predict. In this work, we measure and predict the amount of traffic averaged over a measurement interval of length τ (Eq. (5.1)). This is due to the following two reasons: (i) the LRD characteristic of the Internet traffic implies that the autocorrelation function of the average traffic obeys the same law as that of the original traffic. Hence, the aggregate time series is still a good representative of the

original time series. (ii) The averaging operation in Eq. (5.1) can be viewed as sampling and smoothing operations. As reported in [84], smoothed and sampled traffic exhibits more predictable behaviors, and hence prediction based on the aggregate traffic is more accurate.

Second, we have to determine which prediction method to use (and hence the mathematical model upon which the predictor is based). On the one hand, prediction should provide enough accuracy, but on the other hand, the resulting predictor should be easy to implement, and the parameters that have to be on-line measured/estimated should be kept minimal. Third, we have to determine how far ahead traffic prediction is made. As indicated in the predictability study of network traffic [84], there exists a trade-off between how far ahead prediction can be made and how significant the prediction error is.

Instead of using existing fractional models for time series with LRD, we propose an LMMSE predictor. In what follows, we elaborate on the design of the LMMSE based predictor, and compare it against two widely used fractional models: FBM model and the FARIMA, in terms of accuracy and ease of implementation. We also comment on how we determine the length, τ, of each measurement interval.

5.2.1 LMMSE Predictor

Specifically, given the aggregate series $f^m(k), k = 1, ..., n$, where $f^m(k)$ is given in Eq. (5.1), we predict the aggregate series sample, $f^m(n+1)$, in the next interval as a

weighted sum of the past n average samples:

$$\widehat{f}^m(n+1) = \begin{bmatrix} a_1 & a_2 & ... & a_n \end{bmatrix} \begin{bmatrix} f^m(1) \\ f^m(2) \\ ... \\ f^m(n) \end{bmatrix}, \tag{5.2}$$

where $a_1, a_2, ..., a_n$ are the LMMSE coefficients and can be expressed as

$$\begin{bmatrix} a_1 & a_2 & ... & a_n \end{bmatrix} = \begin{bmatrix} R(n) & R(n-1) & ... & R(1) \end{bmatrix} \times \begin{bmatrix} R(0) & R(1) & ... & R(n-1) \\ R(1) & R(0) & ... & R(n-2) \\ ... & ... & ... & ... \\ R(n-1) & R(n-2) & ... & R(0) \end{bmatrix}^{-1}, \tag{5.3}$$

where $R(n)$ is the covariance function of the time series, and can be estimated (due to the property of asymptotically second-order self- similarity) in practice as

$$R(i) = \frac{1}{n} \sum_{t=i+1}^{n} f^m(t) f^m(t-i), 0 \leq i \leq n-1, \tag{5.4}$$

where n is the number of aggregate series samples kept and is a tunable parameter. (In the simulation study, we use $n = 20$, as in all the simulation runs conducted the performance improvement is marginal as n exceeds 20.) Note that the Hurst parameter H that characterizes the LRD property has been implicitly calculated in the covariance function $R(i)$.

Similarly, to estimate the aggregate series sample, $f^m(n+2)$, in the interval following the next time interval, we take the estimated value of $f^m(n+1)$ as a series sample, and

calculate

$$\widehat{f}^m(n+2) = \begin{bmatrix} a_1 & a_2 & ... & a_n \end{bmatrix} \begin{bmatrix} f^m(2) \\ f^m(3) \\ ... \\ \widehat{f}^m(n+1) \end{bmatrix}.$$

The mean square error of the LMMSE predictor can be calculated (after a few algebraic operations) as

$$\begin{aligned} \sigma^2 &= \sigma_x^2 - \begin{bmatrix} R(n) & R(n-1) & ... & R(1) \end{bmatrix} \cdot \\ & \begin{bmatrix} R(0) & ... & R(n-1) \\ R(1) & ... & R(n-2) \\ ... & ... & ... \\ R(n-1) & ... & R(0) \end{bmatrix}^{-1} \cdot \begin{bmatrix} R(n) \\ R(n-1) \\ ... \\ R(1) \end{bmatrix}. \end{aligned} \tag{5.5}$$

Since the LMMSE predictor does not depend on any fractional model, the close form of σ^2 cannot be easily obtained. Instead, we use the following asymptotic result as an approximation:

$$R(\tau) \sim H(2H-1)\tau^{2H-2}. \tag{5.6}$$

Implementation issues: To practically implement the LMMSE predictor, we consider the following three issues:

1. The LMMSE predictor is derived under the assumption of zero mean stationary stochastic process. As the stochastic time series on-line measured may not be of zero mean, we subtract the mean value from the original time series, apply the LMMSE predictor to estimate the average of the time series in the next interval, and then add back the mean value.

2. We have to determine how far ahead traffic prediction is made. This translates into the problem of determining an appropriate value, τ, for the interval between two calculations of $f^m(k)$. Fortunately the LRD characteristic of the network traffic implies the relatively low decay of the autocorrelation function, and hence the value of τ is not very critical to the performance. In the simulation study, we set the value of τ to be in the range of 0.02 to 0.05 seconds.

3. The operations involved in the calculation of LMMSE coefficients (Eqs. (5.3) and (5.4)) are multiplication of time series samples and matrix manipulation, for which fast algorithms exist [1]. Hence they can be readily implemented in hardware. In particular, the author of [1] gave an adaptive algorithm in linear prediction in which the matrix inverse operation in Eq. (5.4) can be avoided. Succinctly, the algorithm starts with an initial estimate of the coefficient vector A_0. Each time a new data point, $f^m(n)$, is obtained, the algorithm updates A_{n+1} using the recursive equation:

$$A_{n+1} = A_n + \mu \cdot e(n) \cdot f^m(n), \tag{5.7}$$

where $e(n)$ is the prediction error and μ is a constant. If $f^m(n)$ is stationary, A_i is shown to converge in the mean to the optimal solution [1]. The interested reader is referred to [1] for a detailed account.

5.2.2 Comparison with Fractional Model-Based Predictors

Several fractional models and their corresponding predictors have been proposed for time series with LRD characteristics, among which the FBM model and the FARIMA

model may have received the most attention. The interested reader is referred to the extended version of this paper [38] for a brief summary of the two fractional model-based predictors and several important results that are relevant to the discussion of traffic prediction in Section 5.2.3.

The reason why we use the LMMSE predictor instead in predicting incoming traffic at a router is two fold:

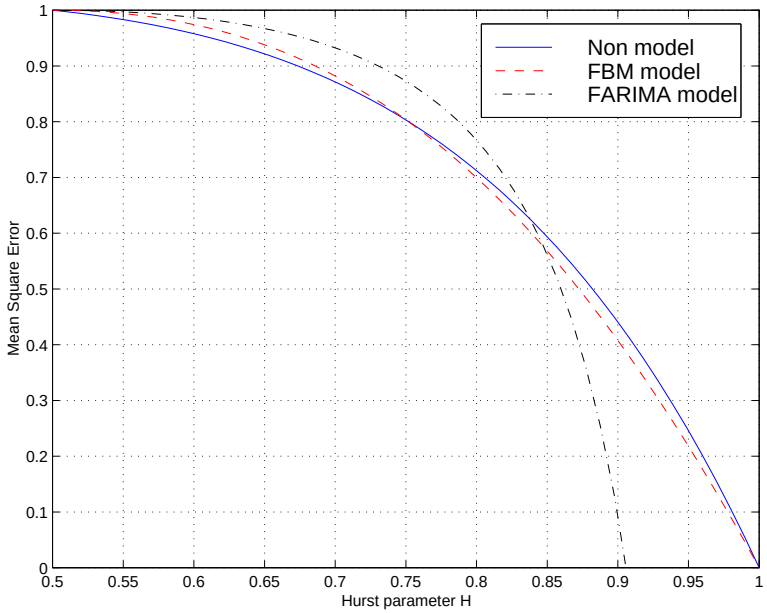


Figure 5.2: Comparison of mean square errors among the FBM, FARIMA, and LMMSE predictors.

Accuracy: The most important criterion in choosing a predictor is the accuracy. We depict the relative mean square errors versus the Hurst parameter H under the LMMSE

predictor (Eq. (5.5)$/\rho_x^2$), the FBM predictor (Eq. (17) in [38]), and the FARIMA predictor (Eq. (22) in [38]) in Fig. 5.2. When $H \rightarrow 1$ the relative error converges to 0 under all three models, suggesting that the more pronounced the LRD characteristic, the better the performance of predictors. Moreover, the three curves are close to one another when $H \leq 0.85$ (beyond which the curve corresponding to the FARIMA model based predictor differs notably). Since analysis of real traffic traces indicates that the H parameter rarely exceeds 0.85 [101], from the theoretic perspective, all three predictors are equally well suited for Internet traffic prediction.

Ease of implementation: To implement the two fractional model based predictors, one has to on-line estimate the H parameter and engage in complicated calculation of weight coefficients. Specifically, one has to estimate the value of H in the FBM model and calculate the weight coefficient in the form of

$$\frac{sin[\pi(H-\frac{1}{2})]}{\pi}[-t(T+t)]^{-H+\frac{1}{2}} \int_0^a \frac{[\tau(\tau+T)]^{H-\frac{1}{2}}}{\tau-t} d\tau.$$

Similarly, one has to estimate the value of $d = H - 1/2$ in the FARIMA model and calculate the weight coefficient in the form of

$$\beta_{kj} = -\begin{pmatrix} k \\ j \end{pmatrix} \frac{\Gamma(j-d)\Gamma(k-d-j+1)}{\Gamma(-d)\Gamma(k-d+1)},$$

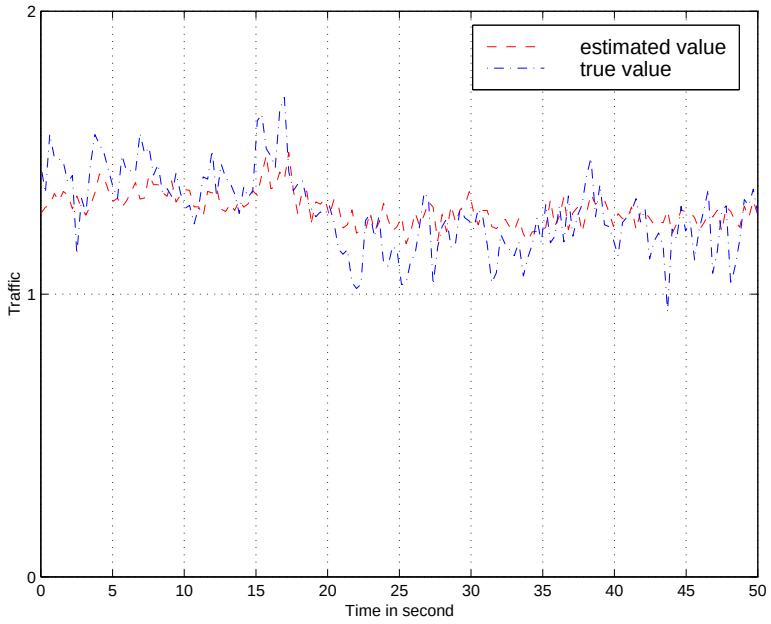
where $\Gamma()$ is the gamma function. (The interested reader is referred to [38] for a detailed account.) Furthermore, to estimate Hurst parameter H for a satisfactory accuracy, a much longer time series of traffic samples is needed. As a result, it is more difficult to practically implement model-based predictors in router hardware/software.

The LMMSE predictor, on the other hand, does not require estimation of such parameters, but instead calculates these parameters (in particular, $R(i)$'s in Eq. (5.4)) directly from the collected traffic samples. Moreover, as mentioned in Section 5.2.1, there exist fast algorithms that can be readily implemented to perform operations involved in the calculation of LMMSE coefficients (Eqs. (5.3) and (5.4)) [1].

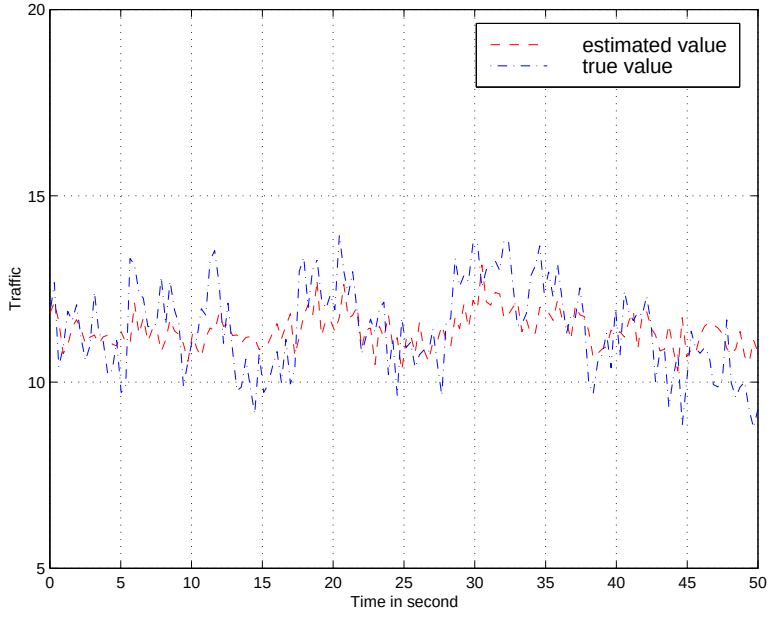
5.2.3 Validation of the LMMSE Predictor

To validate the design of the LMMSE predictor, we have implemented it in a router in *ns-2* and tested its prediction capability in both single-bottleneck networks and networks of arbitrary topology. We found that the actual traffic and the estimated traffic agree very well. To illustrate this, we depict in Fig. 5.3 the actual traffic over a bottleneck link and its corresponding LMMSE estimate. The single bottleneck has a capacity of 20Mbps and a buffer size of 100 packets (each of 1000 bytes). Totally 60 connections are established over the bottleneck link, with each source generating packets either (i) using the on-off traffic model or (ii) using real network traffic traces. The H parameter estimated under both cases in Fig. 5.3 are 0.75 ((a)) and 0.65 ((b)), respectively. This confirms the LRD characteristic of the traffic. Moreover, under both cases, the estimated traffic agrees very well with the actual traffic. In the former case, the ratio of the mean estimate error to the actual value is 0.08, while in the latter case, the ratio is approximately 0.15.

To study the effect of the number of connections on the performance, we repeat the above experiment, but vary the number of connections on the bottleneck link. Fig. 5.4



(a) on-off model



(b) real network traffic traces

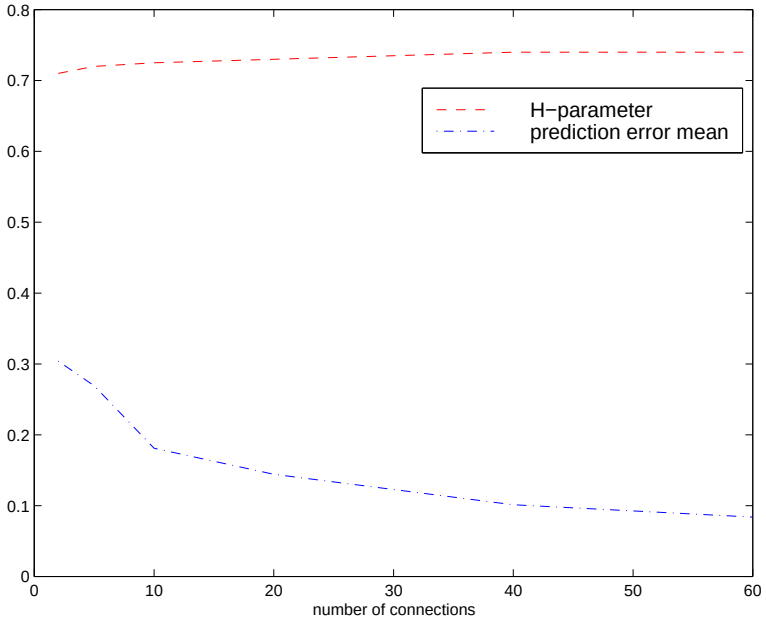
Figure 5.3: Actual and estimated traffic traces when TCP packets are generated using the on-off model or real network traffic traces in *ns-2*.

depicts the Hurst parameter and the estimation error versus the number of connections. As shown in Fig. 5.4, the traffic observed on the bottleneck link exhibits the LRD characteristic under both cases, regardless of the number of connections. In particular, the estimation error of the LMMSE predictor is less than 0.2 when the number of connections exceeds 10 in the former case, and less than 0.1 when the number of connections exceeds 4 in the latter case.

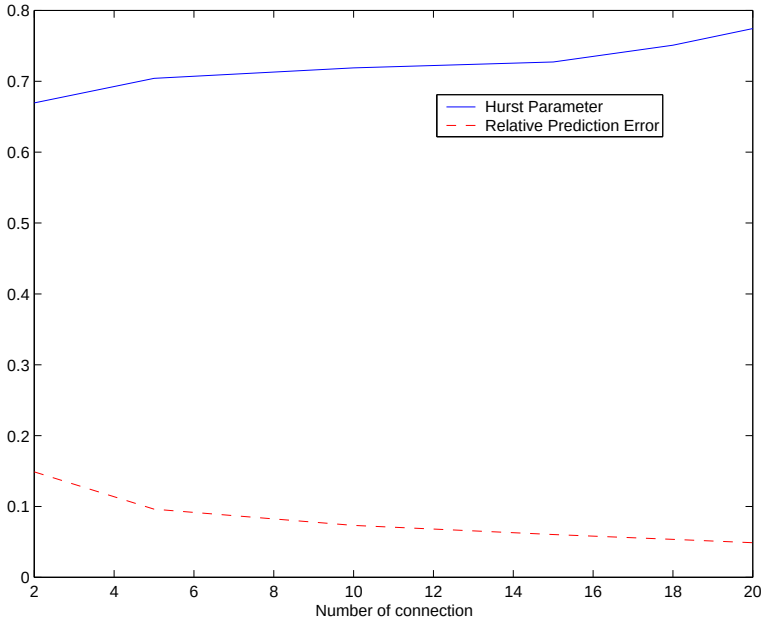
5.3 Design of the Controller

Recall that in Fig. 5.1 the controller utilizes the prediction results (i.e., the amount of traffic that arrives in the next two intervals) to determine the magnitude of the packet dropping probability. The controller in Fig. 5.1 is not easy to analyze, as it is a non-linear system. Hence we replace the controller with an alternative, but equivalent one in Fig. 5.5. In this system, the output, $u(t)$, of the controller is the amount of packets that should be dropped ($0 \leq u(t) \leq f(t)$, where $f(t)$ is the amount of traffic that arrives at the router). By setting $p = \frac{u(t)}{f(t)}$, the two systems are equivalent.

Recall that traffic predication is made on a per-interval basis (where the interval is of duration τ). Let $Q(k)$ denote the queue length at the end of the kth interval, and $f(k+1)$ the amount of traffic that arrives in an interval of duration τ (which may vary from interval to interval). Then, the discrete-time function of the queue length is $Q(k+1) = Q(k) + f(k+1) - u(k+1) - C$, where $C = R \times \tau$ is the number of packets transmitted on the outgoing link and R is the capacity of the outgoing link.



(a) on-off model



(b) real network traffic traces

Figure 5.4: The Hurst parameter and the estimation error versus the number of connections.

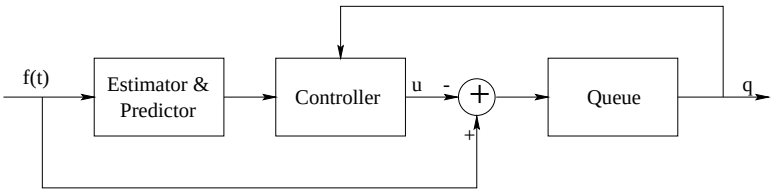


Figure 5.5: An alternate block diagram for AQM with traffic prediction.

The objective function used in the controller is to keep the queue length at an appropriate level or follow a pre-determined, time-variant trajectory. Specifically, let $\widehat{Q}(k)$ and $Q_{opt}(k)$ denote, respectively, the discrete-time functions of the predicted and desirable queue lengths at the end of the kth interval. Then, the objective function we consider is either

$$J = \sum_{i=1}^{M} \left(\widehat{Q}(k+i) - Q_{opt}(k+i) \right)^2, \quad \text{or} \tag{5.8}$$

$$J = \sum_{i=1}^{M} \left(\widehat{Q}(k+i) - Q_{opt}(k+i) \right)^2 + \sum_{i=0}^{M-1} b_i u^2(k+i), \tag{5.9}$$

where M is the number of time intervals to be predicted ahead and b_i is the weight for the controller output. Note that the second term in Eq. (5.9) is used to prevent the system from dropping packets excessively. The optimization problem can then be formally stated as

Problem 5.1 *Find* $u^* = \arg\min_u J$*, where* J *is expressed in either Eq. (5.8) or Eq. (5.9), subject to* $0 \leq u(k+i) \leq \widehat{f}(k+i), 1 \leq i \leq M$.

We first solve Problem 5.1 with the objective function of Eq. (5.8) in the case of $M = 2$ and $Q_{opt}(k) = \overline{Q}, \forall k$, where $\overline{Q}$ is a pre-determined value. To obtain $u^*(k)$ and $u^*(k+1)$,

we take the derivatives of J with respect to $u(k+1)$ and with respect to $u(k+2)$, and set them to zero:

$$\frac{\partial J}{\partial u(k+1)} = 0, \quad \text{and} \quad \frac{\partial J}{\partial u(k+2)} = 0.$$

The results of the above equations are (after a few algebraic operations) are

$$\begin{cases} u^*(k+1) = Q(k) + \widehat{f}(k+1) - C - \overline{Q}, \\ u^*(k+2) = \widehat{f}(k+2) - C. \end{cases} \tag{5.10}$$

Using a similar procedure, we can solve problem 5.1 with the objective function of Eq. (5.9) in the case of $M = 2$ and $Q_{opt}(k) = \overline{Q}, \forall k$. The results are

$$\begin{cases} u^*(k+1) = -\frac{(4b_1-1)(Q(k)+\widehat{f}(k+1)-\overline{Q})+2b_1\widehat{f}(k+2)+(1-6b_1)C}{1-4b_1-2b_0+4b_0b_1}, \\ u^*(k+2) = -\frac{2b_0(Q(k)+\widehat{f}(k+1)-\overline{Q})+(2b_0-1)\widehat{f}(k+2)+(1-4b_0)C}{1-4b_1-2b_0+4b_0b_1}. \end{cases} \tag{5.11}$$

Note that Eq. (5.11) reduces to Eq. (5.10) when $b_1 = b_2 = 0$. As $u^*(k+1)$ and $u^*(k+2)$ obtained in Eq. (5.11) may violate the constraints of $0 \leq u(k+1) \leq \widehat{f}(k+1)$ and $0 \leq u(k+2) \leq \widehat{f}(k+2)$, we consider the following four cases:

(C1) If $u^*(k+1) \in [0, \widehat{f}(k+1)]$ and $u^*(k+2) \in [0, \widehat{f}(k+2)]$, then $u(k+1) \leftarrow u^*(k+1)$ and $u(k+2) \leftarrow u^*(k+2)$.

(C2) If $u^*(k+1) \notin [0, \widehat{f}(k+1)]$ and $u^*(k+2) \notin [0, \widehat{f}(k+2)]$, then the optimal solution is one of the four possible pairs, $(0,0)$, $(\widehat{f}(k+1), 0)$, $(0, \widehat{f}(k+2))$ and $(\widehat{f}(k+1), \widehat{f}(k+2))$. One can compute $J(0,0)$, $J(\widehat{f}(k+1), 0)$, $J(0, \widehat{f}(k+2))$ and $J(\widehat{f}(k+1), \widehat{f}(k+2))$ and select the one that gives the smallest value.

(C3) If $u^*(k+1) \in [0, \widehat{f}(k+1)]$ and $u^*(k+2) \notin [0, \widehat{f}(k+2)]$, then the optimal solution is one of the two possible pairs, $(u^*(k+1), 0)$ and $(u^*(k+1), \widehat{f}(k+2))$. One

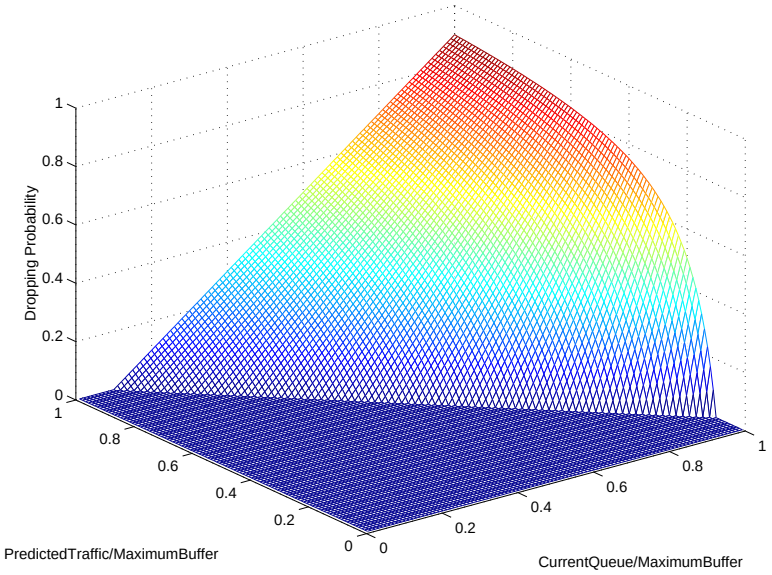
can compute $J(u^*(k+1), 0)$ and $J(u^*(k+1), \widehat{f}(k+2))$ and determine the optimal controller output accordingly.

(C4) If $u^*(k+1) \notin [0, \widehat{f}(k+1)]$ and $u^*(k+2) \in [0, \widehat{f}(k+2)]$, then the optimal solution is one of the two possible pairs, $(0, u^*(k+2))$ and $(\widehat{f}(k+1), u^*(k+2))$. One can compare $J(0, u^*(k+2))$ and J$(\widehat{f}(k+1), u^*(k+1))$ and determine the optimal controller output accordingly.

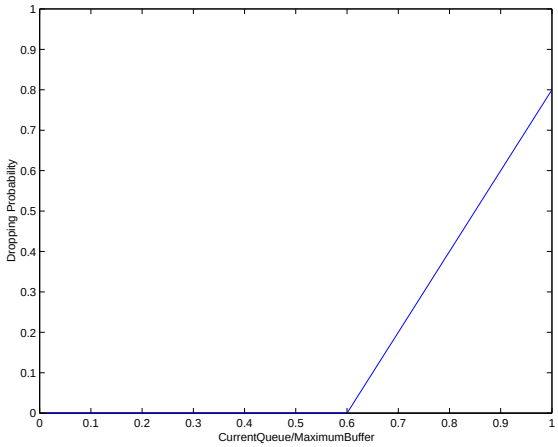
After $u(k+1)$ is determined, we set $p(k+1) = \frac{u(k+1)}{\widehat{f}(k+1)}$ as the packet dropping probability to be used in the next interval. For example, if we use u^* derived in Eq. (5.10), then

$$p(k+1) = \begin{cases} 0, & Q(k) < C + \overline{Q} - \widehat{f}(k+1), \\ \frac{Q(k)+\widehat{f}(k+1)-C-\overline{Q}}{\widehat{f}(k+1)}, & C + \overline{Q} - \widehat{f}(k+1) < Q(k) \\ & \quad < C + \overline{Q}, \\ 1, & Q(k) > C + \overline{Q}. \end{cases}$$

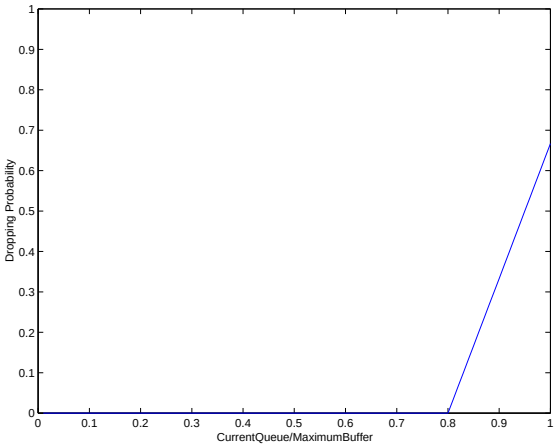
Note that $p(k+1)$ is a linear function of $Q(k)$ when $\widehat{f}(k+1)$ is at a fixed value. Fig. 5.6 gives the values of $p(k+1)$ as a function of $Q(k)$ and $\widehat{f}(k+1)$, with $Q_{opt} = 0.6 \cdot$ MaximumBufferSize and $R = (0.5 \cdot$ MaximumBufferSize $)/\tau$ (i.e., half of the queue can be emptied in one measurement interval). As shown in Fig. 5.6 (b)-(c), the projection of $p(k+1)$ onto the plane that corresponds to a fixed value of $\widehat{f}(k+1)$ is a RED-like curve. That is, RED can be viewed as a special case of PAQM with the estimated future incoming



(a) Dropping probability as a function of current queue size and predicted incoming traffic



(b) $\widehat{f}(k+1) = 0.5 \cdot$ MaxBufSize



(c) $\widehat{f}(k+1) = 0.3 \cdot$ MaxBufSize

Figure 5.6: The packet dropping probability, $p(k+1)$, to be used in the next time interval versus the queue length and the estimated traffic. (The values of $Q(k)$ and $\widehat{f}(k+1)$ are normalized with respect to the maximum buffer size.)

traffic fixed at certain value. Another interpretation of Fig. 5.6 is that a RED queue can be stabilized at a desirable queue length, if it considers the amount of incoming traffic and sets the values of the two thresholds, *min_th* and *max_th*, in compliance with the curves given in Fig. 5.6.

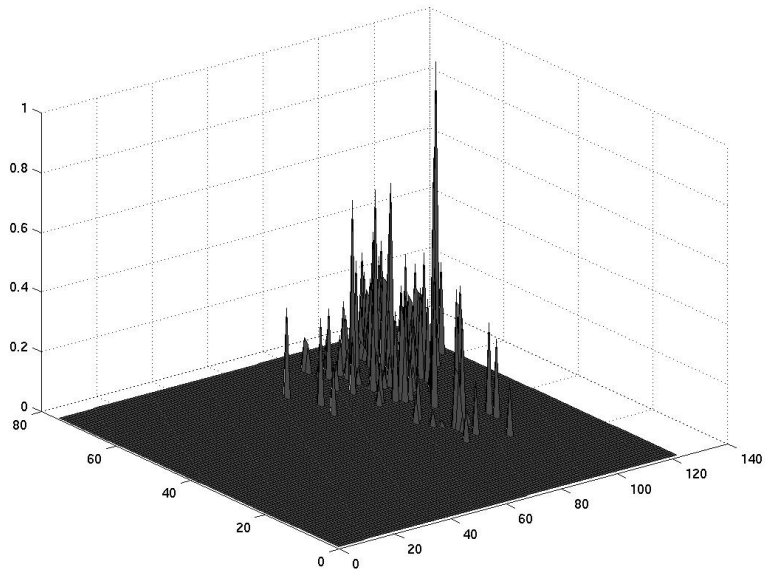


Figure 5.7: The packet dropping probability, $p(k+1)$, calculated in an *ns-2* simulation run.

Fig. 5.7 gives the packet dropping probability, $p(k+1)$, calculated in an *ns-2* simulation with the same setting as in Fig. 5.6. Comparing Fig. 5.7 against Fig. 5.6, we observe that the packet dropping probability used by a router in the simulation indeed lies on the plane depicted in Fig. 5.7.

5.4 Related Work

Several AQM schemes have been proposed after the pioneer work of random early detection (RED) [36], e.g., FRED [58], balanced RED (BRED) [4], BLUE [33], stabilized RED (SRED) [70], random exponential marking (REM) [7], PI controller [41], and AVQ [49]. They can be roughly categorized into three groups with respect to their design objectives: (i) Per-flow fairness: fair RED (FRED) [58], balanced RED (BRED) [4], and Stochastic Fair Blue (SFB) [32]; (ii) Stabilizing the instantaneous queue length: stabilized RED (SRED) [70]; and (iii) Achieving high link utilization and low packet loss ratio simultaneously: BLUE [33], PI controller [41], random exponential marking (REM) [7], and adaptive virtual queue (AVQ) [49]. These schemes differ in (1) the performance objectives (in addition to that of notifying end hosts of incipient congestion by dropping/marking packets); (2) the parameters used as an indicator of congestion; and (3) the policies used to detect (incipient) congestion and to drop/mark packets. In what follows, we summarize these schemes, and give in Table 5.1 a taxonomy with respect to the above three aspects.

Schemes that aim to achieve fairness: In FRED, a router monitors, not only the global average queue length, but also the average queue length, $qlen_i$, of each individual active connection i. Moreover, two minimum and maximum thresholds are defined for the per-flow average queue length. When a packet from flow i arrives, $qlen_i$ is compared against these two thresholds. A flow with $qlen_i$ less than the minimum limit is not subject to random early dropping even if *minth* $\leq$ *avg_queue* $\leq$ *maxth*. On the other hand, a flow, which consistently exceeds the maximum threshold is subject to more aggressive dropping.

Category	Scheme	Congestion index	Policies used to detect congestion and to drop/mark packets
Achieving fairness	FRED [58]	queue length	Monitors the per-flow queue length and fine-tunes the dropping/marking decision w.r.t. the per-flow queue length.
	BRED [4]	queue length	Defines three thresholds and divides the state of per-flow queue length into 4 regions. Fine-tunes the dropping/marking decision w.r.t. the per-flow state.
Achieving high utilization and low packet loss	BLUE [33]	queue length, link idle event	Increases p if the instantaneous queue length exceeds L and has not been updated for over *freeze_time*. Decreases p if the link is idle for over *freeze_time*.
	REM [7]	queue length, input rate	Defines the price function, $c(k)$, as in Eq. (5.12), and calculates the marking probability as $p(k) = 1 - \phi^{-c(k), \phi > 1}$.
	AVQ [49]	input rate	Maintains a virtual queue. At each packet arrival, enqueue a fictitious packet and update the virtual queue capacity using Eq. (5.14). Mark/drop a real packet only if the virtual queue overflows.
Stabilizing queue	SRED [70]	queue length	keeping a zombie list to keep track of recently seen flows, to detect misbehaving flows, and to estimate the number, N, of active flows. Figures in N in the packet dropping probability.
	PI [41]	queue length, input rate	Calculates the marking probability as in Eq. (5.15).
	Scalable control [75]	input rate	Router updates its price function, $p_l(t)$, as in Eq. (5.16), and marks packets with probability as $1 - \phi^{-p_l(t)}, \phi > 1$. Source sets its rate as $x_i(t) = x_{max,i} e^{-\frac{\alpha_i q_i(t)}{M_i \tau_i}}$.

Table 5.1: A taxonomy of AQM schemes.

BRED extends FRED and imposes three thresholds, ℓ_1, ℓ_2, and ℓ_3, on per-flow queue length, $qlen_i$. The three thresholds divide the space of $qlen_i$ into 4 regions: (0, ℓ_1), (ℓ_1, ℓ_2), (ℓ_2, ℓ_3), and (ℓ_3, ∞), each of which is associated with a dropping probability of 0, p_1, $p_2 (> p_1)$, and 1, respectively. A router keeps, for each active flow i, the queue length, $qlen_i$, and the number of its packets accepted into the queue since last drop, gap_i. The dropping probability for a packet from flow i is then a function of (i) the region $qlen_i$ is in and (ii) gap_i. The reason for figuring gap_i into the dropping probability is to prevent

consecutive multiple drops from a flow. In essence, both FRED and BRED aim to improve the fairness of RED at the expense of keeping per-active-flow state information.

Schemes that decouples the congestion index and the performance index: Schemes in this category aim at achieving both high utilization and low packet delay (queue length). The key idea is to decouple the congestion measure from the performance measure. Specifically, these schemes either use additional measures (e.g., link utilization, input rate) as congestion indices, or introduce an intermediate entity (the price function in REM or the virtual queue in AVQ) so that calculation of the dropping probability is not *directly* related to the actual queue length.

In BLUE, the instantaneous queue length and the link utilization are used as the indices of traffic load, and a single dropping probability p is maintained and used to mark or drop packets upon packet arrival. If the instantaneous queue length exceeds a pre-determined threshold, L, a BLUE router increases p by an amount of *delta* (which is a system parameter). To avoid dropping packets too aggressively, BLUE keeps a minimum interval, *freeze_time*, between two successive updates of p. Conversely, if the link is idle (i.e., the queue is empty), the BLUE gateway decreases p by an amount of $delta$ periodically (once every *freeze_time*). By adjusting p with respect to the instantaneous queue length and link utilization (idle events), BLUE is shown through simulation to make the instantaneous queue length converge to an operational point with small buffer sizes, while retaining all the desirable features of RED.

REM decouples the congestion measure from the performance measure by defining the price function, $c(k+1)$, as

$$c(k+1) = \max(0, c(k) + \gamma(\alpha(Q(k) - Q_{opt}) + x(k) - R)), \tag{5.12}$$

where $x(k)$ is the aggregate input rate and R is the capacity of the outgoing link. The $(\alpha(Q(k) - Q_{opt})$ term is the queue mismatch, and the $x(k) - R$ term the rate mismatch. Since $x(k) - R$ measures the rate at which the queue length grows, it can be approximated as $Q(k+1) - Q(k)$, and Eq. (5.12) reduces to

$$c(k+1) = \max(0, c(k) + \gamma(Q(k+1) - (1-\alpha)Q(k) - \alpha Q_{opt})). \tag{5.13}$$

The price increase if the weighted sum of these mismatches is positive, and decrease otherwise. A REM router calculates the marking probability periodically as $p(k) = 1 - \phi^{-c(k)}$, where ϕ is an arbitrary constant that is greater than 1.

The AVQ scheme, on the other hand, takes a dramatically different approach, and uses solely the input rate, $x(t)$, as the congestion index. An AVQ router maintains a virtual queue whose capacity, $\hat{R}$, is adjustable. Upon packet arrival, the virtual queue capacity is updated according to

$$\frac{d\hat{R}}{dt} = \alpha(\gamma R - x(t)). \tag{5.14}$$

where γ is the desired utilization. The rationale behind Eq. (5.14) is to mark/drop packets more aggressively when the arrival rate exceeds the desired utilization (γR) and vice versa. Also, a fictitious packet is enqueued in the virtual queue if space is available. Otherwise, the fictitious packet is not enqueued and the real packet in the real queue is marked/dropped.

The rule for choosing the parameter α is rigorously analyzed using a control theoretic approach to ensure system stability. Through simulation in [49], AVQ is shown to outperform REM in terms of reducing the packet drop rate and average queue length and achieving high utilization.

Schemes that stabilize the instantaneous queue length: SRED argues that the instantaneous queue length may fluctuate dramatically under RED if the number of active flows varies. To stabilize the instantaneous queue, SRED equips each queue with a zombie list that keeps a list of M recently seen flows. When a packet arrives, it is compared with a randomly chosen zombie in the zombie list. The result of a hit or a miss is used to detect potential misbehaving flows for more aggressive dropping and to estimate the number of active flows. The estimated value of N is then figured into the calculation of the dropping probability p (e.g., p is an increasing function of N) so as to avoid, upon packet loss, the situation of significant system throughput decrease in the case that there are only a few active flows. The simulation results indicated that SRED keeps the buffer occupancy close to the specified value and away from overflow or underflow.

The PI controller also aims to stabilize the instantaneous queue length, but it is built upon on a fluid model proposed in [64] and takes a more systematic approach. The PI controller marks each packet with a probability p which is updated periodically using

$$p(k+1) = p(k) + a(Q(k+1) - Q_{opt}) - b(Q(k) - Q_{opt}), \tag{5.15}$$

where $a > 0$ and $b > 0$ are constants chosen according to the design rules given in [41].

The scalable control scheme proposed in [75] uses the link's price $p_l(t)$ as the congestion index and marks packets with probability $1 - \phi^{-p_l(t)}, \phi > 1$. The link then updates its price $p_l(t)$ using the aggregate input rate $y_l(t)$ according to:

$$\dot{p_l}(t) = \begin{cases} \frac{y_l - c_l}{c_l} & \text{if } p_l(t) > 0; \\ \max\{0, \frac{y_l - c_l}{c_l}\} & \text{if } p_l(t) = 0 \end{cases} \tag{5.16}$$

in which c_l is the *virtual capacity* that is strictly less than the actual link capacity. The source will set its sending rate as an exponential function of aggregate price $q_i(t)$, i.e. $x_i(t) = x_{max,i} e^{-\frac{\alpha_i q_i(t)}{M_i \tau_i}}$. To utilize this scheme, the current TCP congestion control and avoidance scheme has to be changed.

The work that comes closest to ours is SRED, as both share the same objective of stabilizing the instantaneous queue. Hence, we will make comprehensive performance comparisons between SRED and PAQM in Section 5.5. On the other hand, as a side effect of using the amount of traffic that arrive in the next few intervals to determine the packet dropping probability, PAQM also decouples the congestion measure and the performance measure and can be classified into the second category. Hence, we will also compare PAQM against AVQ (which is reported to give the best performance in the second category) in Section 5.5.

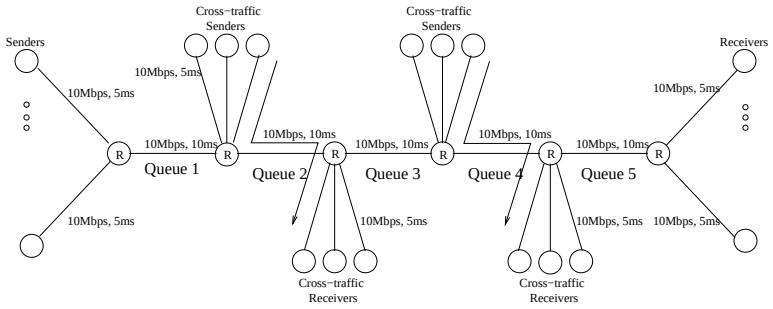


Figure 5.8: The multiple bottleneck simulation topology.

5.5 Simulation Results

We have implemented PAQM along with RED [36], SRED [70], and AVQ [49] in ns-2, and conducted a simulation study to validate the proposed design and compare the performance of PAQM against the other schemes. We examine the behavior of these schemes under a variety of network topologies and traffic sources. In particular, we have considered the network topologies with a single bottleneck link, (e.g., Fig. 3.8), with multiple bottleneck links (e.g., Fig. 5.8), as well as of arbitrary topology. The maximum buffer size of each router is set to 100 packets (of size 1000 bytes) under PAQM, RED, and AVQ, and 20 packets under SRED. The reason for choosing a different maximum buffer size under SRED is that through extensive simulation, we find that SRED always attempts to keep the queue close to full. The value is so chosen that the queue length is comparable to those of PAQM and RED. We use an assortment of traffic sources (namely TCP sources that generate packets according to the on-off model or real traffic traces down-loaded from the

Internet, and constant bit rate UDP sources). Due to the space limitation, we only report on a small set of the simulations that we believe is the most representative.

The parameters used in PAQM are: Q_{opt} is set to 20 packets[10], and τ is in the range of seconds. The parameters of RED are set as recommended in http://www.aciri.org /floyd /REDparameters.txt, and those of SRED are chosen as recommended in [70] (i.e., $M = 1000$, $\alpha = 1/M = 0.001$, and $p_{max} = 0.15$). Finally, the desirable utilization, γ, of AVQ is set to 0.98, and the damping factor, α, is determined in compliance with Theorem 1 in [49] to ensure system stability ($\alpha = 0.15$).

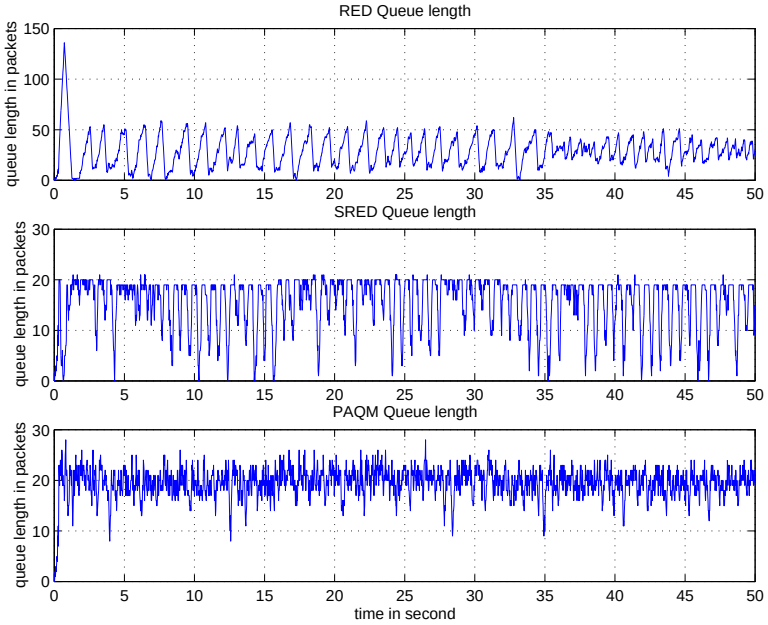
Each data point is the result averaged over 10 simulation runs. In spite of numerous system parameters involved, the results are found to be quite robust in the sense that the conclusion drawn from the performance curves for a representative set of parameter values (reported below) is valid over a wide range of parameter values.

5.5.1 Comparison Between RED, SRED, and PAQM

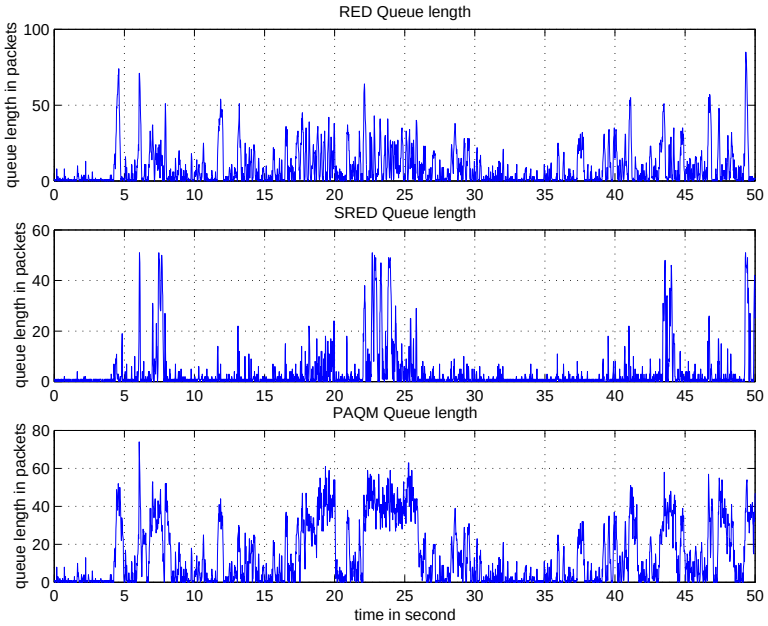
In this set of experiments we compare PAQM against RED and SRED with respect to instantaneous queue length, packet loss ratio, and total throughput attained by receivers under different network topologies and traffic mixes.

Simulation results under the single bottleneck topology: k TCP connections are established over a single bottleneck link of capacity 20 Mbps. where k varies from 20 to 100. Fig. 5.9 gives the instantaneous queue length in the cases that 60 TCP connections

[10] As a matter of fact, the performance of PAQM is not very sensitive to Q_{opt}.



(a) on-off model



(b) real traffic trace

Figure 5.9: Instantaneous queue length in the single bottleneck network with TCP sources.

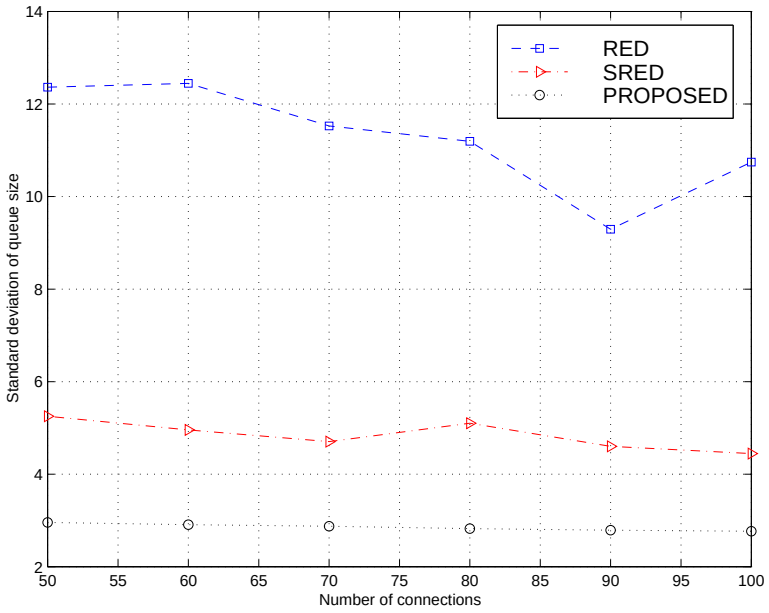
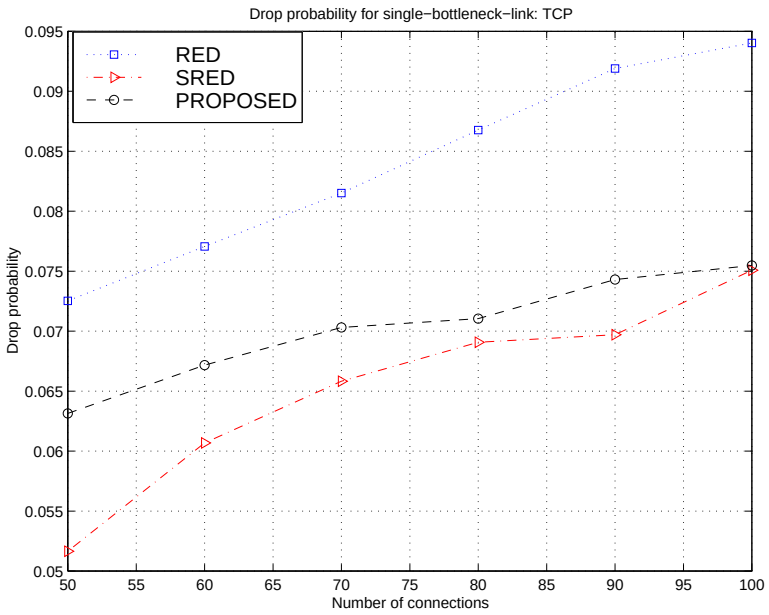


Figure 5.10: The standard deviation of the instantaneous queue length in the single bottleneck network.

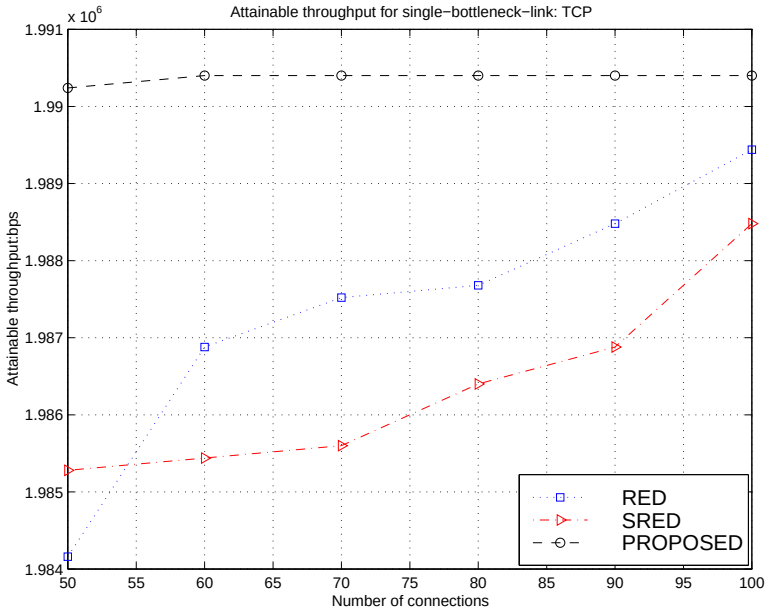
are established and generate packets using either the on-off model ((a)) or the real traffic traces ((b)). Fig 5.10 gives the standard deviation of instantaneous queue length with the same setting in Fig. 5.9 (a) (except that the number of connections varies from 50 to 100). As compared to RED and SRED, PAQM indeed keeps the queue at the desirable level and better stabilizes the queue.

Fig 5.11 gives the packet loss ratio and the throughput attained by all receivers under the case of TCP sources with the on-off model. PAQM performs better than SRED w. r. t. attainable throughput, but slightly worse than SRED w. r. t. packet loss ratio. This is because SRED always attempts to keep the queue (close to) full, and hence does not proactively drop packets that aggressively. However, keeping the queue full also makes the queue more susceptible to the global synchronization effect. This is evidenced in Fig. 5.9 (a) that the instantaneous queue length frequently oscillates between empty and full. This also accounts for the fact that SRED does not attain as much throughput as PAQM.

Simulation results under the multiple bottleneck network: We repeat the same experiments in a network with multiple bottlenecks. As shown in Fig. 5.8, there are 5 queues among which queue 2 and queue 4 are shared with cross traffic of 20 TCP connections. Again we establish k TCP connections with senders at the left hand side and receivers at the right hand side, where k varies from 20 to 100. The cross traffic is composed of TCP connections as well, and the number of TCP connections in each cross traffic bundle is set to $0.5k$. The simulation results show that the queue length at queue 5 is always 0 or 1, suggesting that the link is not a bottleneck link. The other four queues exhibit similar



(a) packet loss ratio



(b) attainable throughput

Figure 5.11: The packet loss ratio and attainable throughput in the single bottleneck network.

trends as far as the performance comparison is concerned. Hence, we arbitrarily choose to depict the instantaneous queue length, the packet loss ratio and the attainable throughput of queue 2 in Fig. 5.12 and 5.14, respectively. PAQM outperforms the other two schemes with respect to all three measures. The reason why SRED does not perform as well in terms of packet loss ratio is because as SRED has the tendency to keep the queue (close to) full, it becomes more likely that packet losses occur as a result of buffer overflow, when the link traffic becomes more bursty/congested with cross traffic.

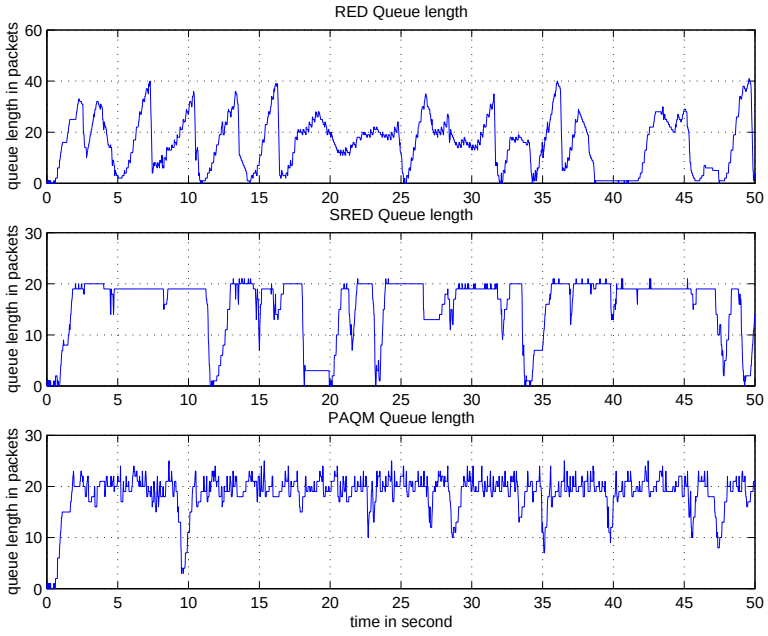


Figure 5.12: The instantaneous queue lengths at queue 2 in the multiple bottleneck network.

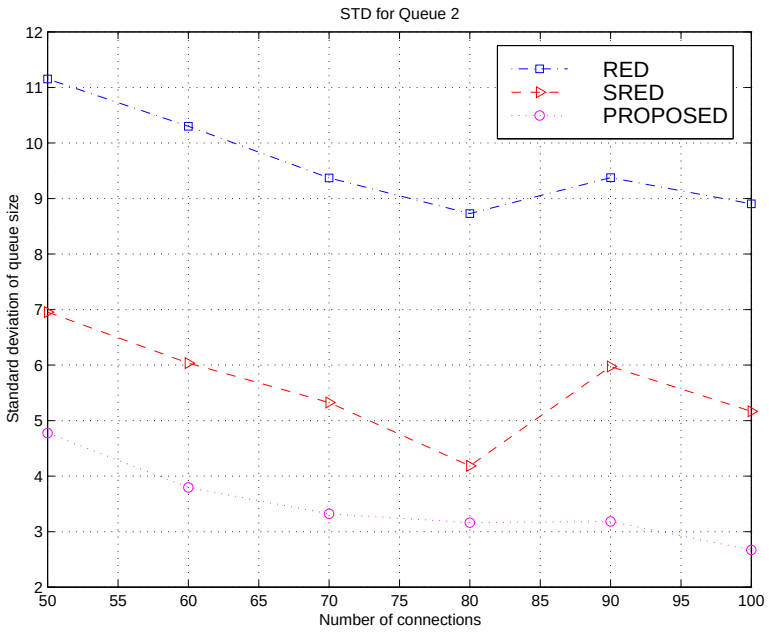
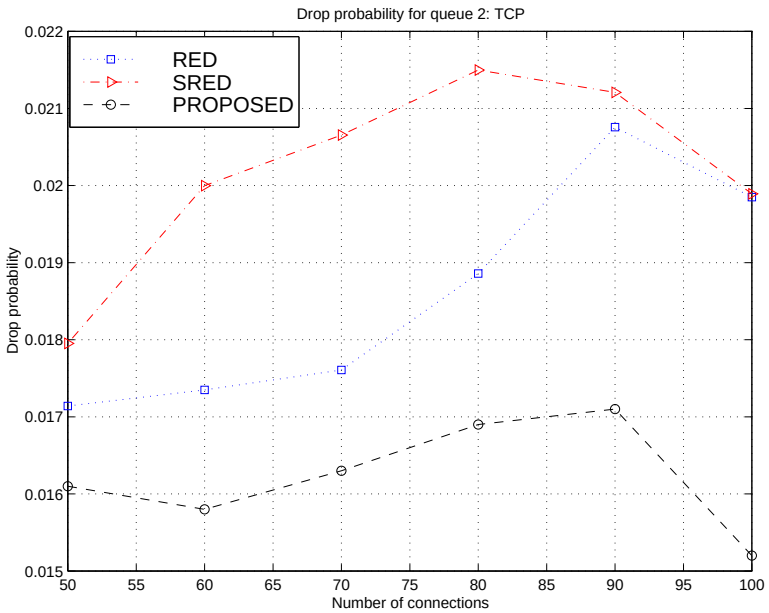
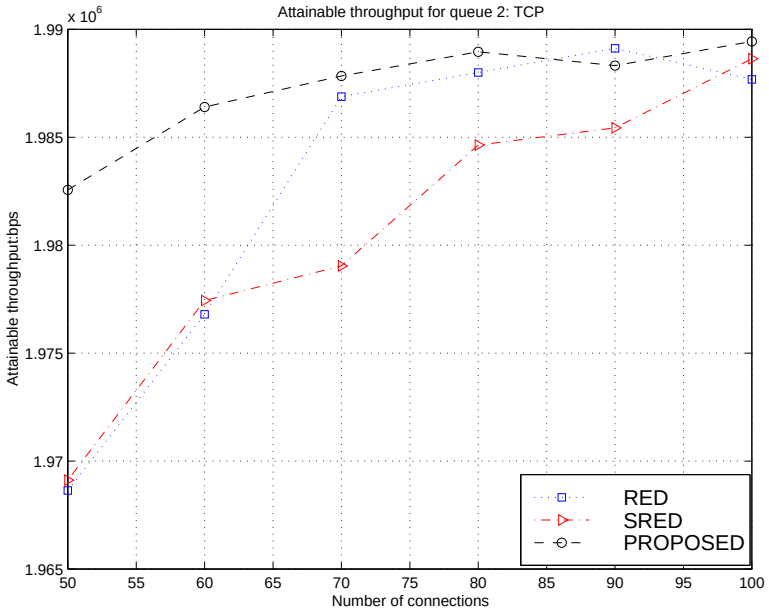


Figure 5.13: The standard deviation of the instantaneous queue length at queue 2 in the multiple bottleneck network.



(a) loss ratio



(b) utilization

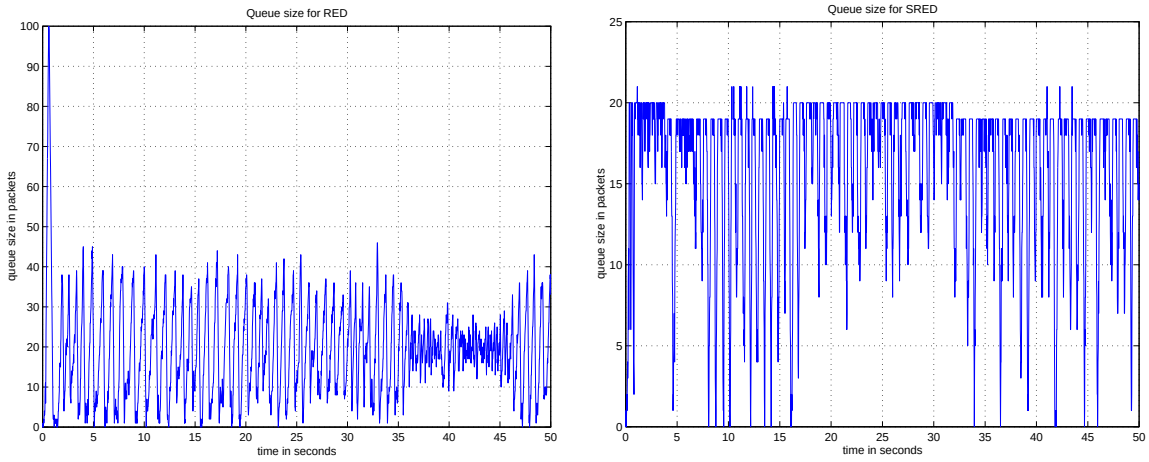
Figure 5.14: The packet loss ratio and link utilization at queue 2 in the multiple bottleneck network.

We have also conducted simulation on several arbitrary network topologies but do not report the results here due to the space limitation. The interested reader is referred to [38] for a detailed account of these results.

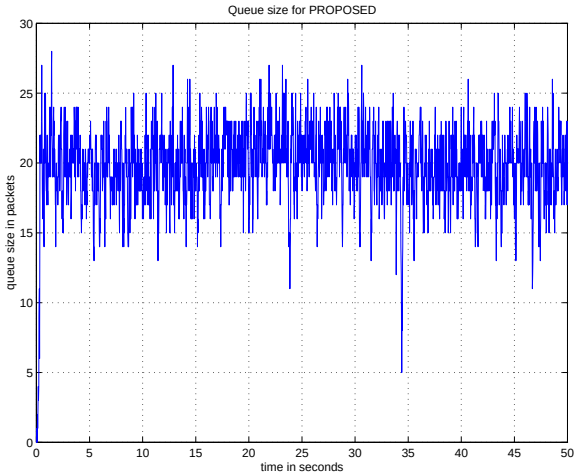
Simulation results in the case of dynamic connection establishment and termination: To test the responsiveness in stabilizing the queue in the case that connections are dynamically established and terminated, we repeat the same experiment in the single bottleneck network except that initially 40 TCP connections commence at time 0, at time 10 another 20 connections are established, and finally at time 30, 20 connections are terminated. All the simulation runs last for 50 seconds. Fig. 5.15 gives the instantaneous queue length under RED, SRED, and PAQM. The queue length under PAQM is always stable around $Q_{opt} = 20$ packets, regardless of change of the number of connections. The queue size under RED, on the other hand, oscillate dramatically almost all the times. SRED again keeps its queue length close to the maximum buffer size, and hence is subject to the global synchronization effect experienced by a drop-tail queue. This is evidenced by the fact that the instantaneous queue length frequently oscillates between empty and full.

5.5.2 Comparison between AVQ and PAQM

As mentioned in Section 5.4, although PAQM is not targeted to decouple the congestion measure and the performance measure, the fact that it takes into account of the amount of future traffic in the next two measurement intervals (or equivalently, the future arrival rate) does help to achieve this objective. To illustrate this, we repeat the same experiment in the



(a) RED (b) SRED

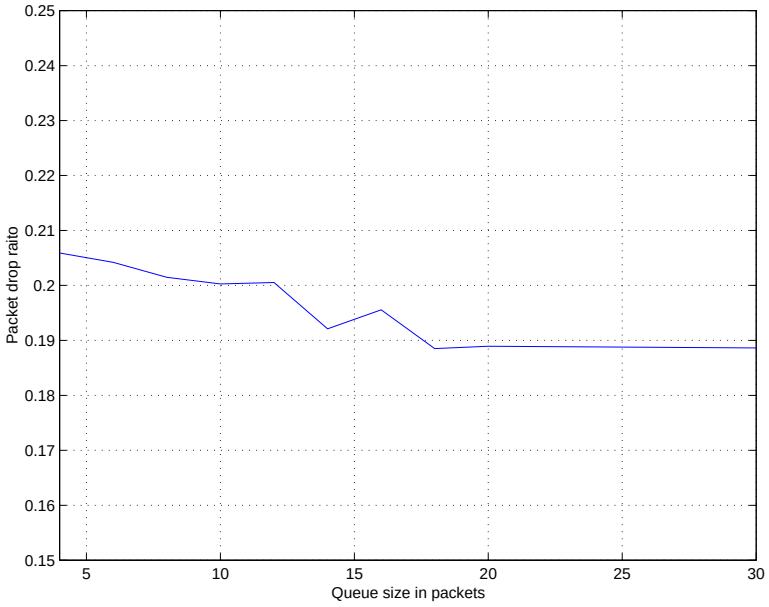


(c) PAQM

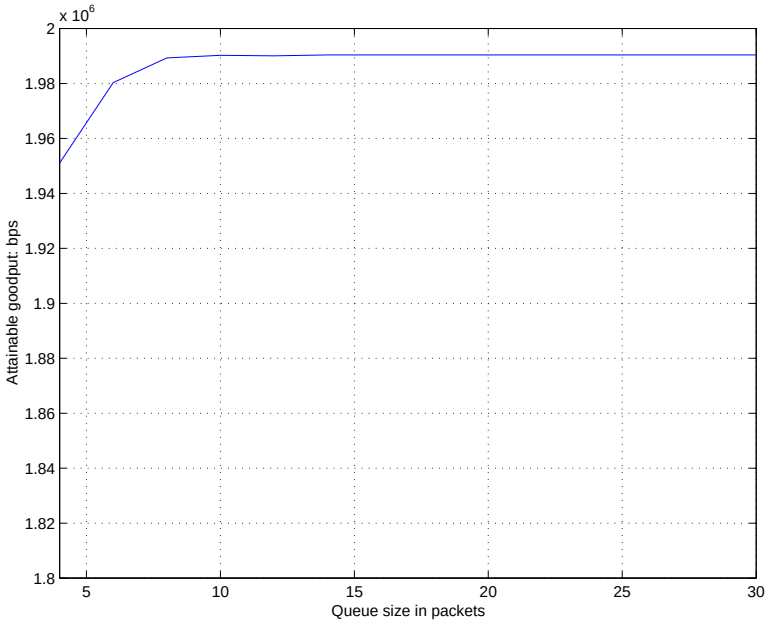
Figure 5.15: The instantaneous queue length in the case of dynamic connection establishment and termination.

single bottleneck network, but change Q_{opt} from 4 to 30, and measure the packets loss ratio and attainable throughput. Fig. 5.16 gives the simulation results. As Q_{opt} changes from 4 to 35, the packet loss ratio changes from 0.206 to 0.19, while the attainable throughput changes from 1.95 Mbps to 1.99 Mbps (the latter levels off when $Q_{opt} \geq 8$). This suggests that under PAQM the equilibrium value of the congestion measure is independent of the equilibrium performance measure (e.g., packet loss or attainable throughput).

As PAQM does exhibit the decoupling behavior, we compare it against AVQ — the scheme currently reported in [49] to give the best performance in the second category (Table 5.1). Again we repeat the same experiment in the single bottleneck network except that we set $Q_{opt} = 2.5$ under PAQM. This is because AVQ usually keeps its average queue length at 2.5 packets under the given simulation setting. By setting $Q_{opt} = 2.5$ packets, a fair comparison can then be made with respect to packet loss ratio and attainable throughput. Fig. 5.17 gives the simulation results. The average queue length both under AVQ and PAQM is kept around 2.5 packets, but PAQM achieves better performance both in terms of packet loss ratio and attainable throughput. This demonstrates the powerfulness of exploiting LRD and taking into account of future arrival rate in determining the packet dropping probability. We have observed similar trends in simulation runs conducted in the multiple bottleneck network and in the network of arbitrary topology, but due to the space limitation, do not include the results.

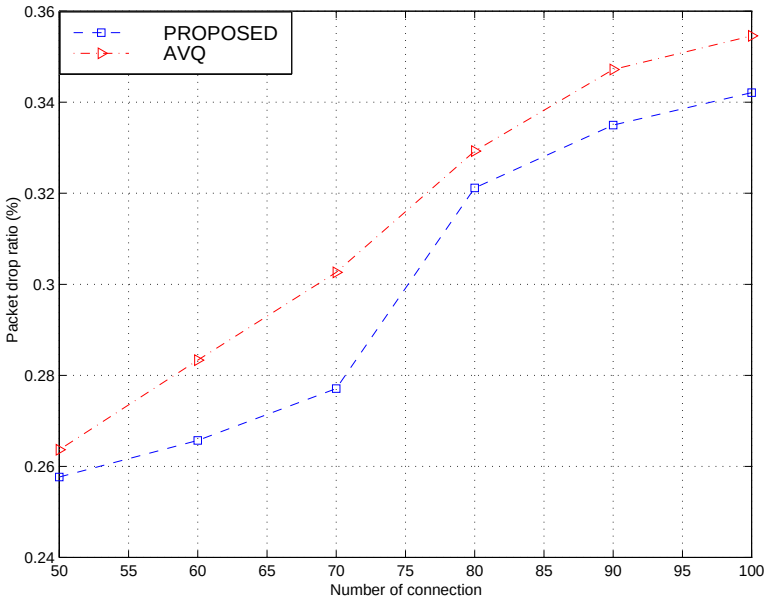


(a) packet loss ratio

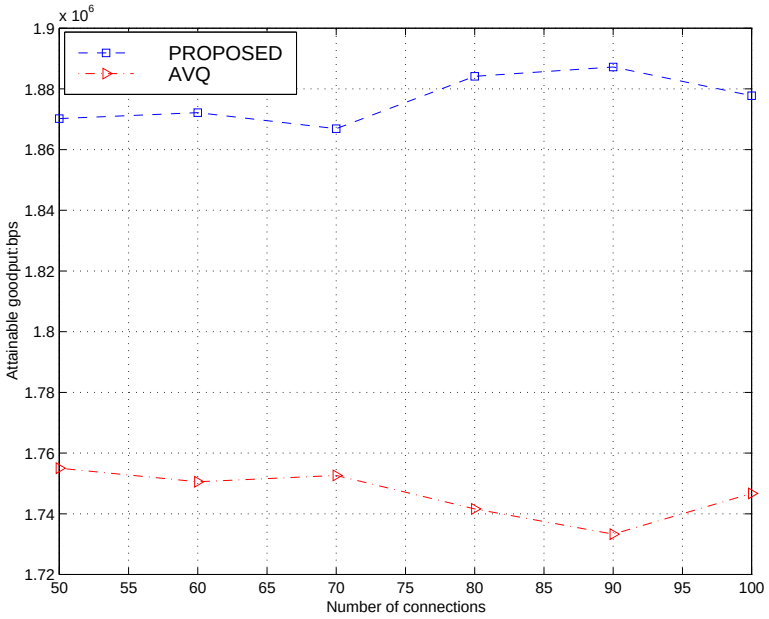


(b) link utilization

Figure 5.16: The packet loss ratio and link utilization versus Q_{opt}.



(a) packet loss ratio



(b) link utilization

Figure 5.17: Performance comparison between AVQ and PAQM.

5.6 Conclusion

We have explored in this chapter the issue of exploiting traffic predictability to enhance the performance of AQM. We show that the correlation structure present in long-range dependent traffic can be detected on-line and used to accurately predict the future traffic. We then figure in in the calculation of the packet dropping probability the prediction results as a new dimension of congestion index. By stabilizing the instantaneous queue length at a desirable level (in anticipation of future traffic), PAQM enables the link capacity to be fully utilized, while not incurring excessive packet loss ratio. Through *ns-2* simulation, we show that under most cases PAQM outperforms SRED in stabilizing the instantaneous queue length, and AVQ in reducing packet loss ratio and utilizing the link capacity.

It is worth mentioning that PAQM is orthogonal to REM, PI, and AVQ in the sense that the effect of incoming traffic (predicted through exploitation of LRD) can be figured in in the calculation of the price function (Eq. (5.12)) or in the adjustment of the virtual queue capacity (Eq. (5.14)) to further improve their performance. As PAQM has been shown to be a generalized scheme of RED with future traffic figured in (Fig. 5.6), one can expect that REM/PI/AVQ, when coupled with PAQM, also gives a more generalized version expectedly with better performance.

CHAPTER 6

A State Feedback Control Approach to Stabilizing Queues for ECN-Enabled TCP Connections

In this chapter, we present an enhanced TCP model that considers effects ignored in the previous models and design a state feedback AQM controller based on the enhanced model. We first give an overview of the states of art in TCP models and indicate their deficiencies (Section 6.1). Then we present our enhanced TCP model (Section 6.2). Following that, we linearize our model and analyze its local stability (Section 6.3), and design a state feedback AQM controller based on our linearized model (Section 6.4). Finally, we discuss through theoretical reasoning, the algorithm implementation and parameter setting issues (Section 6.5), and present simulation results (Section 6.6).

6.1 Overview

In the past few years, significant research efforts have been made to study/improve the performance of RED (or, in general, AQM). These efforts can be roughly classified into three categories. In the first category, approaches are devised to adaptively, on-line adjust

RED parameters according to the condition of network congestion [4, 33, 58, 70]. Most of the approaches proposed in this category are heuristic-based and validated through simulation. In the second category, interactive behaviors of TCP connections and AQM controllers are characterized as a gradient optimization problem [47, 49, 75], with the objective of maximizing the utility of the network. The optimization based approaches proposed in this category primarily focus on the steady state equilibrium, rather than the transient behavior, of the queue. The third category envisions a network that consists of TCP connections and AQM routers as a dynamic feedback control system, in which AQM routers act as controllers and TCP traffic sources act as plants [47, 64]. Several analytical models are proposed to approximate the dynamic AIMD behaviors of TCP in conjunction with AQM [47, 64]. Automatic control theory is then used to analyze (with respect to controllability and stability) and design AQM controllers. Our proposed work falls in the third category.

The analytical models proposed in the third category provide new insights on designing better AQM controllers and detecting problematic parameter settings. Succinctly, in the TCP congestion avoidance phase, a TCP connection uses the AIMD algorithm to adjust its congestion window size *cwnd*. That is, for each positive acknowledgment received, it increases *cwnd* by $\frac{1}{cwnd}$, and in the case of congestion indication (i.e., receipt of three duplicate acknowledgments or ECN), it reduces *cwnd* by half to $\frac{cwnd}{2}$ [43]. These analytical models characterize the AIMD behavior of TCP as follows [47]: as each positive acknowledgment increases *cwnd* by $1/cwnd$ and each congestion indication reduces the *cwnd* by half to $\frac{cwnd}{2}$, the rate at which the expected congestion window size changes is expressed as

$\frac{1-p}{\tau} - \frac{w^2 p}{2\tau}$, where τ is the round-trip delay of a TCP connection, w is the current congestion window size, and p is the dropping or marking probability. The changing rate is then taken as the TCP dynamic behavior for analysis and design.

Although the controllers designed under the aforementioned models are shown, via simulation, to perform well, two effects are not considered in these models. First, the congestion window is not *gradually* decreased at the rate of $\frac{w^2 p}{2}$, but *suddenly* halved upon receipt of congestion indication. Second, the congestion window is halved at most once during one RTT. In this chapter, we present an enhanced model that takes into account of these effects as well as the TCP option of delay acknowledgment. We show that the new model characterizes the TCP dynamics more realistically and that under the new model $cwnd$ decreases faster. We then analyze the stability of its linearized model and design, with the use of state feedback control theory, a controller to stabilize the queue at an AQM router. The resulting AQM controller is called the *state feedback controller (SFC)*. All the algorithm implementation and parameter setting issues are carefully considered and validated through theoretical reasoning. We also evaluate via *ns-2* simulation the performance of the new controller and compare it against other existing schemes. The simulation results show that SFC outperforms other schemes in terms of fluctuation in the queue length, link utilization, and packet loss ratio.

6.2 An Enhanced TCP Model

In this section, we take into account of (i) effects that were previously ignored in other analytical models and (ii) the TCP option of delay acknowledgment, and derive a new model to characterize the expected transient behavior of the TCP congestion window in the congestion avoidance phase. We also compare the new model against existing models.

Similar to the existing models [47, 64], we assume that (**A1**) TCP connections operate in the congestion avoidance phase[11]; (**A2**) the change in the packet dropping/marking probability is insignificant in one RTT, τ; and (**A3**) packets are marked independently. While other models take the expected rate at which *cwnd* changes over one acknowledgment as the approximate *cwnd* change rate, we calculate the expected *cwnd* change, $E(\Delta w)$, over one RTT (τ), and use $\frac{E(\Delta w)}{\tau}$ as the *cwnd* change rate. As will be clearer below, with this subtle change we will be able to figure in effects ignored in other models.

We model the TCP behavior in the congestion avoidance phase in terms of "cycles." An old cycle ends and a new cycle begins when all data packets in the previous congestion window are acknowledged. In the time axis, a cycle takes approximately one RTT, τ. Let the size of the current congestion window and the size of the congestion window one RTT before be denoted, respectively, as w and w'. By definition, totally w' packets are acknowledged in the current cycle. Let the number of packets that are acknowledged by a received acknowledgment (ACK) be denoted as b. If each ACK acknowledges only one

[11] It has been observed in [18] that the majority of Internet traffic is still dominated by long-lived TCP connections and most long-lived TCP connections operate in the congestion avoidance phase most of the time.

packet, $b = 1$. On the other hand, if the delayed ACK option is used (i.e., one ACK is sent for every two data packets received), $b = 2$.

If the ECN bit of the kth acknowledgment is marked, the current congestion window size, $cwnd = w + \frac{k-1}{bw}$, will be halved. As the congestion window is halved at most once in one RTT, after one cycle, the change in the congestion window size (in unit of MSS) is $\frac{1}{b} - \frac{w}{2} - \frac{k-1}{2bw}$. Let p denote the probability that the ECN bit of a packet is marked in the current cycle. Then, under assumptions (**A2**) and (**A3**), the probability that the kth data packet is the first with the ECN bit marked in the current congestion window (so that the kth acknowledgment carry the ECN indication) is $(1-p)^{k-1}p$. If no ECN bit is marked or three duplicate ACKs received, *cwnd* will be increased by $\frac{w'}{bw}$, and the corresponding probability is $(1-p)^{w'}$. By the end of each cycle, the expected change in the congestion window size can be expressed as

$$\begin{aligned} E(\Delta w) &= \sum_{k=1}^{w'}(1-p)^{k-1}p\left(\frac{w'}{bw} - \frac{w}{2} - \frac{k-1}{2bw}\right) + \frac{w'(1-p)^{w'}}{bw} \\ &= -\frac{1-(1-p)^{w'}\ w'p - (1-p)^{w'} - p + p(1-p)^{w'}}{2bwp} \\ &\quad -\frac{w}{2}\left[1-(1-p)^{w'}\right] + \frac{w'}{bw}. \end{aligned} \tag{6.1}$$

When p is small, i.e. $wp \ll 1$, we can consider first-order items of p while ignoring high-order items and simplify Eq. (6.1) to

$$\begin{aligned} E(\Delta w) &\approx -\frac{w'(w'-1)}{2bw}p - \frac{ww'}{2}p + \frac{w'}{bw} \\ &= \frac{w'}{bw} + (\frac{w'}{2bw} - \frac{w'^2}{2bw})p - \frac{ww'}{2}p. \end{aligned} \tag{6.2}$$

The rate at which *cwnd* changes can then be approximated as

$$\frac{dE(w)}{dt} \approx \frac{E(\Delta w)}{\tau}. \tag{6.3}$$

Notice that $w' = w(t-\tau)$ and the packet dropping probability that a TCP connection perceives also incurs a time delay τ, i.e., $p = p(t-\tau)$. Substituting w' and $p(t-\tau)$ into Eq. (6.3), we have

$$\begin{aligned}\frac{dE(w)}{dt} &= \frac{w(t-\tau)}{b\tau w(t)} + \left[\frac{w(t-\tau)}{2b\tau w(t)} - \frac{w^2(t-\tau)}{2b\tau w(t)}\right] p(t-\tau) \\ &\quad - \frac{w(t)w(t-\tau)}{2\tau} p(t-\tau). \end{aligned} \tag{6.4}$$

Comparison against other models: Now we compare the newly derived model with other analytical models. Misra's model [64] (which is scaled by b to take into account of the delayed ACK option) is given below:

$$\frac{dE(w)}{dt} = \frac{1}{b\tau} - \frac{w(t)w(t-\tau)}{2\tau} p(t-\tau). \tag{6.5}$$

Comparing Eqs. (6.4) and (6.5), we can see that when $w(t-\tau) \approx w(t)$ and $w(t-\tau) > 1$ (which is always true, as we assume TCP connections operate in the congestion avoidance phase and hence $w(t-\tau) \geq 3$.),

$$\frac{w(t-\tau)}{2b\tau w(t)} - \frac{w^2(t-\tau)}{2b\tau w(t)} < 0. \tag{6.6}$$

Eq. (6.6) implies the second term in Eq. (6.4) is negative, and hence the congestion window size decreases faster in our model than in Misra's model.

Let the sending rate be $x(t) = \frac{w(t)}{\tau}$. Then Eq. (6.4) can be re-written as

$$\begin{aligned}\frac{dx}{dt} &= \frac{x(t-\tau)}{b\tau^2 x(t)} + \left[\frac{x(t-\tau)}{2b\tau^2 x(t)} - \frac{x^2(t-\tau)}{2b\tau x(t)}\right] p(t-\tau) \\ &\quad - \frac{x(t)x(t-\tau)}{2} p(t-\tau). \end{aligned} \tag{6.7}$$

Kelly's model [47] (which is again scaled by b to take into account of the delayed ACK option) is given below:

$$\frac{dx}{dt} = \frac{x(t-\tau)}{b\tau^2 x(t)} - \frac{x(t-\tau)}{b\tau^2 x(t)} p(t-\tau) - \frac{x(t)x(t-\tau)}{2} \cdot p(t-\tau). \tag{6.8}$$

Comparing Eqs. (6.7) and (6.8), we can see when $x(t-\tau) > \frac{3}{\tau}$, we have

$$\frac{x^2(t-\tau)}{2b\tau x(t)} - \frac{x(t-\tau)}{2b\tau^2 x(t)} > \frac{x(t-\tau)}{b\tau^2 x(t)}, \tag{6.9}$$

i.e., the second term in Eq. (6.7) is less than that in Eq. (6.8). By assumption (**A1**), the congestion window size should always be greater than 3 MSS, and hence $x(t-\tau) > \frac{3}{\tau}$ always holds. As a result, the sending rate decreases more rapidly in our model than in Kelly's model. This fact that the congestion window size decreases more rapidly cautions us for the importance of designing appropriate AQM controllers as the impact of the packet dropping/marking probability on the congestion window change is larger than other models expect.

6.3 Analysis of Interaction Between TCP and AQM

In this section, we consider a system in which N homogeneous TCP connections traverse a bottleneck link with bandwidth C. By homogeneous, we mean all the TCP connections incur roughly the same RTT and share the same bottleneck link, although they do not necessarily traverse the same end-to-end path. Let the queue length on the bottleneck link be denoted q and the congestion window size of each TCP connection w. The dynamic

system can be described by

$$\begin{cases} \dot{q} & \triangleq & g(w(t),q) = \frac{N}{\tau}w - C, \\ \dot{w} & \triangleq & f(w(t),w(t-\tau),p) = \frac{w(t-\tau)}{b\tau w(t)} - \frac{w(t)w(t-\tau)}{2\tau}p(t-\tau) \\ & & + \left[\frac{w(t-\tau)}{2b\tau w(t)} - \frac{w^2(t-\tau)}{2b\tau w(t)}\right] p(t-\tau). \end{cases} \tag{6.10}$$

The first differential equation (Eq. (6.10)) states that the queue length is an integral of the difference between the packet arrival rate and the link capacity. The second differential equation (Eq. (6.10)) describes the dynamic behavior of the TCP congestion window and is developed in Section 6.2.

As the system model (Eq. (6.10)) is nonlinear with a time delay, it is impossible to analyze it analytically. Hence, we will first approximate the system model with its small-deviation linearized model around an operating point, say (w_0, p_0), to analyze its local stability. We assume that τ is constant. Let $\delta w \triangleq w - w_0$ and $\delta p \triangleq p - p_0$, in which δw and δp are, respectively, deviations of the congestion window and the dropping probability from the operating point. By setting $g(w(t), q) = 0$ and $f(w(t), w(t-\tau), p) = 0$, we have

$$\begin{aligned} w_0 &= \frac{\tau C}{N}, \\ p_0 &= \frac{2}{bw_0^2 + w_0 - 1} = \frac{2N^2}{b\tau^2C^2 + \tau CN - N^2}. \end{aligned} \tag{6.11}$$

Also,

$$\begin{aligned} \frac{\partial g}{\partial w} &= \frac{N}{\tau}, \\ \frac{\partial f}{\partial w} &= -\frac{p_0 + 2bw_0p_0}{2b\tau}, \\ \frac{\partial f}{\partial p} &= \frac{1}{2b\tau} - \frac{w_0}{2b\tau} - \frac{w_0^2}{2\tau} = -\frac{1}{b\tau p_0}. \end{aligned}$$

Hence, the equations that characterize the system dynamics around the operating point are

$$\begin{aligned}
\dot{\delta q} &= \frac{\partial g}{\partial w}\delta w = \frac{N}{\tau}\delta w, \\
\dot{\delta w} &= \frac{\partial f}{\partial w}\delta w + \frac{\partial f}{\partial p}\delta p \\
&= -\frac{p_0 + 2bw_0p_0}{2b\tau}\delta w - \frac{1}{b\tau p_0}\delta p(t-\tau).
\end{aligned} \tag{6.12}$$

For the system model to be meaningful around the operating point, we require that $w_0 \geq 0$ and $0 \leq p_0 \leq 1$. As $\tau > 0$, $C > 0$ and $N > 0$, w_0 is always greater than 0. To satisfy $0 \leq p_0 \leq 1$, we should have $0 \leq N < \frac{1+\sqrt{12b+1}}{6}\tau C$. All our analysis and design will be restricted to the parameter region mentioned above, because otherwise the system has no equilibrium and can not be stabilized according to the linear model. The transfer function of the system is

$$T(s) = \frac{W(s)}{P(s)} = -\frac{\frac{1}{b\tau p_0}}{s + \frac{p_0+2bw_0p_0}{2b\tau}}e^{-\tau s} \triangleq G(s)e^{-\tau s}, \tag{6.13}$$

i.e., $T(s)$ is a pure-delay system with a non-delay part $G(s)$. As $G(s)$ has a pole at $-\frac{p_0+2bw_0p_0}{2b\tau}$ that lies on the left half of the complex plane, $G(s)$ is stable.

In summary, the system in which N homogeneous TCP connections traverse a bottleneck link with capacity C can be approximated by the following linear differential equation:

$$\begin{cases}
\dot{\delta q} &= \frac{N}{\tau}\delta w, \\
\dot{\delta w} &= -\frac{p_0+2bw_0p_0}{2b\tau}\delta w - \frac{1}{b\tau p_0}p(t-\tau).
\end{cases} \tag{6.14}$$

In the matrix form, the system can be represented as $\dot{x} = Ax + Dp(t-\tau)$, where $x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T = \begin{bmatrix} \delta q & \delta w \end{bmatrix}^T$, $A = \begin{bmatrix} 0 & \frac{N}{\tau} \\ 0 & -\frac{p_0+2bw_0p_0}{2b\tau} \end{bmatrix}$ and $D = \begin{bmatrix} 0 & -\frac{1}{b\tau p_0} \end{bmatrix}^T$. Since

$\begin{bmatrix} D & AD \end{bmatrix}$ is full ranked, the system is controllable. Hence by using the proper control law, we can take system's states, i.e. the queue length at the bottleneck link and the congestion window size, $cwnd$, of TCP connections, to some desirable equilibrium point.

6.4 State Feedback Control AQM

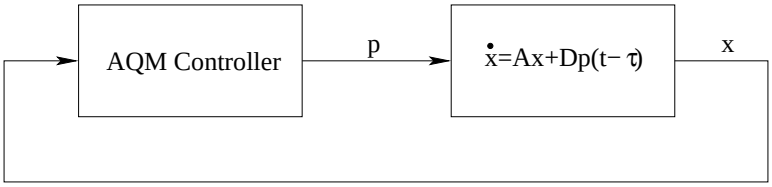


Figure 6.1: The system diagram.

In this section, we design, based on the state feedback control theory, an AQM controller under the linearized model (formulated in Section 6.3), and discuss how tunable parameters should be set to stabilize the system, i.e. to make δq and δw as close to zero as possible. The reasons for using state feedback control are: (i) it is desirable to remove the operation of averaging the queue length (and using it as a congestion index) in several AQM schemes, as it has been believed that this operation brings in more sluggish behaviors to a delay system; (ii) since all system states in state feedback control can be readily obtained or estimated, a state feedback controller can be easily implemented and can quickly respond to system dynamics.

Fig. 6.1 shows the block diagram of the feedback control system that characterizes the interaction between TCP and AQM. The controllable plant is the linearized TCP model and the AQM controller marks the arrived packets with probability p (which is a function of system states) and is the entity to be designed. With the use of state feedback control, we can express the marking/dropping probability as a linear combination of system's states, i.e. $x_1 = \delta q$ and $x_2 = \delta w$. Specifically, let $p\left(t\right) = K \cdot x\left(t\right)$. We have

$$\begin{aligned} \dot{x} &= Ax + Dp(t-\tau) = Ax + D \cdot K \cdot x(t-\tau), \\ &= \begin{bmatrix} 0 & \frac{N}{\tau} \\ 0 & -\frac{p_0+2bw_0p_0}{2b\tau} \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ \frac{-k_1}{b\tau p_0} & \frac{-k_2}{b\tau p_0} \end{bmatrix} x(t-\tau). \end{aligned} \tag{6.15}$$

After Laplace transform, the characteristic polynomial, $D(s)$, of the system can be expressed as:

$$\begin{aligned} D(s) &= \det(sI - A - D \cdot Ke^{-s\tau}) \\ &= \frac{p_0\left(1+2bw_0\right)}{2b\tau}\left[\frac{2N}{\left(1+2bw_0\right)p_0^2\tau}k_1 + \frac{2k_2}{\left(1+2bw_0\right)p_0^2}s\right]e^{-s\tau} \\ &\quad + \frac{p_0\left(1+2bw_0\right)}{2b\tau}\left[1 + \frac{2b\tau}{\left(1+2bw_0\right)p_0}s\right]s. \end{aligned} \tag{6.16}$$

Stable region of k_2: To make the feedback control system stable, we should choose parameters k_1 and k_2 such that the roots of $D(s)$ all lie in the left half complex plane. Here we leverage the results reported in [86]: the sufficient and necessary condition for the system to be stable is that k_2 fulfills the following inequality (after the value of k_2 is determined, k_1 is chosen accordingly; we will elaborate on how to choose k_1 below):

$$\begin{aligned} 0 &\leq k_2 \leq bp_0\sqrt{\theta^2 + \frac{\left(1+2bw_0\right)^2p_0^2}{4b^2}} \\ &= \frac{2bN^2}{b\tau^2C^2 + \tau CN - N^2}\sqrt{\theta^2 + \frac{\left(1+2bw_0\right)^2p_0^2}{4b^2}}, \end{aligned}$$

in which θ is a solution to $\tan(\theta) = -\frac{b\left(b\tau^2C^2+\tau CN-N^2\right)}{N^2\left(1+2b\frac{\tau C}{N}\right)}\theta$ in the interval of $\left(\frac{\pi}{2}, \pi\right)$.

In summary, let $\alpha = \frac{\tau C}{N}$. The sufficient and necessary condition for system stability is that $0 \leq k_2 \leq \frac{2b}{\alpha^2 b+\alpha-1}\sqrt{\theta^2 + \frac{1}{b^2}\left(\frac{2ab+1}{\alpha^2 b+\alpha-1}\right)^2} = K_{2\max}$ in which θ is a solution to $\tan(\theta) = -b\frac{b\alpha^2+\alpha-1}{2\alpha b+1}\theta$ in the interval of $\left(\frac{\pi}{2}, \pi\right)$.

The bound of k_2 as a function of $\frac{N}{\tau C}$ in the case of $b = 1$ and $b = 2$ is depicted in Fig. 6.2. From the figure, we can see that the larger the value of $\frac{N}{\tau C}$, the larger the bound of k_2. This implies when the number, N, of connections gets larger or the RTT, τ, gets smaller, we can choose a larger value of k_2. Alternatively, if we choose the value of k_2 for a minimal number of connections, say N_{min} and a maximal value of τ, say τ_{max}, then for all $N \geq N_{min}$ and $\tau \leq \tau_{max}$, the system is still stable, as $\frac{N}{\tau C} \geq \frac{N_{min}}{\tau_{max}C}$. Also, as shown in Fig. 6.2, when the delayed ACK option is used, i.e. $b = 2$, we should choose a smaller value of k_2.

Stable region of k_1: After the value of k_2 is determined, the region of k_1 to stabilize the system can be determined by finding the roots of the following equation [86]:

$$k_2 + \frac{(2\alpha b + 1)\cos(z)}{(b\alpha^2 + \alpha - 1)^2} - \frac{bz\sin(z)}{(b\alpha^2 + \alpha - 1)} = 0. \tag{6.17}$$

Specifically, let the non-negative roots of Eq. (6.17) be arranged in the increasing order of magnitude and denoted as $z_i, i = 0, 1, 2, \ldots$ (of which $z_0 = 0$). Next we compute $a_i = a(z_i), i = 0, 1, 2, \ldots$ using the following equation:

$$a(z) = \frac{2(2ab+1)}{N}\left(\frac{1}{ba^2+a-1}\right)^2 z\left[\sin(z) + \frac{b\left(ba^2+a-1\right)}{(1+2ab)}\cos(z)\right]. \tag{6.18}$$

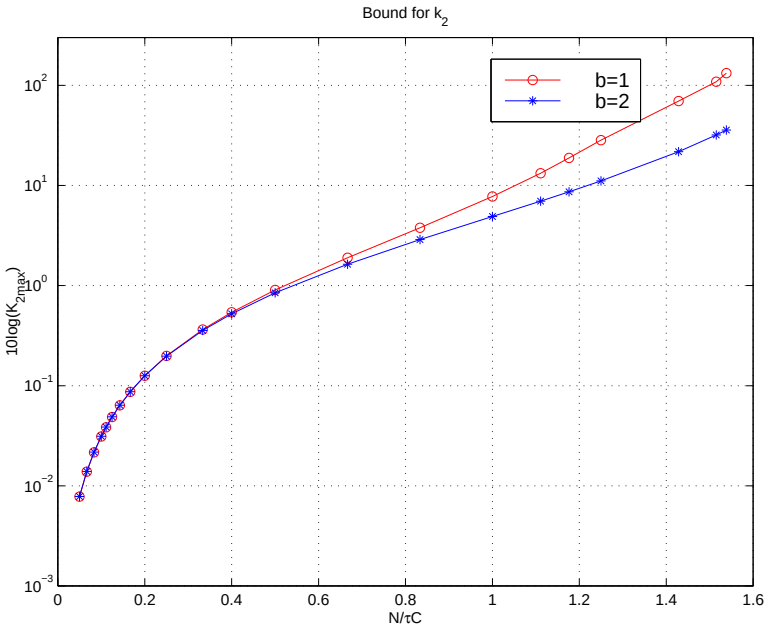


Figure 6.2: The bound of k_2.

The lower and upper bound of k_1 will be $0 < k_1 < \min_{i=1,3,5,\ldots} a_i$. In fact, we do not need to find all the roots of Eq. (6.17). After finding z_{2j+1} that makes $\cos(z_{2j+1}) > 0$, the algorithms can stop and proceeds to find the bound for k_1. In what follows, we give an example to illustrate how to choose these parameters.

Example 6.1 *Given the network parameters: C = 10Mbps = 1250 packets/second with the average packet size 1000 bytes,* $N_{min} = 300$, $\tau_{max} = 0.6sec$, *and* $b = 2$ *(i.e., the delayed* ACK *option is used), only when* $N < \frac{1+\sqrt{12b+1}}{6}\tau C = 750$, *the system equilibrium is meaningful. Also,* $\frac{N_{min}}{\tau_{max}C} = 0.4$. *As shown in Fig. 6.2, if we choose* $0 < k_2 < 0.4$, *the system will be stable for* $N \geq N_{min}$ *and* $\tau \geq \tau_{max}$. *The bound of* k_1 *as a function of* k_2 *in the cases*

of $b = 1$ and $b = 2$ is shown in Fig. 6.3. Also shown in Fig. 6.3 is that when the delayed ACK *option is used, i.e. $b = 2$, we should choose a smaller value of k_1. Given all the above criteria, we choose $k_2 = 0.2$ and $k_1 = 0.0005$.*

The stability of the system with the chosen parameters can be validated by drawing the open-loop Nyquist plot of the system. Specifically, the open-loop transfer function is $G(s) = \frac{1.167s+0.5833}{s(s+0.6548)}e^{-0.6s}$. The system is strictly proper and has a pole on the imaginary axis at $s = 0$, so Γ_s (a clockwise circle that covers the whole right-half complex plane including the imaginary axis) should exclude it. The Nyquist plot is obtained by drawing $G(s)$ for two segments, which are (1) $s = \epsilon e^{j\theta}$ with $\epsilon \to 0$ and θ varying from 0 to $\frac{\pi}{2}$, and (2) $s = j\omega$ with ω varying from ϵ to $+\infty$. For the first segment, by simple substitutions, we have

$$G(s) \approx \frac{0.891}{\epsilon}e^{-j\theta}e^{-0.6\epsilon e^{j\theta}} = \frac{0.891}{\epsilon e^{0.6\epsilon\cos\theta}}e^{-j(\theta+0.6\epsilon\sin\theta)} \tag{6.19}$$

and as θ varies from 0 to $\frac{\pi}{2}$ and $\epsilon \to 0$, this segment follows the quarter circle that starts at $\frac{0.891}{\epsilon}$ and ends at $\frac{0.891}{\epsilon}e^{-j(\frac{\pi}{2})^+}$. The second segment is shown in Fig. 6.4. As both segments of the Nyquist plot do not encircle point (-1, 0), the closed-loop system is stable. Also as shown in Fig. 6.5, the magnitude margin and the phase margin are $9dB$ and 60°, respectively.

6.5 Algorithm Implementation and Parameter Setting

The algorithm of the AQM controller is outlined in Fig. 6.6. Lines 1–4 determine if the queue length already exceeds the buffer limit. If so, the incoming packet is discarded. Lines 5–11 calculate the packet marking/dropping probability, p, according to the current

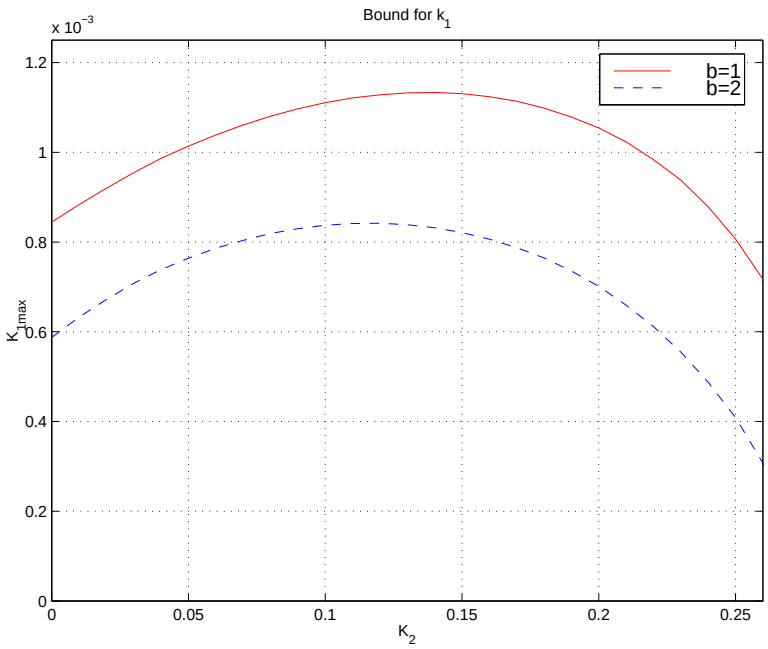


Figure 6.3: The bound of k_1.

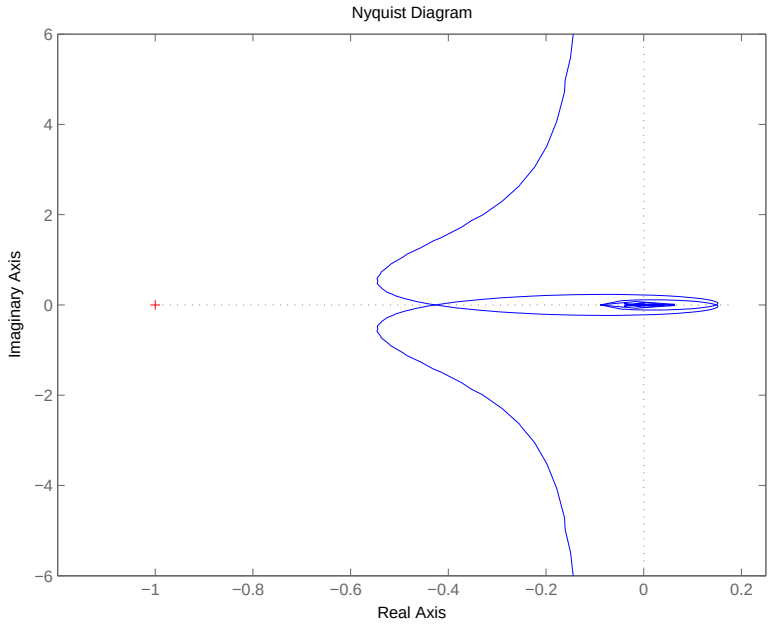


Figure 6.4: The Nyquist diagram of the system of interest.

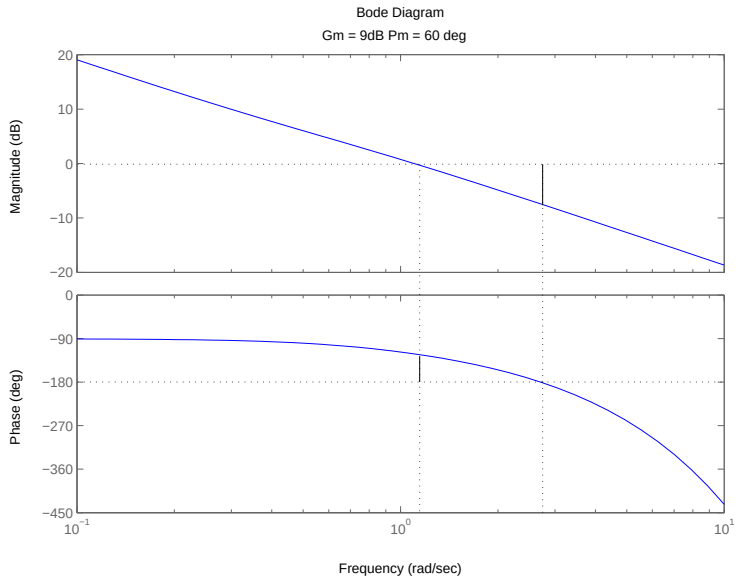


Figure 6.5: Bode plot of the system.

```
/* Called upon arrival of a new packet */
/* Q_lim is the buffer size at the router, C_0 is the capacity of the
* outgoing link, q_0 is the desired queue length, and k_1 and a = k_2/C
* are parameters chosen to stabilize the queue length */
1.  if (q ≥ Q_lim) {
2.      Drop the packet;
3.      Return;
4.  }
5.  R ← R_estimate();
6.  δq ← q − q_0;
7.  p ← k_1 · δq + a · (R − C_0);
8.  if (p < 0)
9.      p = 0;
10. else if (p > 1)
11.     p = 1;
12. drop ← Random_uniform(0,1);
13. if (drop > p) {
14.     Put the packet into the queue;
15. } else if (ECN is enabled) {
16.     Mark the ECN bit of the packet;
17.     Put the packet into the queue;
18. } else
19.     Drop the packet;
20. Return;
```

Figure 6.6: Enqueue procedure.

system states, and reset the probability to 0 or 1, if the calculated value exceeds region $[0, 1]$. In Lines 12–19, the packet is marked/dropped with probability p.

Line 7 in the algorithm is worthy of further discussion, as it is related to the issues of how to practically implement the AQM controller and how to measure/gather all the parameters such as k_1 and k_2. One key objective of choosing these parameters is that we should ensure that the controller is robust to a wide variety of network condition changes. The value of k_1 can be determined as described in Section 6.4, after the value of k_2 is determined. On the other hand, to practically implement the algorithm, we replace $k_2\delta w$ with $a(R-C_0)$. The value of δw cannot be directly obtained at the router, but can be estimated by $\frac{\tau}{N}\delta\dot{q}$, where $\delta\dot{q}$ is the difference between the incoming rate and the link capacity, i.e. $R-C_0$. The remaining problem is how to determine the values of τ and N. As mentioned in Section 6.4, the equilibrium state of the system is meaningful only when $N < \frac{1+\sqrt{12b+1}}{6}\tau C$. Hence, we can use $N_{max} = \frac{1+\sqrt{12b+1}}{6}\tau C$ (which is $0.77\tau C$ and τC, respectively, when b is 1 and 2.). Thus, in the case of $b = 2$, we have $|\frac{\tau}{N_{max}}\delta\dot{q}| = |\frac{1}{C}\delta\dot{q}| \leq |\frac{\tau}{N}\delta\dot{q}| = |\delta w|$. Because $|\frac{k_2}{C}\delta\dot{q}| \leq |k_2\delta w|$, the net effect of setting $a = \frac{k_2}{C}$ is that we choose a smaller value of k_2 than that determined according to the method described above, and hence the system is still stable.

$p = k_1 \cdot \delta q + a \cdot (R - C_0)$ implies that the packet dropping/marking probability in SFC consists of two parts: the first part is proportional to the difference between the current queue length and the target queue length; and the second part is proportional to the ratio of the difference between the incoming rate and the link capacity to the link capacity.

Continuing Example 6.1, we discuss in the following example how the parameters are practically chosen:

Example 6.2 *As shown in Fig. 6.3, for the nominal value of $k_2 = 0.2$ obtained in Example. 6.1, the appropriate value of k_1 ranges from 0 to 0.0007. As described above, we estimate the number, N, of connections to be N_{max}, the net effect of which is that we actually use a smaller value of k_2 than the chosen value of k_2. Consequently, we should choose the value of k_1 so that the system remains stable given a smaller value of k_2. From Fig. 6.3, we can see that $k_1 = 0.0005$ is in the stable region for all values of $k_2 \in [0, 0.2]$ and hence is a safe choice.*

6.6 Simulation Results

We have implemented our scheme along with RED [36], SRED [70], PI [41] and AVQ [49] in *ns-2* [50], and conducted a simulation study to validate the performance of proposed design and compare its performance against other schemes. The performance study was conducted with respect to queue stability, packet loss rate, link utilization, and parameter robustness. The work that comes closest to ours is SRED and PI controller, as both of them share the same objective of stabilizing the instantaneous queue. Hence we will conduct a comprehensive performance study between SRED, PI and SFC. On the other hand, as a side effect of using both the deviations of the queue length and the incoming rate from the operating point to determine the packet dropping/marking probability, SFC also decouples

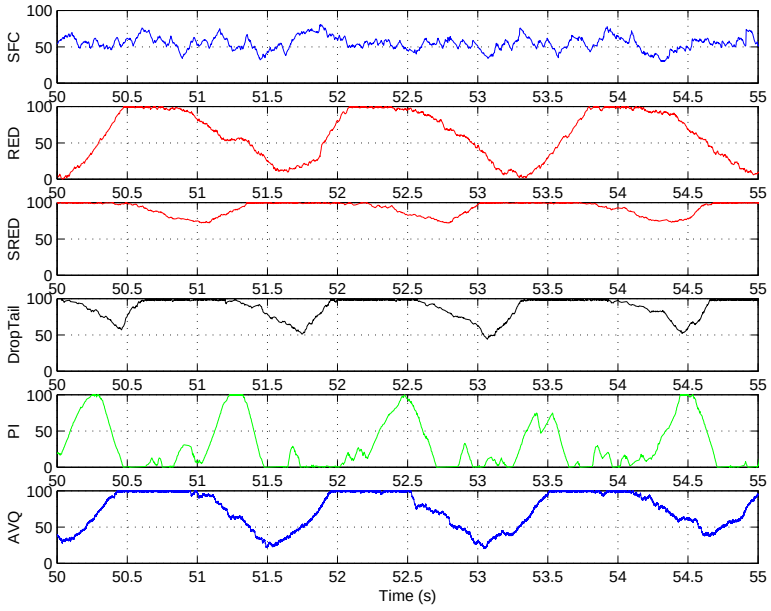


Figure 6.7: Performance comparison with respect to instantaneous queue length among different schemes.

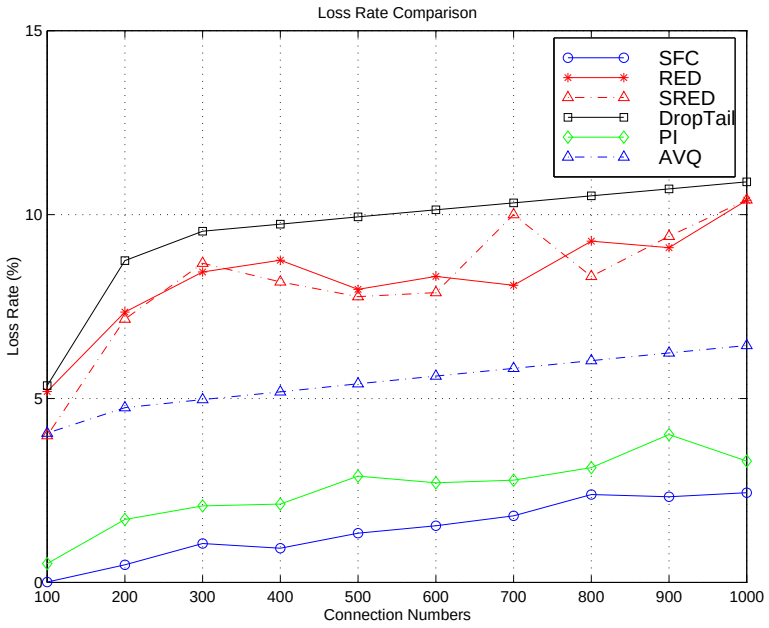


Figure 6.8: Performance comparison with respect to packet loss rate among different schemes.

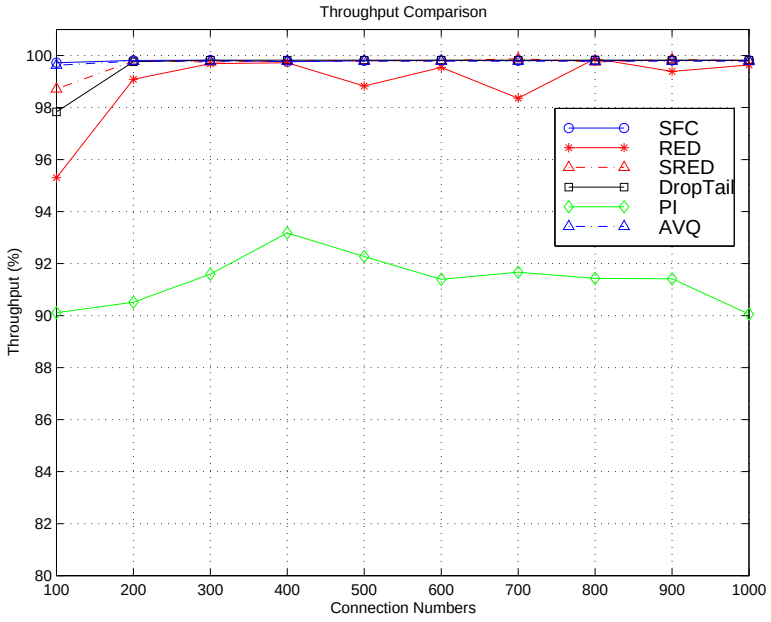


Figure 6.9: Performance comparison with respect to link utilization among different schemes.

the congestion measure and the performance measure. Hence, we will also compare SFC against AVQ (which is reported to give the best performance in the second category).

We examine the behavior of these schemes under a variety of network topologies and traffic sources. In particular, we have considered the network topologies with a single bottleneck link of various RTTs (Fig. 3.8) and network topology with multiple bottlenecks (Fig. 5.8). In the single bottleneck topology, the bottleneck link has a capacity of 10 Mbps, a delay of 20 ms. The links between a sender and the left router and between a receiver and the right router have a capacity of 10 Mbps and a delay of 40ms. The link bandwidth and propagation delay used in the multiple bottleneck topology are specified in Fig. 5.8.

The average packet size is 1000 byte and buffer size on each link is 100 packets. The traffic sources we use include long-termed TCP connections and short-termed TCP connections, both of which support ECN but do not enable the delayed ACK option. The number of connections varies from 100 to 1000. The target queue length is set to 50 packets.

The setting of parameters in the various AQM schemes is as follows. The parameters of RED are set as recommended in http://www.aciri.org/floyd/REDparameters.txt, and those of SRED are chosen as recommended in [70] (i.e., $M = 1000$, $\alpha = 1/M = 0.001$, and $p_{max} = 0.15$). The desirable utilization, γ, of AVQ is set to 0.98, and the damping factor, α, is determined in compliance with Theorem 1 in [49] to ensure system stability ($\alpha = 0.15$). The parameters of our scheme are $k_2 = 0.2$ and $k_1 = 0.0005$ as calculated in the previous example. Each data point is the result averaged over 20 simulation runs. In spite of numerous system parameters involved, the results are found to be quite robust in

the sense that the conclusion drawn from the performance curves for a representative set of parameter values (reported below) is valid over a wide range of parameter values.

6.6.1 Performance Comparison Under the Single Bottleneck Topology

In this set of experiments, we compare SFC against other schemes in the single bottleneck topology (Fig. 3.8). Totally k TCP connections are established over a single bottleneck link of capacity 10 Mbps, where k varies from 100 to 1000. Fig. 6.7 gives the instantaneous queue length in the cases that 200 TCP connections are established and continuously transmit packets. As shown in the figure, the instantaneous queue length under SFC fluctuates around the target level, while the queue lengths under other schemes are either always full or oscillates between empty and full.

Figs. 6.8–6.9 depict, respectively, the packet loss ratio and the goodput attained by all receivers. SFC outperforms than other schemes with respect to packet loss rate (by reducing as much as 50% packet losses), because it keeps the queue size at the desirable level. As a result, buffer overflow seldom occurs and fewer packets are dropped. As the PI controller also attempts to keep the queue length at a target level, it also achieves good performance. Although SRED shares the same objective of stabilizing the queue at a desirable level, it incurs much higher packet losses. This is attributed to the fact that SRED always attempts to keep the queue full (as shown in Fig. 6.7). On the other hand, as shown in Fig. 6.9, AVQ, SRED, and SFC (almost) fully utilize the bandwidth of the bottleneck link. This is because the queue under SFC and AVQ is seldom empty, while SRED always keeps the

queue full (which in turns leads to high packet losses). The PI controller, on the other hand, does not perform as well (10% less utilization), as the queue under the PI controller cannot control the queue length constantly at the desirable level and the queue is sometimes empty. Overall, SFC strikes a balance between reducing packet losses and queuing delay, and utilizing link bandwidth.

6.6.2 System Response

In this set of experiments, we set the number of connections to 100, the target queue length to 100, the buffer size to 300 packets, and compare the time it takes for the queue to stabilize at the desirable level under the PI controller and SFC. As shown in Fig. 6.10, SFC stabilizes the queue much faster than the PI controller. Furthermore, the former incurs very low overshoot. The fact that the PI controller incurs slower response and larger overshoot is attributed to its integral part in the controller.

6.6.3 Performance Comparison Under Dynamic Traffic Changes

In this set of experiments, we compare RED, PI, and SFC in terms of their responses to dynamic traffic changes in the single bottleneck topology (Fig. 3.8). 200 TCP connections (with bulk data transfer) are established over a single bottleneck link of capacity 10 Mbps. 50 TCP connections stop their transmitting at the 60th second, and resume at the 70th second again. Fig. 6.11 depicts the instantaneous queue length under PI controller, RED and SFC. The instantaneous queue length under PI and RED fluctuates significantly

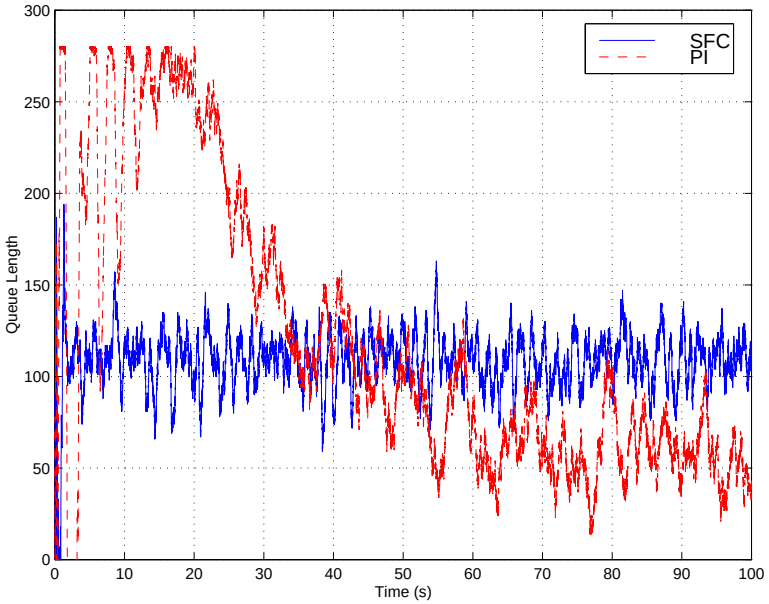


Figure 6.10: Performance comparison (in terms of the time taken for the queue to stabilize) between PI and SFC.

between the 60th and 70th seconds, while that under SFC is less susceptible to dynamic traffic changes. The reason why RED does not respond well to dynamic traffic changes is because the packet marking/dropping probability of RED is a linear function of the average queue length, and averaged queue length responds slowly to instantaneous queue length changes and even more slowly to the incoming rate change. The integral part of the PI controller makes it response slowly to dramatic traffic changes. In contrast, the marking/dropping probability in SFC is a linear combination of the instantaneous queue length and the incoming rate, and hence can adapt to traffic changes.

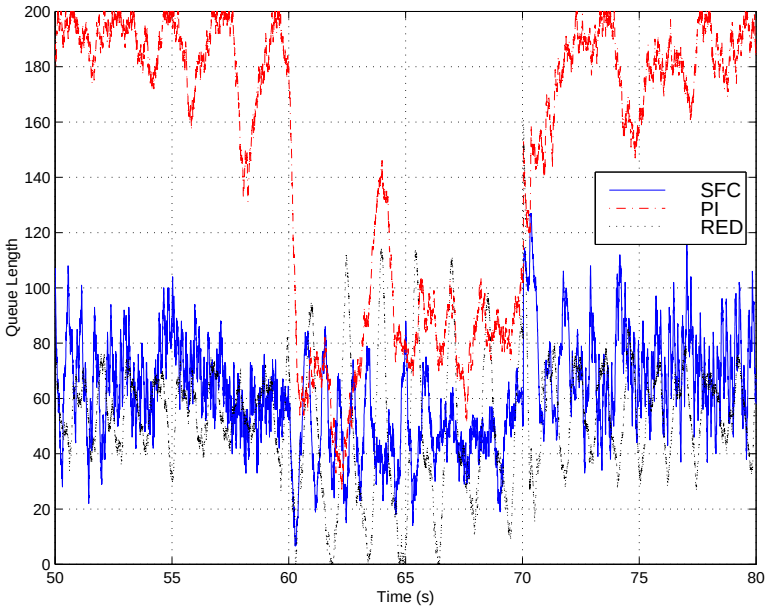


Figure 6.11: Performance comparison (in terms of instantaneous queue length) under dynamic traffic changes.

6.6.4 Robustness w.r.t. RTT and # Connection Changes

In this set of experiments, we test the robustness of the system parameters chosen in SFC with respect to different values of RTT and different # of TCP connections. The simulation setup is the same as in the first experiment, except that the number of connections varies from 200 to 600 and the RTT value varies from 200ms to 800ms. Figs. 6.12 and 6.13 depict, respectively, the link utilization (i.e., the goodput attained by all receivers) and the packet loss ratio for different values of RTTs and different numbers of TCP connections. It is clear that although the system parameters chosen in SFC are for the case of $\tau_{max} = 600ms$ and $N_{min} = 300$, the controller still achieves high link utilization and low packet loss ratio, regardless of the RTT and connection number changes.

6.6.5 Performance Comparison under the Multiple Bottleneck Topology

Although SFC is designed for the case of homogeneous TCP connections sharing a single bottleneck, we have conducted simulation to evaluate its multiple in the bottleneck topology (Fig. 5.8). As shown in Fig. 5.8, there are 5 queues among which queue 2 and queue 4 are shared by cross traffic of some other TCP connections. Again we establish k TCP connections with senders at the left hand side and receivers at the right hand side, where k varies from 100 to 1000. The cross traffic is composed of TCP connections as well, and the number of TCP connections in each cross traffic bundle is set to $0.2k$.

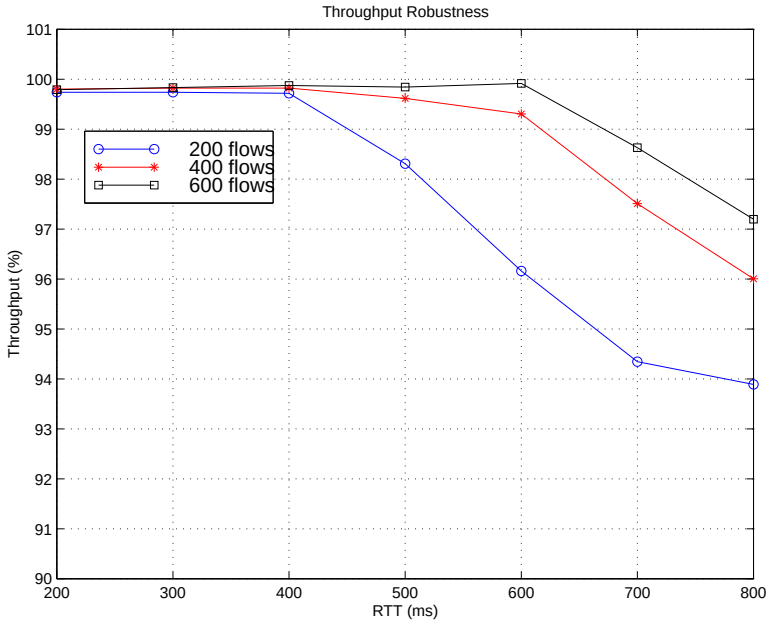


Figure 6.12: Robustness of system parameters chosen in SFC (link utilization with respect to different values of RTTs and # of connections).

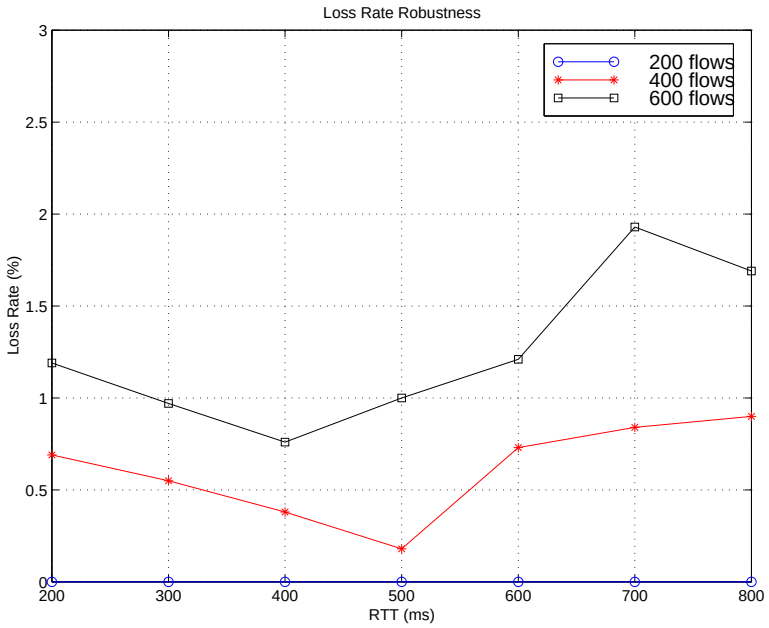


Figure 6.13: Robustness of system parameters chosen in SFC (packet loss ratio with respect to different values of RTTs and # of connections).

The simulation results show that the queue length at queue 5 is always 0 or 1, suggesting that the link is not a bottleneck link. The other four queues exhibit similar trends as far as the performance comparison is concerned. Hence, we arbitrarily choose to depict the instantaneous queue lengths of queue 2 in Fig. 6.14 and the packet loss ratio and the link utilization of queue 4 in Figs. 6.15–6.16, respectively. As shown in Fig. 6.14, although the capability of SFC to stabilize the instantaneous queue length degrades in the multiple bottleneck topology, its queue length still oscillates around the desirable level (except that the level of oscillation is larger than that in the single bottleneck topology). This is perhaps due to the fact that every router attempts to control its queue level locally, so that the interaction among them becomes complicated.

The link utilization achieved under SFC is the highest, as the controller attempts to keep the queue length stabilized and responds quickly to the queue length and incoming rate changes. The packet loss ratio under SFC is the second smallest and that under PI is the smallest. However, PI suffers from low link utilization. The reason why SRED does not perform as well in terms of packet loss ratio is because SRED has the tendency to keep the queue (close to) full, and hence packet losses occur as a result of buffer overflow.

6.7 Conclusion

In this chapter, we introduce an analytical TCP model that takes into account of several issues that were ignored in other existing models (such as those in [47, 64]), i.e., (i) the congestion window is not *gradually* decreased at the rate of $\frac{w^2 p}{2}$, but *suddenly* halved upon receipt of congestion indication and (ii) the congestion window is halved at most once

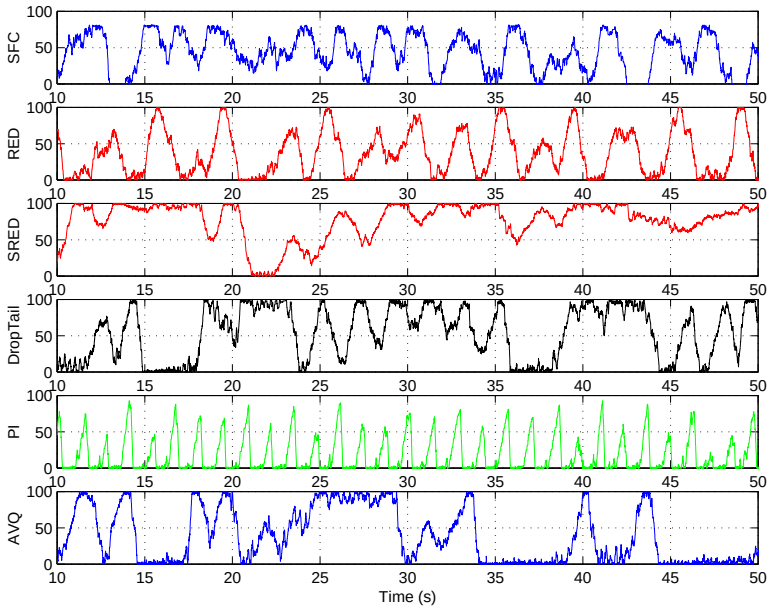


Figure 6.14: Instantaneous queue length at queue 2 under different schemes in the multiple bottleneck topology.

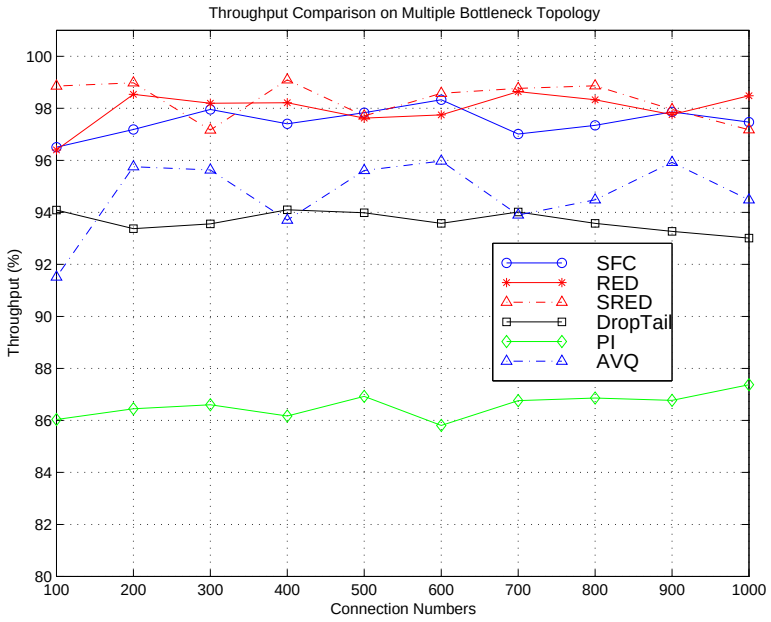


Figure 6.15: Link utilization at queue 4 under different schemes in the multiple bottleneck topology.

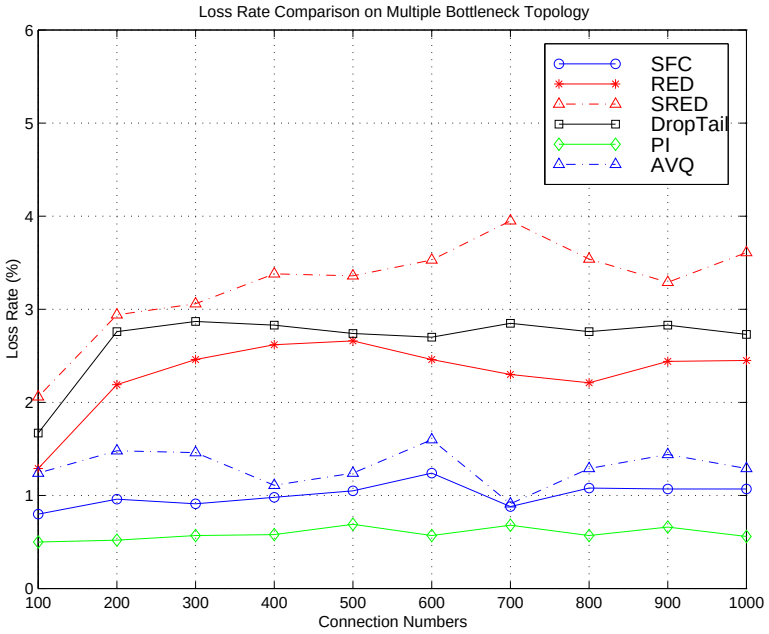


Figure 6.16: Packet loss ratio at queue 4 under different schemes in the multiple bottleneck topology.

during one RTT. We also include the delayed ACK option in the model. We show that this enhanced model more realistically characterizes the TCP dynamics and that under this model the change in the congestion window size is more significant in response to packet losses. The latter cautions us for designing an appropriate AQM controller. Based on this model, we then design, with the use of state feedback control theory, an AQM controller, called *SFC*, to stabilize the queue at a router. The performance of the new controller is shown via *ns* simulation to outperform other schemes in terms of fluctuation in the queue length, link utilization, and packet loss ratio. In particular, SFC can reduce packet loss by more than 50% as compared to other schemes and yet achieve full link utilization. As compared to PI controller proposed in [41], SFC achieves 10% more link utilization.

CHAPTER 7

Conclusions and Future Work

In this chapter, we summarize our research accomplishments and suggest several research avenues for future work.

7.1 Summary of Research Work

The major contributions of this thesis are:

(I) We present a rate-based multicast congestion control scheme, *RACCOOM* for continuous multimedia applications. In the absence of packet loss, a *RACCOOM* sender adjusts its sending rate in a TCP-Vegas fashion, based on the congestion status of the on-tree path with the largest RTT (called the *target* path). In case of packet loss, a *RACCOOM* sender then responds by reducing its sending rate by half. An ACK aggregation method is judiciously devised to prevent ACK implosion and yet to provide the sender with a simple but comprehensive view of congestion conditions in the multicast tree. *RACCOOM* also achieves (weighted) fairness among competing connections by exploiting feedback control theory and appropriately selecting

the parameters used in the rate adjustment mechanism. On the other hand, if TCP friendliness is the performance criterion, then a simple iterative approach can be used to on-line adjust the parameters α and β so as for a *RACCOOM* session to exhibit TCP-friendliness. Simulation experiments indicate that *RACCOOM* connections can achieve, irrespectively of the RTTs of individual connections, TCP-friendliness, can handle membership/network traffic changes, can deal with persistent congestion, and can achieve (weighted) fairness among competing connections with different RTTs. In terms of the level of TCP friendliness, *RACCOOM* also outperforms RAP and other formula-based approaches (e.g., CMTCP).

(II) We present an alternate endpoint congestion management scheme, called *COCOON*. *COCOON* groups concurrent TCP and UDP connections destined for the same destination host or subnet, enables them to share congestion information (but not the congestion window), and coordinates congestion control activities among them. The size of a *COCOON* group can be dynamically adjusted, and the overhead of group management has been analytically shown to be reasonably small (no more than 2.5%) under an extreme wide spectrum of subnet distribution and network loads. *COCOON* also expedites the start up of a new connection by allowing it to commerce with a congestion window that is large enough to catch up with other connections but not to induce congestion. Finally, *COCOON* takes into account non-responsive UDP connections in a group and "bundles" them into a virtual connection, which is subject to TCP-like congestion control. *COCOON* requires modification only at the server hosts and can be readily deployed over the Internet. We implement *COCOON* in

ns-2 and in *FreeBSD* and perform simulation/empirical studies with respect to various network topologies and traffic loads. We observe that as compared to *TCP-Reno, TCP-Int, CM*, HTTP 1.0, and HTTP 1.1, *COCOON* indeed reduces packet loss rate of concurrent connections, while sustaining throughput comparable to the best scheme. The encouraging results show that a good trade-off between maximizing the attainable bandwidth and minimizing the packet loss is to treat connections as individual flows but coordinate congestion management activities among them.

(II) We explore the issue of exploiting traffic predictability to enhance the performance of AQM. We show that the correlation structure present in long-range dependent traffic can be detected on-line and used to accurately predict the future traffic. We then figure in in the calculation of packet dropping probability the prediction results as a new dimension of congestion index. By stabilizing the instantaneous queue length at a desirable level (in anticipation of future traffic), the new AQM scheme, called *PAQM*, enables the link capacity to be fully utilized, while not incurring excessive packet loss. Through *ns-2* simulation, we show that under most cases *PAQM* outperforms SRED in stabilizing the instantaneous queue length, and AVQ in reducing packet loss ratio and utilizing the link capacity.

(IV) We develop an analytical TCP model that takes into account of several issues that were ignored in other existing models (such as those in [47, 64]), i.e., (i) the congestion window is not *gradually* decreased at the rate of $\frac{w^2 p}{2}$, but *suddenly* halved upon receipt of congestion indication and (ii) the congestion window is halved at most once during one RTT. We also include the delayed ACK option in the model. We

show that this enhanced model more realistically characterizes the TCP dynamics and that under this model the change in the congestion window size is more significant in response to packet loss. The latter cautions us for designing an appropriate AQM controller. Based on this model, we then design, with the use of state feedback control theory, an AQM controller, called *SFC*, to stabilize the queue at a router. The performance of the new controller is shown via *ns-2* simulation to outperform other schemes in terms of fluctuation in the queue length, link utilization, and packet loss ratio. In particular, *SFC* can reduce packet loss by more than 50% as compared to the other schemes and yet achieve full link utilization. As compared to PI controller proposed in [41], SFC achieves 10% more link utilization.

7.2 Future Work

Based on our previous research work and experiences, we have identified several avenues for conducting congestion control issues for next-generation Internets.

Prototype and evaluate RACCOOM over Internet 2: Since all the *RACCOOM* operations can be implemented at end hosts, we plan to prototype *RACCOOM* on *FreeBSD* and conduct experiments over Internet 2. (Currently NCSA [67] provides Illinois academic institutions the opportunity to do research on Internet 2 through connectivity to Abilene.) This is made possible by the fact that Internet2 has experienced a reasonable amount of success in deploying multicast. Both vBNS [45] and

Abilene current run PIM-SM [23]/MBGP [11]/MSDP [31] as the inter-domain multicast routing protocol and source discovery protocol. We will study how to implement *RACCOOM* on top of these protocols. The implementation will be DR-based and require minimum router support. We also plan to investigate how *RACCOOM* sessions interact with TCP connections when a different queue management mechanism, such as RED, is used.

Combine traffic prediction and TCP/AQM model in AQM design: We have shown that with traffic prediction and state feedback control, AQM schemes can be better designed and give better performance. A natural extension is then to exercise predictive control by combining traffic prediction and TCP/AQM models to stabilize the queue.

Use non-linear control in AQM design: The current research trend TCP-model based AQM design is to linearize the system model at the equilibrium point. The limitation of linearization is that the resulting linear system model is only accurate around the operating point. To remedy this problem, we will exploit non-linear, adaptive control on the original non-linear system model and enable the controller to be on-line adaptive to system parameter changes.

BIBLIOGRAPHY

[1] A. M. Adas. Using Adaptive Linear Prediction to Support Real-Time VBR Video Under RCBR Network Service Model. *IEEE/ACM Transactions on Networking*, 6(5), October 1988.

[2] J.S. Ahn, P.B. Danzig, Z. Liu, and Y. Yan. Experience with TCP Vegas: Emulation and Experiment. *Proceedings of ACM SIGCOMM'95*, August 1995.

[3] M. Allman. A Web Server's View of the Transport Layer. *ACM Computer Communication Review.*, October 2000.

[4] F.M. Anjum and L. Tassiulas. Fair Bandwidth Sharing Among Adaptive and Non-Adaptive Flows in the Internet. *Proceedings of IEEE INFOCOM'99*, March 1999.

[5] J. Apisdorf, K. Claffy, K. Thompson, and R. Wilder. OC3MON: Flexible, Affordable, High-Performance Statistics Collection. *http://www.nlanr.net/NA/Oc3mon/*, August 1996.

[6] M. Arlitt and T. Jin. Workload Characterization of the 1998 World Cup Web Site. *HP Technical Report http://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.html*, September 1999.

[7] S. Athuraliya, S. H. Low, V. H. Li, and Q. Yin. REM: Active Queue Management. *IEEE Network Magazine*, 15(3), May/June 2001.

[8] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. *Proceedings of IEEE INFOCOM'98*, March 1998.

[9] H. Balakrishnan, H. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. *Proceedings of ACM SIGCOMM'99*, August 1999.

[10] A. Ballardie. Core Based Trees (CBT version 3) Multicast Routing: Protocol Specification. *Internet draft*, August 1998.

[11] T. Betes, R. Chandra, D. Katz, and Y. Rekhter. Multiprotocol Extensions for BGP-4. *RFC-2283, IETF*, February 1998.

[12] S. Bhattacharyya, D. Towsley, and J. Kurose. The Loss Path Multiplicity Problem in Multicast Congestion Control. *Proceedings of IEEE INFOCOM'99*, March 1999.

[13] J. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-based Error Control for Interactive Audio in the Internet. *Proceedings of IEEE INFOCOM'99*, March 1999.

[14] R. Braden. Extending TCP for Transaction - Concepts. *RFC-1379, IETF*, November 1992.

[15] R. Braden. T/TCP - TCP Extensions for Transactions. Functional Specification. *RFC-1644, IETF*, July 1994.

[16] L. S. Brakmo and L. L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, October 1995.

[17] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, June 1989.

[18] K. Claffy, G. Miller, and K. Thompson. The Nature of the Beast: Recent Traffic Measurements from An Internet Backbone. *Proceedings of INET'98*, 3(3), September 1998.

[19] A. Costello. Search Party: An Approach to Reliable Multicast With Local Recovery. *Proceedings of IEEE INFOCOM'99*, March 1999.

[20] C. R. Cunha, A. Bestavros, and M. E. Crovella. Characteristics of WWW Client-based Traces. *Technical Report Boston University CS Department*, April 1995.

[21] L. Eggert, J. Heidemann, and J. Touch. Effects of Ensemble-TCP. *ACM Computer Communication Review*, January 2000.

[22] A. Erramilli, O. Narayan, and W. Willinger. Experimental Queuing Analysis with Long-Range Dependent Traffic. *IEEE/ACM Transactions on Networking*, April 1996.

[23] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast Sparse-mode (PIM-SM): Protocol Specification. *RFC-2362, IETF*, June 1998.

[24] B. Braden et al. Management and Congestion Avoidance in the Internet. *RFC-2309, IETF*, April 1998.

[25] D. Estrin et al. Protocol Independent Multicast (PIM) Sparse Mode/Dense Mode. *ftp://netweb.usc.edu/pim. Working drafts*, 1996.

[26] M. Allman et al. TCP Congestion Control. *RFC-2581 IETF*, April 1999.

[27] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. *RFC-2616, IETF*, June 1999.

[28] T. Berners-Lee et al. The World Wide Web. *Communications of the ACM*, August 1994.

[29] T. Berners-Lee et al. Hypertext Transfer Protocol – HTTP/1.0. *RFC-1945, IETF*, May 1996.

[30] T. Speakman et al. PGM Reliable Transport Protocol. *Internet draft*, April 2000.

[31] D. Farinacci, Y. Rekhter, P. Lothberg, H. Kilmer, and J. Hall. Multicast Source Discovery Protocol (MSDP). *Internet Draft*, June 1998.

[32] W. Feng, D. Kandlur, and K. Shin. Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness. *Proceedings of IEEE INFOCOM 2001*, April 2001.

[33] W. Feng, K. Shin, D. Kandlur, and D. Saha. A Self-Configuring RED Gateway. *Proceedings of IEEE INFOCOM'99*, March 1999.

[34] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications. *Proceedings of ACM SIGCOMM 2000*, September 2000.

[35] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[36] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1993.

[37] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *Proceedings of ACM SIGCOMM'95*, October 1995.

[38] Y. Gao, G. He, and J. Hou. On Leveraging Traffic Predictability in Active Queue Management (Extended version). *Available at http://eewww.eng.ohio-state.edu/∼heg.*, August 2001.

[39] H. W. Holbrook and D. R. Cheriton. IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications. *Proceedings of ACM SIGCOMM'99*, September 1999.

[40] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton. Log-based Receiver-reliable Multicast for Distributed Interactive Simulation. *Proceedings of ACM SIGCOMM'95*, October 1995.

[41] C. Hollot, V. Misra, D. Towsley, and W. Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. *Proceedings of IEEE INFOCOM 2001*, April 2001.

[42] S. Jacobs and A. Eleftheriadis. Providing Video Services over Networks without Quality of Service Guarantees. *World Wide Web Consortium Workshop on Real-Time Multimedia and the Web*, October 1996.

[43] V. Jacobson. Congestion Avoidance and Control. *Proceedings of ACM SIGCOMM'88*, 1988.

[44] S. Jagannathan, K. Almeroth, and A. Acharya. Topology Sensitive Congestion Control for Real-Time Multicast. *Proceedings of NOSSDAV 2000*, June 2000.

[45] J. Jamison, R. Nicklas, G. Miller, K. Thompson, R. Wilder, L. Cunningham, and C. Song. vBNS: Not Your Father's Internet. *IEEE Spectrum*, July 1999.

[46] M. Kadansky. Reliable Multicast Tree-building Techniques. *RMRG Meeting*, July 1998.

[47] F. Kelly. Mathematical Modeling of the Internet. *Mathematics Unlimited - 2001 and Beyond (Editors B. Engquist and W. Schmid)*, Springer-Verlag 2001.

[48] H.T. Kung and S.Y. Wang. TCP Trunking: Design, Implementation, and Performance. *Proceedings of IEEE ICNP'99*, October 1999.

[49] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. *Proceedings of ACM SIGCOMM 2001*, August 2001.

[50] LBNL. UCB/LBNL/VINT Network Simulator - *ns-2*. *http://www-mash.cs.berkeley.edu/ns/*.

[51] K. W. Lee, S. Ha, and V. Bharghavan. IRMA: A Reliable Multicast Architecture for the Internet. *Proceedings of IEEE INFOCOM'99*, March 1999.

[52] A. Legout and E. W. Biersack. Pathological Behaviors for RLM and RLC. *Proceedings of NOSSDAV 2000*, June 2000.

[53] L. W. Lehman, S. J. Garland, and D. L. Tennenhouse. Active Reliable Multicast. *Proceedings of IEEE INFOCOM'98*, March 1998.

[54] B. N. Levine and J. J. Garcia-Luna-Aceves. Improving Internet Multicast with Routing Labels. *Proceedings of IEEE ICNP'97*, October 1997.

[55] B.N. Levine, S. Paul, , and J.J. Garcia-Luna-Aceves. Organizing multicast receivers deterministically according to packet-loss correlation. *Proceedings of ACM Multimedia 98*, September 1998.

[56] D. Li and D. R. Cheriton. OTERS (On-tree Efficient Recovery using Subcasting): a Reliable Multicast Protocol. *Proceedings of IEEE ICNP'98*, October 1998.

[57] J. Lin, S. Paul, K. Sabnani, and S. Bhattacharyya. A Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, April 1997.

[58] W. Lin and R. Morris. Dynamics of Random Early Detection. *Proceedings of ACM SIGCOMM'97*, September 1997.

[59] B.A. Mah. An Empirical Model of HTTP Network Traffic. *Proceedings of IEEE INFOCOM'97*, April 1997.

[60] J. Mahdavi and S. Floyd. TCP-Friendly Unicast Rate-Based Flow Control. *Technical Note*, January 1997.

[61] G. Malkin. RIP version 2. *RFC-2453, IETF*, November 1998.

[62] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communication Review*, July 1997.

[63] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. *Proceedings of ACM SIGCOMM'96.*, 1996.

[64] V. Misra, W. Gong, and D. Towsley. A Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. *Proceedings of ACM SIGCOMM 2000*, September 2000.

[65] J. Mo, R. J. La, V. Anantharam, and J.Walrand. Analysis and Comparison of TCP Reno and Vegas. *Proceedings of IEEE INFOCOM'99*, March 1999.

[66] J. Moy. OSPF version 2. *RFC-2178, IETF*, July 1997.

[67] NCSA. NCSA Projects Database. *http://www.ncsa.uiuc.edu/TechFocus/Projects/.*, 1996.

[68] J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-based Loss Recovery for Reliable Multicast Transmission. *IEEE Transactions on Networking*, January 1998.

[69] T. J. Ott, J. H. B. Kemperman, and M. Mathis. The Stationary Behavior of Ideal TCP Congestion Avoidance. *Proceedings of IEEE INFOCOM'99*, March 1999.

[70] T.J. Ott, T.V. Lakshman, and L.H. Wong. SRED: Stabilized RED. *Proceedings of IEEE INFOCOM'99*, March 1999.

[71] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: a Simple Model and its Empirical Validation. *UMass-CMPSCI Technical Report TR 98-004*, Feburary 1998.

[72] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A TCP-friendly Rate Adjustment Protocol for Continuous Media Flows over Best Effort Networks. *UMass-CMPSCI Technical Report TR 98-004*, October 1998.

[73] V. N. Padmanabhan. Addressing the Challenges of Web Data Transport. *PhD Dissertation Univ. of California at Berkeley*, September 1998.

[74] V. N. Padmanabhan and R. H. Katz. TCP Fast Start: A Technique for Speeding Up Web Transfers. *Proceedings of IEEE GlOBECOM '98*, November 1998.

[75] F. Paganini, J. C. Doyle, and S. H. Low. Scalable Laws for Stable Network Congestion Control. *Proceedings of Conference on Decision and Control*, December 2001.

[76] C. Papadopoulos, G. Parulkar, and G. Varghese. An Error Control Scheme for Large-scale Multicast Applications. *Proceedings of IEEE INFOCOM'98*, March 1998.

[77] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach.* Kaufmann, Second edition, 1999.

[78] T. Pusateri. Distance Vector Multicast Routing Protocol. *Draft, IETF*, March 2000.

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

[80] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. *Proceedings of IEEE INFOCOM'99*, March 1999.

[81] I. Rhee, N. Balaguru, and G. N. Rouskas. MTCP: Scalable TCP-like Congestion Control for Reliable Multicast. *Proceedings of IEEE INFOCOM'99*, March 1999.

[82] L. Rizzo. Pgmcc:a TCP-friendly Single-rate Multicast Congestion Control Scheme. *Proceedings of ACM SIGCOMM 2000*, September 2000.

[83] D. Rubenstein, S. Kasera, D. Towsley, and J. Kurose. Improving Reliable Multicast using Active Parity Encoding Services (APES). *Proceedings of IEEE INFOCOM'99*, March 1999.

[84] A. Sang and S. q. Li. A Predictability Analysis of Network Traffic. *Proceedings of IEEE INFOCOM 2000*, March 2000.

[85] S. Shenker, L. Zhang, and D. Clark. Some Observations on the Dynamics of a Congestion Control Algorithm. *Proceedings of ACM SIGCOMM'90*, 1990.

[86] G. Silva, A. Datta, and S. P. Bhattacharyya. PI Stabilization of First-order Systems with Time Delay. *Automatica*, December 2001.

[87] D. Sisalem and H. Schulzrinne. The Loss-delay Adjustment Algorithm: a TCP-friendly Adaptation Scheme. *Proceedings of NOSSDAV'98*, July 1998.

[88] W. Stallings. *Data and Computer Communications.* Prentice-Hall, Sixth edition, 1996.

[89] W. Stevens. *TCP/IP Illustrated, Volume 3: The Protocols.* Addison-Wesley, 1998.

[90] W. Richard Stevens. *TCP/IP Illustrated, Volum 1: The Protocols.* Addison-Wesley, 1994.

[91] R. Talpade and M. H. Ammar. An Architecture for Providing a Reliable Multicast Transport Service. *Proceedings of ICDCS'95*, 1995.

[92] K. Thompson, G.J. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network Magazine*, November 1997.

[93] J. Touch. TCP Control Block Interdependence. *RFC-2140, IETF*, July 1994.

[94] T. Turletti, S. Parisis, and J. Bolot. Experiments with a Layered Transmission Scheme over the Internet. *Technical report RR-3296 INRIA*, 1996.

[95] L. Vicisano, L Rizzo, and J. Crowcroft. TCP-like Congestion Control for Layered Multicast Data Transfer. *Proceedings of IEEE INFOCOM'98*, March 1998.

[96] W. Willinger W. E. Leland, M. S. Taqqu and D. V. Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Transactions on Networking*, February 1994.

[97] L. Wei, D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S.Deering, M. Handley, V. Jacobson, C. Liu, and P. Sharma. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification. *Internet draft*, November 1999.

[98] B. Whetten and J. Conlan. A Rate Based Congestion Control Scheme for Reliable Multicast. *Technical White Paper, GlobalCast Communications*, October 1998.

[99] J. Widmer and M. Handley. Extending Equation-based Congestion Control to Multicast Applications. *Proceedings of ACM SIGCOMM 2001*, August 2001.

[100] W. Willinger, V. Paxson, and M. S. Taqqu. Self-Similarity and Heavy Tails: Structural Modeling of Network Traffic. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, 1998.

[101] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. *Proceedings of ACM SIGCOMM'91*, 1991.

[102] R. Yavatkar, J. Griffioen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. *ACM Multimedia*, 1995.