Congestion Control for Streaming Media

by

Jae Won Chung

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Computer Science

by

October 2005

APPROVED:

Professor Mark Claypool
Advisor

Professor Robert Kinicki
Committee Member

Professor Craig Wills
Committee Member

Professor Kevin Jeffay
External Committee Member
Computer Science, UNC-Chapel Hill

Professor Michael Gennert
Head of Department

Abstract

The Internet has assumed the role of the underlying communication network for applications such as file transfer, electronic mail, Web browsing and multimedia streaming. Multimedia streaming, in particular, is growing with the growth in power and connectivity of today's computers. These Internet applications have a variety of network service requirements and traffic characteristics, which presents new challenges to the single best-effort service of today's Internet. TCP, the de facto Internet transport protocol, has been successful in satisfying the needs of traditional Internet applications, but fails to satisfy the increasingly popular delay sensitive multimedia applications. Streaming applications often use UDP without a proper congestion avoidance mechanisms, threatening the well-being of the Internet.

This dissertation presents an IP router traffic management mechanism, referred to as Crimson, that can be seamlessly deployed in the current Internet to protect well-behaving traffic from misbehaving traffic and support Quality of Service (QoS) requirements of delay sensitive multimedia applications as well as traditional Internet applications. In addition, as a means to enhance Internet support for multimedia streaming, this dissertation report presents design and evaluation of a TCP-Friendly and streaming-friendly transport protocol called the Multimedia Transport Protocol (MTP). Through a simulation study this report shows the Crimson network efficiently handles network congestion and minimizes queuing delay while providing affordable fairness protection from misbehaving flows over a wide range of traffic conditions. In addition, our results show that MTP offers streaming performance comparable to that provided by UDP, while doing so under a TCP-Friendly rate.

Acknowledgments

First of all, thank God for guarding and guiding me all the times, and giving me power, strength and patience to finish this dissertation.

I would like to express my deepest gratitude to Professor Claypool for sharing his passion for research with me and being such a great advisor for both academic and personal matters. I would also like to express the same deepest gratitude to Professor Kinicki for guiding me throughout my undergraduate and graduate studies at WPI. Professor Claypool is like a big brother to me, and Professor Kinicki is like father to me. I would like to thank Professor Wills for leading me to the world of computer science. His CS2005 class in my early undergraduate years stimulated me to change my major from civil engineering to computer science. I still remember how fun was the CS2005 group project: airline reservation system. Many thanks to Professor Jeffay at UNC-Chapel Hill for giving a ready consent to join my Ph.D. committee and attending my dissertation defense in his own expense. In addition, I would like to thank him for his great research papers that inspired me with Master thesis and my other researches.

I would like to thank my parents for giving birth to, raising and educating me, and of course for everything. Also, I would like to express my gratitude to my parents in law for giving birth to, raising and educating my lovely wife Bo. Lastly, but not least, I would like to thank Bo for sharing all the joys and pains with me during all these years, and my son Donghyun for teaching me a new joy of life.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The Internet has become an essential part of public life while assuming the role of the underlying communication network for diverse applications such as file transfer, electronic mail, Web browsing, media streaming and interactive video games. These various Internet applications have a variety of network service requirements and traffic characteristics, which provides new challenges to Internet congestion control. This chapter motivates and introduces the research in this dissertation.

1.1 Motivation

The Internet is a collection of interconnected networks that offer a best-effort data transmission service to users using the Internet Protocol (IP). The current Internet relies on the end-host congestion avoidance mechanisms of modern TCP to resolve traffic congestion and prevent congestion collapse. Modern TCP traffic sources monitor their own transmission to detect network packet losses,[1] take them as implicit congestion signals from routers and reduce their transmission rate to avoid conges-

[1]Although not used in practice on the Internet today, TCP has an option to use the Explicit Congestion Notification (ECN) bit [109] set by ECN enabled routers to detect network congestion.

tion using bandwidth adaptation mechanisms. The congestion avoidance bandwidth adaptation behavior of TCP is mainly characterized by the Additive Increase Multiplicative Decrease (AIMD) algorithm [65].

In the network, IP routers use outbound queues to accommodate traffic bursts and achieve high link utilization. Most current routers use FIFO queues that passively drop incoming packets when queues are full. Unfortunately, when faced with persistent congestion, FIFO drop-tail queues, often over-provisioned with large buffers to yield maximum throughput, oscillate and fill up resulting in high transmission delays. Also, the simple drop-tail queue mechanism is open to the threat of congestion collapse due to a malicious or even unintentional misuse of the network [43]. Moreover, FIFO queues have difficulty in satisfying the diverse network service requirements of today's applications, especially the ever more popular delay sensitive applications.

The congestion control challenges the current Internet faces can be categorized into three sub-problems: first is to improve network support for an efficient congestion control feedback system; second is to protect the network from potentially misbehaving or unresponsive traffic; third is to concurrently support various Internet application domains with diverse Quality of Service (QoS) requirements.

The efforts to improve the congestion control feedback system can be divided into two approaches. The first approach is to completely redesign a congestion control feedback system by replacing both the TCP congestion avoidance mechanism and the router traffic management mechanisms [5, 69, 99]. This type of approach has the potential to achieve a highly efficient congestion control structure, but faces the possibly insurmountable problems of gradual deployment.

The second approach is to replace drop-tail queue management with Active Queue Management (AQM) [14] in order to enhance network support for end-to-end congestion control. AQM enabled routers include a congestion controller unit [35, 45, 47, 56, 58, 75] that detects and notifies end-systems of impending congestion, allowing responsive traffic sources to reduce transmission rates before the congested router queue overflows. When properly designed and configured, AQM can offer a low queuing delay while achieving high link utilization. Moreover, since AQM routers are able to predict impending congestion before buffer overflows they may explicitly signal end-systems of network congestion by marking the Explicit Congestion Notification (ECN) [41, 109] bit in the IP headers. ECN improves congestion notification efficiency and system goodput over implicit packet-drop congestion notification by avoiding network packet losses [25]. Yet another gain from the early congestion prediction is that routers have a choice in selecting to which end-hosts to signal congestion. By fairly notifying end-hosts of congestion proportion to their usage, an AQM router may improve bandwidth distribution fairness over drop-tail queue management [36, 45].

The efforts to protect the network from potentially misbehaving or unresponsive traffic can be made both at end-systems and routers. The Internet research community is trying to build TCP-Friendly[2] [43] transport protocols [44, 114, 113, 131] for applications that cannot benefit from using TCP but would rather use UDP. At the same time, drastic approaches to protect the network or to enforce bandwidth distribution fairness at routers have been proposed [17, 23, 33, 36, 46, 85, 81, 91, 103, 119]. The bandwidth controller units are often designed as a part of the AQM mechanisms.

The efforts to concurrently support various Internet application domains with diverse QoS requirements are made mainly at the network service architecture level. The Internet research community is considering restructuring the service architecture to support differentiated classes of service, known as the Differentiated Services

[2]TCP-Friendly refers to the congestion responsiveness of traffic sources that do not receive more throughput than conforming TCP flows would under the same network conditions.

(DiffServ) architecture [8]. The DiffServ architecture can offer a variety of network services that may be sufficient to support the diverse QoS requirements of different Internet applications. However, what services should be offered, how to configure network components to support specific services or how to price services are very complex challenges not yet resolved. Furthermore, gradual deployment of DiffServ is non-trivial since it requires a different network service interface from that of the current Internet.

Alternatively, a few approaches have been made to offer a limited support for QoS such as controlling delays and packet loss rates using a QoS packet scheduling instead of FIFO scheduling discipline [60, 106]. These mechanisms consider delay requirement hints in each IP header and tradeoffs the delay gain with the throughput loss and vice versa using a QoS scheduling queue control unit. The merit of the delay-throughput exchange approaches is that they require a minor modification to the Internet service interface and IP header, and provide a "better-than-best-effort" service. However, a concern is that the delay-throughput exchange mechanisms may not preserve fairness among flows, since routers may not be able to collect and use all the information required to ensure fairness due to the information acquiring complexity and overhead. Moreover, few were able to provide a suitable support for the diverse delay requirements.

The delay sensitivity of the QoS requirements makes the Internet support for QoS more difficult rather than the variety of the QoS requirements. For example, concurrently satisfying throughput sensitive FTP and Email applications is relatively easy compared to concurrently satisfying Web browsing and Internet videophone applications because of the stricter delay constraints imposed by the latter applications. From this simple observation, we can deduce that if the queuing delays at the congested routers can be minimized even at the cost of a small decrease in link utilization, a significant number of QoS issues will be reduced. In other words, if the current Internet cannot handle diverse delay QoS requirement, a possible solution other than a network service structure change, which requires time, vast investment and risk, is to significantly reduce network delays.

The Internet can offer "best-delay-effort" service without any change in the service architecture or interface using a delay optimized AQM at routers improving both the congestion control efficiency and the support for diverse QoS requirements. A delay optimized AQM minimizes queuing delay by making early and efficient imminent congestion prediction based on the incoming traffic rate rather than on the queue length. An analogy is a preemptive time-sharing machine that configures its time-slice length for the QoS needs of interactive applications can support both the interactive and computation-rich applications concurrently up to a certain workload. Likewise, a best-delay-effort Internet could concurrently serve various applications with different delay constraints.

In addition to improving efficiency and QoS, it is inevitable that the next generation Internet should offer the public network affordable protection from threats of potentially misbehaving or unresponsive traffic. The current Internet allows applications to use the network with arbitrary data rates and congestion response, potentially in a harmful way. Protection of the public network may not be a practically important problem when the majority of Internet applications uses TCP. However, it becomes serious with the growth of delay sensitive applications such as streaming media, which often prefer UDP over TCP as their transport protocol choice. Moreover, the growth of the end-user Internet connection capacity further increases the level of threats from misbehaving or unresponsive flows due to the increased traffic volume limits of each flow.

Streaming media is sensitive to delay and jitter, but can tolerate some data loss.

Thus, TCP with reliable transmission service at the cost of potentially large delay at congestion may not be an optimal choice for streaming applications. As discussed earlier, recent research has proposed rate-based TCP-Friendly protocols [44, 114, 113, 131] in the hope that streaming media applications will use them, but such protocols are not yet widely part of most operating system distributions. For these reasons, streaming media applications often use UDP as a transport protocol rather than TCP. Moreover, with the use of repair techniques [11, 82, 95, 102], UDP packet losses can be partially or fully concealed, reducing the impact of loss on the quality of the media by the user, and thus reducing the incentive for multimedia applications to lower their bitrate in the presence of packet loss during congestion. Moreover, as the end-user Internet connection capacity offered by Internet Service Providers (ISP) has significantly increased (up to 3 Mbps for typical cable modem services), even the highest quality media, about 2-4 Mbps for broadcast quality video, can be streamed without imposing congestion at the local Internet connection links. Thus, high-bandwidth Internet connections are pushing the streaming media performance bottleneck closer to the servers threatening the well-being of the public Internet [130].

In addition to the design and evaluation of a delay optimized AQM practical for best-delay-effort Internet service implementation, this dissertation research seeks to find a proper protection of the Internet from misbehaving traffic in two different angles: 1) As a pessimistic approach we develop a light-weight bandwidth control mechanism at routers that works both with AQM or drop-tail queue management. 2) As an optimistic approach we modify TCP and provide end-systems Multimedia Transport Protocol (MTP), an alternate to UDP for streaming and other delay sensitive Internet applications that favor prompt and timely datagram delivery service over the reliable transmission service of TCP. The new unreliable transport protocol has the exact congestion avoidance mechanism and hence proven stability of TCP.

Also, MTP can be easily made available for all operating system distributions, since an MTP implementation can reuse most of a TCP implementation.

Thus, this dissertation presents the design and evaluation of two IP router traffic management mechanisms and an end-system transport protocol that can be deployed individually or together to improve streaming media performance on the Internet: 1) MTP provides TCP-Friendly and streaming-friendly transport service to streaming media applications. 2) The bandwidth controller at IP routers punishes misbehaving high-bandwidth UDP media streams encouraging them to use MTP. 3) The delay optimized AQM congestion controller reduces IP router queuing delays to meet the delay requirements of interactive streaming applications.

1.2 Approach

We build and evaluate a fairness protection and delay optimized AQM mechanism as a part of the *Crimson*[3] architecture as shown in Figure 1.1. Without a modification to the service architecture or interface of the current Internet, the Crimson network protects well-behaving congestion responsive flows from misbehaving traffic and efficiently minimizes network queuing delays to help delay sensitive applications without compromising performance of throughput sensitive applications.

The Crimson router AQM mechanism is composes of two parts. First is a Bandwidth Controller, referred to as the Stochastic Fairness Guardian (SFG), that protects bandwidth utilization fairness. The objective of SFG is not to achieve per-flow fairness but to protect the network from potentially harmful misbehaving flows. To make the Crimson router as lightweight and independent of other routers as possible, we develop a lightweight pseudo per-flow protection mechanism that uses a statisti-

[3]The name *Crimson* comes from a word play on the acronym RED for Random Early Detection [45]. Considering RED as a color, then crimson is one of the school colors of WPI.

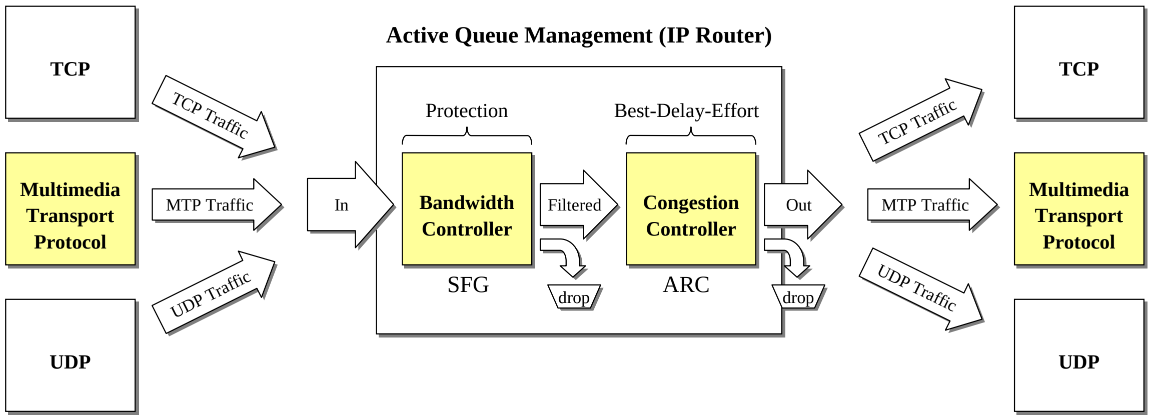


Figure 1.1: Proposed Mechanisms: Crimson Architecture

cal traffic filtering technique to effectively regulate misbehaving flows with minimal traffic state information. SFG uses a multi-level hash scheme that places incoming flows into different flow groups at each level and approximates a proper packet drop rate for each flow by monitoring the incoming traffic rates for the groups to which the flow belongs. SFG can be used in conjunction with an AQM or with drop-tail queue management.

The incoming traffic that passes the SFG meets Crimson's rate-based Congestion Controller targeted to minimize the queuing delay and support a wide range of traffic load and conditions. Recently, the most promising approaches model AQM as a feedback controller on a time-delayed response system, and develop an efficient proportional integral (PI) controller, a dominating feedback controller design in modern control systems due to their simplicity and effectiveness, for TCP traffic [47, 58, 75]. These approaches demonstrate that a PI controller can be an effective AQM mechanism for a range of traffic conditions. Yet, the PI control approaches omit to provide complete parameter configuration guidelines to support a practical range of traffic conditions, since the configuration of PI control parameters in a time-delayed feedback system (i.e., the Internet) is challenging [118]. We address the practical configuration issue by carefully reducing the PI parameters and find a complete configuration guideline for the parameter-reduced PI controller, called the Aggregate Rate Controller (ARC). For early and accurate congestion estimation, which is essential for a delay optimized AQM to achieve a high link utilization, ARC takes a rate-based control information acquisition approach. For a small amount of data collection overhead, the direct incoming traffic rate measurement approach can significantly reduce control noise from the estimate of incoming rate based on queue samples. Lastly, ARC concurrently supports ECN as well as implicit packet dropping congestion notification.

At the end-systems, Crimson supports an additional unreliable transport protocol beside TCP and UDP to improve Internet support for media streaming and other delay sensitive applications. There has been several TCP-Friendly transport protocols proposed for non-TCP applications, or more specifically for streaming media applications. These protocols are mostly rate-based congestion adaptation approaches that may contribute to the instability for systems with large end-to-end delays, where network system instability is characterized by large queue and throughput oscillations and bursty packet drops at network routers. Furthermore, it is not clear the proposed protocols are indeed practical for streaming applications. Few studies have been made to understand the streaming end-system protocol requirements and verify the suitability of the TCP-Friendly protocols.

As a part of this dissertation, we conducted a video streaming Internet measurement study using Real Networks[4] video streams to understand the requirements and characteristics of streaming Internet applications. Throughout the study, we were able to identify issues that TCP-friendly transport protocols should address to support streaming applications, and found several key incentives for video streams

[4]http://www.real.com

to use UDP rather than TCP. It is difficult for streaming applications to efficiently estimate available bandwidth upon TCP, since the TCP API hides network information such as packet loss rate and RTT. Moreover, the huge penalty of overestimating available bandwidth upon TCP discourages streaming applications from using TCP. However, with the window-based mechanism of TCP, we did not find transmission timing issues that may discourage streaming applications from using TCP.

Thus, we modify TCP by removing the retransmission feature that is unnecessary or even harmful for streaming purposes, and evaluate the modified TCP as a streaming transport protocol. We call the modified TCP for streaming the Multimedia Transport Protocol (MTP) that forms another part of the *Crimson* architecture as shown in Figure 1.1. By removing retransmission from TCP, MTP becomes an unreliable transport protocol like UDP, but has the exact TCP congestion avoidance mechanism that has proven to be effective in practice.

With the unreliable datagram delivery semantics, the MTP senders have the freedom at the application side input queue to trade the queuing delay that contributes to the huge transmission delay upon capacity overestimation by the streaming application to packet losses. In addition, MTP receivers deliver packets to the application above as soon as they are received avoiding extra transmission delay and jitter that can be caused by TCP receivers in order to guarantee the in-order lossless data stream service. These extra delays discourage delay sensitive applications from using TCP. Moreover, MTP resolves most of the streaming unfriendly TCP API issues by revealing the network layer packet loss information to the application layer receivers. Existing UDP-based streaming applications that make media scaling decisions based on network packet loss or frame loss information can also use MTP with a little modification on the streaming channel setup modules. We also propose to add a method to the MTP API to reveal network path round-trip time information measured at the transport layer, since this information may help new streaming applications make prompt and informed streaming decisions.

MTP and the Crimson AQM mechanisms are evaluated via simulations using the NS [127] network simulator. The evaluation of MTP for streaming use requires a proper streaming application that is not provided by NS. Therefore, we build a video streaming application, called Goddard,[5] in NS based on the findings in our video streaming Internet measurement study and other streaming measurement studies [80, 93, 130]. Our simulation results show that the Crimson AQM mechanisms (ARC and SFG) can effectively support best-delay-effort service with protection from misbehaving flows. Also, it is shown in the evaluation that MTP can well support streaming applications offering an efficient low-delay streaming channel such that the offered capacity is TCP-Friendly at all times.

The Crimson architecture may not offer the same level of QoS support as the DiffServ architecture. However, the Crimson architecture resolves a very significant portion of the issues related to supporting diverse QoS requirements, offering improved service quality for interactive applications. The biggest advantage of Crimson over DiffServ is that Crimson can be gradually deployed in the current Internet. Additionally, Crimson attempts to improve support for streaming and other delay sensitive applications by offering a TCP-Friendly and application-friendly multimedia transport protocol.

[5]We named the streaming application after Robert Goddard, the "Father of Modern Rocketry" who attended WPI. Legend has it that while using a lab in WPI campus, his explosions caused some damage and he was then moved to the Magnetic Lab (now Skull Tomb). Even there, neighbors complained of hearing loud noises.

1.3 Contributions

The main contribution of this dissertation is the design and evaluation of the Crimson architecture to improve support for today's diverse Internet applications. The specific contributions include:

- The design and configuration of the rate-based Crimson Congestion Controller, called Aggregated Rate Controller (ARC), targeted to minimize queuing delay over a wide range of traffic conditions. Through a simulation study, ARC is evaluated and compared against similar Congestion Controllers including the PI controller [58], AVQ [75] and SFC [47], and drop-tail queue management over a wide range of network and traffic conditions. The results show that ARC efficiently handles network congestion in all the tested traffic conditions, and when considering all traffic scenarios, outperforms the other mechanisms in terms of queuing delay and link utilization.

- The design of the Crimson Bandwidth Controller, called Stochastic Fairness Guardian (SFG), which provides fair bandwidth protection without requiring structure changes to the current Internet. We provide an evaluation of the SFG (with drop-tail queue management) and the combination of SFG and ARC, referred to as Stochastic Fair ARC (SFA), in comparison with other preferential-based dropping techniques that do not require an edge-core architecture for scalability, including Random Early Detection with Preferential Dropping (RED-PD) [85], Stochastic Fair Blue (SFB) [36] and CHOKe [91], and drop-tail queue management. Through a simulation study, we demonstrate that SFG provides simple and effective fairness protection that complements the weakness of drop-tail alone. The results also show that SFA outperforms other statistical flow management mechanisms considering complexity of the mechanisms, protection, stability, queuing delay and overall TCP performance over a wide range of realistic traffic mixes and loads.

- Characterization of the Internet streaming requirements and identification of the features of TCP that are inappropriate for streaming. In addition, we provide the TCP-friendliness evaluation for RealVideo UDP streams chosen in head-to-head competition with a TCP stream for limited local bandwidth. The measurement study shows that overall, most streaming RealVideo clips are not capacity-constrained for a typical broadband connection. In cases with reduced connection capacity, video streams over TCP take more time to adapt to the capacity than streams over UDP due to the streaming unfriendly TCP API that hides network information. This gives a incentive for video streams to use UDP rather than TCP, suggesting that potentially unresponsive streaming media over UDP will likely persist for some time.

- The design and evaluation of Multimedia Transport Protocol (MTP), a TCP-friendly and user-friendly streaming transport protocol that offers the applications choice to use the available TCP-Friendly bandwidth as they like. Our simulations show that MTP video streams adapt as quickly to the available bandwidth as do UDP streams, and significantly reduce re-buffering events compared to TCP streams while maintaining other media qualities such as frame rate or picture resolution at the level of the TCP streams. The results also show that existing UDP streaming application can use MTP with little modification to their media scaling mechanisms, achieving better quality streams than streams over TCP.

- The design and evaluation of MTP is yet another contribution. The Internet community proposes to build an unreliable transport protocol incorporating end-to-end congestion control, called Datagram Congestion Control Protocol (DCCP).[6] DCCP proposes to support a TCP-like window-based congestion control mechanism (Congestion Control ID 2) similar to MTP and to support TFRC [44] (Congestion Control ID 3), a rate-based end-to-end mechanism. The design and evaluation of MTP for streaming media is a valuable contribution toward the design and evaluation of the DCCP ID 2 congestion control mechanism.

- Design and implementation of the Goddard streaming application in NS. The Goddard streaming client and server use packet-pairs [10, 65, 70] to estimate the bottleneck capacity and select an appropriate media before streaming. During streaming, the Goddard client and server re-select media to stream (i.e. performs media scaling) in response to network packet losses or playout buffer re-buffering events at the client. Goddard, which also simulates play of the received media at the client, is the first and only realistic streaming application in NS.

- Evaluation of the Crimson system. We evaluate the performance gains that the Crimson architecture can offer to streaming applications. Our simulations show that Crimson AQM mechanisms improve congestion control efficiency for TCP and MTP flows using ECN, and protect the network from potentially misbehaving UDP media streams during congestion.

1.4 Roadmap

The rest of the dissertation report is organized as follow: Chapter 2 presents related research in the area of AQM, the Differential service architecture, streaming traffic

[6]http://www.icir.org/kohler/dccp/

characterization, and TCP-friendly transport protocol design. Chapter 3 presents additional in depth background materials that help in understanding this dissertation research. The background discussions include the TCP/IP networking structure, Internet congestion control issues, and QoS requirements and trends in current Internet applications. Chapter 4 presents the design, configuration and evaluation of the Crimson AQM mechanisms, ARC and SFG. Chapter 5 presents the results and findings from our Internet streaming measurement study followed by the design and evaluation of MTP for streaming media, including the design of Goddard. Chapter 6 presents evaluation of streaming media under the Crimson architecture. Chapter 7 concludes this dissertation research and lists possible future work.

Chapter 2

Related Research

This chapter introduces research work related to the three sub-areas of this dissertation: Active Queue Management, streaming characterization, and TCP-friendly transport protocols.

2.1 Active Queue Management

Active Queue Management (AQM) refers to traffic management techniques at a router that detect and notify traffic sources of imminent network congestion to prevent outbound buffer overflow and control queuing delay [14]. When notified of network congestion, cooperative traffic sources like TCP reduce their transmission rates to participate in the congestion control. In the case network congestion cannot be managed voluntarily by the traffic sources, AQMs may use buffer management techniques to suppress traffic to the targeted traffic level and achieve the QoS goal. In this section, we first propose an AQM taxonomy that provides a systematic way to classify and analyze AQM mechanisms. We examine various AQM mechanisms using the taxonomy.

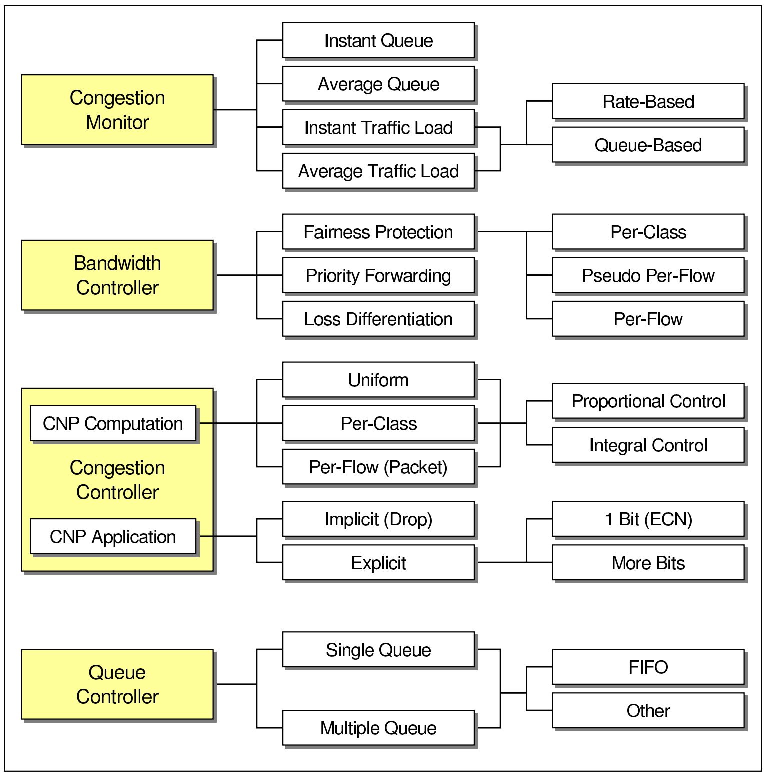


Figure 2.1: AQM Taxonomy

2.1.1 AQM Taxonomy

In general, the tasks of AQM can be divided into that of a *Congestion Monitor* which detects and estimates congestion, a *Bandwidth Controller* that manages use of the output bandwidth, a *Congestion Controller* which computes and applies the congestion notification probability (CNP) to incoming traffic and a *Queue Controller* which manages buffer usage and packet scheduling. We have developed an AQM taxonomy based on the four AQM tasks, as shown in Figure 2.1.

2.1.1.1 Congestion Monitor

The first task of an AQM is to monitor, detect and estimate congestion. This estimation is used for bandwidth management decisions by the Bandwidth Controller, or for congestion notification probability (CNP) computations in the Congestion

Controller.

In general, AQM congestion detection and estimation mechanisms can be classified by the monitoring policy that uses either *queue length* as a measure of congestion or the incoming *traffic load* as a measure of congestion. In each case, either the *instant* or *average* measure of the quantity can be used. Traditionally, AQM mechanisms used queue statistics such as instant or average queue length against queue thresholds as a measure of impending congestion. Yet, AQMs may measure incoming traffic load and declare congestion when the measured load is greater than the target load. The target traffic load is typically set to near 1, which defines congestion as a state that the estimated incoming traffic rate (the offered load) is greater than the service rate or link capacity. As in the case of the queue-based congestion estimation policies, either an instant or an average measure of the traffic load can be used in the load-based policies. An instant load refers to the load measured in the last measurement interval, whereas an average load can be defined as the average of instant loads over a specified period.

The traffic load based congestion estimation policies can be further classified by the monitoring method (traffic rate or queue length) to estimate the traffic load. It is more intuitive to measure the traffic load in terms of incoming traffic rate over service rate. Yet, the traffic load can also be estimated in terms of queue length differences over a measurement interval. Both rate-based and queue-based load estimation methods have advantages and disadvantages. Rate-based methods usually have a little more overhead than queue-based methods since rate-based methods need to collect every incoming packet size while queue-based methods can sample the queue size every measurement interval. However, an important advantage is that the rate-based congestion estimation methods can reduce the estimation noise and detect impending congestion before the queue starts to grow, allowing the Bandwidth Controller or Congestion Controller to more accurately and promptly respond to the imminent congestion.

Of importance in load estimation is determining an effective measurement interval, which may affect the stability of the feedback control system, link utilization and queuing delay and buffer overflows. For example, choosing an insufficiently small interval can lead to the Congestion Controller making an over reactive decision on network congestion while choosing an exceedingly large interval can make the Congestion Controller less responsive. Similarly, when an averaging technique is used for congestion estimation, the averaging factor will affect the responsiveness and performance of the controller.

2.1.1.2 Bandwidth Controller

Following the Congestion Monitor, an AQM may have a Bandwidth Controller that manages the use of the outbound bandwidth. Bandwidth controllers can be categorized based on the nature of services they provide and the QoS goals. A bandwidth controller may provide *priority forwarding* or *loss differentiation* service. Priority forwarding is a priority class-based protection mechanism in which, upon congestion, packets from a lower priority class are dropped before dropping packets belonging to higher priority classes. A loss differentiation service is also a class-based protection mechanism in which, upon congestion, a predefined proportion of traffic is dropped from each class.

The most common type of bandwidth controller is one that provides *fairness protection* in which individual flows or groups of flows are protected from one another. We refer to such a Bandwidth Controller as a *Bandwidth Guardian*. Bandwidth Guardians can be sub-categorized into using class-based, per-flow or pseudo per-flow bandwidth management. Class-based bandwidth fairness protection mechanisms usually have the least overhead among the three, whereas per-flow fairness protection mechanisms that maintain per-flow state have the most overhead. Pseudo per-flow bandwidth management mechanisms detect outstanding high-bandwidth flows without keeping per-flow information or by using a minimum per-flow information and rate limit the flows. Pseudo per-flow management protects flows from misbehaving high-bandwidth flows rather than enforcing per-flow fairness, but at a lower cost than per-flow management.

Even within the same subcategory of the Bandwidth Guardians, the accuracy and performance of the mechanisms can significantly differ in the complexity and traffic information used for bandwidth management. For example, a simple class-based Bandwidth Guardian can pre-assign a fixed congestion bandwidth to each class, while a more advanced class-based mechanism can dynamically assign a fair bandwidth to each class for the price of estimating the number of active flows in each class.

2.1.1.3 Congestion Controller

Incoming traffic that passes the Bandwidth Controller is forwarded to a Congestion Controller. The job of the Congestion Controller is to prevent or control network congestion by notifying traffic sources of the impending congestion earlier so that congestion responsive traffic sources such as TCP can reduce their transmission rate. Although an explicit binary congestion notification method called Explicit Congestion Notification (ECN) was proposed about a decade ago [41], the implicit mechanism of dropping incoming packets has been historically used. For this reason, it is sometimes hard to distinguish Congestion Controllers from Bandwidth Controllers as packet drops resulting from the bandwidth management also act as implicit congestion notification. Yet Bandwidth Controllers attempt to repressively manage outbound bandwidth usage at congestion, while Congestion Controllers attempt to prevent congestion with the help of responsive traffic sources. An easy way to distinguish between a Bandwidth Controller and a Congestion Controller is, if it does not make sense for a mechanism to use ECN instead of packet drop, then it is a Bandwidth Controller, otherwise, it is a Congestion Controller. Note that an AQM may have either a Bandwidth Controller or a Congestion Controller, or both controllers.

More precisely, a Congestion Controller determines a congestion notification probability (CNP) based on the estimated congestion level, its control history and possibly other traffic information, and notifies traffic sources by randomly marking (or dropping) the incoming packets with the estimated CNP. Every Congestion Controller has its own QoS goal and thus has a specific CNP computation policy of which flows should reduce or increase their transmission rate and by what amount. The QoS goals can be simply to prevent congestion with minimized queuing delay, to yield fair bandwidth allocation among responsive sources while preventing congestion, or to provide a diverse QoS while preventing congestion. To achieve these QoS and performance goals, the AQM may perform a *uniform*, *class-based*, *per-flow* or *per-packet* CNP computation. For example, to achieve only the basic goal of preventing congestion, a Congestion Controller may compute and apply a uniform CNP to all incoming traffic. In order to additionally yield fair bandwidth allocations among different classes of responsive traffic, a Congestion Controller may compute and apply per-class CNPs. Furthermore, a congestion controller may choose to perform per-flow CNP computation to yield per-flow fairness among responsive flows, or to provide a customized QoS to each flow.

The CNP computation methods can be further classified into two categories based on how the CNP is computed. The first category is a *Proportional (memory less)* controller [100] that does not consult the recent control history but computes the CNP based only on the current estimated congestion level and traffic information. Proportional congestion controllers typically require knowledge on the traffic sources to successfully control congestion. The fact that the stable state average transmission rate of window-based traffic sources (like TCP) given a CNP can differ based on the average RTTs they experience implies that a router should have some knowledge of the average RTT and the number of flows (N) to control aggregated average incoming traffic rate and to control per-flow average throughput. For example, an AQM that has knowledge of the average RTT of the incoming flow aggregate and N can compute a proper CNP for the flow aggregate using the queue law [38] assuming ideal TCP traffic sources. Or, knowing the fair bandwidth share (link capacity divided by N) and RTTs of individual flows, a router can compute a proper CNP for each flow that will bring each average flow rate to the fair share. To be practical, since not all TCP traffic is greedy and long-lived, a proportional congestion controller is required to know per-flow transmission rates ($cwnd/RTT$) and congestion estimation. That is, the router should know exactly how much bandwidth to take away from each flow to compensate for the overloaded amount of the service bandwidth to increase congestion control precision. Without knowing the transmission rate of each flow in the first place, it is not possible to determine how much bandwidth will be reduced per notification. However, knowing the per-flow transmission rate (or just $cwnd$ since RTT is already known) helps little under the current congestion notification structure of the Internet due to the inefficient and inaccurate router-to-host binary congestion control communication method of marking with the CNP. In addition, it is not possible for a router to instantly make coarsely-responding TCP to transmit at a specific rate. Thus, precision congestion control using CNP alone cannot readily be done. This is the basic argument behind the design of XCP [69], one of the most recent mechanisms, that proposes to use a window based traffic source that inform routers of *RTT* and *cwnd* and transmit at the rate that the most congested router explicitly specifies in terms of allowable *cwnd*.

One critical problem facing AQMs in this category is that per-flow traffic information such as *RTT* and *cwnd* may not be practically and securely obtainable under the current Internet structure. On the other hand, AQMs that compute CNP based only on the congestion estimate or use incomplete traffic information may not be able to support a wide range of traffic without confronting configuration problems or failure to meet QoS and/or performance goals for some traffic mixes.

The second class of CNP computation methods is *Integral* control[1] that heuristically searches for a stable state CNP that will bring the aggregated traffic to a desired level based on the recent control history and congestion estimates measured by the Congestion Monitor. More precisely, integral congestion controllers continuously update the CNP of the previous interval based on an estimated congestion control error. A significant advantage of integral CNP computation methods over proportional methods is that integral methods require no additional traffic information to converge to a CNP that accomplishes the aggregate QoS goals such as bounded queuing delay. While the integral CNP computation techniques are usually used for incoming traffic aggregates, they can be applied for groups of flows or even possibly for individual flows assuming per-flow throughput and fair bandwidth share (or N) can be measured. An important integral congestion controller issue is to find an appropriate CNP update interval and increment/decrement steps in order to ensure the congestion control stability and responsiveness under both steady and

[1]In fact, an Integral controller [100] is a specific type of feedback controller. However, our AQM taxonomy uses "integral controller" to represent congestion controllers that use an integral feedback control mechanism, and encompasses Proportional-Integral (PI) controllers and Proportional-Integral-Derivative (PID) controllers.

changing network traffic conditions.

So far, two types of CNP determination methods have been discussed and illustrated. The integral method is usually used to implement uniform or per-class based congestion notification services. On the other hand, the proportional CNP computation method utilizing the complete traffic information is a typical per-flow CNP computation method that can be used to implement per-flow fair congestion notification services or customized QoS congestion notification services. Alternatively, a per-flow QoS service may be implemented using the integral method with per-flow QoS requirement information from traffic sources. That is, a router may heuristically adjust the updated CNP of the traffic aggregate for each flow considering the QoS requirement of the flow.

As briefly mentioned in a previous paragraph, Congestion Controllers can use either the *implicit* congestion notification method of packet dropping or the *explicit* congestion notification method of marking. Currently, the Internet Protocol only supports binary ECN bit marking. ECN marking can offer significant performance gain in terms of packet loss rate compared to the implicit packet drop congestion notification, as discussed in Section 3.2 in detail. However, it is possible that multiple bits can be used to enhance congestion control precision in the future.

2.1.1.4 Queue Controller

The last component of an AQM is a Queue Controller. A Queue Controller manages the transmission of packets forwarded by the Congestion Controller or Bandwidth Controller. Typically, AQM mechanisms keep only a single packet queue. However, a mechanism may assign a packet queue for each incoming flow and perform link scheduling (although it is arguable that this mechanism is not an AQM). Alternatively, an AQM may assign a packet queue for each class of traffic. To encompass these possibilities, the AQM taxonomy includes the *number of packet queues* for the Queue Controller categorization.

Every packet queue has a *management discipline*, the simplest being FIFO queue management. Other management disciplines include support for a uniform QoS such as bounded average queuing delay or per-flow or per-class delay. The Queue Controller can support diverse per-flow QoS requirements using a packet scheduling rather than FIFO in cooperation with the Congestion Controller, although little work has been done in this direction. The per-flow QoS parameters that an Internet router can support are CNP and queuing delay. As mentioned earlier, a Congestion Controller may consider the QoS requirements in determining the CNP for a flow given QoS information from traffic sources. Similarly, the Queue Controller can consider the delay requirement of each flow using QoS packet scheduling. As long as the flow uses bandwidth less than or equal to the fair share, trying to meet the QoS requirements of individual flows may be desirable. However, QoS packet scheduling may need to address issues such as starvation that can affect the throughput of window based traffic sources like TCP.

2.1.2 AQM Mechanisms

The previous section introduced our AQM taxonomy. In this section, popular AQM mechanisms are examined using the taxonomy. First, we introduce AQM mechanisms that mainly offer Congestion Controller functionalities, followed by AQM mechanisms that emphasize the functionality of a Bandwidth Controller. Then, we introduce AQM mechanisms that support diverse QoS using Queue Controllers.

2.1.2.1 Congestion Controllers

Random Early Detection (RED) [45] is one of the first AQM mechanisms to offer the functionalities of a Congestion Controller. The Congestion Monitor of RED uses a weighted average queue length (q_{avg}) and a threshold (th_{min}) to detect congestion, and estimates the congestion amount in terms of $q_{avg} - th_{min}$. The RED Congestion Controller is a proportional controller that computes the uniform CNP based only on the estimated degree of congestion while q_{avg} is less than the upper operating bound of the Congestion Controller (th_{max}). The Congestion Controller linearly maps q_{avg} ranging from th_{min} to th_{max} onto the CNP from 0 to p_{max}, and supports both drop and ECN mark congestion notification. RED does not have a Bandwidth Controller but it does have a special Queue Controller. The Queue Controller of RED uses a single queue and operates in two modes. When q_{avg} is less than th_{max}, the Delay Controller operates with FIFO packet scheduling. Otherwise, it operates in the QoS mode dropping all incoming packets until q_{avg} drops under the th_{max} to bound the average queuing delay. Later versions of RED support the "gentle" setting for the QoS Queue Controller that replaces the strict uniform delay control of RED by allowing q_{avg} to go above th_{max}, in order to reduce bursty packet drop rates. RED is known to be difficult to configure for a wide range of traffic [12, 21, 34].

State Feedback Controller (SFC) [47] is a Congestion Controller that provides a proportional feedback control as in RED. However, instead of using average queue length, the Congestion Monitor of SFC estimates congestion by monitoring both the instant queue length and the instant traffic load. The Congestion Controller, then, computes the uniform CNP proportional to the instant queue length and also to the estimated incoming traffic load. SFC is based on a classical control theory and has a wider stability margin and a faster congestion response time than RED. However, as in other proportional controllers, SFC has difficulty in controlling the queue length to a target as traffic load varies, which is shown in Chapter 4.1.3. SFC does not have a Bandwidth Controller. The Queue Controller of SFC uses FIFO queue management on a single queue.

BLUE [35] controls congestion using an integral Congestion Controller to find a stable state CNP that may bring the aggregated traffic to a desired level. The BLUE Congestion Monitor takes a buffer overflow event in the last measurement interval as an indication of congestion and a link idle event as a reduction of congestion. However, the Congestion Monitor does not estimate the congestion level, since this information is not used in the CNP computation process. The BLUE Congestion Controller periodically updates the CNP of the previous measurement interval using fixed sized steps upon congestion or congestion reduction event detection. The use of fixed CNP potentially has CNP convergence and stability problems for fast changing traffic conditions. The updated CNP is uniformly applied to the incoming traffic until the next measurement interval with the preferred notification method being ECN marking. The BLUE Queue Controller uses a single queue with simple FIFO packet scheduling.

Proportional-Integral (PI) controller [58] and Random Early Marking (REM) [5] are typical integral congestion control mechanisms that use traffic load-based congestion monitoring techniques on the incoming traffic aggregate. In PI, the Congestion Monitor detects and estimates congestion, or lack of it, based on the instant queue length and instant traffic load measured by the difference in the queue length over the measurement interval. Then, the Congestion Controller updates the CNP of the previous measurement interval based on the current congestion estimates (queue length and load), and drops or marks the incoming packets with the updated CNP. PI supports both mark front and mark back options. PI does not have a Bandwidth Controller. The Queue Controller of PI uses FIFO queue management on a single queue.

The Congestion Monitor of REM detects and estimates congestion, or lack of it, based on the instant queue length and average traffic load as measured in terms of average incoming traffic rate over the link capacity. Yet, in its implementation REM recommends estimating the traffic load using the queue length differences to reduce the measurement overhead. The REM Congestion Controller updates the CNP of the previous measurement interval based on the current congestion estimates. Yet, the CNP application is different from ordinary ECN marking. A REM router marks the unmarked incoming packets with a transformed probability of $1-\phi^{CNP}$, where ϕ is a constant shared by REM traffic sources and routers. REM capable traffic sources monitor their own transmissions to detect the marking probability, convert back to the CNP and adjust the transmission rates to TCP-friendly rates. Thus, REM is also a congestion control protocol beyond simply an AQM. REM does not have a Bandwidth Controller. The Queue Controller of REM uses FIFO queue management on a single packet queue.

Adaptive Virtual Queue (AVQ) [75] has a special integral Congestion Controller that does not directly calculate or update the CNP, but the CNP is indirectly determined by a virtual queue overflow assuming that incoming traffic is serviced at a virtual service rate. Periodically, the Congestion Monitor of AVQ uses the incoming traffic rate and target traffic rate to determine congestion. The Congestion Controller updates the virtual service rate of the previous period according to the determined congestion level. Then, the Congestion Controller marks the incoming packets for congestion notification when the virtual queue overflows. The virtual service rate adjustment causes a change in the virtual queue overflow rate that in turn determines the CNP. The Congestion Controller of AVQ may not efficiently control queuing delay, as it does not use any queue statistics in determining the level of congestion. Furthermore, AVQ has potential for bursts of congestion marking when there are a few flows. AVQ does not have a Bandwidth Controller, and the Queue Controller uses a normal FIFO queue management on a single packet queue.

eXplicit Congestion Protocol (XCP) [69] and FAST [99] are congestion control protocols that include new end-to-end congestion control mechanisms as well as AQMs. XCP assumes window based traffic sources that inform the network of their RTT and *cwnd*, and adjusts *cwnd* as explicitly specified in the returning ACK packets. Thus, XCP requires significant changes in the Internet structure and IP header. XCP routers have a Congestion Monitor that computes the overloaded or underutilized traffic amount of the previous measurement interval, set to the average RTT of the incoming flows. The proportional Congestion Controller of an XCP router equally distributes the underutilized traffic amount to each flow, or takes the overloaded traffic amount from the flows proportional to their bandwidth usage. The fair underutilized traffic share to give and the proportional overloaded share to take away from each flow is computed and converted to per-packet *cwnd* increment or decrement amounts using a technique that utilizes the ACK paced congestion window adjustment characteristics of a reliable window based transmission protocol considering the RTT and *cwnd*. XCP also shuffles and reallocates a small fraction ($\gamma = 0.1$) of pre-distributed bandwidth through this feedback computation to increase bandwidth distribution fairness, possibly compromising efficiency. XCP routers may update the *cwnd* originally given by the traffic source in the IP header according to their computation, which will be returned to the source via ACK packets by the receiver end-system. Thus, the XCP Congestion Controller offers per-flow fair congestion notification service. XCP does not have a Bandwidth Controller, and has a normal Queue Controller offering FIFO packet scheduling.

The FAST proposed for high bandwidth-delay product networks to overcome the inefficiency of using TCP, on the other hand, does not require modification to IP specifications, since the FAST sources and routers use the REM [5] marking method described above for congestion communication. The design of the FAST protocol is based on two theories, a market theory of utility price appropriation for a fair utility resource allocation (i.e. a fair allocation of outbound link bandwidth at congested router), and a control engineering discipline for a stable implementation of the utility price appropriation mechanism (i.e. realization of a stable feedback control system). In short, FAST routers use AQM that implements an integral Congestion Controller to notify the CNP (or the link price in their term) to FAST traffic sources via REM marking method. FAST traffic sources use a utility function that uses only the CNP to compute the fair transmission rate (or demand) for the notified CNP (or the link price). Thus, FAST traffic sources, in theory, can fairly use the network resources regardless of their end-to-end RTTs. However, the REM communication efficiency is important for the performance of FAST in a practical networking environment. Due to the REM communication (encoding/decoding) delay that is a function of the transmission rate of the traffic source, FAST may not perform well for low bitrate flows or when backward path congestion causes a significant amount of ACK compressions. In addition, the complexity of the FAST configuration is yet another issue that has not yet been addressed. FAST does not have a Bandwidth Controller, and has a normal Queue Controller offering FIFO packet scheduling.

2.1.2.2 Bandwidth Controllers

Bandwidth controllers can be categorized into ones providing *bandwidth fairness protection* (Bandwidth Guardian), *priority forwarding* or *loss differentiation* services, among which bandwidth fairness protection mechanisms are the most common. It is believed that the network should provide a certain level of bandwidth protection although it is arguable how and to what extent the protection should be enforced due to the price-performance tradeoff.

Differential Congestion Notification (DCN) [78] offers a loss differentiation service to two classes of flows, “small” and “large” high-bandwidth flow classes. In addition, DCN provides a per-flow bandwidth fairness protection service within the large flow class. DCN heuristically classifies large flows from small flows using a threshold on the hashed flow’s packet count. DCN does not regulate the bandwidth usage of small flows. For “large” flows, DCN applies differentiated preferential packet drop rates to regulate their bitrates such that the outbound queue length stay at the pre-configured target. DCN is motivated by the fact that short connections dominate many Internet links (more than 84% of all flows in some cases [132]), yet they use only a small fraction of the capacity on most links (less than 20% of all bytes [132]). In order to compute differentiated per-flow packet drop probability for large flows, DCN takes a two-step approach. DCN uses a PI controller [58] as a Bandwidth Controller to compute a uniform drop probability (“fair share”) that makes the outbound queue stays at the target length, where the traffic from small flows acts as a noise to the feedback control system. Then, DCN computes and applies per-flow drop probability to an incoming large flow considering its bandwidth usage in packet count. Le et. al. in [78] show that DCN improves response times of HTTP request/response exchanges without using explicit congestion notification (ECN).

Class-Based Queuing (CBQ) [46] and Class-Based Threshold (CBT) [103] are lightweight router queue mechanisms that offer traffic class-based protection by assigning a fixed amount of bandwidth to each traffic class upon congestion. While CBQ is not an AQM, if one should apply the AQM taxonomy to CBQ, it can be seen as an AQM with only a Queue Controller that has multiple packet queues, where each queue is managed with FIFO packet scheduling discipline.

CBT extends RED to support class-based traffic protection by inserting a Bandwidth Controller that performs a “threshold test”. The Bandwidth Controller has a pre-determined average buffer share of each supported traffic class. The Bandwidth Controller monitors the average buffer usage of each traffic class, and drops incoming packets for classes that use the outbound buffer more than their fair share. Thus, CBT ensures a pre-determined bandwidth share for each traffic class during congestion.

More heavyweight class-based approaches not only offer protection but also ensure fair bandwidth distribution among traffic classes. For example, Dynamic Class-Based Threshold (D-CBT) [23] extends the Bandwidth Controller of CBT to count the number of active flows for each class in order to determine fair average buffer shares for each supported traffic class instead of using pre-determined class thresholds. The Bandwidth Controller monitors the average buffer usage of each traffic class, and drops incoming packets from the classes that use the outbound buffer more than their fair share. Thus, D-CBT ensures a fair class bandwidth for each traffic class during congestion.

Approaches to provide per-flow or pseudo per-flow bandwidth fairness can be divided into scheduling-based and preferential-based packet dropping mechanisms as shown in Figure 2.2. Scheduling-based techniques, such as Weighted Fair Queuing (WFQ) [33] and Stochastic Fair Queuing (SFQ) [89], allocate a separate queue to each flow or group of flows passing through a router’s outgoing link and transmit packets from the queues in round-robin fashion. Scheduling-based mechanisms are generally expensive to implement due to the complexity of the link/packet scheduling. Moreover, it may be undesirable to combine a scheduling-based mechanism with a Congestion Controller due to the redundancy inherent in providing queue buffers needed to support both mechanisms. As in the case of CBQ, it is hard to classify

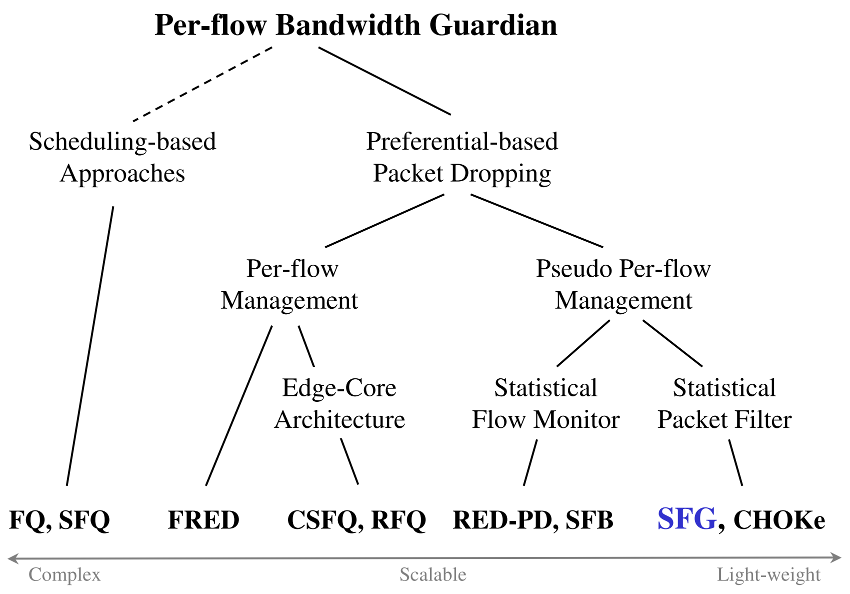

Per-flow Bandwidth Guardian
Scheduling-based Approaches
Preferential-based Packet Dropping
Per-flow Management
Pseudo Per-flow Management
Edge-Core Architecture
Statistical Flow Monitor
Statistical Packet Filter
FQ, SFQ
FRED
CSFQ, RFQ
RED-PD, SFB
SFG, CHOKe
Complex
Scalable
Light-weight

Figure 2.2: Per-Flow Fairness Protection Mechanisms

WFQ and SFQ as an AQM. However, should it be classified, it can be seen as an AQM with only a Queue Controller that assigns a FIFO managed packet queue for each incoming flow.

Preferential-based packet dropping techniques, also referred to as per-flow Bandwidth Guardians,[2] monitor, detect and regulate misbehaving flows before forwarding packets to an outbound link queue that may or may not be managed by a Congestion Controller. Preferential-based dropping mechanisms can be further categorized by their complexity and the amount of state information maintained. The most complex mechanisms, including Fair Random Early Drop (FRED) [81], Core Stateless Fair Queuing (CSFQ) [119] and Rainbow Fair Queuing(RFQ) [17], require per-flow state information. The fact that per flow state information does not scale for high capacity networks with many flows is a significant weakness for FRED. However,

[2]Although scheduling-based fairness protection approaches share the same goal with Bandwidth Guardians, the non-AQM scheduling-based approaches may not be classified as a Bandwidth Guardian. The dotted line in Figure 2.2 implies this relationship.

CSFQ and RFQ reduce this problem by requiring per-flow state information only at DiffServ [8]-like edge routers and forwarding the information to core routers in each IP header.

FRED, providing a per-flow bandwidth fairness protection upon RED, uses the same Congestion Monitor, Congestion Controller and Queue Controller as RED. Additionally, FRED has a Bandwidth Controller that offers a per-flow bandwidth protection service. That is, the FRED Bandwidth Controller monitors per-flow buffer usage and punishes flows that use more buffers than heuristically determined limits before performing RED congestion control functions. In addition, the FRED Bandwidth Controller heuristically identifies fragile flows and bypasses the Congestion Controller, directly forwarding the packets to the Queue Controller. FRED has similar configuration problems to RED.

CSFQ does not have a Congestion Controller, but relies on the Congestion Monitor and the Bandwidth Controller to handle congestion. CSFQ assumes an edge-core network architecture, where edge routers estimate per-flow transmission rates (r_i) and forward the information to the core routers. The Bandwidth Controller of CSFQ uses the per-flow transmission rates along with the aggregated traffic rate measured by the Congestion Monitor to estimate the fair bandwidth share (α). Then, the controller probabilistically drops the exceeded amount of traffic of each flow from their fair share by applying a per-flow drop probability ($1 - \frac{\alpha}{r_i}$) to the incoming packets. CSFQ does not support ECN marking as it does not have Congestion Controller.

RFQ also has edge and core router behavior. The RFQ edge router monitors per-flow transmission rates and incoming packets are assigned and marked with priority levels (or colors) such that packets of a flow are assigned to a lower class up to an allowed per-flow rate of the class before being assigned to the next priority class. The allowed per-flow rate for each priority class is predetermined using a non-linear encoding scheme. The RFQ core routers have a Congestion Monitor and a Bandwidth Controller that provide a priority forwarding service. However, RFQ does not have a Congestion Controller. The Congestion Monitor uses the instant queue length and traffic load in combination to detect congestion, or lack of it, and estimates the degree of congestion. The Bandwidth Controller adjusts the priority level threshold according to the estimated amount of congestion. The threshold can roughly be decreased up to a quarter of the current priority level and can be increased one priority level in a measurement interval. Then, the Bandwidth Controller simply admits packets marked lower than or equal to the priority level threshold given as long as there is a space in the outbound buffer. The bandwidth management performance of RFQ can be sensitive to the measurement interval and the allowed per-flow rate assignment of the priority classes.

Other preferential-based dropping techniques uses approximation approaches that can protect fairness among individual flows without requiring per-flow state information or the edge-core DiffServ network architecture. Techniques such as Random Early Detection with Preferential Dropping (RED-PD) [85], Stochastic Fair Blue (SFB) [36] and CHOKe [91], use statistical flow management to address scalability. RED-PD and SFB employ statistical flow monitoring to identify and then regulate misbehaving flows. Although statistical flow monitoring mechanisms can significantly reduce the flow state information needed to be maintained when a small number of flows account for the majority of the Internet traffic [101], the mechanisms used to identify misbehaving flows are complex and may induce significant processing overhead. To avoid the complexity of flow identification, CHOKe uses a stateless statistical traffic filtering technique that does not require any flow state information trading for some performance loss. Although these mechanisms cannot guarantee total per-flow fairness, the fact that they can be deployed independently at any router with a moderate and scalable overhead is an attractive property of this approximation technique. Moreover, whether total per-flow fairness is necessary or even beneficial is arguable.

RED-PD extends RED to support pseudo per-flow bandwidth fairness protection by inserting a Bandwidth Controller that monitors the per-flow packet drop history to statistically detect high-bandwidth flows in times of congestion. Although RED-PD's drop history-based method for selecting flows to monitor reduces the amount of state information required, the flow selection algorithm is complex and requires a significant overhead due to dynamic the history list lookup and maintenence. Moreover, the flow selection performance is sensitive to the history collection epoch length, computed based on a TCP-friendly rate formula [43]. Once a flow is identified as an outstanding high-bandwidth flow, the RED-PD Bandwidth Controller monitors its traffic rate and regulates it under the estimated TCP-friendly rate given by the TCP-friendly formula using the CNP computed at the Congestion Controller and a hard-coded round-trip time as parameters.

SFB is a fairness and protection enhanced version of BLUE. SFB adds a Bandwidth Controller to BLUE, and uses a Bloom filter in the Congestion Controller to offer a fair congestion notification service. The Congestion Controller assigns incoming flows into different groups multiple times and maintains the CNP for each flow group using the BLUE AQM [35]. The Congestion Controller determines the pseudo per-flow CNP for an individual flow by taking the minimum CNP of the groups to which the flow belongs, and marks the packets of the flow with this minimum. To determine if a specific flow is consuming more bandwidth than other flows, the Bandwidth Controller monitors the maximum CNP of the bins to which a flow belongs. When the maximum CNP of the flow is greater than or equal to a given threshold (0.98 is the recommended value), the flow is put in to a penalty box and gets rate limited to a pre-set data rate for a fixed amount of time.

CHOKe is a Bandwidth Controller that implements a lightweight statistical packet filtering mechanism. For each incoming packet, CHOKe randomly chooses a packet from the outbound queue and drops both packets if they are from the same flow. This algorithm is derived from the observation that the higher a flow's bitrate, the more the chance for a packet from the flow to be found in the outbound queue. However, the basic CHOKe algorithm cannot effectively control the target flow rate for unresponsive flows as the offered traffic load changes. Thus, CHOKe extends the number of the random packet matches per incoming packet as offered load increases. To estimate offered load, CHOKe proposes to use RED's average queue length maintained by the RED Congestion Monitor. The extended CHOKe algorithm divides RED's minimum and maximum queue thresholds range into m even subregions and applies $2i + 1$ drop comparisons for an incoming packet, where $i = \{0, 1, 2, 3, ..., m - 1\}$ is the subregion ID. Although simple in practice, CHOKe can have significant overhead that increases with the offered load. Moreover, CHOKe's stateless design makes it difficult to configure the target per-flow bitrate under changing traffic loads. Furthermore, CHOKe may punish well-behaved flows that are unluckily selected and noticeably degrade TCP performance under light traffic loads. Lastly, CHOKe may not work well with delay optimized AQMs, such as the one proposed in this dissertation, since there are not as many packets to randomly compare against as in a RED queue.

Lastly, Stochastic Fairness Guardian (SFG) in Figure 2.2 that uses a multi-level hash filter requiring minimal state information is described in detail in Section 4.2. SFG offers filtering performance comparable to that provided by more complicated statistical flow monitoring mechanisms such as RED-PD [85] while avoiding the complexity of flow identification.

2.1.2.3 Queue Controllers

We introduce two Queue Controller mechanisms that control class-based or per-flow delay, trading a delay gain for a packet loss rate or CNP loss. These mechanisms assume that traffic sources give their QoS requirement hints in the IP packet headers.

Alternative Best Effort (ABE) [60] is a Queue Controller that offers a strictly bounded queuing delay service to a delay sensitive traffic class. ABE may not be seen as an AQM as it does not have a Congestion Monitor, Bandwidth Controller, or Congestion Controller. ABE supports two classes of traffic, delay sensitive ("green") and throughput sensitive ("blue") classes in the following way: The blue packets are serviced with queuing delays no more than they would experience when serviced with a single FIFO packet queue for the outbound link. Within this blue packet forwarding service constraint, incoming green packets that cannot make the queuing delay deadline (pre-set by the router) are dropped, and ones that can make the deadline are scheduled such that they may precede the blue packets as much as possible. This packet scheduling policy is referred to as *duplicate scheduling with deadlines* (DSD), and is implemented using a virtual queue and two specially scheduled FIFO packet queues. DSD trades off the packet loss rate for a delay gain for delay sensitive traffic, and does not harm the throughput sensitive traffic. DSD (or ABE) can in theory replace the FIFO single queued Queue Controllers of other AQM mechanisms.

RED-Boston [106] extends RED to adjust the target average queue length based on the traffic mixes in order to better support the average delay requirements. RED-Boston modifies the Congestion Controller of RED to keep track of the average delay hints and adjust the target queue length accordingly. Thus, the Congestion Controller decreases the target average queue length when delay sensitive traffic dominates, and increases the target when throughput sensitive traffic dominates.

In addition, RED-Boston supports per-flow QoS, offering freedom to choose the delay sensitivity treatment level at the cost of a CNP loss. A CNP loss, which means an increased packet loss rate for implicit congestion notification or an increased marking rate for ECN, results in a throughput loss for TCP and TCP-Friendly traffic sources. For each incoming packet, the Congestion Controller heuristically calibrates the uniform RED CNP to compute the per-flow (or per-packet) CNP considering the delay hints. Thus, a flow that requests more delay sensitive queuing delay treatment may experience a higher CNP. The Queue Controller of RED-Boston uses QoS packet scheduling referred to as *weighted insert*. Using the arrival time and the delay hint in each packet, the Queue Controller weighs each admitted packet and inserts it in sorted packet queue that is serviced in the reverse weight order. Thus, packets requesting lower delays are preferred in the scheduling, while the automatically aging weights prevent starvation for packets requesting higher delays for a lower CNP. By heuristically calibrating the per-flow CNP inversely proportionally to the delay hints, RED-Boston tries to even out the throughput gains due to the queuing delay gains by the delay sensitive TCP-Friendly traffic sources. How effectively this mechanism can preserve fair bandwidth allocation among TCP and TCP-Friendly sources is ongoing work.

2.2 Differentiated Services Architecture

Although AQMs may resolve QoS issues to some extent, it is hard to meet the diverse QoS requirements of all applications using a single best-effort service architecture. Understanding this fundamental limitation of the current Internet, the Internet research community is considering restructuring the service architecture to support multiple differentiated classes of services, known as the Differentiated Services (Diff-Serv) architecture [8], shown in Figure 2.3.

DiffServ is intended to offer an end-to-end differentiated service by concatenation of per-domain services and Service Level Agreements (SLAs) between adjoining domains along the travel path, which specifies the handling of the customer's traffic. Within a network domain (i.e., an ISP), services specified in SLAs are realized by traffic conditioning at the edge and simple differentiated forwarding at the core routers, where traffic conditioning includes classification, metering, policing and shaping. This edge-core architecture is targeted to achieve scalability by pushing "intelligence" toward the edge and making core routers as simple and therefore as fast as possible. DiffServ uses Per Hop Behaviors (PHBs) at the core routers, specifying Assured Forwarding (AF) PHB and optionally Expedited Forwarding (EF) PHB [7, 52, 64].

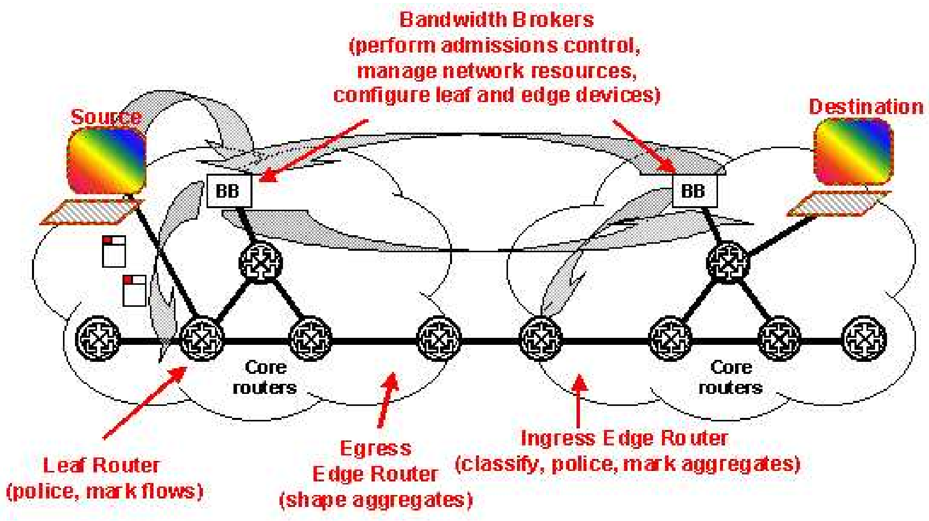


Figure 2.3: DiffServ Architecture (from [121])

The EF PHB refers to router support for a traffic class for which a configurable minimum bandwidth is guaranteed at any time. In cooperation with traffic conditioning at the edge routers, the EF PHB can be used to implement a premium service such as Virtual Leased Line (VLL) ensuring no data loss and very little queuing delay for an incoming traffic aggregate rate that is under a minimum threshold. Yet, the reservation-based premium service will not likely be implemented in the near future due to deployment issues [63]. The EF PHB is not a mandatory part of the DiffServ architecture, and is not a necessary feature of a DiffServ-compliant router.

The AF PHB supports four distinct traffic classes (e.g., similar to CBQ [46] with four queues) and each traffic class supports three priority classes (e.g., similar to an extended RIO [28]), where AF_{ij} denotes the i^{th} AF class with priority j. The AF PHB, which requires an AQM for the priority queuing in each class, can be used to implement "better than best effort" services in cooperation with the edge routers by distributing the incoming traffic to the AF_{ij} class according to the policies in each SLA. For example, an ISP may implement a video streaming AF class and distribute packets into 3 priority classes as shown in [24] to improve the quality of video streams during network congestion. As a second example, the edge routers of an ISP can distribute the incoming traffic such that class AF_{i+1} gets more traffic than AF_i possibly having a higher congestion notification rate or queuing delay at the higher AF class. Also, AF classes can be configured to offer different average queuing delays upon congestion.

Thus, the DiffServ architecture can offer differentiated network services that may be sufficient to support the diverse QoS requirements of different Internet applications. However, what services should be offered or how to configure PHBs as well as traffic policers to support specific services are not yet well-defined. Furthermore, gradual deployment of DiffServ is a non-trivial issue since it requires a network service interface that is different than that of the current Internet. DiffServ is being considered as the service architecture of *Internet2* [62].

2.3 Measurement Studies Characterizing Streaming Traffic

This section, introduces recent research work that measures and characterizes Internet streaming traffic and applications. Understanding streaming characteristics and requirements is important for both AQM and TCP-friendly transport protocol designs that better accommodate streaming traffic.

McCreary and Claffy [88] present trends in application usage seen at the NASA Ames Internet Exchange (AIX) over 10 months, from May 1999 through March 2000, and include analysis of streaming data. Despite the increase in the number of UDP applications in recent years, growth in the total amount of UDP traffic is offset by growth in TCP traffic as well, according to the fraction of packets due to TCP and UDP seen at AIX. RealAudio traffic seen at AIX decreased in packet volume compared to non-RealAudio traffic, although this trend seems to have flattened out in the last few months of their study. This represents either a slowing in the growth of RealAudio traffic or a decline in the growth relative to non-RealAudio traffic. Lastly, the overall fraction of online game traffic seems to be on the rise. However, it is a moving target since the increase is primarily generated by new games as they gain popularity, while older games decrease in popularity over time.

Loguinov and Radha [83] characterize end-to-end network dynamics experienced by streaming low-bitrate CBR MPEG-4 UDP video streams using dial-up PPP connections. The dial-up PPP connections were made from more than 600 major U.S. cities to a dedicated video server connected to the Internet via a T1 line. The 7-month measurement study establishes the feasibility of video streaming in the current best-effort Internet and provides an insight into the dynamics of real-time streaming from the perspective of an average Internet user. They find that Internet packet loss is bursty, and the distribution of loss burst lengths and the RTT appear to be heavy-tailed. RTTs on the order of several seconds are possible along paths in the current Internet with a dial-up modem link. The average RTT appears to be positively correlated with the number of end-to-end hops, whereas packet loss does not seem to depend on the number of hops or the average RTT along the same path. Delay jitter appears to be much more harmful to low-bitrate real-time applications than either packet loss or large RTTs. Packet reordering can be experienced in the current Internet even by paths with very slow bottleneck links, although the reordering delay and reordering distance are relatively small. Lastly, the majority of Internet paths sampled by their experiment were asymmetric.

Merwe, Sen and Kalmanek [125] characterize requests for streaming content (VOD and live broadcasting) and the associated server and network workload distributions by analyzing 4.5 million session-level log entries for two commercial streaming services over a 4-month of period, and also by integrating information from the logs with BGP routing information gleaned from multiple border routers on a tier-1 ISP. They find requests for content encoded at a higher bitrate dominate where high and low encoding rates are available, making up about 94% of the traffic. Windows Media streams account for more than 75% of all requests when the content is available in both Windows and Real formats. TCP based transport protocols dominate over UDP, with TCP being used in about 70% of all bytes transferred. Object popularly exhibits substantial skew with a few objects accounting for most of the load. A small percentage of IP addresses (or routing prefixes or origin autonomous systems (ASes)) account for most of the traffic demand across a range of performance metrics. Lastly, it is reported that streaming traffic exhibits regular daily patterns with high variability in terms of request, traffic volume and concurrent number of connections.

Mena and Heidemann [90] capture and analyze Real Audio traffic from five different audio traces at a commercial media site in March and Jun 1999. They find that 60-70% of audio flows use UDP for data while using TCP for control. Real Audio flow durations (mean duration of 78 minutes) are significantly longer than typical Internet Web flow durations. Lastly, Real Audio flows exhibit a significant amount of regularity in packet lengths, bit rates and inter-packet departures. It is reported that Real Audio data is sent at consistent bit rates at medium time-scales (tens of seconds), but at small time-scales (single seconds) it is best modeled as a bursty on/off source with off periods in multiples of 1.8 seconds.

Wang, Claypool and Zuo [130] collect and analyze the traces of RealVideo clips streamed to 60 clients, playing about 2800 video clips from 11 servers worldwide in June 2001. They find the average RealVideo clip streamed over the Internet has good quality, playing out at 10 frames per second and, aided by a large, initial delay buffer, with nearly imperceptible amounts of inter frame jitter. Second, while users connecting to the Internet with modems and/or slow computers still have their PC network connection as the video performance bottleneck, typical new computers connecting the Internet via DSL or Cable modems achieve even slightly better performance than corporate network connections to the Internet. This suggests that broadband connections from home users are pushing the bottlenecks for video performance closer to the servers. Lastly, nearly half of RealVideo flows use TCP to stream video data.

Li, Claypool and Kinicki [80] collect traces of video streams from RealNetworks RealServers and Microsoft MediaServers during March and April of 2002, and compare the two different video streams in terms of bandwidth, packet size, packet inter-arrival times, start-up buffering and frame rate. They find distinctly different behavior characteristics of RealVideo and MediaVideo streams and provide data for building more realistic streaming media simulations. They also find that the packet sizes and inter-packet arrival times of MediaVideo streams are typical of CBR flows, while those of RealVideo streams vary considerably more. High bandwidth MediaVideo traffic can have up to 80% IP fragmentation rates, while RealVideo traffic has none. Lastly, RealVideo streams are more elaborate during the initial buffering phase.

2.4 Transport Protocols for non-TCP Applications

This section introduces four types of TCP-friendly congestion avoidance mechanisms: General AIMD (window-based), rate-based AIMD, TCP model-based flow control and TCP emulation, which limits the rate at which TCP would transmit on average under the same network condition. These techniques can be used to build TCP-friendly transport protocols for non-TCP applications, such as delay sensitive media streaming applications, that otherwise may choose UDP instead of TCP for performance reasons. Also, this section introduces two approaches to modify TCP to enhance support for delay sensitive applications.

TCP uses a (1, 0.5) Additive Increase Multiplicative Decrease (AIMD) congestion avoidance mechanism that, during congestion avoidance, increases *cwnd* by 1 packet per window of packets acknowledged and decreases *cwnd* to 1/2 of the previous *cwnd* upon congestion. General AIMD (GAIMD) [131] denotes a general case (α, β) AIMD, where α is the additive increment step and β is the multiplicative decrement factor. [131] presents the average sending rate of an AIMD traffic hosts as a function of α, β, loss rate, mean RTT, mean RTO and the number of packets acknowledged by each ACK, which is shown to be quite accurate for loss rates up to 20%. In addition, [131] shows the relationship between α and β for an AIMD flow to be TCP-friendly, and reports that AIMD flows with $\alpha = 0.31$ and $\beta = 7/8$ deduces rate fluctuations compared to TCP flows.

The Rate Adaptation Protocol (RAP) [113] adapts AIMD to a rate-based congestion control mechanism. The RAP receiver acknowledges every packet received to allow the sender to measure RTT and lost packets. The RAP sender additively increases the transmission rate periodically (once in a RTT) until congestion is detected, in which case the sender decreases the transmission rate multiplicatively. To adapt the window-based flow control mechanism to rate-based, RAP also uses a mechanism for the sender to stop transmitting in the absence of feedback from the receiver. RAP does not support reliable transmission at its core and does not have retransmission timeouts (RTO) nor account for the impact of RTOs. Thus, RAP is TCP-friendly as long as the TCP's congestion control is dominated by the AIMD algorithm, but may be greedier than TCP during severe congestion.

TCP-Friendly Rate Control (TFRC) [44] is an equation-based congestion control mechanism for best-effort streaming multimedia build upon a stochastic TCP throughput model [96]. The TCP throughput model takes average RTT, steady-state loss event rate, RTO value and average packet size as input and approximates the upper bound of TCP's transmission rate. The TFRC sender maintains RTT and RTO values and obtains the loss event rate in feedback messages from the receiver at least once per RTT. Every time a feedback message is received, the sender calculates the new allowed sending rate using the TCP throughput model, and limits the actual sending rate to at or below the allowed rate. The TFRC receiver acknowledges packets received to allow the sender to measure the RTT. The receiver also measures the loss event rate and periodically notifies the sender of this rate. Possible limitations of model-based congestion control mechanisms like TFRC stem from limitations of the underlying TCP throughput models, which may not hold for some networking environments.

TCP Emulation At Receiver (TEAR) [114] emulates a TCP sender's flow control functions at each receiver to estimate a TCP-friendly rate for the congestion conditions observed in their forward paths, and notifies the estimated rate to the sender which limits the transmission rate to the reported rate. For the emulation, the TEAR receiver keeps track of *cwnd* at the receiver, and updates *cwnd* based on the arrival of packets (instead of ACKs) using a TCP window adjustment algorithm. A transmission session is partitioned into non-overlapping time periods, *rounds*, which contain roughly an arrival of the *cwnd* number of packets. At the end of each round, the emulated TCP rate is computed using the current *cwnd* and a current estimate of the RTT instead of the real-time duration of the round, and a weighted average TCP-friendly rate is updated. This average rate is reported at the end of a *feedback round* which is a parameter to the system (1 feedback round = n rounds), or the updated average TCP-friendly rate is less than the previously reported rate. Thus, the perceived rate fluctuations at the application are smoother than in TCP. Since TEAR shifts most of flow control mechanisms to the receiver, it can be extended to a multicast transport protocol. As a possible side benefit, the receiver-based mechanism may avoid problems associated with congestion in the reverse path and reduce the feedback frequency, although it is unclear how accurately TEAR receivers can estimate the RTT.

The next two approaches propose to make minor modifications to TCP to enhance the support for delay sensitive applications:

Goel et. al. in [48] identify that a significant portion of the TCP protocol latency occurs due to TCP's send buffer, and show that a simple send-buffer size adaptation mechanism can eliminate this latency without requiring changes to the TCP protocol. Their experiments show that the send-buffer size adaptation can reduce the average protocol latency to well within the interactive latency limits of approximately 200 ms [61] when the underlying network RTT is less than 100 ms (a typical coast-to-coast RTT in the US [59]) at a small expense in throughput. Yet, the send-buffer adaptation mechanism alone is not sufficient for interactive streaming applications, since instant TCP protocol latencies often spike well beyond 200ms due to packet drops and retransmissions. The authors support use of ECN as a means to significantly reduce packet drops and thus reduce the protocol latency spikes. In order to benefit from the TCP send-buffer adaptation approach, applications should use poll and non-blocking write calls on the sending side to avoid re-introducing the reduced latency at the application layer.

McCreary et. al. propose a receiver-centered TCP modification, called TCP-RC [87], to sacrifice the full reliability of TCP protocol for low protocol latency. TCP-RC modifies the TCP receiver to forge lost packets and pass them on to a TCP-RC enabled application to avoid significant delay in the stream due to TCP retransmissions. A TCP-RC receiver, on detecting a out-of-order packet sequence, sets a timer to expire three times the average packet inter-arrival time assuming that the missing packet, if not lost, will arrive within 3 additional packets. When the timer expires, the TCP-RC receiver generates three duplicate ACKs to notify the TCP sender of the packet loss event, and forges the lost packet. On receiving three duplicate ACKs, the TCP sender performs fast retransmission cutting the congestion window to half. Although TCP-RC tries to mimic TCP congestion control behavior as such, TCP-RC becomes TCP-unfriendly as network packet loss rate or RTT increases, mainly since TCP-RC avoids retransmission timeouts even when the TCP sender window is small.

Chapter 3

Background: Internet Congestion Control

This chapter first reviews the congestion control structure of the current Internet networking. Then, it discusses open issues in the areas of bandwidth control, congestion control and QoS control.

3.1 TCP/IP Networking

The Internet is a collection of interconnected networks that offers a data transmission service to users. Figure 3.1 shows a view of the protocol stack. Internet Protocol (IP) is a network layer (corresponding to layer 3 of OSI reference model) protocol [120] which provides the means to connect different OSI layer 2 networks such as Ethernet, TokenRing, FDDI, SONET, HDLC and ATM. Thus, the primary concern of the IP layer is to deliver packets from traffic sources to destinations across networks. IP is composed of senders, receivers and routers, each of which is associated with an unique IP address for each OSI layer 2 network it communicates on. An IP sender is

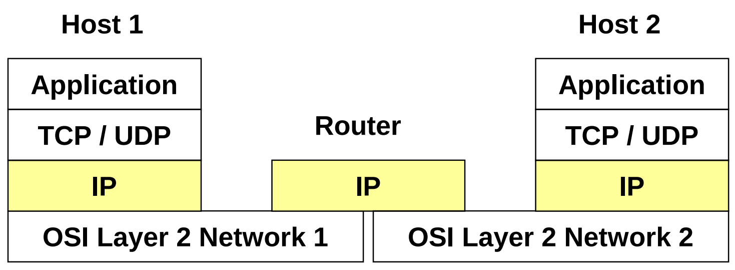


Figure 3.1: Layered View of IP Components

a lightweight protocol that packetizes the data segments from the layer above (OSI layer 4) and sends them to the layer below, and an IP receiver is the reverse of an IP sender. An IP router connects two or more OSI layer 2 networks and finds the proper outbound link to forward an inbound packet so that it might be delivered correctly to its destination.

Another important role of an IP router is to control network congestion, although very few routers in practice specifically address this issue. The network service provided by an IP network is often referred to as best effort since packets may or may not be delivered to the destinations. IP packets can be lost during their travel through the network due to transmission medium signaling errors, or more likely due to network congestion at IP routers. Figure 3.2 shows a simple model of an IP router. Typical IP routers today use a simple First-In-First-Out (FIFO) drop-tail outbound queue for each outbound link to absorb bursts of packets. However, when a router's outbound link gets a burst of packets that is beyond the limit of the queue capacity or consistently gets more packets than the outbound link can handle, the queue overflows and packets are dropped and lost.

Routers with FIFO drop-tail queues designed to handle traffic in bursts can only passively manage congestion by dropping incoming packets when the queue is full. Also, IP senders and receivers have no mechanism to detect network congestion or

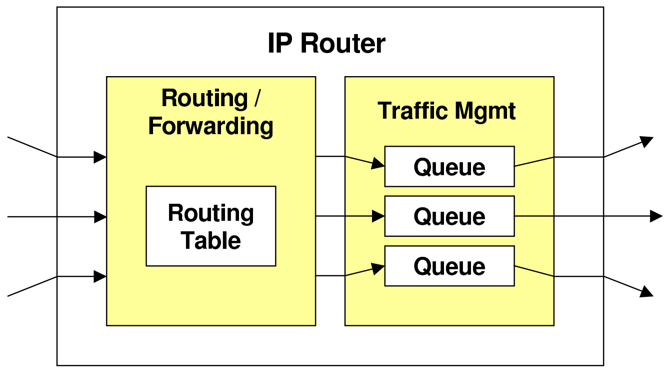


Figure 3.2: Simplified Model of An IP Router

control their packet transmission rates. Thus, IP networks have few mechanisms to control or prevent network congestion, and the well being of the Internet depends upon the behavior of upper layer traffic sources. The Internet collapse first observed in 1986 [65] illustrates the fragility of IP networks and the need for a proper congestion control mechanism. Shortly after the observation, the Internet research community devised a sender side (or source based) solution to monitor congestion and control transmission rates appropriately. However, this solution was invented not for the IP sender (OSI layer 3) but for a transport protocol TCP (OSI layer 4) that was and still is dominantly used among Internet applications.

There are two de facto transport protocols above the IP layer, the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP) [120]. UDP is a connectionless, unreliable transport protocol in which data is exchanged in discrete units called datagrams, which are similar to IP packets. In fact, the only features that UDP offers over raw IP packets are port numbers and an optional checksum. TCP offers a connection-oriented, reliable end-to-end data transportation service. TCP uses a sliding-window [107] mechanism to achieve reliable data segment delivery using acknowledgment (ACK), retransmission timeout (RTO) and retransmission, and to achieve flow control by negotiating the sliding-window size upon connection establishment. Although UDP was originally deployed by a few system level applications such as domain name service (DNS) for performance reasons, TCP dominated and still dominates the Internet.

When TCP was first widely available (BSD 4.2 TCP) in 1983, it did not have a proper congestion control mechanism. It merely had the sliding-window protocol and transmitted segments at the rate that the receiver window could handle in a very bursty manner as shown in Figure 3.3. Thus, the first generation of TCP was basically unresponsive to network congestion. Although BSD 4.3 TCP improved upon 4.2 with a RTO interval calculation method to behave better for longer delay networks, BSD 4.3 TCP was also unresponsive to network congestion and caused the Internet congestion collapse in 1986 [65]. The reaction from the research community was to add a congestion control mechanism to TCP leaving the IP layer untouched. This required a minimal change, such as applying a kernel patch, and allowed for gradual deployment. However, this did introduce the potential layer violation of 2 different layers (TCP traffic source and IP router) collaborating for congestion control.

The first TCP with a proper congestion control mechanism was introduced in 1988 and implemented in the BSD 4.3: TCP Tahoe version [65]. The basic objectives behind the Tahoe modification were twofold. One was to make TCP senders start transmission in a conservative but a fairly efficient manner. The other was to give TCP senders the capability to monitor network congestion and control the transmission rate accordingly by adjusting the sliding-window size. The Tahoe TCP congestion control mechanisms that realized these goals were *slow-start*, *congestion avoidance* and *fast retransmit*, which are the foundation of modern TCP/IP congestion control.

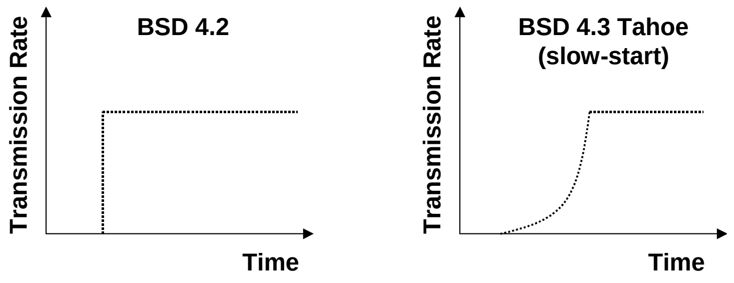


Figure 3.3: BSD 4.2 TCP vs. BSD 4.3 Tahoe (Slow-Start)

Starting from the initial window size of 2 segments,[1] *slow-start* increases the sender's sliding window by 1 segment for each acknowledgment (ACK) of new data received, which has the same effect of doubling *cwnd*[2] every round-trip time (RTT). Figure 3.3 shows congestion window growth from slow-start in an unconstrained bandwidth condition. This transmission startup behavior is specifically targeted to properly start TCP's "self clocking" mechanism [65]. Slow-start reduces the risk of queue overflow at IP routers in the path compared to the sudden traffic burst startup behavior of BSD 4.2 and early release of BSD 4.3. However, with the exponentially growing window, slow-start is also capable of increasing the transmission rate quickly to a target. Slow-start is used whenever a TCP source starts or restarts transmission in BSD 4.3 Tahoe TCP, although it is triggered only by RTOs in later versions.

As the aggregated TCP transmission rates grow beyond its service rate (or outbound link bandwidth), a congested router drops IP packets causing TCP segment losses. TCP senders detect the segment losses by monitoring ACK packets sent back by the receiver to signal successful or unsuccessful reception of segments transmitted. Modern TCP senders assume segment losses as an indication of network congestion and try to adapt to the available bandwidth using a *congestion avoidance* mechanism. TCP uses two segment loss detection methods, retransmission timeout

[1]The initial congestion window size was set to 1 segment in the old specification.

[2]The TCP sender's sliding-window is typically referred to as congestion window and its size will be denoted as *cwnd* hereafter.

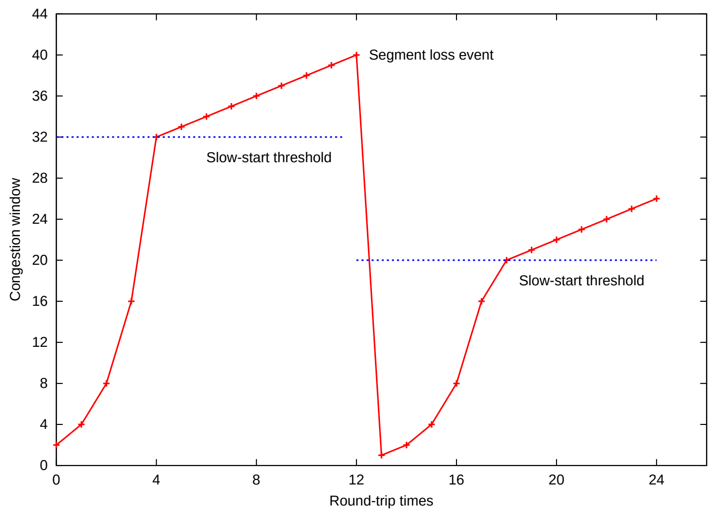


Figure 3.4: An Example of Slow-Start and Congestion Avoidance

(RTO) and triple duplicate ACKs, which will be discussed further shortly. The TCP sender in congestion avoidance mode increases the congestion window size ($cwnd$) by $1/cwnd$ for each ACK for new data, which has the same effect of increasing $cwnd$ by 1 segment for each RTT when there is no congestion, and decreases $cwnd$ by half on detection of a segment loss. This is the TCP implementation of the *Additive Increase Multiplicative Decrease (AIMD)* policy suggested in [66] targeted to achieve network stability. The TCP AIMD mechanism is often noted as (1, 1/2) AIMD, where 1 is the additive increase factor in terms of segments per RTT and 1/2 is the multiplicative decrease factor, and is the core mechanism that has been serving as the congestion safeguard of the Internet.

While the slow-start and congestion avoidance mechanisms are independent of each other and have completely different objectives, they are used in combination in all modern TCP implementations. However, since they both are triggered by a segment loss event and both manipulate $cwnd$, it is not trivial to how see how the two algorithms are combined. The following example illustrates the combination deployed by all modern TCP implementations. Note that the TCP sender, at any instance, transmits segments up to the minimum of *cwnd* and advertised window size of the receiver (often noted as *awnd*). The TCP sender maintains a new state variable called slow-start threshold (*ssthresh*) as well as *cwnd* to switch between slow-start and congestion avoidance algorithms. When a segment loss is detected, *ssthresh* is set to *cwnd* and TCP goes back to slow-start, setting *cwnd* to 1 segment. When new data is ACKed, the sender checks if *cwnd* is less than *ssthresh*. If so, *cwnd* is increased in slow-start ($cwnd$ += 1 *segment*), or in congestion avoidance ($cwnd$ += $1/cwnd$). Figure 3.4 shows an example behavior of the combined algorithm assuming *ssthresh* is initialized to 32 segments and receiver's window is larger than 40 segments.

As briefly mentioned above, TCP has two ways of detecting segment loss at the sender. Retransmission time out (RTO) is the most basic but inefficient method of detecting the loss because of the difficulty in finding an optimal RTO interval. The RTO method is especially expensive on the transmission startup phase as the initial RTO interval is set to a large value, typically 3 seconds [15]. As a means of overcoming the inefficiency of the RTO method, *fast retransmit* [2] was invented to use the arrival of 3 duplicate ACKs (4 identical ACKs without the arrival of any other intervening packets) as an indication of a segment loss due to congestion. Upon receiving 3 duplicated ACKs, the sender retransmits what appears to be the missing segment without waiting for a RTO. After the fast retransmission, BSD 4.3 Tahoe TCP reduces *cwnd* by half (during congestion avoidance) and restarts transmission using slow-start. The loss detection mechanism of fast retransmit can significantly improve TCP throughput by avoiding RTOs when the congestion window is large enough to generate 3 duplicate ACKs for a lost segment. Otherwise, RTO dictates the performance of the TCP transmission in the presence of loss.

So far, we looked into the 3 main mechanisms that characterized the congestion control behavior of modern TCP, which were available through BSD 4.3 Tahoe

TCP. In 1990, an additional algorithm referred to as *fast recovery* was proposed to improve the post fast retransmit behavior of Tahoe TCP that restarts transmission using slow-start. Slow-start is designed to start TCP's ACK paced 'self clocking' transmission without knowing network bandwidth or delay, and can be over-conservative for restarting transmissions after fast retransmit. Fast recovery was implemented with the other 3 congestion control mechanisms in BSD 4.3 Reno TCP [2, 42]. After a missing segment is re-sent by the fast retransmit algorithm and *cwnd* is reduced by half by the congestion avoidance algorithm, the fast recovery algorithm governs the transmission of new data until the arrival of a non-duplicate ACK. During fast recovery, the sender inflates its congestion window (*cwnd*) by 1 segment for each duplicated ACK received, and transmit new segments when *cwnd* allows. In other words, the sender, after retransmitting and reducing *cwnd* to half, waits until *cwnd* duplicate ACKs are received, and then sends a new segment for each additional duplicated ACK received. This sender congestion window inflation mechanism is targeted to preserve ACK "clocking" [65] during the recovery stage, so that the sender can continue to transmit new segments eliminating the need to use slow-start when the recovery process is over. Upon receipt of a non-duplicate ACK (a "recovery ACK") informing that the lost segment is recovered, the sender deflates the congestion window and exits fast recovery.

TCP Reno's fast recovery mechanism significantly improves the performance over Tahoe TCP for a single segment loss from a window of data (or a single loss in a RTT). However, Reno performs poorly for multiple segment losses in a window of data due to lack of outstanding segments that can generate 3 duplicate ACKs (for example, for the second lost segment after the recovery of the first lost segment) causing a RTO. In 1996, modification to TCP Reno's fast recovery mechanism was proposed to address the RTO problem, and implemented in a new TCP version

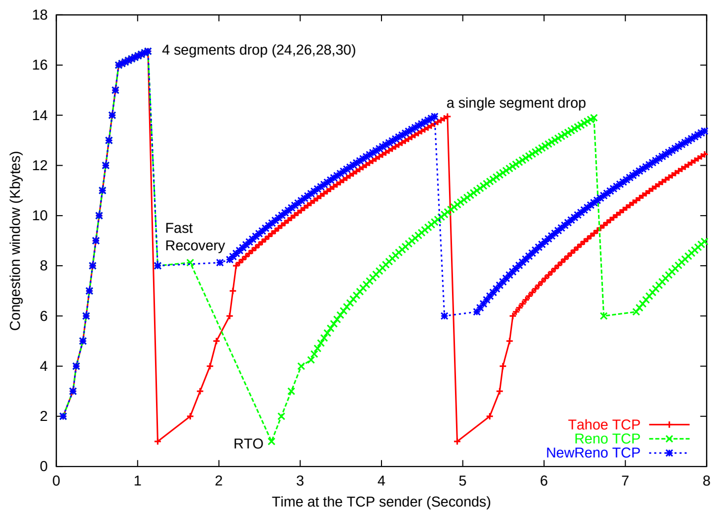


Figure 3.5: Comparison of Tahoe, Reno and NewReno TCP Behaviors

referred to as NewReno [39, 42].[3] The extended fast recovery algorithm of NewReno assumes a *partial* ACK received during the recovery phase as an indicator that the segment immediately following the acknowledged segment in the sequence space has been lost and needs to be retransmitted. Thus, a NewReno TCP sender tries to recover all the lost segments outstanding when fast recovery was initiated before returning back to congestion avoidance by retransmitting one lost segment per RTT until all of the lost segments from the window have been retransmitted.

Figure 3.5, obtained via a simple simulation using NS [127], illustrates the *cwnd* behavior of Tahoe, Reno and NewReno TCP on a single loss and multiple losses (4 in a window), where the initial *ssthresh* is 16 segments (1 segment = 1 Kbyte). Figure 3.5 shows that fast recovery can improve transmission restarting behavior (over Tahoe) for single segment loss per RTT as targeted. Yet, Figure 3.5 also shows the negative impact caused by the fast recovery of Reno TCP for multiple losses in

[3]Discussion on the Selective ACK (SACK) approach [86] that also proposes to address the problem Reno faces is omitted, since SACK is not used in practice because it requires both end-hosts to be SACK-enabled. According to [97], a large percentage of hosts (more than half) advertise SACK, but only 6% of hosts use SACK correctly. Other SACK deployment statistics can be found at http://www.icir.org/floyd/sack-questions.html.

a window of data. NewReno significantly improves the fast recovery of Reno (see 1 to 3 seconds in Figure 3.5). However, the benefit from the fast recovery of NewReno over Tahoe is not trivial for multiple losses in a window of data as the recovery time increases by an RTT for each loss.

More importantly from the congestion control standpoint, Figure 3.5 exhibits two congestion responsiveness characteristics of modern TCP implementations. First, regardless of the version, TCP's congestion window growth is bounded by the (1,1/2) AIMD bandwidth adaptation algorithm. Second, except for the TCP Reno senders, TCP's multiplicative decrease is triggered by a loss event rather than individual segment losses, where a loss event can be defined as one or more segment losses in a window of data (or a RTT). These characteristics are typically used along with the RTT to model TCP throughput as a function of network packet loss rate (p) and average RTT [96]. A TCP throughput model can be used to determine the TCP friendliness of non-TCP flows [43], design a rate-based TCP-friendly traffic source [44] or help in designing an efficient router queue management mechanism [38, 58, 69].

[98] reports the distribution of TCP versions used by 4550 Web servers as of May 2001. Among the tested servers, about 41% were using NewReno TCP (35%) or an equivalent version called RenoPlus (6%). About 15% were using Reno, 4% were using Tahoe and 22% were still using TCP without fast retransmit. This gives a statistic that about 82% of the Web servers are using a TCP version with a congestion responsiveness behavior dominated by (1,1/2) AIMD leaving 18% using unknown versions of TCP. Since the majority of TCP connections are HTTP connections to Web servers [122], the TCP version distribution report verifies that the congestion control mechanism of current TCP/IP networking is the source-based (1,1/2) AIMD bandwidth adaptation algorithm.

This section explained the congestion control structure of the current Internet, where the IP layer addresses congestion control passively by dropping incoming packets when the router queue is full. The vast majority of congestion control mechanisms in the Internet is the (1,1/2) AIMD congestion avoidance mechanism used by modern TCP. The Internet research community is currently enhancing IP router queue management mechanisms to better handle network congestion in cooperation with responsive traffic sources. Before introducing specific router side approaches and mechanisms, the next section discusses issues in TCP/IP networking and relevant congestion control problems.

3.2 Congestion Control Issues

Congestion at an IP router is controlled by multiple distributed TCP traffic sources independently. TCP uses a window-based transmission mechanism in which the transmission rate depends upon the RTT and *cwnd*. The *cwnd* is affected only by the network packet drop rate (p) [96]. Congestion control issues describe the potential performance drawbacks of current TCP/IP congestion mechanisms, and suggest ways to improve the congestion feedback control mechanism at the end-hosts and routers. Typically, many studies in this area involve Active Queue Management (AQM), especially the Congestion Controller, discussed in Chapter 2.1. Congestion Controllers offer advanced congestion detection and signaling methods at IP routers to better handle network congestion with the help of TCP-like cooperative traffic sources.

Global synchronization of TCP connections decreasing their window at the same time is one of the traditional TCP/IP congestion control issues brought up in [51] and first attempted to be addressed in [45]. Global synchronization typically affects the utilization of a router's outbound link when a relatively small number of

TCP connections are sharing it, and is often seen at routers with a FIFO drop-tail queue mechanism. [45] tried to avoid global synchronization by introducing a new router queue management mechanism called Random Early Detection (RED) which actively monitors incoming congestion and drops incoming packets randomly with a probability (p) proportional to the estimated congestion level, as determined by a weighted average queue length. Insisting on a low random drop rate, RED could manage the global synchronization problem. However, RED, at saturation, revealed other performance drawbacks such as more bursty packet drops than in a typical FIFO drop-tail queue [12] for a wide range of traffic loads. Although not practical for deployment, RED is important since it was among the first and well-known IP router traffic (or queue) management techniques (or AQM) designed to actively control congestion, which directly and indirectly brought up important TCP/IP congestion control issues.

Recently, [69, 84] showed that even applying random drops with low probability will not completely solve the link utilization problem related to flow synchronization, especially for links with a very large capacity (a couple hundred Mbps to Gbps) and a relatively small outbound queue. When a small number of TCP connections are sharing the bandwidth of a high capacity link with a relatively small outbound queue, they cannot fully utilize the link capacity due to too large a packet drop rate bounded by the queue limit and the inefficient bandwidth probing/adaptation of TCP. That is, when a packet drop forces a TCP sender to reduce its transmission rate by half, the slow bandwidth probing during TCP congestion avoidance cannot consume the released bandwidth before the queue drains, which results in a lower link utilization. Although it is possible to fully utilize the high capacity link using an extremely large buffer and by lowering the drop probability, [69, 84] illustrates the inherent limit of aggregate (1,1/2) AIMD bandwidth adaptation. The inability of TCP to fully utilize the available bandwidth needs to be addressed for high bandwidth business-to-business communications such as Web proxy server updates. However, it is out of scope of typical consumer side congestion control solutions, where issues are the efficient control and fair allocation of insufficient network resources.

Another inherent congestion control issue of the current Internet is the tradeoff between queuing delay and data (packet) loss rate at a congested router. This delay-loss tradeoff was implicitly suggested in [21] and theoretically verified in [38] using a notion of a "queue law". The queue law, which can be derived from a TCP throughput model, defines the relationship between random packet-drop congestion notification probability and the stable state average queue length of a congested router. Given an offered TCP traffic load and a drop notification rate of the congested router, the queue law gives a stable state queuing delay that makes the aggregated TCP throughput equal to the service rate (or link bandwidth). The queue law, in which queuing delay is roughly inversely proportional to the square root of the drop notification probability, suggests that there is a higher packet drop rate for a lower queuing delay to achieve the same throughput. This subtle delay-loss tradeoff implies that an AQM will incur a higher packet loss rate than FIFO drop-tail queue management for a lower queuing delay. Thus, the benefit of AQM over the drop-tail management may not worthwhile given the complexity of typical mechanisms.

The loss rate and queuing delay tradeoff resulting from the implicit drop congestion signaling of the current TCP/IP network disappears when Internet routers use explicit congestion notification (ECN) [25, 58]. ECN [109] uses binary congestion signaling, where a congested router can set a bit in an IP packet header to signal an ECN compliant TCP receiver of the network congestion. The notified receiver, in turn, signals the TCP sender by setting the ECN-Echo flag in the ACK segment. On receiving an ECN-Echo ACK segment, the TCP sender reacts as if it detected a segment loss. That is, the sender in congestion avoidance mode halves the congestion window (*cwnd*) and reduces the slow start threshold (*ssthresh*), but does not react to congestion indications (multiple ECN-Echo ACKs) more than once every window of data (or RTT). By randomly marking packets instead of randomly dropping packets to notify TCP sources of congestion, packet loss due to congestion signaling is avoided. In other words, the new queue law for ECN enabled TCP/IP networks [25] defines the relationship between random marking probability and stable state queuing delay that is independent of packet loss rate, implying that a well-designed ECN-enabled AQM can achieve high utilization with both a low queuing delay and a low loss rate at the same time. There is growing interest in the ECN deployment within the Internet community [109] since ECN can be relatively easily and gradually deployed while offering great potential benefits when used with AQM.

The last congestion control issue of TCP/IP networking discussed in this section is the fairness in distributing congested link bandwidth to TCP flows with different round trip times (RTT). Assuming a single congested router with a certain packet loss rate, TCP connections with longer RTTs (fragile flows) will have lower throughput than TCP connections with lower RTTs (robust flows). This fundamental fair bandwidth distribution problem can be addressed either at the TCP traffic sources or at the IP routers.

TCP congestion avoidance mechanism can be modified to allow connections with longer RTTs to more aggressively probe for bandwidth by adjusting the additive increase steps [54]. TCP Westwood [18] is another sender side approach that modifies TCP congestion avoidance mechanism to utilize RTT information in order to improve system fairness. A closely related end-system congestion control topic gaining attention recently is modification to the TCP congestion avoidance mechanisms to better adapt to high speed networks thus increasing TCP throughput on a high capacity link [40, 84, 99].

In parallel with approaches to enhance TCP fairness, end-host approaches are made to develop *TCP-friendly* transport protocols for non-TCP applications such as interactive and continuous media applications. General AIMD (GAIMD) [131], TCP-Friendly Rate Control (TFRC) [44], TCP emulation at receivers (TEAR) [114] are TCP-friendly transport mechanisms targeted to achieve smoother transmission rate fluctuations than TCP. Yet, it is not clear that these mechanisms are practical for the use by media streaming applications.

Router side approaches to improve fairness among TCP and/or TCP-Friendly flows estimate and apply an appropriate congestion notification probability for each flow such that the system fairness can be achieved by the cooperative traffic sources. For example, an AQM may use per-flow traffic information such as RTT to determine the fair per-flow congestion notification probability [79]. One drawback of this approach is that it requires network architectural support for routers to securely obtain per-flow traffic information. An alternative approach to approximate fair per-flow congestion notification probability without demanding per-flow traffic information is introduced in Stochastic Fair Blue (SFB) [36]. SFB uses Bloom filters to estimate a proper congestion notification probability for each flow.

Approaches to achieve system fairness via per-flow congestion signaling at routers have no ability to protect system fairness from non-TCP flows. The use of UDP has been increased with the use of continuous media and/or delay sensitive Internet applications such as interactive streaming and online games, because these relatively new applications often are not able to achieve their targeted performance using TCP due to its transmission characteristics. As UDP traffic is expected to significantly grow in the near future, protection of the Internet and TCP flows from misbehaving or unresponsive non-TCP flows becomes an important issue. A closely related issue is the protection of the network from Denial of Service (DoS) attacks such as UDP flooding. Although support for lightweight UDP is essential for efficient network management and gives applications freedom in utilizing the network, it can be a security hole for Internet congestion control. Thus, it is believed that the network should provide protection although it is arguable how and to what extent the protection should be enforced due to the price-performance tradeoff. The bandwidth fairness protection issues are addressed in AQM approaches within the context of the Bandwidth Controller discussed in Chapter 2.1.

3.3 Support for Diverse QoS

One of the major challenges that the Internet faces today is to concurrently support diverse Quality of Service (QoS) requirements of various applications. This section first examines the QoS requirements and characteristics of various Internet application domains, focusing on World Wide Web (WWW) browsing, streaming media applications, and network games. Then, trends are identified in the various QoS requirements and traffic characteristics, and discuss the possibility of supporting the diverse QoS requirements using AQM and without deploying a Differentiated Services (DiffServ) [8] architecture.

One of the most dramatic events in Internet history was the advent of World Wide Web (WWW). The Hypertext Transfer Protocol (HTTP) [6, 37] that uses TCP as the underlying transport carrier, is an application layer protocol used by the WWW to manage the globally distributed information system communication and Web object transmission.

One of the major impacts of the Web on the Internet traffic is the creation of "flash crowds", a phenomenon that a large volume of traffic that comes and goes in a burst manner. For example, the France 98' World Cup Web site[4] was popular during the 1998 FIFA (Federation Internationale de Football Association) World Cup game getting 1.35 billion requests during the tournament, reaching up to approximately ten million request per hour during the matches [4]. The resulting traffic heavily burdened nearby routers and gateways. The performance of the Internet today depends on the ability of IP routers to efficiently handle sudden such increases in the volume of Web traffic while meeting the QoS needs.

Web browsing is moderately interactive with more strict QoS requirements compared to FTP or Email. The performance of Web browsing is sensitive to HTTP response time, which is affected by both the end-to-end delay and packet loss rate of the network, primarily by the latter for short-lived HTTP/TCP transmissions that take a significant portion of Web transmissions [49]. Traditional Internet applications such as FTP and Email are not as sensitive to the end-to-end delay or the network packet loss rate. For example, the response time requirement for a FTP transfer is on the order of minutes and that of an Email transmission is on the order of tens of minutes. Thus, the Internet had no strict QoS requirements to meet when FTP and Email applications were dominant, but now must meet the stricter response time requirements of Web browsing.

Another relatively new but fast growing Internet application domain which has unique QoS requirements is *media streaming* applications that transmit audio and video that is played out at a specific rate. Media streaming applications can be categorized into *interactive streaming* such as Internet phone and video conferencing and *non-interactive streaming* such as audio jukebox and video on demand (VOD). Both interactive and non-interactive streaming applications share many characteristics and QoS requirements except that interactive streaming applications have more

[4]http://www.france98.com

strict delay and transmission timing requirements than do non-interactive ones. Non-interactive streaming applications are not very sensitive to delay introduced by the network, since a fairly large initial playout delay is acceptable in most cases and the relatively large delay buffer can smooth out most of the timing related issues.

In general, streaming media applications are less sensitive to data loss than are other Internet applications, since a little data loss in the streamed media can be repaired without noticeably degrading the user perceived quality of the stream [11, 50, 82, 95, 102, 104]. Rather, streaming applications are sensitive to the available network bandwidth as streaming media over a network with less available bandwidth than is required will incur either a periodic transmission delay resulting in a non-continuous media playout or an unreasonable data loss rate significantly degrading the user perceived quality of the stream. Therefore, most of the streaming applications today use *media scaling* to adapt to changing network conditions by estimating the available network bandwidth and selecting a version of the encoded media from different quality versions to best fit the estimated bandwidth [9, 32, 53, 74]. This improves the overall quality of the stream on a condition that the application sender (or receiver) is able to efficiently estimate or at least not overestimate the available bandwidth.

Streaming applications prefer to use UDP because it is harder to efficiently estimate the available bandwidth over TCP since it hides detailed network information such as network packet loss rate. Also, the fact that TCP offers only fully reliable data transmissions discourages streaming applications from using TCP. For example, a streaming sender cannot force TCP to drop frames in the transmission buffer even if they become useless due to a transmission delay during network congestion and get in the way of subsequent frame transmissions. TCP is especially not a suitable transport protocol for delay sensitive interactive streaming applications, since unwanted

TCP retransmissions can add a significant amount of end-to-end transmission delay and jitter.

Another fast growing Internet application domain that often uses UDP is *network games*, where the most popular genres of games are First Person Shooter (FPS) and Massively Multiplayer Online Role Playing games (MMORP), followed closely by Real Time Strategy (RTS) games [29]. Having slightly different QoS requirements, networked games in general have strict delay requirements especially for the highly interactive FPS. Also, the games are typically targeted to work on a low bandwidth ($< 56Kbps$) modem connection. Thus, the extra transmission delay and the packet acknowledgment overhead introduced by TCP can cause slowdown in a game, and the lightweight, unreliable and congestion-unresponsive UDP is often preferred. UDP gaming traffic is not likely to respond to network congestion and behavior is significantly different from game to game [29].

Although the bandwidth consumed by individual connections may be small, the volume of online game traffic has increased over the years [88], and is expected to grow further as the latest generation of consoles systems from Sony and Microsoft all include network support for multiplayer game play on the Internet. An evidence that gaming traffic indeed occupies a significant portion of UDP traffic is shown in [88] which reports about 20% of UDP traffic (in terms of bytes) of the AIX backbone in February 2000 was gaming traffic while 24% was streaming traffic, 23% was DNS traffic and 33% of the UDP traffic was not identified.

Table 3.1 summarizes the QoS requirements, characteristics and proportional network (AIX backbone) usage of the Internet applications discussed in this section, which reveals a couple of interesting trends. First, the delay QoS requirements are tightened as more interactive Internet applications are used. Second, various Internet application domains impose the diverse QoS requirements. Third, HTTP traffic is

Internet Applications	Sensitivity to			Characteristics		Traffic Ratio (Bytes)*
	Delay	CNR	BW	BW Usage	Duration	
FTP (TCP)	low	low	low	high	mid	7%
Email (TCP)	low	low	low	low	short	16%
Telnet (TCP)	high	high	low	very low	long	0%
HTTP (TCP)	low	high	mid	low/mid	short/mid	61%
Jukebox/VOD (UDP)	low	mid	high	low/mid	long	<15%
A/V Phone (UDP)	high	mid	high	low/mid	long	
Video Game (UDP)	high	mid	high	low $< 56k$	long	
DNS (UDP)	high	high	low	very low	very short	1%

Table 3.1: Internet Applications: QoS Requirements and Characteristics (CNR = Congestion Notification Rate, BW = Bandwidth changes). **Traffic Ratio (Bytes) is based on February 2000 AIX backbone trace data in [88]*

consuming more than half of the all bytes transferred.

When applications that are not very sensitive to delay or congestion notification probability (CNP) were dominant, meeting the required QoS was relatively easy. As more Internet applications with strict QoS requirements became popular, Internet congestion control became harder due to the strict QoS requirements and the diversity of QoS requirements for various application domains to meet concurrently. The current Internet offers a *single-class*, *best-effort* packet delivery service, where IP routers are typically configured to maximize throughput over delay, and can fail to meet the QoS requirements of delay sensitive applications.

Our observation on the delay QoS requirements indicates that the Internet support for diverse QoS requirements of different application domains can be significantly improved by deploying Active Queue Management (AQM) at routers to minimize queuing delays. In other words, a router in congestion can almost eliminate delay QoS issues by minimizing queuing delay at the price of a small link utilization loss. Furthermore, by using ECN, IP routers can significantly reduce packet drops, which improve TCP throughput and system goodput avoiding unnecessary retransmissions and retransmission timeouts that degrade HTTP response time for short-lived HTTP/TCP transmissions. Thus, there is a potential to support the diverse QoS requirements by gradually deploying delay optimized AQM with ECN instead of restructuring the Internet service architecture to support multiple differentiated classes of services [8].

Chapter 4

Crimson: AQM Support for Streaming Media

This chapter presents design, configuration and evaluation of two IP router traffic management mechanisms that are the building blocks of Crimson. Crimson's goal is to offer best-delay-effort Internet service with affordable fairness protection from misbehaving traffic. Section 4.1 presents the Crimson Congestion Controller, called Aggregate Rate Controller (ARC). ARC minimizes outbound queuing delay while achieving a high link utilization by efficiently estimating and signaling impending congestion to congestion responsive traffic sources like TCP. Section 4.2 presents the Crimson Bandwidth Controller, called Stochastic Fairness Guardian (SFG). SFG uses a statistical traffic filter to limit the network usage of misbehaving flows.

4.1 Aggregate Rate Controller for Low Delay Packet Switching Network

TCP, the de-facto Internet transport protocol, has an end-host congestion control mechanism that has largely been effective in managing Internet congestion. Yet, TCP alone can be inefficient in controlling congestion, mainly since end-hosts typically must wait until router buffers overflow before detecting congestion. Active queue management (AQM) with explicit congestion notification (ECN) [109] promises to overcome the limitations of end-host only congestion control by providing congestion feedback information to the end-hosts before router buffers overflow.

The most promising approaches model AQM as a feedback controller on a time-delayed response system and apply control-engineering principles to design an efficient controller for TCP traffic [47, 58, 75]. In modern control systems, proportional integral derivative (PID) designs dominate due to their simplicity and effectiveness. Without exception, this applies to recent active queue management developments and has altered AQM research from basic framework design into detailed controller design and practical implementation issues.

Among PID principles, only the proportional integral (PI) feedback control approach is primarily considered for AQM since the effect of the derivative control is often insignificant under practical Internet environments. While the PI control approach seems promising, a critical deployment challenge is the configuration of PI control parameters in a time-delayed feedback system, (i.e., the Internet); there are no simple and effective PI control parameter configuration available for time-delayed system [118]. The existing PI control-based AQM mechanisms such as the PI controller [58] or Adaptive Virtual Queue (AVQ) [75] lack complete configuration guidelines, making their practical deployment difficult.

We address the practical configuration issue by carefully reducing the PI parameters and find a complete configuration guideline for the parameter reduced PI controller, called Aggregate Rate Controller (ARC). We model a TCP-ARC feedback control system using a linear TCP model [58] and develop practical yet effective ARC configuration guidelines. The guidelines cover issues in choosing a target stable boundary system for ARC configuration and provide a method for selecting control parameters that help avoid system instability even when the system is out of the stability boundary. The guidelines also address the effects of the control information sampling interval on system stability, a consideration often neglected in other AQM studies.

Controllers with the same underlying principle design can result in noticeably different implementations, both in terms of complexity and performance depending on the way in which control information is obtained and feedback is processed. AQM requires information on the incoming traffic load to make accurate congestion control decisions, which can be obtained in two different ways: derivation of queue samples or incoming traffic rate over the service rate. ARC takes a rate-based control information acquisition approach. For a small amount of data collection overhead, rate-based data acquisition reduces sampling noise that can significantly degrade the accuracy of congestion measurement. In addition, rate-based mechanisms can more effectively react to impending congestion making control decisions before outbound queue buildup, and thus can minimize queuing delay while achieving a high link utilization enhancing support for the quality of service (QoS) needs of various Internet applications.

Through an extensive simulation study, we evaluate ARC and compare it with similar AQM mechanisms including the PI controller [58], AVQ [75] and SFC [47], and drop-tail queue management over a wide range of network and traffic conditions including Web flash crowd and multiple bottleneck cases. Our simulations validate the stability and practicality of ARC, showing that ARC efficiently handles network congestion in all the tested traffic conditions, and when considering all traffic scenarios, outperforms the other mechanisms considering queuing delay, link utilization, data loss rate, and object response time for Web traffic.

In addition to TCP support, ARC can be extended to concurrently support FAST [99] traffic. FAST is a recent non-TCP congestion control mechanism proposed for high bandwidth-delay product networks to overcome the inefficiency of using TCP. The most significant advantage of FAST over other non-TCP mechanisms such as eXplicit Congestion Protocol (XCP) [69] is that FAST can be deployed in the current Internet without modification to IP specifications. Noting that the router traffic management behavior required for FAST is fundamentally the same as that of active queue management, we make a simple modification to ARC to concurrently support FAST traffic as well as TCP traffic.

The rest of section is organized as follows: Section 4.1.1 describes the design of ARC; Section 4.1.2 discusses issues with configuring ARC and provides practical guidelines for ARC configuration; Section 4.1.3 evaluates ARC performance through simulation and compares ARC to several popular Congestion Controllers; Section 4.1.4 shows how ARC can be extended to seamlessly support FAST as well as TCP; and Section 4.1.5 presents summary and possible future work.

4.1.1 Design

In modern control systems, proportional integral derivative (PID) designs are the most widely used for feedback controllers because of their simplicity and effectiveness, and have recently been applied to active queue management. The PI controller [58] adapts a proportional-integral (PI) controller into a TCP congestion control system using a linear TCP behavioral model, and converts the continuous time-domain PI control function into an algorithmic implementation through discretization. The PI controller can be configured to support a wide range of traffic conditions. However, it is still not mature enough for practical deployment due to some configuration and stability concerns. The first concern is the configuration complexity. Configuration of PI involves determining at least two parameters, a proportional parameter (K_P) and an integral parameter (K_I), in order to work effectively over a wide range of traffic conditions. There is a guided, but still complicated, process to configure a PI controller [118] but lack of expert knowledge may leave the controller ineffective. Second, the discretization of the continuous time-domain control function introduces a state measurement interval such as the queue-sampling epoch in the PI controller that also affects the stability of the system. However, this measurement interval is neglected in the stability analysis adding uncertainty to the already complicated controller configuration issues. Lastly, a small but fundamental stability concern for recent TCP model-based AQM Congestion Controller designs [47, 58, 75] is that a linear TCP model is used for the stable control parameter range analysis without validating the linearity of the TCP system. That is, the linear models obtained from the non-linear stochastic TCP behavioral model may not accurately represent the behavior of the actual TCP congestion control system.

Based on sound understanding of PI control for Internet traffic and stability analysis using a linear TCP model, this section explores the possibilities of reducing the PI configuration parameters for active queue management, and presents the ARC logic for a configuration-optimized PI control algorithm for TCP congestion control systems. First, we design a rate-based algorithm that implements the general PI traffic controller algorithm in [58], enhancing the traffic rate estimation mechanism to provide an efficient and flexible controller configuration. Then, we carefully reduce

Algorithm 1 Rate-Based PI Controller for AQM

```
Every $d$ seconds (epoch):
1: $p \leftarrow p + \alpha(b - d\gamma C) + \beta(q - q_0)$;
2: $b \leftarrow 0$;
Every $packet$ arrival:
3: $b \leftarrow b + sizeof(packet)$;
4: $notify(packet, p)$;
Variables:
  $p$: congestion notification probability
  $q$: queue length in bytes
  $b$: total bytes received this epoch
Parameters:
  $C$: link capacity (bytes per second)
  $\gamma$: target link utilization ($0 < \gamma \leq 1$)
  $q_0$: target queue length in bytes
  $d$: measurement interval
  $\alpha$: virtual queue control constant
  $\beta$: queue control constant
```

the control parameters to obtain the ARC logic. Next, we model the ARC dynamics into a differential equation and use it along with the linear TCP behavior model from [57] to perform TCP-ARC system stability analysis that also shows the effect of the state measurement interval on the system stability. Lastly, we validate the linearity of the TCP system as well as correctness and usefulness of the ARC-TCP system model through simulation.

Algorithm 1 shows our initial rate-based enhancement of the queue-based PI controller algorithm from [58], which adds support for QoS configuration by introducing an additional target link utilization parameter (γ) in the PI control logic at *line 1*. When $\gamma = 1$, the rate-based PI control logic is identical to the queue-based control logic except for the incoming traffic amount measurement. However, when $0 < \gamma < 1$, the PI control logic can be regarded as maintaining a virtual link as in AVQ [75] and making control decisions proportional to the virtual queue length, where the virtual link capacity is dynamically adjusted to handle the traffic still present in the physical queue due to control error from the previous measurement

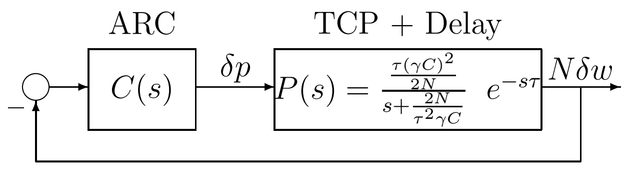

ARC
TCP + Delay
$C(s)$
δp
$P(s) = \frac{\frac{\tau(\gamma C)^2}{2N}}{s+\frac{2N}{\tau^2\gamma C}} \; e^{-s\tau}$
$N\delta w$
$-$

Figure 4.1: TCP-ARC Feedback Control System with Transmission Delay

interval (epoch). Thus, this PI first reserves a portion of the physical link capacity proportional to the queue displacement from the targeted length $(q - q_0)$, and then determines the virtual capacity used to control aggregated traffic for the next epoch based on γ and the remaining physical capacity. This view of the PI control behavior is clearly shown when rewriting the logic in the following form:

$$p \leftarrow p + \alpha(b - \gamma(dC - \tfrac{\beta}{\gamma\alpha}(q - q_0)))$$

When the overfilled queue draining capacity reserved is proportional to the queue displacement, we immediately see that the necessary range condition for the proportional queue control parameter is:

$$0 < \tfrac{\beta}{\gamma\alpha} \leq 1$$

since the controller must reserve no more capacity than needs to be served for the upcoming epoch. We now consider the most conservative case of setting the proportional queue control parameter to one (i.e. $\beta = \gamma\alpha$). Then, the PI control logic is reduced to the following ARC logic:

$$p \leftarrow p + \alpha(b - \gamma(dC - (q - q_0))) \tag{4.1}$$

Next, we model the dynamics of the ARC logic for stability analysis. Figure 4.1 shows the block diagram of TCP-ARC feedback control system modeling N TCP sources and a single congested ARC router using the linear TCP model from [57], where τ is round trip time the system, C is the capacity of the congested link, and w is the expected TCP window size in packets given a congestion notification probability of p from the system. The system model also uses delta (δ) notation for each system variable to express the displacement from the equilibrium state. Thus:

$$\begin{aligned}\delta p &= p - p_0\\ \delta w &= w - w_0\\ \delta q &= q - q_0\end{aligned}$$

where, p_0, w_0 and q_0 are the values of corresponding system variables at system equilibrium (assuming equilibrium exists). Using the number of packets received ($Nw(t)$) instead of the number of bytes received (b) to be compatible with the TCP model we use, and the unit of link capacity (C in packets per second), the difference equation that models the dynamics of the ARC logic is:

$$\triangle p = \alpha(Nw(t) + \gamma(q(t) - q_0) - d\gamma C) \tag{4.2}$$

The ARC difference equation is transformed to use delta notation in order to adapt to the input and output of the TCP model given in delta format, and then converted into a differential equation:

$$\begin{aligned}\triangle\delta p &= \delta p - \delta p_{prev} = (p - p_0) - (p_{prev} - p_0) = \triangle p\\ &= \alpha(Nw - Nw_0 + \gamma(q - q_0) + Nw_0 - d\gamma C)\\ &= \alpha(N\delta w + \gamma\delta q + Nw_0 - d\gamma C)\\ \delta\dot{p} &= \lim_{\triangle t\to 0}\frac{\triangle\delta p}{\triangle t}\\ &= \lim_{\triangle t\to 0}\frac{\alpha(N\delta w + \gamma\delta q + Nw_0 - d\gamma C)}{\triangle t}\end{aligned}$$

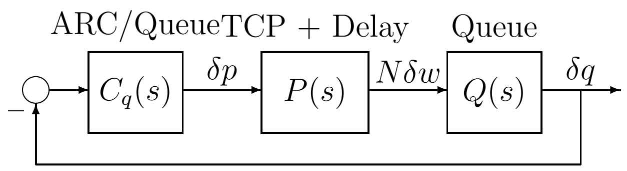


Figure 4.2: TCP-ARC System with the Queue Model Removed from the ARC Transfer Function

$$\approx \frac{\alpha}{d}(N\delta w + \gamma\delta q + Nw_0 - d\gamma C) \tag{4.3}$$

Note in the last step of the above discrete to continuous time domain conversion the traffic state measurement interval (d) uses an approximation for $\lim_{\triangle t\to 0}\triangle t$. This approximation, also used in the discretization processes, is valid for sufficiently small d. Note also that the ARC differential equation has a control parameter represented by $\frac{\alpha}{d}$ that needs to be configured for system stability and responsiveness. Once the stable range of $\frac{\alpha}{d}$ is found, α can be determined by choosing a sufficiently small d value. By applying a Laplas transformation to the ARC differential equation (Equation 4.3), the transfer function of ARC ($C(s)$) is:

$$C(s) = \frac{\frac{\alpha}{d}\left(s + \frac{1+\gamma}{\tau}\right)}{s\left(s + \frac{1}{\tau}\right)} \tag{4.4}$$

Before continuing our stability analysis, in order to identify the controller type, we take the queue behavior model embedded in the ARC model out from the ARC transfer function such that $C(s) = Q(s)C_q(s)$. Figure 4.2 represents this view of the system. Now:

$$Q(s) = \frac{\frac{1}{\tau}}{s + \frac{1}{\tau}} \tag{4.5}$$

$$C_q(s) = \frac{\alpha(1+\gamma)}{d}\frac{\frac{\tau s}{1+\gamma}+1}{s} = K_I\frac{\frac{s}{T_p}+1}{s} \tag{4.6}$$

Equation 4.6 indicates that the queue-based implementation of ARC is a special PI controller that fixes the constant $T_p = \frac{1+\gamma}{\tau}$ such that the proportional constant ($K_P = \frac{K_I}{T_p}$) and the integral constant (K_I) of the compensator has the following relationship:

$$K_P = K_I \frac{\tau}{1+\gamma} \quad \textit{and} \quad K_I = \frac{\alpha(1+\gamma)}{d}$$

We next perform frequency response analysis on the TCP-ARC system model, applying Bode stability criteria [94] to find the stable operating range of the $\frac{\alpha}{d}$ parameter for a chosen range of traffic conditions. The Bode method evaluates the stability of a closed-loop system by examining the open-loop system response to sinusoidal inputs with various frequencies. The open-loop transfer function of TCP-ARC system is:

$$G(s) = C(s)P(s) = \frac{\frac{\alpha\tau^3\gamma^3C^3(1+\gamma)}{4dN^2}\left(\frac{\tau}{1+\gamma}s+1\right)e^{\tau s}}{s\left(\frac{\tau^2\gamma C}{2N}s+1\right)(\tau s+1)} \tag{4.7}$$

The two open-loop system responses the Bode method uses are the magnitude gain in decibels ($20\log_{10}|G(j\omega)|$) and the phase shift ($\angle G(j\omega)$) of the output sinusoid given as functions of input sinusoid frequency, which for the TCP-ARC system are computed as follows:

$$\begin{aligned}
\mu(\omega, \tfrac{\alpha}{d}) &= 20\log_{10}|G(j\omega)| \\
&= 20\log_{10}\left(\frac{\frac{\alpha\tau^3\gamma^3C^3(1+\gamma)}{4dN^2}\sqrt{\left(\frac{\tau\omega}{1+\gamma}\right)^2+1}}{\omega\sqrt{\left(\frac{\tau^2\gamma C\omega}{2N}\right)^2+1}\sqrt{(\tau\omega)^2+1}}\right) \\
\phi(\omega) &= \angle G(j\omega) \\
&= \tan^{-1}\left(\frac{\tau\omega}{1+\gamma}\right) - \tan^{-1}\left(\frac{\tau^2\gamma C\omega}{2N}\right) - \tan^{-1}(\tau\omega) - \frac{180^\circ}{\pi}\tau\omega - 90^\circ
\end{aligned}$$

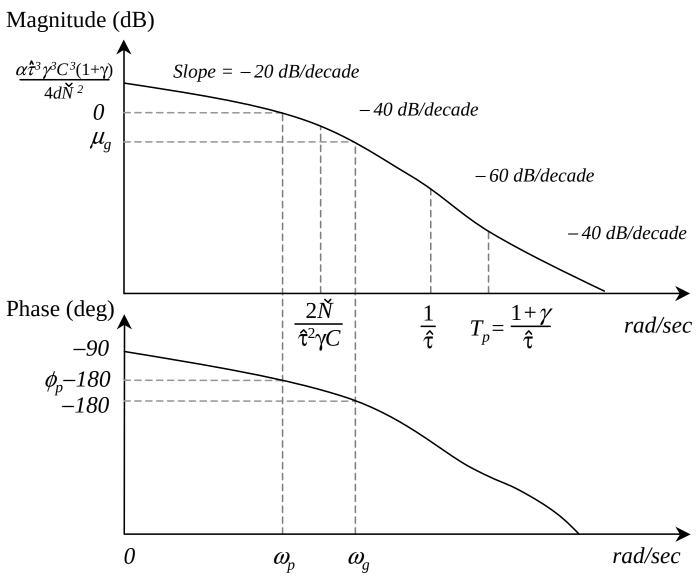


Figure 4.3: Bode Plot of TCP-ARC System

Figure 4.3 shows Bode plot for an example TCP-ARC system. Before discussing the characteristics of the TCP-ARC system described by the Bode plots, we briefly introduce the Bode stability criterion that states the following: A closed-loop system is stable when the magnitude gain of the open-loop system is less than $0dB$ at the input frequency (ω_g) that causes a 180° phase lag to the output signal. The Bode stability criterion provides a necessary and sufficient condition for a closed-loop stability, yet it does not guarantee that a system exhibits desirable transient response characteristics. Therefore, the following rule of thumb is often used as an extention to the Bode criterion: a well-designed feedback control system has the magnitude gain less than $-6dB$ at ω_g, and the phase lag in between 30° and 60° at the input frequency (ω_p) that results in zero magnitude gain. That is, $\mu_g < -6$ and $30^\circ < \phi_p < 60^\circ$ in Figure 4.3 where:

$$\left\{\begin{array}{lcl} \mu_g & = & \mu(\omega_g, \frac{\alpha}{d}) \\ \phi_p & = & \phi(\omega_p) + 180^\circ \end{array}\right. \left\{\begin{array}{lcl} \omega_g & = & \phi^{-1}(-180^\circ) \\ \omega_p & = & \mu^{-1}(0, \frac{\alpha}{d}) \end{array}\right.$$

Using Bode stability criteria to configure a general PI controller, such as in [58], with two control parameter involves deformation of both the magnitude and phase curves as the parameter values change, adding complications. And unfortunately, most other available PI tuning methods are not valid for systems with time delay. Moreover, the few available PI tuning methods for time-delayed systems may require expert decisions and only work for first-order systems [118]. ARC eases the configuration problems for TCP systems by fixing the controller constant $T_p = \frac{1+\gamma}{\hat{\tau}} > \frac{1}{\hat{\tau}}$. This results in the phase shift response curve of the TCP-ARC system that is not a function of the $\frac{\alpha}{d}$ parameter, and thus fixes the phase shift curve during the tuning process. In addition, the magnitude gain curve only moves vertically without deformation as a function of $\frac{\alpha}{d}$, easing the tuning process.

Given the link capacity (C) and the target utilization (γ), the stability boundary condition of the TCP-ARC system is characterized by the minimum expected number of TCP flows ($\check{N}$) and the maximum expected round-trip time of the system ($\hat{\tau}$). These parameters determine the location of the zero and poles of $G(s)$ characterizing the system into one of the following three cases:

$$\begin{aligned}
Case\ 1 &: \frac{2\check{N}}{\hat{\tau}^2\gamma C} < \frac{1}{\hat{\tau}} < \frac{1+\gamma}{\hat{\tau}} \\
Case\ 2 &: \frac{1}{\hat{\tau}} \leq \frac{2\check{N}}{\hat{\tau}^2\gamma C} < \frac{1+\gamma}{\hat{\tau}} \\
Case\ 3 &: \frac{1}{\hat{\tau}} < \frac{1+\gamma}{\hat{\tau}} \leq \frac{2\check{N}}{\hat{\tau}^2\gamma C}
\end{aligned}$$

Case 1: This system depicted by Figure 4.3 has phase lags of $\phi(0^+) = -90^\circ$ and $\phi(\frac{1}{\hat{\tau}}) < -180^\circ$ indicating that at least one $\omega_g < \frac{1}{\hat{\tau}}$ exists.

$$\phi\left(\tfrac{1}{\hat{\tau}}\right) = \tan^{-1}\left(\frac{\hat{\tau}}{1+\gamma}\,\frac{1}{\hat{\tau}}\right) - \tan^{-1}\left(\frac{\hat{\tau}^2\gamma C}{2\check{N}}\,\frac{1}{\hat{\tau}}\right) - \tan^{-1}(1) - \frac{180^\circ}{\pi} - 90^\circ$$

$$
\begin{aligned}
&= (45^\circ - \epsilon_1) - (45^\circ + \epsilon_2) - 45^\circ - \frac{180^\circ}{\pi} - 90^\circ \\
&< -180^\circ
\end{aligned}
$$

Further examining the shape of $\phi(\omega)$ using the derivative $\frac{d\phi(\omega)}{d\omega}$ reveals that $\phi(\omega)$ is a decreasing function of ω except for a possible small mound in the neighborhood of $\frac{1+\gamma}{\hat{\tau}}$. However, since $\frac{1}{\hat{\tau}} < \frac{1+\gamma}{\hat{\tau}}$, the smallest (and usually the only) $\omega_g < \frac{1}{\hat{\tau}}$ with $\frac{d\phi(\omega)}{d\omega}|_{\omega<\omega_g} < 0$ can always be found. This, plus the fact that the magnitude gain curve $\mu(\omega, \frac{\alpha}{d})$ is a strictly decreasing function of ω starting from a positive value at $\omega = 0^+$, guarantees that it is always possible to find a range of $\frac{\alpha}{d}$ that makes $\omega_p(\frac{\alpha}{d}) < \omega_g$ and thus have a positive ϕ_p. If a TCP-ARC system can ever be stabilized for a given $\check{N}$ and $\hat{\tau}$ boundary, one should be able to refine the range of $\frac{\alpha}{d}$ such that $30^\circ < \phi_p < 60^\circ$. A useful necessary condition for the system stability is $\mu(\frac{1}{\hat{\tau}}, 0) < -6$.

Case 2 and *Case* 3: The same analysis used for a *Case* 1 system can be applied to *Case* 2 and *Case* 3 systems with two differences. The first difference for both systems is that $\phi(\frac{2N}{\hat{\tau}^2\gamma C}) < -180^\circ$. The second difference for *Case* 3 systems is that $\phi(\omega, \frac{\alpha}{d})$ is a strictly decreasing function of ω. It is always possible to find a range $\frac{\alpha}{d}$ that makes $\omega_p(\frac{\alpha}{d}) < \omega_g$ and have a positive ϕ_p, where a necessary condition for system stability is $\mu(\frac{2N}{\hat{\tau}^2\gamma C}, 0) < -6$.

For all cases, we can determine if a TCP-ARC system can be stabilized for the chosen $\check{N}$ and $\hat{\tau}$ boundary, and find the stable operating range of $\frac{\alpha}{d}$ in two steps:

$$
\begin{aligned}
&\left\{\tfrac{\alpha}{d} \mid \mu(\omega_g, \tfrac{\alpha}{d}) < -6\right\} \quad \textit{and} \\
&\left\{\tfrac{\alpha}{d} \mid -150^\circ < \phi(\omega_p(\tfrac{\alpha}{d})) < -120^\circ\right\} \qquad (4.8)
\end{aligned}
$$

Algorithm 2 Aggregate Rate Controller

```
Every $d$ seconds:
1: $p \leftarrow p + \alpha(b - \gamma(dC - (q - q_0)))$;
2: $b \leftarrow 0$;
Every $packet$ arrival:
3: if $(uniform(0,1) \leq p)$ then
4:     if $(mark(packet) == false)$ then
5:         $drop(packet)$;
6:         $return$;
7:     end if
8: end if
9: $b \leftarrow b + sizeof(packet)$;
10: if $(enqueue(packet) == false)$ then
11:     $drop(packet)$;
12: end if
Functions:
  $mark(packet)$: ECN mark the packet. Return $false$ on error.
  $enqueue(packet)$: Enqueue the packet. Return $false$ if queue is full.
  $drop(packet)$: Drop the packet.
Variables:
  $p$, $q$, $b$
Parameters:
  $C$: link capacity (bytes per second)
  $\gamma$: target link utilization ($\gamma = C_0/C$)
  $q_0$: target queue length in bytes
  $d$: measurement interval
  $\alpha$: TCP congestion feedback constant
```

Thus, ARC configuration involves choosing a reasonable minimum number of flows ($\check{N}$) and maximum expected round-trip time ($\hat{\tau}$) for the system we wish to support, finding the $\frac{\alpha}{d}$ range that satisfies the first condition of Equation 4.8, and refining the range to satisfy the second condition of Equation 4.8. Once the stable range is found, choosing a sufficiently small d gives the range of α. The ARC configuration issues of choosing reasonable $\check{N}$ and $\hat{\tau}$ stability boundary conditions and the measurement interval d is discussed in Section 4.1.2.

We implemented the ARC algorithm shown in Algorithm 2 into NS [127] and designed experiments to validate the linearity of the TCP system and correctness of our model. Linearity of a system can be validated via simple frequency analysis by

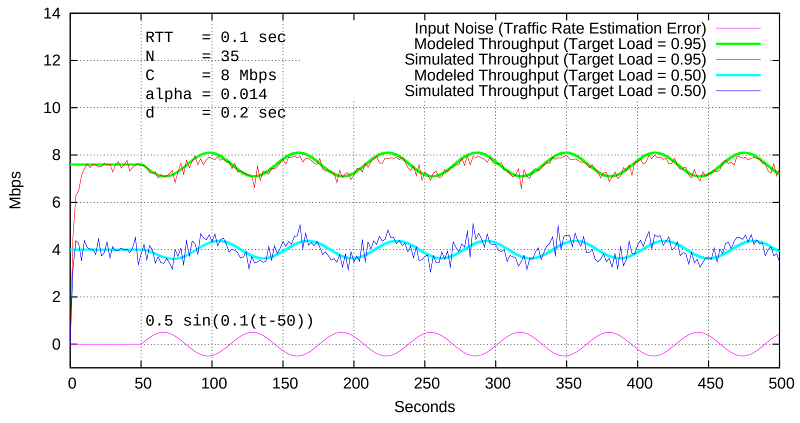


Figure 4.4: TCP-ARC Model Validation: estimation error vs. throughput

introducing a sinusoid into the system. If a system has linearity, the output of the system should also be a sinusoid with a possibly altered magnitude and frequency. We artificially injected a sinusoid into a simulated TCP-ARC system representing the incoming traffic rate estimation error at the congested ARC router and measured the system throughput. Then, we compared the simulated throughput with the output of the analytic TCP-ARC system model.

We used a dumbbell topology for the simulations, uniformly varying the round-trip times of 35 FTP-TCP connections from $90ms$ to $110ms$. We set the physical queue length to 500 packets, where the system used 1 Kbyte packets. For the congested-link, we used $C = 8$ Mbps once with the target utilization $\gamma = 0.95$, and once with $\gamma = 0.50$. The ARC control parameters were set to $d = 200$ ms and $\alpha = 0.112$ for the simulations. For the ARC rate estimation error, we used a low frequency sinusoid of $0.5 \sin(0.1(t - 50))$ Mbps.

Figure 4.4 compares the throughput of the simulated systems with that of our analytic model for the sinusoidal input under the two different target loads (γ). The simulated results closely match the output of the TCP-ARC analytical model, showing the linearity of the TCP congestion control system and also the correctness of our

TCP-ARC model. In addition, the ability to meet custom target traffic loads illustrates the configuration flexibility of ARC. More importantly, the resilient TCP-ARC congestion control system performance for the sinusoidal traffic rate estimation errors indicates potentially resilient ARC performance in the presence of TCP-unfriendly, unresponsive and/or short-lived Web traffic.

4.1.2 Configuration

In the previous section, we have shown that the ARC control parameter $\frac{\alpha}{d}$ can be found to stabilize a TCP system characterized by $\check{N}$ and $\hat{\tau}$ boundary conditions. This section addresses the configuration issue of selecting the boundary conditions epoch length (d) to enable ARC to be resilient over a wide range of traffic conditions.

Although a PI controller considerably broadens the range over which a controller with a given configuration can be stable [58], fundamental configuration issues still remain. Consider the case where ARC is configured for an average-case traffic scenario with a relatively large number of flows ($\check{N}$) and a typical round-trip time $\hat{\tau}$. The resulting TCP-ARC system may encounter instability in cases with fewer TCP flows than average and/or cases where the average round-trip time is unusually large. We will refer to this configuration issue as the "overshoot problem". On the other hand, if ARC is configured to support a worst-case scenario with a small number of flows ($\check{N}$) and a large round-trip time ($\hat{\tau}$), the resulting system will have a significantly larger response time when there is the average-case traffic. Thus, the system will suffer from significant queuing delays and queue overflows before adapting to any changing traffic conditions. We will refer to this configuration issue as the "undershoot problem".

Most TCP flows are limited by one or more of: the capacity of the ISP access points or local networks; the maximum TCP window sizes; and the sizes of the objects being downloaded [67]. Therefore, the overshoot problem is not a significant concern for core routers since backbones are not likely to be congested by a small number of TCP flows. Also backbone link flows typically have a relatively low average round-trip time [67, 20], making them unable to create an overshoot problem. However, the overshoot problem may occur in enterprise or access network routers as the traffic load and round-trip time may vary significantly during a single day. In fact, the overshoot problem is more critical than the undershoot problem since it can severely degrade network stability even under light traffic loads. At the same time, we do not want to create undershoot problems since that deteriorates the long sought after benefits of AQM.

We address the overshoot and undershoot configuration problems for ARC by carefully interpreting the meaning of the analytically determined system stability analysis for practical network operations. Finding a reasonable $\check{N}$ and $\hat{\tau}$ for the $\frac{\alpha}{d}$ configuration starts from the fact that it is inadequate to apply stability determined through our stochastic system model to fragile systems with a small number of TCP flows and a large average round-trip time. That is, no matter how well we tune ARC using the model, it is impossible for an AQM to achieve a high link utilization and low queuing delay for a fragile system since the aggregate traffic rate fluctuates reflecting the bursty characteristics of TCP. Therefore, tuning ARC for a fragile system is an ineffective use of the controller capability and it may cause unnecessary queuing delays and overflows for normal traffic loads. Thus, the ARC configuration objective is to effectively manage average traffic conditions while avoiding large overshoot problems that can significantly degrade network stability.

In order to determine when an overshoot threatens stability for the fragile lower bound of a system we wish to support, denoted by $\check{N}_{min}$ and $\hat{\tau}_{max}$, we need to determine the sustainable congestion notification probability estimation error ($\delta p_s <$

1) in order for the lower bound system to be least operational. We develop a simple frequency analysis on a TCP plant ($P(s)$) to quantify δp_s. We define the least operational state of a TCP system as the following: a TCP system is least operational if for congestion notification errors in the range $[-\delta p_s,\ \delta p_s]$ introduced to the TCP plant, the output range of the system is $[-\gamma C,\ \gamma C]$. The equivalent mathematical representation is:

$$|P(j\omega)| \;=\; \frac{\frac{(\hat{\tau}_{max})^3\gamma^3C^3}{4(\check{N}_{min})^2}}{\sqrt{\omega^2\left(\frac{(\hat{\tau}_{max})^2\gamma C}{2\check{N}_{min}}\right)^2+1}} \;\leq\; \frac{\gamma C}{\delta p_s}$$

This inequality always holds for high frequency inputs. For low frequency congestion notification probability estimation errors $\left(\omega \ll \frac{2\check{N}_{min}}{(\hat{\tau}_{max})^2\gamma C}\right)$, we have:

$$\delta p_s \;\leq\; \frac{4(\check{N}_{min})^2}{(\hat{\tau}_{max})^3\gamma^2C^2}$$

Applying this statement to the TCP-ARC system, if ARC makes congestion estimation errors within $[-\delta p_s,\ \delta p_s]$ for the fragile lower bound of the system we wish to support, the system will be least operational. In order to accomplish this, it is necessary to set the $\frac{\alpha}{d}$ parameter such that ARC increases p sufficiently less than δp_s during one $\hat{\tau}_{max}$ interval:

$$\frac{\alpha}{d} \;\leq\; \frac{\alpha_{max}}{\hat{\tau}_{max}} \;\ll\; \delta p_s$$

This minimum stability condition can be used for ARC configuration to avoid stability threatening, large overshoot problems for the fragile boundary system we choose to support. We first choose an average TCP-ARC system to stably support, described by $\check{N}$ and $\hat{\tau}$, and use the ARC configuration guidelines introduced in

Section 4.1.1 to find the stable $\frac{\alpha}{d}$ range. Then, we choose the fragile boundary system described by $\check{N}_{min}$ and $\hat{\tau}_{max}$, and use the minimum stability condition to see if the parameter range is safe for the fragile boundary system.

Next, we need to determine the measurement epoch d to complete the configuration. The stability of backbone routers may not be sensitive to the choice of d, but proper selection of d is critical for enterprise or access network router configurations. The stochastic TCP system model may not accurately depict the stability of the system with a large average round-trip time and a small number of TCP flows as packets would tend to arrive at the router in bursts, breaking the assumption of stochastic packet arrivals. However, we can relax the stochastic assumption for the TCP-ARC system by choosing a sufficiently large d, larger or at least equal to the maximum average round-trip time we wish to support ($\hat{\tau}_{max}$). In this way, the effect of the traffic bursts on the ARC control decision can be minimized, reducing the noise in the incoming traffic rate estimation caused by the traffic bursts. However, choosing too large a d will affect the system responsiveness, weakening the small interval assumption made for the algorithmic discretization. As a compromise, we recommend setting the measurement epoch to the maximum expected round-trip time, or $d = \hat{\tau}_{max}$. According to a recent Internet measurement study [67], the median of the median round-trip times is less than 1 second, and 90% of the median round-trip times are under 2 seconds. Therefore, depending on the targeted level of network resilience and the specific traffic characteristics of the network, setting $\hat{\tau}_{max}$ in between 1 and 2 seconds is appropriate for most enterprise or access network routers. For backbone routers, selecting an even smaller $\hat{\tau}_{max}$ to enhance the granularity of congestion control may also be effective.

Summarizing our analysis for configuring ARC over a wide range of traffic loads, we present the final guidelines that are recommended along with Equation 4.8:

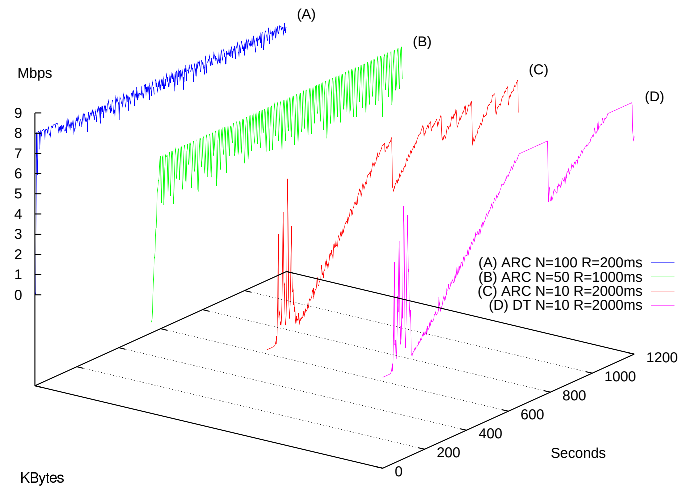
$$d = \hat{\tau}_{max} \quad and \quad \alpha \ll \frac{4(\check{N}_{min})^2}{(\hat{\tau}_{max})^3\gamma^2C^2} \tag{4.9}$$

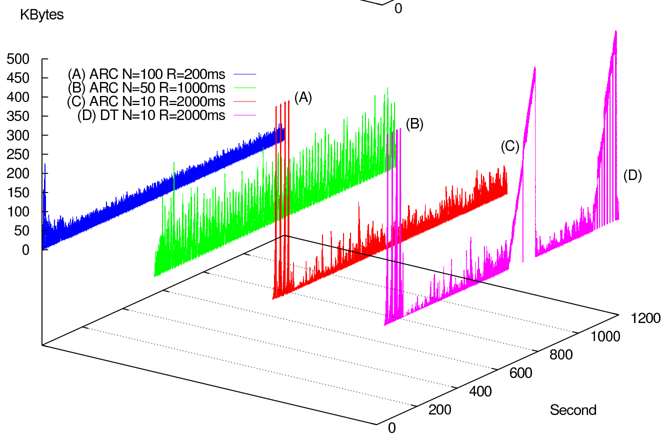
We validate the minimum stability condition for fragile boundary systems and show that ARC can be resilient over a wide range of traffic loads through simulation. We use a dumbbell topology with $C = 8Mbps$, $\gamma = 0.95$ and the queue limit set to $500Kbytes$ ($1packet = 1Kbyte$). We vary the round-trip time (τ) uniformly across the range [0.150,0.250] seconds, [0.9,1.1] seconds and [1.9,2.1] seconds for each simulation. We also vary the number of ECN enabled long-lived TCP flows from 10 to 50 to 100, where each flow starts randomly distributed from 0 to 100 seconds. The ARC control parameters are set to $\alpha = 1.11 \times 10^{-5}$ and $d = 2sec$ in order to effectively support the system with $\hat{\tau} = 500ms$ and $\check{N} = 50$, giving a fair per-flow bitrate of $160Kbps$. We further assume that the fragile boundary system has $\check{N}_{min} = 10$ and $\hat{\tau}_{max} = 2sec$, satisfying the minimum stability condition:

$$\alpha \ll \frac{4(\check{N}_{min})^2}{(\hat{\tau}_{max})^3\gamma^2C^2} = 5.54 \times 10^{-5}$$

Figure 4.5 shows three ARC simulations: (A) N=100 and τ=200ms, (B) N=50 and τ=1.0sec, and (C) N=10 and τ=2.0sec. We include one fragile boundary system simulation using drop-tail, (D), as a yardstick to assess the performance of ARC for the fragile boundary system. Simulation (A) shows that ARC handles the undershoot situation, achieving average throughput close to the target capacity with a consistently low queuing delay and a nearly zero packet loss rate. Simulation (B) shows that ARC can also handle the medium overshoot situation, utilizing 93% of the target capacity (.95) with a low queuing delay and no packet loss.

Simulations (C) and (D) verify the inefficiency of TCP when there is a large





(0-1200 Seconds)	Throughput	Loss
(A) ARC N=100, τ=200ms	7589 $Kbps$	0.001%
(B) ARC N=50, τ=1.0sec	7053 $Kbps$	0.000%
(C) ARC N=10, τ=2.0sec	1707 $Kbps$	0.515%
(D) DT N=10, τ=2.0sec	1709 $Kbps$	0.519%

Figure 4.5: Throughput and Queue Dynamics

capacity compared to the number of flows (C/N). More importantly, the simulation results show that ARC, even when not optimized for the fragile boundary system and only satisfying the minimum stability condition, can still perform better than drop-tail queue management under the same conditions. ARC is able to keep a consistently low queuing delay while having a throughput and packet loss rate compatible with that of drop-tail queue management.

In practice, router access speed is an order of magnitude slower than the link speed, and the bottleneck may often be at the network processor rather at the link. Therefore, the access speed (or effective service rate) of the router should be used instead of the link capacity (C) to configure ARC. A related issue is that network processor output is bounded by the number of packets, independently of the packet size. Therefore, when small packets dominate the incoming traffic, the effective service rate of the router can be noticeably decreased. This effective service rate variation serves as another source of control noise for AQM congestion controllers. Yet, an ARC router configured to be robust over wide range traffic load will still be stable in this situation and perform well. We show the performance of ARC as C changes in Section 4.1.3.3.

A more advanced approach to ARC configuration may consider a dynamic control parameter adaptation mechanism. One possible design may slowly change the value of α (assuming a fixed d) based on the displacement of the average traffic rate from the target capacity (γC) within the initially configured α range. While this approach may sound promising, it introduces another controller (a low pass filter) into the system. Therefore, any dynamic control parameter adaptation would demand another major study of implementation and stability analysis. We leave this as future work.

4.1.3 Evaluation

This section compares the performance of ARC with PI [58], AVQ [75], SFC [47] and drop-tail through detailed simulations varying long-lived FTP traffic load, round-trip times (RTT) and bottleneck link capacity, with 2-way traffic (which can result in ack compression) and background Web traffic. We also simulate Web flash crowds and multiple congestion bottlenecks to test the AQM controllers under more realistic Internet traffic environments. For all evaluations, we use the IP packet simulator NS which includes source code for PI and AVQ. We extend NS to support ARC and implement SFC based on [47].

Unless otherwise noted, we use a dumbell network topology with a bottleneck link capacity of 10 Mbps and a maximum packet size of 1000 bytes. Round-trip link delays are randomly uniformly distributed over the range [60:1000], based on measurements in [67]. The physical queue limit for each AQM and the drop-tail queue is set to 500 Kbytes, which is approximately equal to the bandwidth-delay product for the mean round-trip time.

The settings for the parameters of the various AQMs are based on the recommendations by their authors. The target utilization, γ, of AVQ is set to 0.98 and the damping factor, α, is set to 0.15 according to Theorem 1 in [75]. The parameters for SFC are $k_1 = 0.0005$ and $k_2 = 0.2$, as used in [47]. The parameters for PI are $\alpha = 1.822 \times 10^{-5}$, $\beta = 1.816 \times 10^{-5}$ and sampling frequency $w = 170$, as in [58]. The settings for ARC have the measurement epoch $d = 1$ seconds, $\alpha = 1.42 \times 10^{-5}$, the target utilization $\gamma = 0.98$ and the target queue $q_0 = 0$. The ARC control parameters are chosen in order to effectively support the system with $\hat{\tau} = 500ms$ and $\check{N} = 50$, and to satisfy the minimum stability condition assuming the fragile boundary system has $\hat{\tau}_{max} = 2sec$ and $\check{N}_{min} = 10$.

In all simulations, we use ECN enabled NewReno TCP for both long-live FTP flows and Web-like sessions. Each simulation has a number of forward and backward direction bulk transfer FTP flows of which the numbers are specified in each experiment subsection. Also, each simulation has 300 background Web-like sessions (using the Webtraf code built into NS) that start evenly distributed during the first 30 seconds. Each Web session requests pages with 2 objects drawn from a Pareto distribution with a shape parameter of 1.2 and an average size 5 Kbytes, where each of these settings is based on values from [4, 55]. The Web sessions have an exponentially distributed think time with a mean of 7 seconds, which results in an average utilization of about 0.25 of the 10 Mbps capacity, a typical fraction of Internet traffic such as reported by [115].

4.1.3.1 Long-Lived TCP Flows

This experiment compares the performance of ARC, PI, AVQ, SFC and drop-tail over a range of traffic loads with long-lived bulk transfer TCP flows. Each simulation begins with 10 forward direction FTP flows with start times uniformly randomly distributed over [0:50] seconds, and is accompanied by 300 background Web sessions of which the parameters given in the previous paragraph and 50 backward direction FTP flows. After 200 seconds, an additional 40 FTP flows are added with start times uniformly randomly distributed over the subsequent 50 seconds. The total number of FTP flows doubles every 200 seconds thereafter, resulting in 100 flows at time 400, 200 flows at time 600, 400 flows at time 800 and 800 flows at time 1000. In each interval new FTP flows arrive, their start times are uniformly randomly distributed over a 50 second range. We ran ARC twice, once with the settings specified earlier ($d = 1$ and $\alpha = 1.42 \times 10^{-5}$) and then with $d = 2$ (and $\alpha = 2.84 \times 10^{-5}$ to preserve the α/d ratio), in order to test the sensitivity of the rate measurement interval.

Figure 4.6 (top) depicts the queue dynamics for the four AQMs with drop-tail (DT) shown as a reference. Drop-tail exhibits the expected queue dynamics with large queue oscillations when there are few flows and stable, but large, queue sizes when there are many flows. SFC has stable queue dynamics for all numbers of flows. However, without an integral control component, SFC exhibits a steady increase in the average queue size as the number of flows increases, approaching the physical queue limit when there are 800 FTP flows. AVQ exhibits unstable queue dynamics when there are few flows but stabilizes with relatively low queues once there are 200

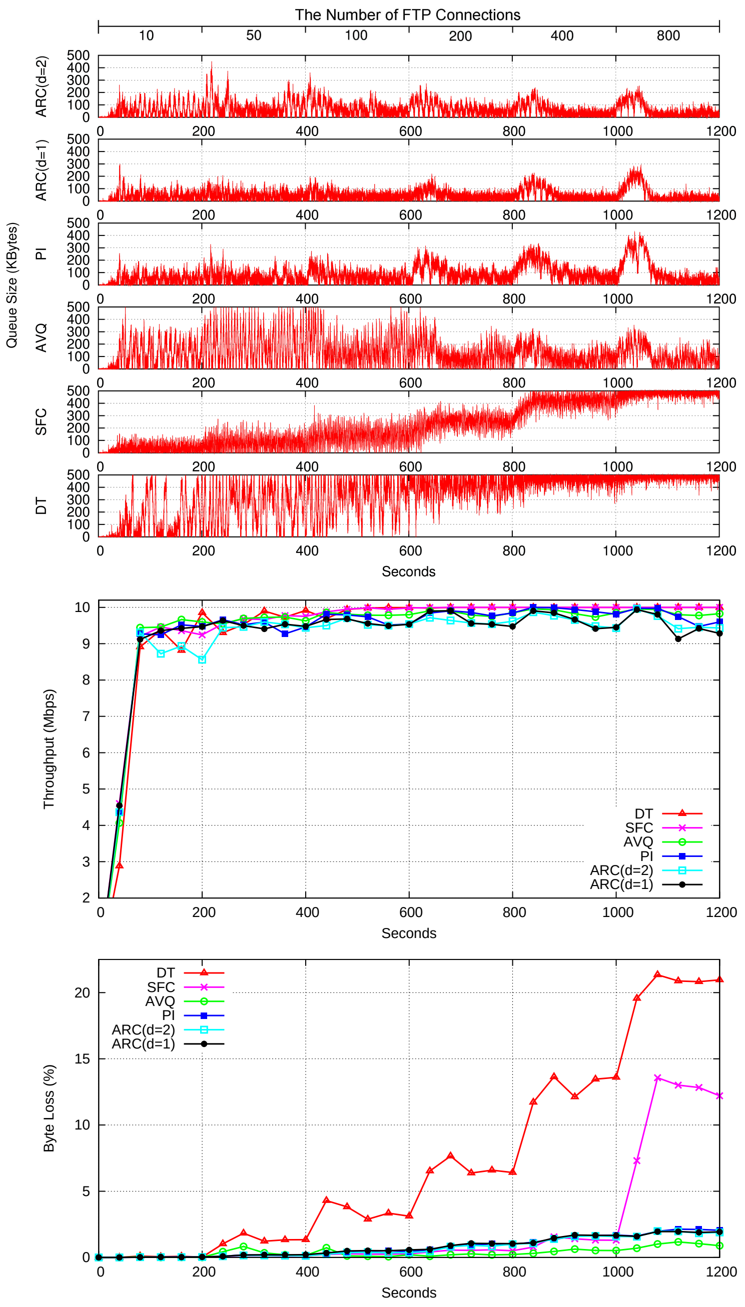


Figure 4.6: Network Statistics: Increasing the number of FTP flows

or more FTP flows. PI and ARC ($d = 1$ and $d = 2$) are similar, with stable queue dynamics for all numbers of flows, and with short-term queue size increases each time a large group of flows arrives. Comparing ARC with $d = 1$ and $d = 2$ shows that ARC is not very sensitive to the selection of the measurement interval.

Figure 4.6 (middle) depicts the throughput for the AQMs and drop-tail. Once the number of FTP flows is 50 or more, each queue management scheme is able to obtain a throughput over 9.5 Mbps. For more than 200 FTP flows, drop-tail has the highest throughput, with SFC having a throughput nearly equal to that of drop-tail. However, drop-tail has the lowest utilization when there are only 10 FTP flows. For 50+ FTP flows, AVQ and ARC keep their throughput close to their target utilization, and PI and AVQ achieve nearly the same throughput for all numbers of flows. Overall, PI and AVQ have less control over the queue length than ARC with a correspondingly slightly higher throughput than ARC.

Figure 4.6 (bottom) depicts the loss rates, where most of losses seen for AQMs at the lower range of traffic loads are due to ECN incapable packets such as the reverse traffic acknowledgments. The figure verifies that AQM can significantly reduce data losses for ECN traffic over drop-tail queue management as long as AQM can avoid or minimize buffer overflows. However, when there are consistent buffer overflows as in SFC for high traffic loads, the benefits of AQM quickly diminishes. A case where an AQM (AVQ, in this case) can incur even greater data losses than drop-tail is illustrated in Section 4.1.3.3.

4.1.3.2 Round-Trip Time

The next experiment illustrates behavior of the different queue mechanisms over a range of round-trip times, from about 0.3 to 2.4 seconds. We gradually increase the round-trip link delay (RTLD) of the congested link by 300 ms every 200 simulation seconds with 300 background Web sessions. We have run multiple sets of simulations with different numbers of forward and backward FTP flows to see the effect of the RTT spectrum on different levels of load. For brevity, we show the results for 5

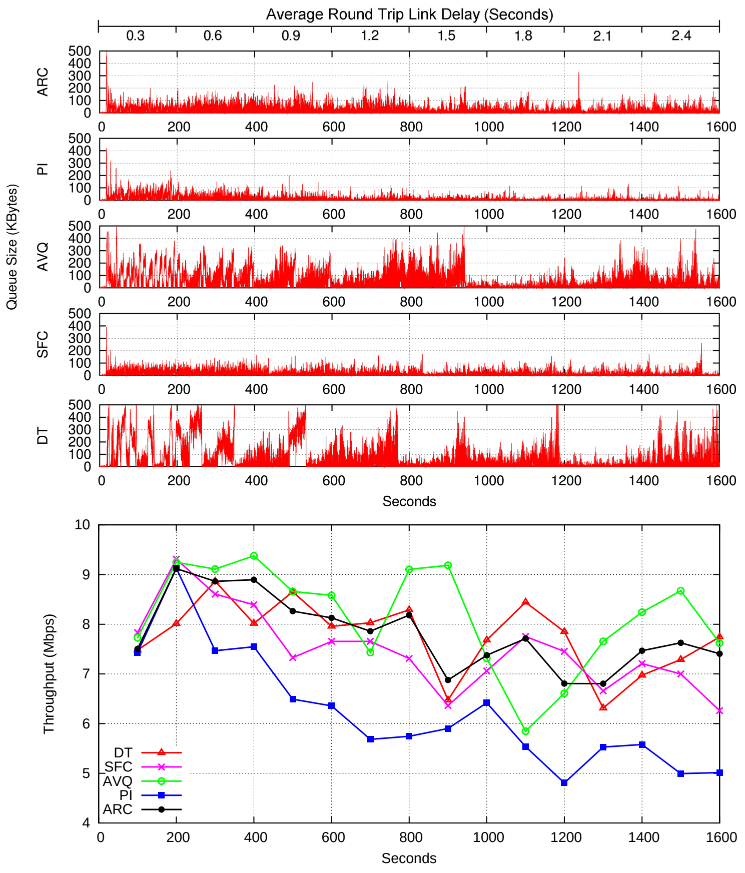


Figure 4.7: Network Statistics: Increasing the average round-trip time

forward FTP flows and 10 backward FTP flows, but the overall performance trends are similar for other numbers of flows.

The queue dynamics in Figure 4.7 (top) show that ARC, PI and SFC keep the queue size low and dampening the queue oscillations, while AVQ and drop-tail queue have poor control over the queue. Furthermore, the throughput comparisons in Figure 4.7 (middle) show that PI under-utilizes the link capacity far more then the other queue management schemes. This inefficiency is due to PI's method of traffic rate information acquisition that uses a high frequency of queue sampling. When a system becomes fragile, traffic arrives in bursts. In this case, incoming traffic rate estimation via derivation of frequent queue samples introduces a large estimation noise. ARC and SFC[1] avoid this noise by using a direct traffic rate measurement method over an interval of 1 second and so can more accurately estimate incoming traffic rate than PI. AVQ's packet-paced rate measurement mechanism is also compatibly accurate.

4.1.3.3 Congested Link Capacity

For the next set of simulations, we increase the bottleneck link capacity initially set to 10 Mbps by 10 Mbps every 200 simulation seconds up to 50 Mbps, with 200 forward and 50 backward direction FTP flows and 300 background Web sessions. During these increases, we do not change the AQM parameter settings. This creates two effects: the increasing the capacity makes the system more fragile and it also brings forth the undershoot configuration problem (described in Section 4.1.2) of an AQM slowly responding to network congestion. Thus, this experiment illustrates the control parameter sensitivity of ARC, PI, AVQ and SFC.

Figure 4.8 illustrates the queue dynamics, throughput and data loss rates. AVQ is very sensitive to the initial control parameters. While AVQ is able to maintain the throughput close to its target utilization, it does not effectively control the queue oscillations and incurs even greater data losses than does drop-tail queue management.

PI produces under-utilization and a sluggish response as the capacity increases, another drawback of a queue based rate estimation. As the traffic load decreases due to the capacity increase, PI cannot efficiently estimate the traffic rate, since the queue oscillations tell little about the underlying traffic rate.

Both ARC and SFC effectively control both queue oscillation and achieve high throughput over the range of traffic loads. Yet, when the link capacity increases to 40 Mbps and over, SFC is not able to achieve as high a throughput as ARC due

[1]SFC [47] does not specify how the rate estimation mechanism should be implemented nor recommend a range of values for the measurement epoch. We used the rate estimation mechanism of ARC to implement SFC in NS, and set the measurement epoch to 1 second as in ARC.

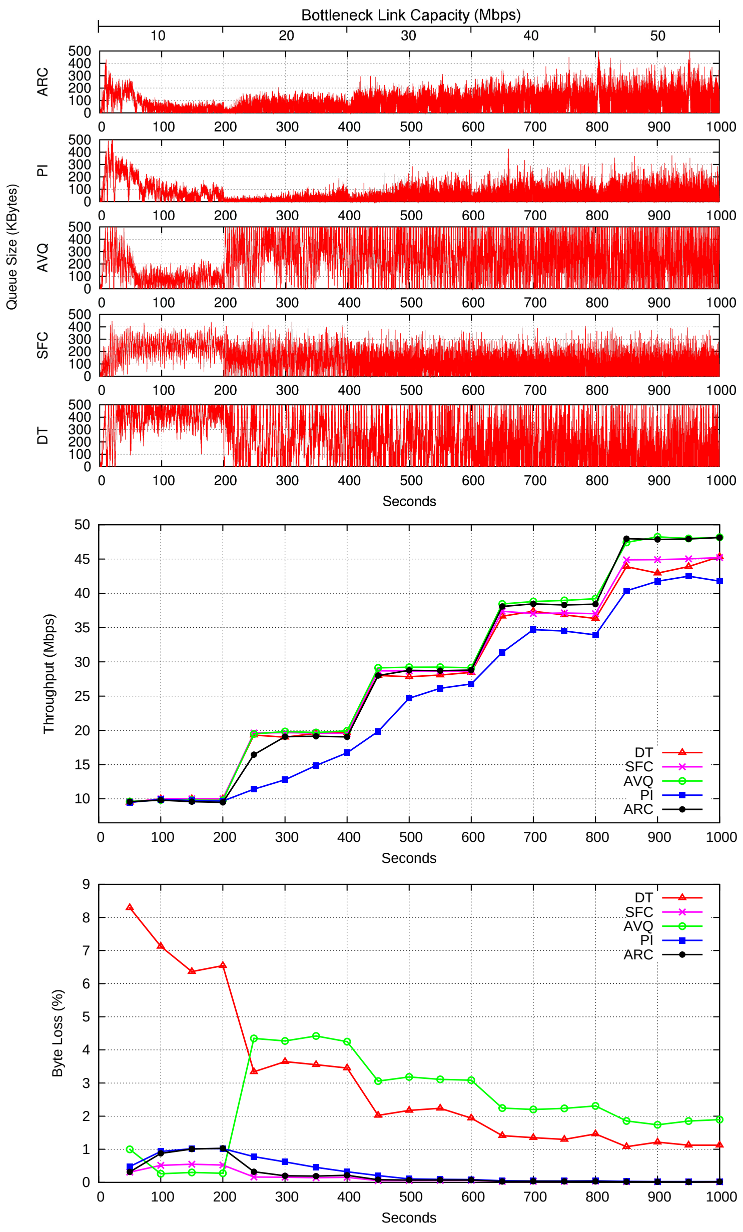


Figure 4.8: Network Statistics: Increasing the bottleneck link capacity

to its traffic load dependent control behavior. However, SFC is still able to achieve throughput at about the level of drop-tail.

4.1.3.4 Web Flash Crowd

In this section, we simulate a Web flash crowd. For this simulation, we have 25 forward direction and 50 backward direction FTP flows as background traffic, and an initial 300 Web sessions. After a warm up period of 100 seconds, 1000 more Web sessions are uniformly injected during the first 6000 simulation seconds and then detached during the next 6000 simulation seconds, giving a Web flash rate of 10 sessions per minute. Thus, we have a peak of 1300 Web sessions, providing an offered load of about 1.35.

We use a Web flash rate of 10 sessions per minute chosen based on the peak Web flash rate seen at the FIFA World Cup 98 Web server analysis [4]. The peak flash rate (both increase and decrease) was from 2 to 10 million requests per hour in 2 hours during the France-Croatia game. This means an acceleration of about 1,110 $requests/min^2$ from the minimum object request rate of 33,333 $requests/min$ (= 2 million $requests/hour$).

The starting 300 Web sessions in our simulation offer an average utilization of about 0.25, a typical fraction of Internet traffic reported in [115], with the minimum object request rate of 5,143 $requests/min$ (300×2 $objs/click$ $\times$ $60/7$ $clicks/min$). This is about 15% of the FIFA 98 base request rate. In order to get the proportional acceleration rate for our simulated minimum request rate, we take 15% of the FIFA 98 acceleration rate, that is about 170 $requests/min^2$ or 10 sessions per minute, and used it as our flash rate.[2]

Figure 4.9 depicts the network statistics including queue dynamics, throughput and data loss rate. In order to better understand the performance of the AQMs for Web traffic, we also analyzed the average object service time. To compare perfor-

[2]We also tried the absolute flash rate of the FIFA 98 trace (1,110 $requests/min^2$ = 60 sessions per minute) and received simular overall results.

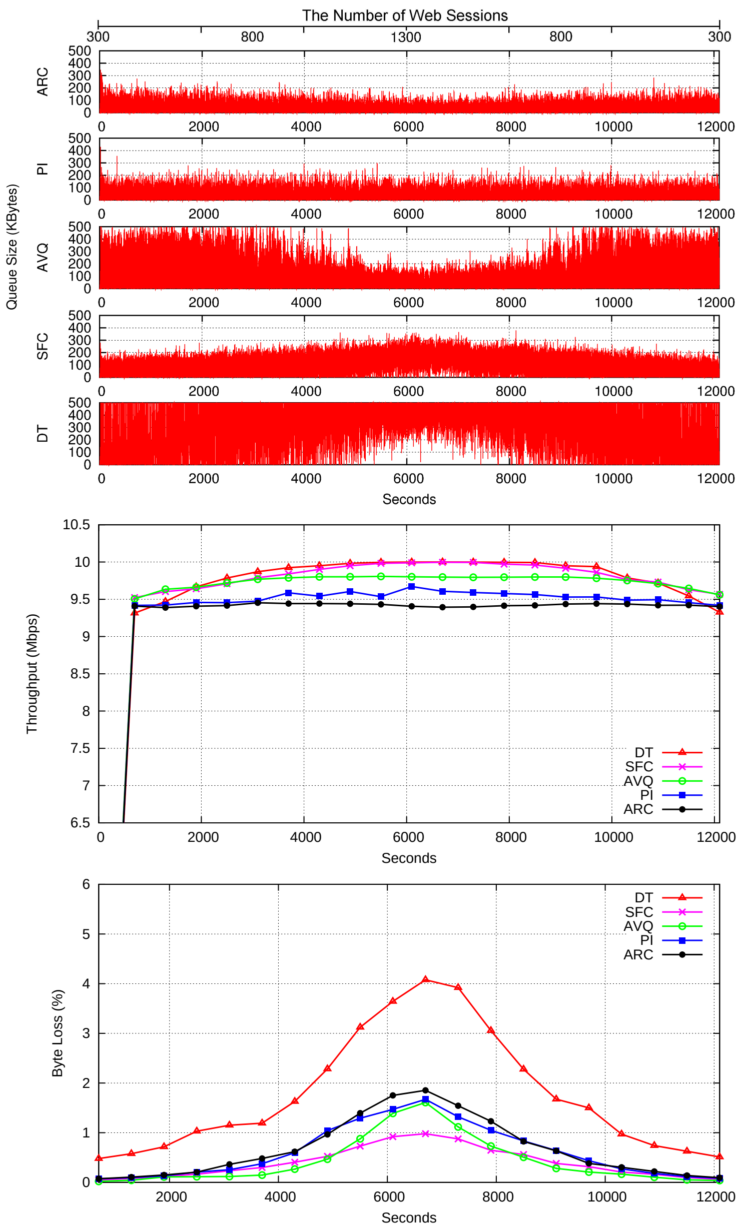


Figure 4.9: Network Statistics: Web flash crowd

mances of the AQMs on short-lived flows, Figure 4.10 plots average service time for objects less than 12 Kbytes which can be transmitted in about 4 round-trip times in the best case including TCP connection setup and represents about 95% of the all objects generated in a simulation.

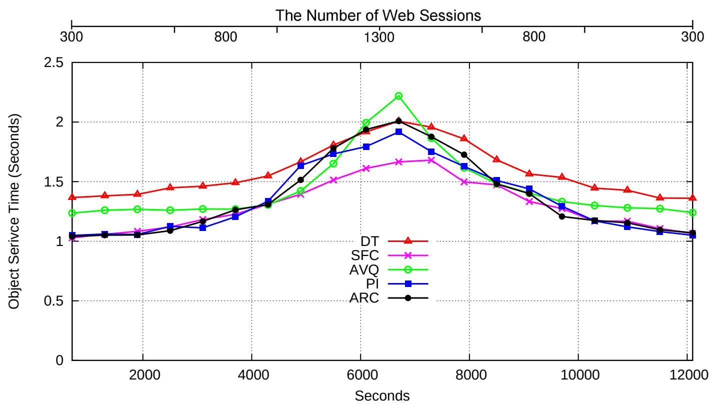


Figure 4.10: Average Service Time: Objects less than 12 Kbytes

In general, the queue dynamics shown in Figure 4.9 (top) are very similar to those of Figure 4.6 (top). Likewise, Figure 4.9 (middle) shows that all AQMs achieve a throughput around 9.4 Mbps or higher throughout the simulation. However, in Figure 4.9 (bottom), the data loss rates for AQMs are significantly increased (compared with Figure 4.6 (bottom)) for the traffic dominated by Web due to the considerably increased number of TCP-SYN packets that can not support ECN. This illustrates that gains for AQM with ECN traffic is reduced for short Web traffic.

Drop-tail management performs the worst overall having the highest data losses, queuing delay and object service times. All AQMs performed well under the offered Web traffic load of 0.9 (before 4,000 seconds or after 10,000 seconds), except AVQ which has average service times consistently higher than those of other AQMs due to its queue oscillations.

As the Web traffic load further increases over 0.9, SFC performs slightly better than other AQMs by stabilizing the queue higher than the other AQMs, thus achieving the overall highest throughput and the lowest data loss rate at the expense in queuing delay. ARC and PI perform very similarly in all performance aspects. Beyond the offered load of 0.9, AVQ starts to gain control over queue oscillations

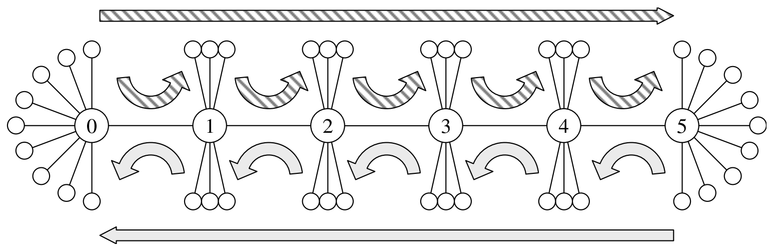


Figure 4.11: Multiple Bottleneck Simulation Setup

and perform comparable to ARC and PI.

4.1.3.5 Multiple Bottleneck Congestion

Figure 4.11 shows the network topology and scenario used for the multiple bottleneck simulations. The simulated network has 5 bottleneck links (numbered 0 to 4) with capacities of 10 Mbps and the transmission delay of 20 ms. The edge links are 100 Mbps with 20 ms of transmission delay. Each striped arrow in Figure 4.11 represents 25 FTP flows plus 150 Web sessions, and each solid arrow represents 25 backward FTP flows.

Figure 4.12 depicts the queue dynamics of the each network link for ARC, PI, AVQ, SFC and drop-tail queue management for the first 300 seconds (each simulation runs for 600 seconds). The queue dynamics show that ARC and other AQM systems are globally stable, and that ARC has the best control over queue oscillations followed by PI, SFC then AVQ. Figure 4.13 (top and bottom) show the average throughputs and the average data loss rates of the bottleneck links, where all the AQM routers achieve a high throughput and a low data loss rate, consistent with previous single bottleneck analysis.

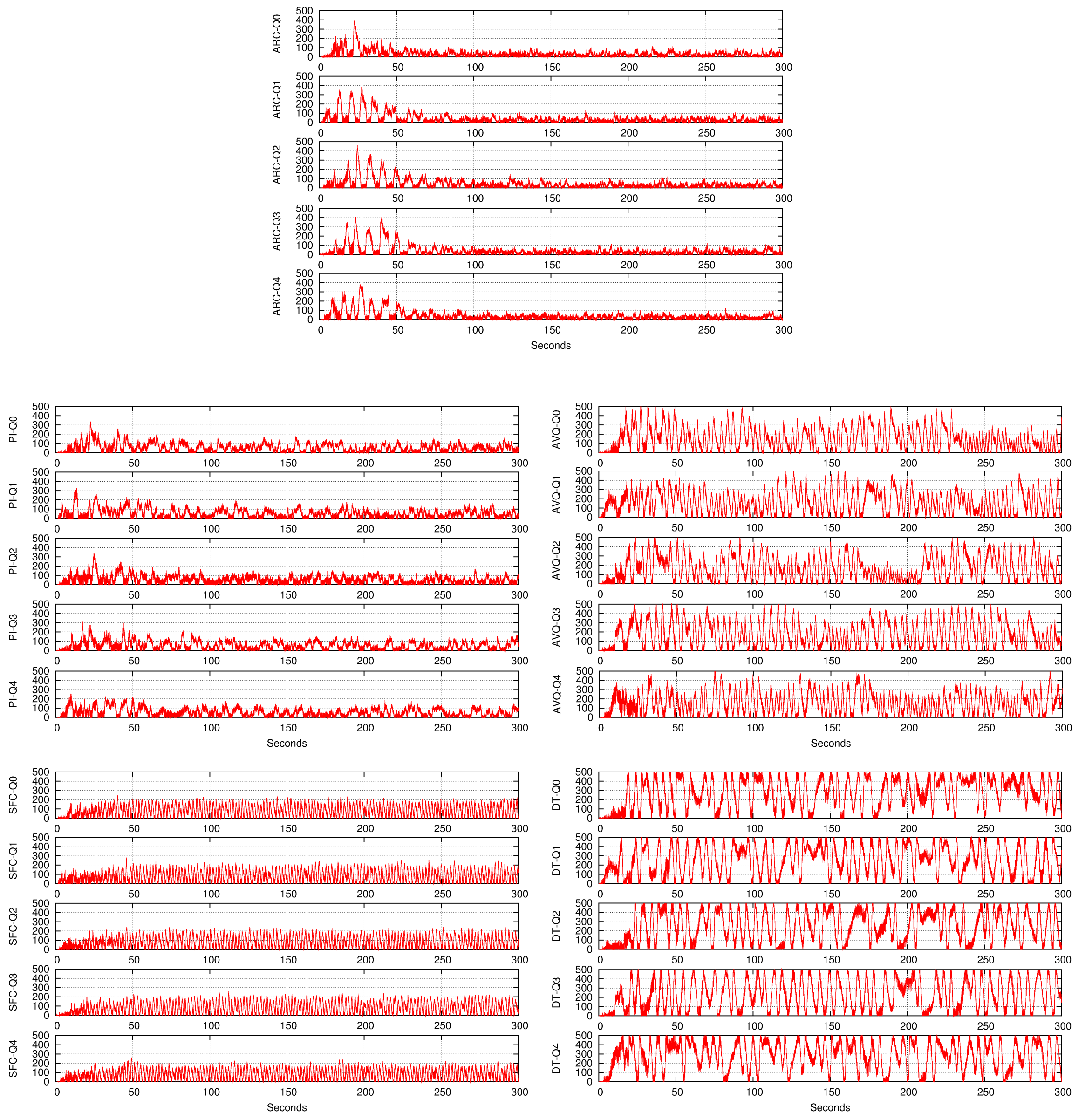


Figure 4.12: Queue Dynamics: Multiple Bottleneck Simulation (y-axes: queue size in KBytes)

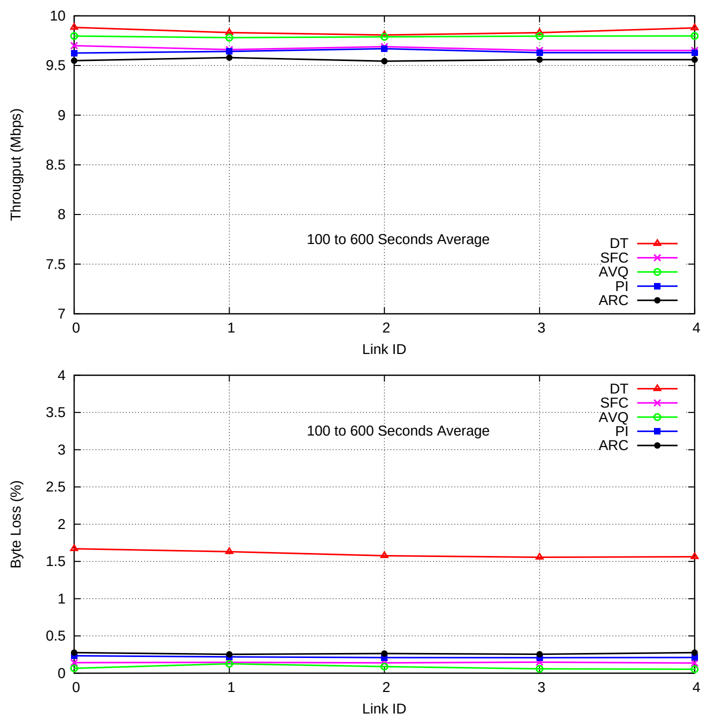


Figure 4.13: Network Statistics: Multiple bottleneck simulation

4.1.4 Extention to Support FAST

The ARC we have presented expects TCP with an additive increase multiplicative decrease (AIMD) source/network utility/congestion control protocol. This section discusses extending ARC to seamlessly support other similar utility control protocols, in particular, FAST [99] proposed to overcome the inefficiency of AIMD for high bandwidth-delay product networks.

Utility/congestion control protocols can either have a network centric control design, where network nodes explicitly determine utility (i.e., transmission rate) of individual traffic sources as in the eXplicit Congestion Protocol (XCP) [69], or have a distributed control design, where each individual traffic source determines its own utility using a demand function implicitly or explicitly chosen by the protocol and utility price of its network path implicitly or explicitly notified by the network. Most congestion control protocols proposed for the Internet, including various versions of TCP, TFRC [44], TCP Vegas [16] and FAST [99] are in the latter category.

Utility prices are determined based on a type of queuing delay along the network path: physical queuing delay for protocols with no explicit utility control support, and virtual queuing delay for protocols that assume the network provides the utility price. For example, in TCP or TFRC systems, utility price of a congested drop-tail link given in form of packet drop rate is determined implicitly by average queuing delay [25]. TCP Vegas, on the other hand, explicitly measures estimated queuing delay of congested drop-tail links in a network path for its utility price of the path. The utility price of a congested ARC or other PI-based AQM link, given in ECN marking probability, is computed based on the virtual queuing delay. Similarly, FAST, which has a demand control mechanism for traffic sources and a utility/congestion controller at network routers, uses virtual queuing delay over the virtual capacity at each link as the estimated price of the congested link.

The network utility price computation requirements being fundamentally the same for most distributed utility control protocols – a function of the network queuing delay – provides an opportunity for a single network to concurrently support similar distributed utility/congestion control protocols with little overhead. To illustrate this, we make a simple modification to the original ARC in Algorithm 2 to concurrently support FAST, which can be deployed in the current Internet without modification to IP specifications, as well as TCP.

Algorithm 3 shows the FAST-extended ARC, where q_v is an implementation of a virtual queue after physical queue error correction. The extended ARC uses $\alpha \times q_v$ as the link utility price for TCP traffic, which is basically the same implementation with the original ARC logic, and $q_v/(\gamma C)$ for FAST traffic. The only minor difference between this FAST utility price computation from [99] is that while the original FAST

Algorithm 3 FAST-Extended ARC

```
Every $d$ seconds:
 1: $q_v \leftarrow q_v + (b - \gamma(dC - (q - q_0)))$;
 2: $p_{tcp} = \alpha \times q_v$;
 3: $p_{fast} = \phi^{-q_v/(\gamma C)}$;
 4: $b \leftarrow 0$;
Every packet arrival:
 5: if ($typeof(packet) == fast$) then
 6:     $p = p_{fast}$;
 7: else
 8:     $p = p_{tcp}$;
 9: end if
10: if ($uniform(0,1) \leq p$) then
11:     if ($mark(packet) == false$) then
12:         $drop(packet)$;
13:         $return$;
14:     end if
15: end if
16: if ($enqueue(packet) == false$) then
17:     $drop(packet)$;
18:     $return$;
19: end if
20: $b \leftarrow b + sizeof(packet)$;
Functions:
  $mark(packet)$: ECN mark the packet. Return $false$ on error.
  $enqueue(packet)$: Enqueue the packet. Return $false$ if queue is full.
  $drop(packet)$: Drop the packet.
Variables:
  $p$, $p_{tcp}$, $p_{fast}$, $q_v$, $q$, $b$
Parameters:
  $C$: link capacity (bytes per second)
  $\gamma$: target link utilization ($\gamma = C_0/C$)
  $q_0$: target queue length in bytes
  $d$: measurement interval
  $\alpha$: TCP congestion feedback constant
  $\phi$: REM [5] communication constant shared by all FAST sources
```

algorithm uses pure virtual queuing delay with no physical queue error control (i.e. uses an integral controller) this new version considers physical queue error control as well and uses a PI utility/congestion controller. Replacing the integral utility controller with the PI controller should have little effect on stability of the FAST system, since adding a proportional control, in general, does not significantly affect stability of a system.

For congestion notification, TCP uses a plain ECN marking or packet dropping communication scheme. In contrast, FAST uses REM [5] encoding/decoding, an alternative way to use the single ECN bit in the IP header for congestion communication. In Algorithm 3, ϕ is the REM communication constant shared by all FAST sources and FAST-enabled routers in the system.

Thus, by keeping track of the fundamentally compatible link utility prices with the support for the two different congestion communication methods, this extended ARC can simultaneously support both TCP and FAST with little overhead. In order to make ARC support a distributed utility control without an explicit utility control at network routers, an extra step is required to modify the protocol to adapt a compatible network-evaluated link price instead of the source-estimated link price. The conversion issues are not discussed further, since it is out of scope of this research. Also, we leave stability analysis and evaluation of the modified FAST system as future work.

4.1.5 Summary

In this section, we present Aggregate Rate Controller (ARC), a reduced parameter proportional integral controller for Internet traffic. ARC resolves configuration difficulties that have crippled past AQM approaches by taking a well-understood and efficient proportional-integral controller design for AQM, carefully reducing the control parameters based on a sound understanding of Internet congestion control, and providing practical configuration guidelines though control engineering for a resilient performance under a wide range of traffic conditions. Also, by taking a low frequency, direct rate-based control information acquisition design rather than queue-based traffic rate estimation approach, ARC significantly reduces control noise with little extra overhead, and provides flexible QoS tunability.

ARC configured to minimize queuing delay makes a good candidate for the Crimson Congestion Controller offering best-delay-effort service to fulfill needs of various Internet applications. Our simulations demonstrate that by complying with the configuration guidelines, ARC can efficiently support a wide-range of traffic conditions, dampening queue oscillations, keeping queue low and throughput high, even when the configuration is not optimized for the current traffic. Overall, ARC out-performs PI, SFC and AVQ under all tested conditions in terms of queue dynamics, throughput, data loss rate and Web service time. Especially, ARC can provide significantly improved performance over PI for lightly loaded conditions, the norm for many Internet routers, as well as for sudden traffic load changes, owing to the low frequency rate-based control data acquisition design.

In addition to TCP support, ARC can be easily extended to support other similar distributed utility/congestion control protocols with little overhead, since most of the protocols use queuing delay-based link utility price estimation methods that are fundamentally compatible with one another. As an example, we extend ARC to concurrently support TCP and FAST [99], an IP compatible distributed utility control protocol proposed to overcome the inefficiency of AIMD for high capacity networks.

4.2 Stochastic Fairness Guardian for Bandwidth Fairness Protection

Emerging delay sensitive Internet applications such as streaming media and network games often prefer to use UDP over TCP as their transport protocol. As the use of non-TCP applications increases, the Internet must deal with more flows with improper or no end-to-end congestion control. Moreover, as Internet connection capacity provided by ISPs is significantly increased (up to 3 Mbps for typical cable modem services), use of high bandwidth demanding non-TCP applications such as broadcast quality streaming video become possible. This trend carries the potential for a significant imbalance in the public link capacities used by TCP and UDP flows. This imbalance threatens Internet stability and, in the worst case, an extrapolation of this trend could lead to Internet congestion collapse [14].

In this section, we introduce a novel statistical traffic filtering technique, called the Stochastic Fairness Guardian (SFG), that can effectively regulate misbehaving flows with minimal traffic state information. SFG offers misbehaving traffic management performance comparable to that provided by more complicated statistical flow monitoring mechanisms such as RED-PD [85] without requiring misbehaving flow identification. Such identification is computationally intensive and error prone, and thus may not be affordable for broadband network routers.

SFG uses a multi-level hash scheme that places incoming flows into different flow groups at each level and approximates a proper packet drop rate for each flow by monitoring the incoming traffic rates for the groups to which the flow belongs. SFG can be used in conjunction with a drop-tail queue as an effective network protection mechanism. When SFG is used in combination with an AQM congestion feedback controller, the combination can improve both network protection and efficiency. When TCP traffic is effectively controlled by the AQM, the interference between SFG and the AQM for TCP traffic can be minimized such that SFG serves only as a traffic filter for misbehaving, unresponsive flows.

SFG is evaluated in conjunction with Aggregate Rate Controller (ARC) presented in Section 4.1, and compared with RED-PD, SFB and CHOKe, and drop-tail queue management through simulations. The results show that SFG with drop-tail queue management provides simple and effective fairness protection that complements the weakness of drop-tail alone. Also, the combination of SFG and ARC, referred to as Stochastic Fair ARC (SFA), outperforms other mechanisms in terms of protection, stability, queuing delay and overall TCP performance over a wide range of realistic traffic mixes and loads that includes a few high bitrate CBR flows and many MPEG video-like VBR streams.

The remainder of this section is organized as follows: Section 4.2.1 describes the design of SFG and SFA; Section 4.2.2 develops an error model to predict the number of false positives for SFG and provides SFG configuration guidelines; Section 4.2.3 uses simulations to evaluate the performance of SFG and SFA along with other statistical flow management approaches; and Section 4.2.4 presents a summary and future work.

4.2.1 Design

Stochastic Fairness Guardian (SFG) is a highly scalable statistical traffic filter that uses a small amount of state information to provide stochastically fair network resource allocation and network protection. Using a pre-queue management mechanism, SFG preferentially drops incoming packets in proportion to a flow's approximated unfair resource usage. SFG can be deployed either with a drop-tail queue or with an AQM mechanism.

Algorithm 4 Stochastic Fairness Guardian (SFG)

```
Every $d_s$ seconds:
 1: for $i = 0$ to $L - 1$ do
 2:     for $j = 0$ to $N - 1$ do
 3:         $prob[i][j] \leftarrow (bytes[i][j] - d_s C/N)/bytes[i][j]$;
 4:         $bytes[i][j] \leftarrow 0$;    /* update drop_p for all bins */
 5:     end for
 6: end for
Every $packet$ arrival:
 7: $p = 1$;
 8: for $i = 0$ to $L - 1$ do
 9:     $j = hash(i, packet)$;
10:     $p = min(p, prob[i][j])$;   /* take min drop_p seen so far */
11:     $bytes[i][j] \leftarrow bytes[i][j] + sizeof(packet)$;
12: end for
13: if $(uniform(0, 1) \leq p)$ then
14:     $drop(packet)$;
15:     $return$;
16: end if
17: $queue(packet)$;
Functions:
  $hash(key, packet)$ Returns hash ($<N$) for given key and packet.
  $drop(packet)$: Drops the packet.
  $queue(packet)$: Passes the packet to the queue manager.
Variables:
  $prob[L][N]$, $bytes[L][N]$, $i$, $j$, $p$
Parameters:
  $C$: link capacity (bytes per second)
  $L$: number of levels
  $N$: number of bins in a level
  $d_s$: measurement interval
```

SFG defines a flow as an abstract entity that can be identified by a combination of source/destination address, protocol and port numbers. A closely related issue that makes flow monitoring and accounting challenging for approaches such as RED-PD [85] and SFB [36] is determining the lifetime of a flow. However, SFG does not need to monitor nor account for individual flows to filter traffic. Thus, in the rest of Section 4.2 the terms "incoming packet" and "incoming flow" are used interchangeably.

To approximate and regulate unfair network usage, SFG uses a multi-level traffic group management technique. SFG, shown in Algorithm 4, clusters incoming flows into N different traffic groups in each of L levels using an independent hash function for each level. Thus, SFG maintains N x L bins, where each bin in a level is assigned an equal share $(1/N)$ of the outbound link capacity (C). Every d_s second epoch, SFG computes and updates the packet drop probability for each bin $(prob[i][j])$ by taking the incoming traffic rate of the last measurment epoch $(bytes[i][j]/d_s)$ as an estimate of this epoch's packet arrival rate for the flows in the bin, and setting the drop probability such that no more than C/N capacity is used by a bin.

When a packet arrives, SFG looks up the packet drop probabilities for the L bins to which the packet belongs and applies the minimum drop probability to the packet. The motivation behind choosing the minimum drop probability is to protect TCP flows that share one or more accounting bins with other high bitrate flows. Figure 4.14 shows an example of SFG selecting drop probabilities for three different flows, where rounded-corner boxes represent the accounting bins and shaded boxes represent the bitrate of each flow. In this example, packets of $flow1$ are dropped with a probability $p = 0.03$ since it is the minimum drop probability of all the bins to which the flow belongs. Similarly, $flow2$ gets $p = 0.02$ and $flow3$ gets $p = 0.00$.

A potential drawback of using static hash filters for flow group assignments is that a well-behaving flow that has the misfortune of sharing all its bins with misbehaving flows can be unfairly treated for the lifetime of the flow. SFG eases this concern by a simple modification to Algorithm 4 such that two hashes in each level are used, one for the drop probability access for the current epoch and the other for the control data collection for the next epoch. In this way, a flow assigned to poluted bins in all levels in the current epoch can be re-hashed into different bins in the next epoch. This additional fairness enhancement is easily added to SFG, since SFG flow group managment for the current epoch is independent of previous epochs.

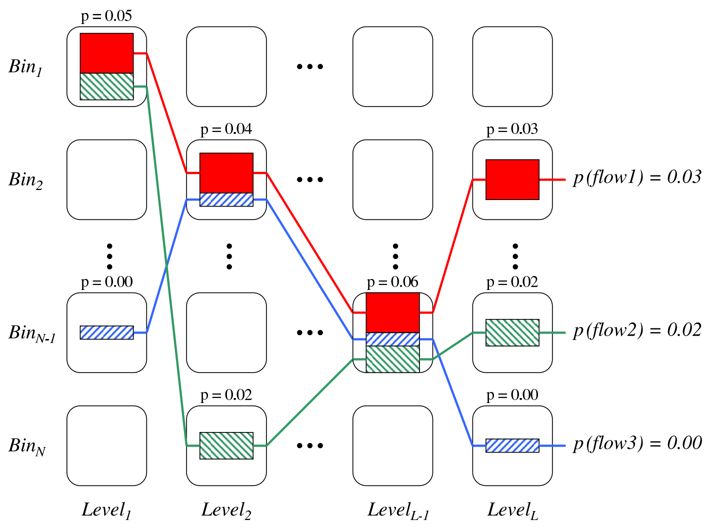


Figure 4.14: An Example SFG showing three flows. The size of the shaded blocks indicate the flow bitrates. The drop probability applied to each flow is indicated on the right.

SFG can be used both with a drop-trail queue or with an AQM. The combinaton of SFG with an AQM can enhance TCP performance by avoiding packet drops through the AQM while still providing network protection through SFG. To maximize the benefit of an SFG and AQM combintaion, a careful configuration of SFG and the AQM is required. Note that although SFG is not designed to be a congestion feedback controller for TCP traffic, it may perform the roll of an implicit (packet dropping) congestion feedback controller when faced with fewer than N bandwidth-hungry TCP flows, or when SFG is configured to offer lower link utilization than the following queue manager can offer. Under such circumstances, SFG may degrade TCP performance by interfering with the AQM congestion feedback control. The next section provides SFG configuration guidlines for setting L, N and d_s, and addresses issues associated with combining SFG with a queue manager to maximize preformance benefits.

SFG shares structural similarities with the Bloom filter technique used in SFB [36] in that both mechanisms use multi-level hashing to group flows. However, the major difference is that the Bloom filter in SFB is used as an unresponisve flow identification tool, while SFG uses the multi-level hash packet filter to prevent a few misbehaving flows from dominating the outbound link utilization. By periodically updating packet drop probabilities for accounting bins, SFG inherently has less overhead than does SFB with the Blue AQM [35] inside each accounting bin where the congestion notification probabilities of the relevent Blue bins are updated for every arriving packet.

4.2.2 Configuration

We develop a false positive model to estimate the probability of well-behaving flows being incorrectly identified by SFG as one of the misbehaving flows. Based on the model and performance considerations, we provide SFG configuration guidelines with a practical SFG integration mechanism that can be applied to both a drop-tail queue and an AQM to maximize the potential benefit of SFG.

An analytic model is developed to determine the false positive flow punishment ratio for SFG, i.e. how often a well-behaving flow is unfairly penalized because it shares all of its bins with misbehaving flows. Parameters in the model include: L - the number of levels supported by SFG , N - the number of bins in each level, and B - the number of misbehaving flows in the system. The first step is to determine the expected number of bins occupied by one or more misbehaving flows (referred to as polluted bins) in a level.

Let $T(B, i)$ be the number of ways to distribute B flows into i bins such that no bin is empty, where $B > i$. This is a well-understood probability problem that can be computed as follows:

$$T(B,i) = \sum_{k=0}^{i} (-1)^k \binom{i}{k} (i-k)^B \tag{4.10}$$

Let $P_w(N,B,i)$ be the probability that exactly i bins from the N total bins are polluted with B misbehaving flows. Computing P_w requires determining the total number of possible instances of the event. Let $W(N,B,i)$ be the number of ways to pollute exactly i bins from N total bins with B misbehaving flows. This is equal to the number of ways to choose i bins from N total bins and distribute B flows into the chosen i bins such that no bin is empty. Thus, $P_w(N,B,i)$ is determined by dividing $W(N,B,i)$ by the number of ways to put B flows into N bins. Thus, we have:

$$P_w(N,B,i) = \frac{W(N,B,i)}{N^B} = \frac{\binom{N}{i} T(B,i)}{N^B}$$

Let $E_w(N,B)$ be the expected number of polluted bins in a level, given N total bins and B misbehaving flows. E_w can be computed as the sum of each possible outcome number of polluted bins times its occurrence probability $P_w(N,B,i)$:

$$E_w(N,B) = \sum_{i=0}^{B} (i\ P_w(N,B,i))$$

Knowing $E_w(N,B)$, the false positive probability, $P_{fp}(L,N,B)$, that a well-behaving flow shares its bins in all levels with misbehaving flows and thus can be unfairly treated can be computed as:

$$P_{fp}(L,N,B) = \left(\frac{E_w(N,B)}{N}\right)^L = \left(\frac{1}{N^{B+1}} \sum_{i=0}^{B} i \binom{N}{i} T(B,i)\right)^L \tag{4.11}$$

Equation 4.11 can be used as a secondary SFG configuration tool to find an appropriate number of levels (L) that lowers the false positive error rate after configuring the number of bins per level (N), based on an expected maximum number of misbehaving flows ($\hat{B}$). A misbehaving flow, defined in this context, is flow that is not a TCP-friendly [43] flow, where a TCP-friendly flow is a flow with a data rate that does not exceed the maximum rate of a conformant TCP connection under the same network conditions. In practice it is difficult to determine if a flow is TCP-friendly. For example, a relatively low bitrate unresponsive flow that is classified as a TCP-friendly flow under light traffic loads can turn into a TCP-unfriendly flow at a higher load. Yet, the above definition is sufficient for the purposes of SFG. Once N is determined, the rate limit for misbehaving flow classification becomes apparent, i.e. C/N and $\hat{B}$ can be estimated.

A primary performance factor to consider in choosing N, the number of bins in a level, is the maximum per-flow bitrate that SFG will permit during congestion. Choosing N directly determines the maximum allowed per-flow bitrate (C/N) for a fixed capacity C. If N is too small, SFG will not effectively filter misbehaving flows that have a low data rate and will also have a high false positive flow punishment ratio. On the other hand, if N is too large, the small maximum allowed per-flow data rate can affect link utilization at low traffic loads dominated by a few greedy flows and prevent applications with bandwidth requirements larger than C/N from utilizing unused capacity.

One way to address this SFG configuration issue is to enable SFG only when the outbound link is congested, while carefully setting N such that the maximum allowed per-flow rate is small enough to effectively filter misbehaving flows and greater than or equal to a TCP-friendly rate [43] at the SFG enabling/disabling thresholds. This simple approach offers a static, maximum allowed per-flow rate during congestion regardless of the actual load level. A more sophisticated approach is to dynamically adjust N every control/measurement epoch using a TCP-friendly rate estimator. The TCP-friendly rate can be estimated using a TCP-friendly rate formula where the congestion notification rate (CNR) is measured at the router and the average system round-trip time is included as an extra SFG configuration parameter. This dynamic configuration approach is elegant but has increased complexity because the SFG hash functions will have to be adjusted frequently as N changes. This dissertation research explores the feasibility of the simple static on/off approach and leaves dynamic bin adjustment as future work.

To provide an on/off mechanism for SFG, a high/low watermark mechanism (m_h, m_l) for the average CNR estimate is used. The CNR estimate at the router is considered as a measure of the congestion level. To estimate the average CNR, SFG takes a weighted average on CNR every epoch, where CNR (p_n) is computed as the relational sum of the packet drop rate of SFG (p_d) and the congestion notification probability of the queue manager (p_e):

$$p_n = p_d + (1 - p_d)\, p_e \tag{4.12}$$

where, p_e can be measured in terms of the packet loss rate from queue overflow for a drop-tail queue, or explicitly reported by the AQM queue manager.

The SFG configuration process is illustrated by example. Setting $m_h = 0.02$ and $m_l = 0.01$, SFG assumes congestion when the CNR is over 2% and under 1%, respectively. The maximum allowed per-flow rate enforced by SFG at congestion can be determined by computing the low boundary TCP-friendly rate at the low watermark using a rate formula from [43]:

$$T_{tcp} \le \frac{1.5\sqrt{2/3}\, S}{\tau\sqrt{p_n}} \tag{4.13}$$

where, T_{tcp} is the upperbound TCP-friendly rate, S is average packet size and τ is estimated system round-trip time. By setting $S = 1500$ bytes, a typical MTU, and $\tau = 300$ ms, a value chosen from a valid range of average round-trip times [20, 67], T_{tcp} is 0.5 Mbps. To achieve this maximum allowed per-flow rate, SFG should set $N = 20$ (C/T_{tcp}) for a 10 Mbps output link.

After configuring N, the minimum number of levels (L) that can provide an optimal false positive error rate can be determined using Equation 4.11 given a range of the expected number of misbehaving flows $(\hat{B})$. A reasonable $\hat{B}$ can be estimated based on an Internet measurement study [88] that reports about 10% of the traffic is UDP traffic. Based on this statistic, an average of 1 Mbps UDP traffic is expected for a 10 Mbps link. Assuming all UDP bandwidth is potentially misbehaving, medium quality video, the typical bitrate will be about 300 Kbps. Thus, $\hat{B}$ is about 3 to 4 misbehaving flows for a 10 Mbps link. Assuming the UDP flows are low quality 56 Kbps video streams, a 10 Mbps link may carry as many as 17 misbehaving flows.

Figure 4.2.2 plots the false positive error rates of an $N = 20$ system, varying L for $\hat{B} = 1, 5, 10, 15$, to find the number of levels that reduces the per-packet processing overhead from hashing as well as the false positive error rates. Figure 4.2.2 shows that $L = 3$ provides both a low packet processing overhead and a low false positive error rate for the selected range of $\hat{B}$. For example, $P_{fp}(3, 20, 5) \approx 0.01$ and $P_{fp}(3, 20, 10) \approx 0.06$ indicates that the chances that a well-behaving flow is unfairly treated in an epoch by SFG with $L = 3$ and $N = 20$ is about 1% when $\hat{B} = 5$ and about 6% when $\hat{B} = 10$. Similarly, $P_{fp}(3, 20, 15) \approx 0.15$ shows that the chosen SFG setting can also offer relatively low false positive error rates for the higher range of $\hat{B}$.

Consider now the router memory requirement for SFG. Assuming each bin requires a 4-byte integer for counting bytes received and a 8-byte double-precision

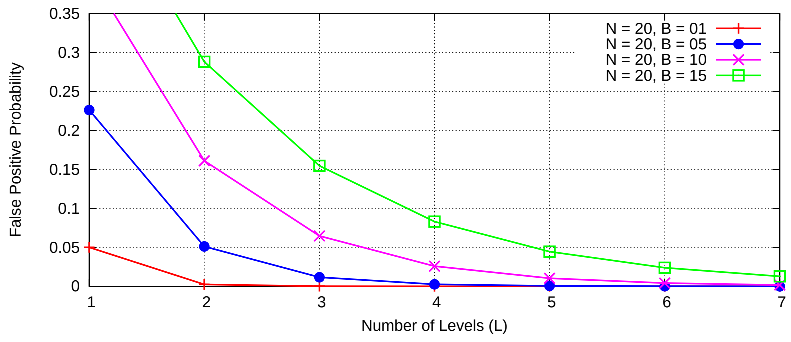


Figure 4.15: False Positive Probability (N = 20)

floating number for storing the drop probability, the memory requirement for a 10 Mbps SFG link with $L = 3$ and $N = 20$ is 720 bytes per output port (3 levels $\times$ 20 bins $\times$ 12 bytes/bin). Similarly, a 10 Gbps link with an equivalent SFG setting of $L = 3$ and $N = 20,000$ requires only 720 Kbytes of memory per output port.

The last SFG parameter to discuss is the control/measurement epoch length (d_s). We recommend setting d_s to a couple of seconds (d_s is set to 2 seconds in this investigation), such that it is approximately twice the upperbound average round-trip time seen on the Internet [20, 67]. This avoids control error due to insufficient control data acquisition and minimizes congestion control interference with the AQM controller. Since potentially misbehaving flows such as streaming media and network games often have long lifetimes, the large epoch length, and hence slow response time, is acceptable. A more responsive system would pay a high price in terms of fairness and efficiency for packet drops caused by inaccurate SFG control.

4.2.3 Evaluation

We compare the performance of SFG (with drop-tail queue management) and the combination of SFG and ARC, referred to as Stochastic Fair ARC (SFA), with RED-PD [85], SFB [36], CHOKe [91] and drop-tail through detailed simulations. The simulations attempt to incorporate realistic traffic conditions by including long-lived FTP flows (that vary in number over time to induce a range of offered loads), background Web traffic, and 2-way traffic (which can result in ack compression). The IP packet simulator NS is used for all simulations. The NS distribution comes with source code for RED-PD and makes available the source code for SFB as contributed code. We extended NS to support CHOKe, ARC, SFG and SFA.

The simulations model a dumbell network topology with a bottleneck link capacity of 10 Mbps and a maximum packet size of 1000 bytes. Round-trip link delays are randomly uniformly distributed over the range [60:1000], based on measurements in [67]. The physical queue limit for each AQM and the drop-tail queue is set to 500 Kbytes, approximately equal to the bandwidth-delay product for the mean round-trip time.

The settings for the parameters of the various statistical preferential drop and AQM mechanisms are based on the recommendations of their authors (see Related Work in Section 2.1.2.2 for details on other preferential-based dropping mechanisms). The settings for RED have minimum and maximum thresholds of 50 and 300 respectively, maximum drop probability of 0.15, weighted average factor $W_q = 0.002$, and the gentle option enabled. The additional RED-PD settings include: a target system round-trip time of 100 ms that is used to determine the epoch length for monitoring and for the TCP-friendly rate, flow monitor history window of 5, minimum time to un-monitor a monitored flow and its drop rate threshold of 15 seconds and 0.005, respectively, and maximum drop probability increment step of 0.05.

CHOKe, which works in conjunction with RED, is set to divide RED's minimum and maximum queue threshold range into 5 even subregions and apply $2i + 1$ drop comparisons for an incoming packet, where $i = \{0, 1, 2, 3, 4\}$ is the subregion ID.

For SFB, the number of levels and bins are set to $L = 3$ and $N = 20$, the unresponsive detection CNP threshold is set to 0.98, and the penalty box time is set to 15 ms. SFB is set to switch hash functions every 20 seconds. For the Blue AQM inside each SFB bin, the CNP increment step is 0.005 and the decrement step is 0.001 with a freeze time of 100 ms.

The settings for ARC include the measurement epoch $d_a = 1$ second, $\alpha = 1.42 \times 10^{-5}$, the target utilization $\gamma = 0.98$ and the target queue $q_0 = 0$. For SFG, based on the analysis made in the previous section, the on/off thresholds are $m_h = 0.02$ and $m_l = 0.01$, the control/measurement interval $d_s = 2$ seconds, the number of levels $L = 3$ and the number of bins $N = 20$.

All simulations use ECN enabled NewReno TCP for both the long-live FTP flows and the Web sessions. Each simulation has 50 backward direction bulk transfer FTP flows and 300 forward direction background Web sessions (using the Webtraf code built into NS) that start evenly distributed during the first 30 seconds. Based on settings from [4, 55], each Web session requests pages with 2 objects drawn from a Pareto distribution with a shape parameter of 1.2 and an average size of 5 Kbytes. The Web sessions have an exponentially distributed think time with a mean of 7 seconds, which results in an average utilization of about 2.5 Mbps of the 10 Mbps capacity, a fraction typical of some Internet links, such as in [115]. Each simulation has forward direction bulk transfer FTP flows. To test the various mechanisms under different traffic loads, the number of forward direction FTP flows varies every 200 simulation seconds from 10-50-100-200-400 flows and then back down from 400-200-100-50-10 flows.

To more intuitively characterize the degree of congestion experienced by the link beyond simply the number of flows, the drop-tail queue simulation with the above network settings was run with only the Web traffic, varying the number of Web session from 1200 to 1800, and recording the packet drop rate for each load. Sub-

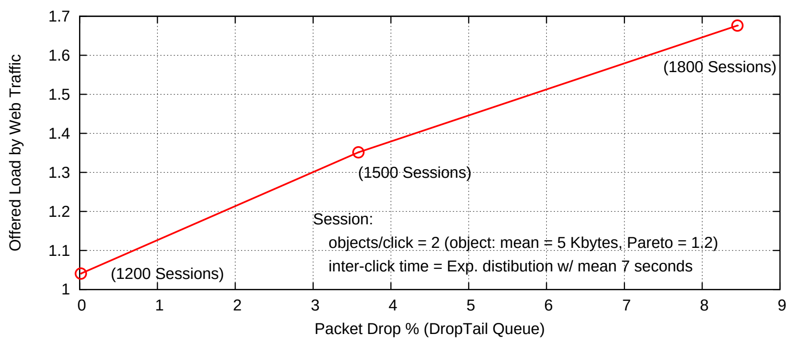


Figure 4.16: Offered Load by Web Traffic versus Packet Drop Rate (drop-tail queue: qlim = 500 Kbytes)

sequently, the congested link bandwidth was changed from 10 to 100 Mbps and the simulation re-run to measure the offered traffic rate for each number of Web sessions under a capacity unconstrained condition. The offered traffic rates were then converted to offered loads in relation to the 10 Mbps link capacity, and plotted in relation to the packet drop rates measured for the same number of Web sessions under the 10 Mbps link. Figure 4.2.3 shows the linear relationship between the drop-tail packet drop rate and offered load.

The offered loads given as a function of the drop-tail packet drop rates are useful for characterizing the load created by the TCP traffic mix (i.e., forward direction FTP, backward direction FTP, and background Web traffic), by converting the load of the mixed TCP traffic expressed in terms of the number of FTP flows into the equivalent Web offered traffic load expressed in terms of the packet drop rates of a drop-tail queue. Thus, the equivalent offered loads for the TCP traffic mix when the number of forward direction FTP flows is 10, 50, 100, 200 and 400 are about 1.0, 1.1, 1.2, 1.4 and 1.7 respectively. This means, for example, that when the number of FTP flows is 400, the congested link is experiencing about 1.7 times the offered load it can handle without having to drop any packets.

While an offered load of 1.7 is probably beyond any realistic load for most routers on today's Internet, this high load serves as a stress test of the various preferential dropping and AQM mechanisms, showing insight into how they handle the traffic in terms of fairness, throughput, stability, queuing delay, packet and byte loss rate, and Web performance. RED-PD, CHOKe, SFB, SFG and SFA are evaluated using the TCP traffic mix, in comparison with drop-tail and ARC, queue management mechanisms without a preferential dropping mechanism. The next set of simulations includes one unresponsive 10 Mbps CBR flow added to the TCP traffic mix, replacing the one unresponsive flow with five unresponsive 2 Mbps CBR flows. The last set of simulations replaces the five unresponsive flows with five unresponsive MPEG video VBR flows.

4.2.3.1 TCP Traffic Mix

As a baseline, this experiment compares the performance of the various statistical preferential drop mechanisms with that of a drop-tail (DT) and ARC over the range of loads with the TCP traffic mix (ie - no unresponsive flows). Figure 4.17 shows the queue dynamics (top) and system throughput (bottom) of DT, ARC, SFB, CHOKe, RED-PD, SFG and SFA. The byte loss rate and packet drop rate are shown in Figure 4.18, and the average Web object service time for each system is shown in Figure 4.19.

First, comparing the queue dynamics, throughput and byte loss rate of drop-tail with ARC verifies the benefits of ARC. ARC is able to stably control traffic over the entire load range, keeping the queue length low at around 100 Kbytes, and maintaining a high link utilization. ARC loss rates are low (less than 2%), even at the offered load of 1.7, resulting in a higher goodput than the drop-tail system. The low and stable queue length is desirable to concurrently support QoS needs of

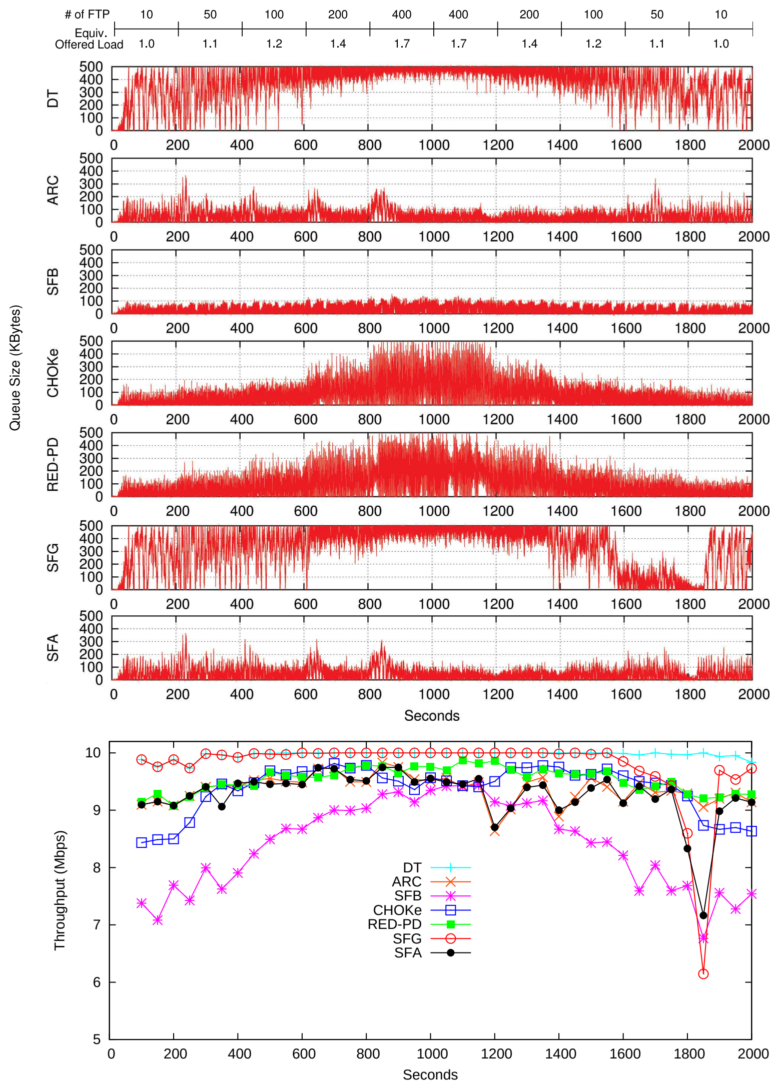


Figure 4.17: TCP Traffic Mix - Queue Dynamics (top) and Throughput (bottom)

various applications, and can also greatly reduce the buffer size required to achieve a high link utilization [3].

The packet loss rate and the average Web object service time of drop-tail and ARC show some of the less well-known performance aspects of ECN when viewed over the range of traffic loads. Although the network efficiency measured in byte loss rate is consistently better for ARC, the packet losses for ARC, a combination

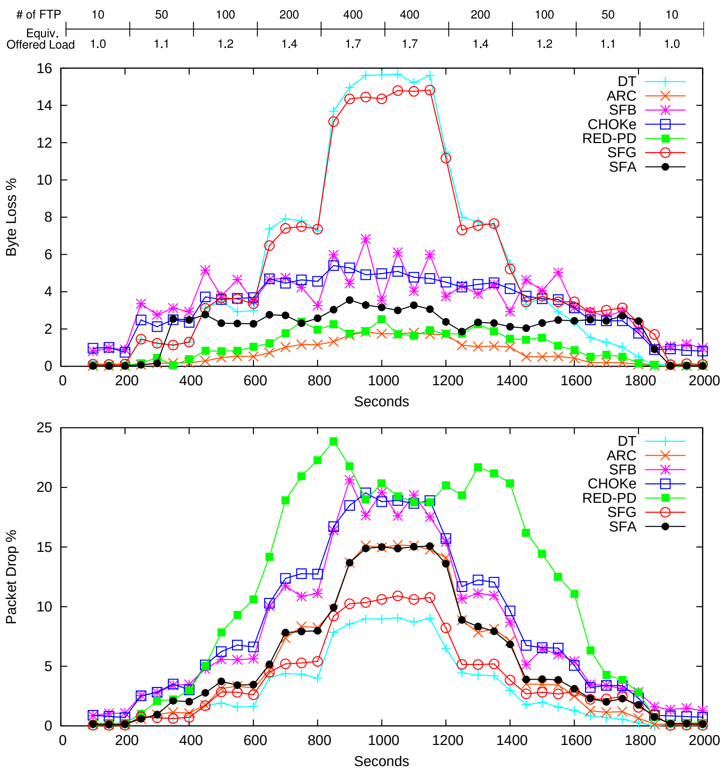


Figure 4.18: TCP Traffic Mix - Packet Drop Rate (top) and Byte Loss Rate (bottom)

of ECN-incapable SYN packets for the Web traffic and the backward direction TCP ACK packets, are about twice as high as that of drop-tail as the traffic load increases beyond an offered load of 1.2. To control traffic at congestion, ARC, and more generally any AQM, using ECN must maintain a higher congestion notification probability (CNP) than the packet-drop congestion notification rate of a drop-tail queue. As a result, ARC, by dropping non-ECN packets with the CNP, favors ECN-capable packets over non-ECN packets especially at high traffic loads, yielding a higher packet drop rate than drop-tail for conditions in which small, non-ECN enabled packets dominate.

This phenomenon creates the Web object delivery performance crossover for the drop-tail and ARC systems as the offered load changes from 1.2 to 1.4, at which

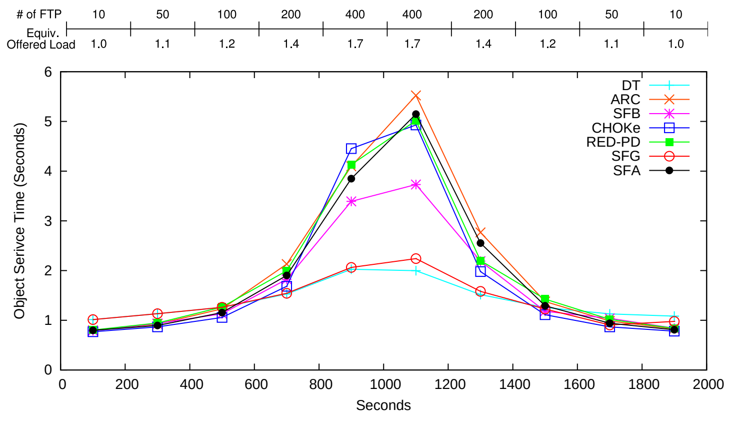


Figure 4.19: TCP Traffic Mix - Average Web Object Service Time

point the initial TCP timeout for SYN packet drops becomes the dominating factor for Web object service times. At the peak load of 1.7, the average Web object service time for ARC is about 5 seconds, while the Web object service time for drop-tail is about 2 seconds. For the traffic load ranges below 1.2, the Web performance results are consistent with the experimental measurement results from [77], showing AQM with ECN can significantly benefit Web traffic at offered loads from 0.9 to 1.0. In contrast, for traffic load ranges above 1.2 or 1.3, Web performance can be significantly degraded by AQM with ECN, although such high loads are uncommon in practice.

The various performance measures of SFG (with drop-tail queue management) shown in Figure 4.17, Figure 4.18 and Figure 4.19 closely match those of drop-tail, indicating that SFG, activated from 300 to 1900 seconds, works well with drop-tail queue management for the TCP traffic mix. Similarly, the performance of SFA closely matches that of ARC except for the slightly higher byte loss rates, indicating SFG interfered little with the ability of ARC to control TCP traffic. The sharp link utilization drops for SFG and SFA between 1800 and 1900 seconds are due to the maximum allowed per-flow rate of SFG being less than the TCP-Friendly rate as the system load goes down. However, both SFG and SFA detect the decrease in load within a minute and then turn off the fairness enforcement mechanism.

Comparing the performance of SFB, CHOKe and RED-PD with that of SFA in Figure 4.17 and Figure 4.18, SFB, CHOKe and RED-PD have consistently higher packet drop rates and byte loss rates than SFA, except for RED-PD's slightly lower byte loss rate caused by RED-PD's higher operating queue length. Yet CHOKe, which works in conjunction with a RED controller, was not able to benefit from this higher RED queue length due to its rather inefficient statistical preferential dropping mechanism, and had a low average throughput of 8.5 Mbps with the low offered loads during the first and last 200 seconds. Throughout the simulation, SFB suffers from low link utilization caused by the inefficient rate control afforded by Blue for the traffic mix.

4.2.3.2 An Unresponsive, High-Bitrate Flow

In this set of simulations, one unresponsive 10 Mbps CBR UDP flow is added to the TCP traffic mix used in Section 4.2.3.1, starting at time 100 seconds and stopping at time 1700 seconds in order to test the performance of the different preferential dropping mechanisms. For comparison, the performance of drop-tail and ARC are examined to determine the impact of the unimpeded CBR flow. Figure 4.20 shows the queue dynamics, Figure 4.21 shows the system throughput (top) and the throughput of the single CBR stream (bottom), and Figure 4.22 shows the average Web object service time of the systems.

Figure 4.20 shows that the drop-tail queue remains full from the time the high-bitrate CBR flow starts until it stops and Figure 4.21 (bottom) shows that drop-tail is very unfair as about 95% of the link capacity is used by the CBR flow. The average Web service time for drop-tail ranges from about 50 to 300 seconds, too high to be seen in Figure 4.22.

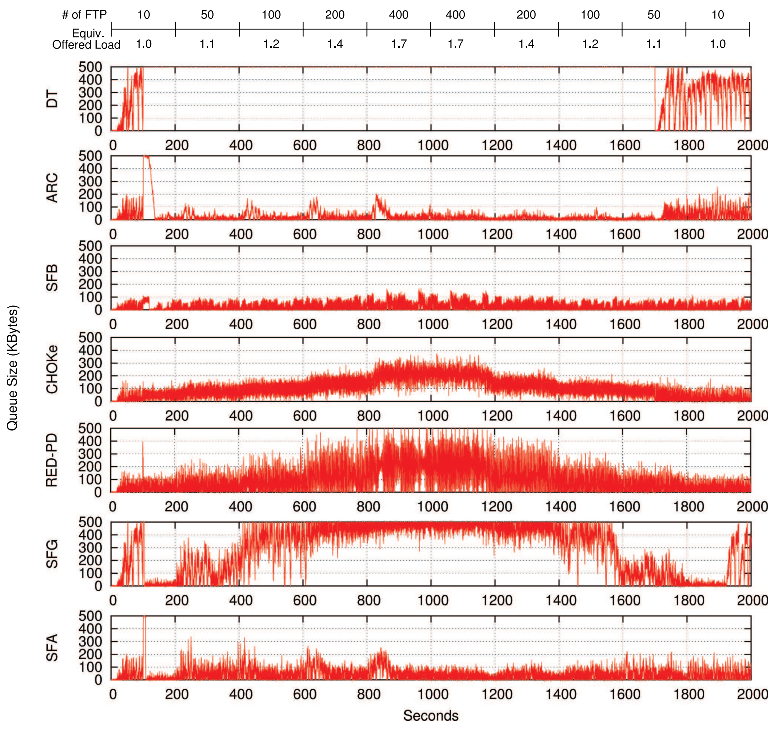


Figure 4.20: An Unresponsive High-Bitrate CBR Flow - Queue Dynamics

ARC controls the aggregated traffic and the unresponsive flow better than drop-tail, applying a high congestion notification probability (CNP) to drop the UDP packets while marking the ECN-enabled packets. Furthermore, as shown in Figure 4.20, ARC is still able to keep the queue length consistently low while maintaining a high link utilization even in the supersaturated conditions. However, like drop-tail, ARC is unfair, and the Web traffic experiences high service times, ranging from about 2 to 13 seconds throughout the simulation.

Figure 4.21 (bottom) shows that SFB is able to effectively handle the high-bitrate CBR flow, reducing its achieved bandwidth to the target rate. Yet, as in the case of the TCP-only traffic mix, SFB experiences severe link underutilization. CHOKe, using its statistical filtering mechanism, is able to regulate the unresponsive, high-bitrate flow. However, CHOKe's fairness is coarse as CHOKe heuristically increases

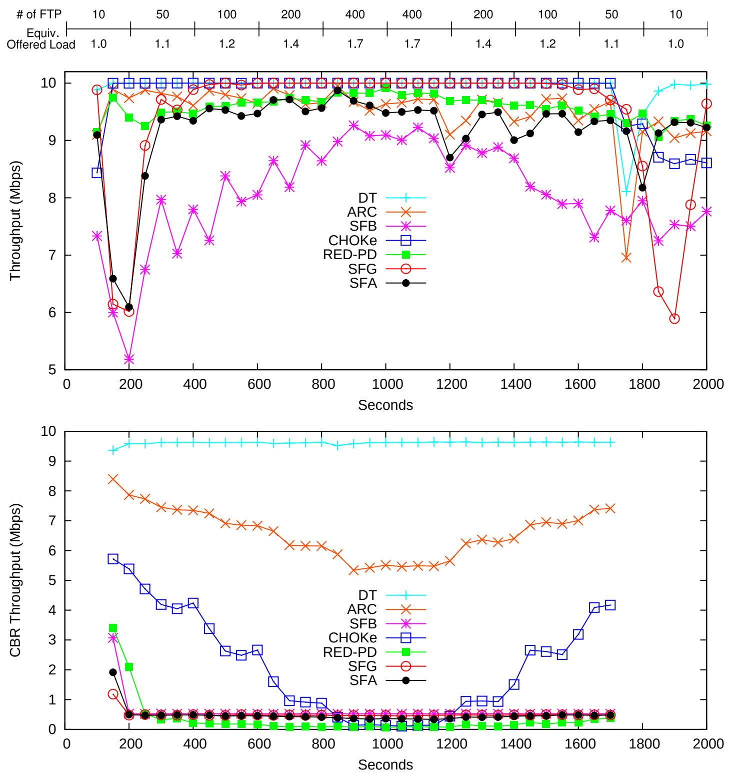


Figure 4.21: An Unresponsive High-Bitrate CBR Flow - System Throughput (top) and CBR Throughput (bottom)

the number of random match drops for each incoming packet as the load increases. RED-PD is able to effectively regulate the high-bitrate CBR flow by monitoring and then restricting the flow to no more than the periodically adjusted TCP-friendly rate. As observed from the queue dynamics of both the RED AQM based CHOKe and RED-PD, by frequently adjusting the maximum allowed flow rate for the CBR flow, RED-PD affects the stability of the RED congestion control, causing the queue in RED-PD to oscillate more than the queue in CHOKe.

Similarly to SFB and RED-PD, both SFG (with drop-tail) and SFA (SFG + ARC) are able to effectively restrict the rate of the unresponsive high-bitrate CBR flow to the target maximum. Yet, Figure 4.21 (top) shows that a high-bitrate, unresponsive flow in lightly loaded traffic conditions can degrade link utilization

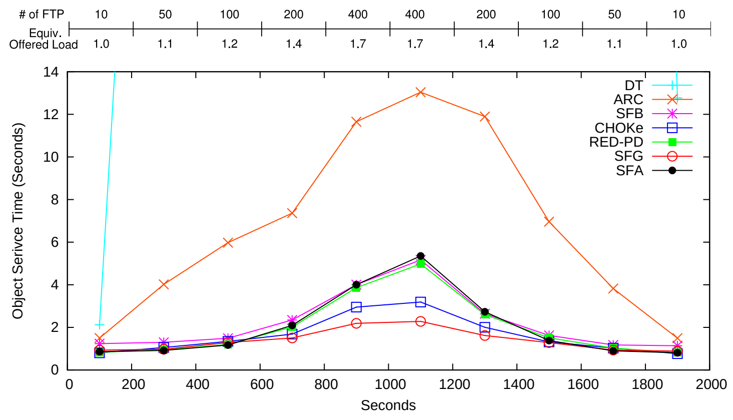


Figure 4.22: An Unresponsive High-Bitrate CBR Flow - Average Web Object Service Time

by forcing a pre-configured target rate to be imposed on all incoming flows. This potential shortcoming can be relaxed by dynamically adjusting the configuration of N, as briefly discussed in Section 4.2.2. Finally, the consistently stable and low queue dynamics in Figure 4.17 (top) of Section 4.2.3.1 and Figure 4.20 show that the statistical filtering mechanism of SFG does not noticeably affect the congestion control of ARC.

4.2.3.3 Multiple Unresponsive, Medium-Bitrate Flows

For the simulation in this section, the one 10 Mbps unresponsive CBR flows used in Section 4.2.3.2 is replaced with five unresponsive 2 Mbps CBR flows. As in the previous simulation, the unresponsive CBR flows are started at 100 seconds and stopped at 1700 seconds. Figure 4.23 shows the system throughput (top) and the average throughput of the five unresponsive CBR streams (bottom), and Figure 4.24 shows the average Web object service times. The queue dynamics are not shown, since they are very similar to the queue dynamics in Figure 4.20 in Section 4.2.3.2 and Figure 4.25 in the next section.

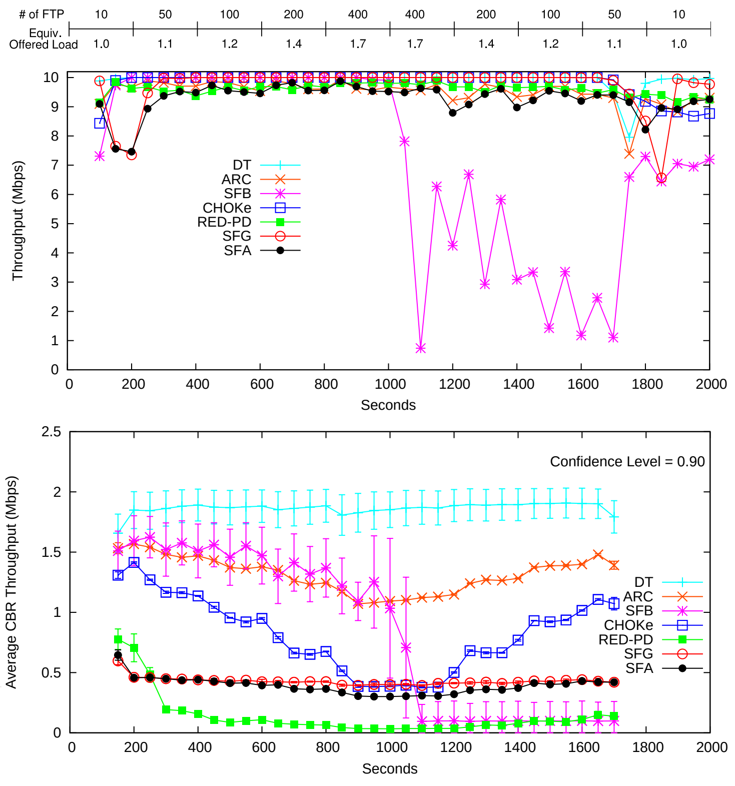


Figure 4.23: Multiple Unresponsive Medium-Bitrate CBR Flows - System Throughput (top) and CBR Throughput (bottom)

The effect of five unresponsive 2 Mbps CBR flows on the performance of drop-tail and ARC is similar to that of a single unresponsive 10 Mbps CBR flows. However, the five smaller capacity flows cause a remarkable degradation in performance for SFB. Figure 4.23 (bottom) shows SFB fails to detect the unresponsive medium bitrate flows until the offered load reaches 1.7, and performs even more unfairly than ARC, despite ARC not having any fairness protection mechanisms. Moreover, when SFB finally detects the five unresponsive streams and restricts their rate by putting them into a penalty box, there is significant link underutilization, since SFB fails to lower the congestion notification probability (CNP) accordingly. SFB's failure to properly adjust the CNP when there is an increase in the available capacity is also apparent in the Web object service time, shown in Figure 4.24, that is similar to or larger than

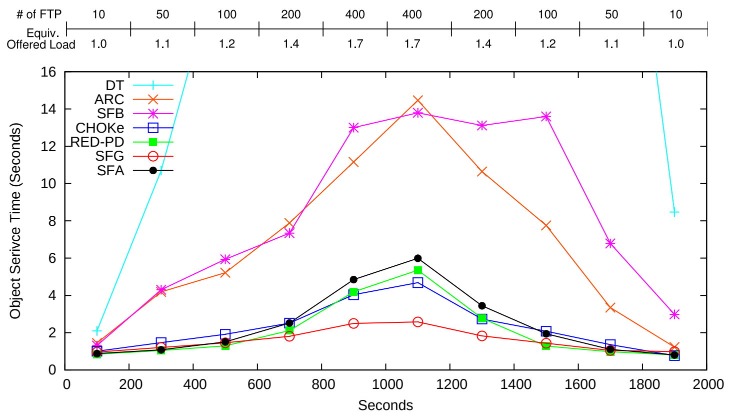


Figure 4.24: Multiple Unresponsive Medium-Bitrate CBR Flows - Average Web Object Service Time

that of ARC for the second half of the simulation.

Other preferential dropping mechanisms perform as designed. RED-PD effectively regulates each unresponsive flow to the estimated TCP-friendly rate at each control epoch, and SFG and SFA prevent each flow from using more bandwidth than the pre-assigned target rate of 0.5 Mbps.

4.2.3.4 Multiple Unresponsive, MPEG Video Streams

In order to test SFB, CHOKe, RED-PD, SFG and SFA with more realistic unresponsive flows, the five unresponsive 2 Mbps CBR flows used in Section 4.2.3.3 are replaced with five unresponsive MPEG-like video streams, implemented using tools from [22]. The MPEG streams have average I-, P- and B-frame sizes of 11, 8, and 2 Kbytes, respectively based on a trace of typical MPEG-1 video, sent at 30 frames per second for an average bitrate of slightly over 1 Mbps.

As in the previous experiments, the five MPEG streams are started at 100 seconds to stopped at 1700 seconds. For completeness, this section includes the same set of performance results as Section 4.2.3.2. Figure 4.25 shows the queue dynamics of the

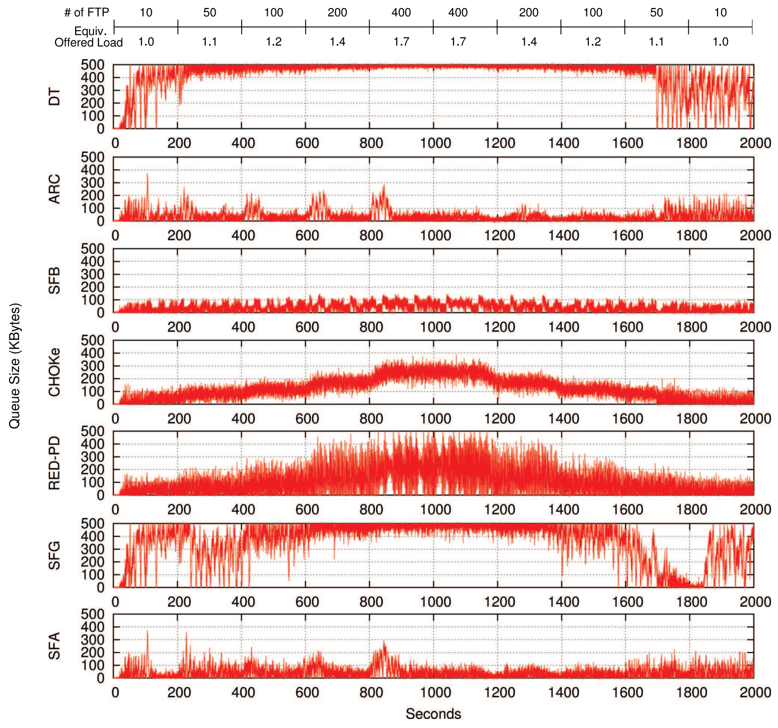


Figure 4.25: Multiple Unresponsive MPEG Video Streams - Queue Dynamics

different systems, Figure 4.26 shows the system throughput (top) and the average throughput of the five MPEG streams (bottom), and Figure 4.27 shows the average Web object service times.

As in the case with five unresponsive 2 Mbps CBR flows, for the MPEG streams, all preferential dropping mechanisms perform well, except SFB which again fails to detect the unresponsive flows. Figure 4.25 and Figure 4.26 (top) show SFA performs best in terms of queue length, stability and control of link utilization. Considering the Web performance in Figure 4.27, at an offered load of 1.2 or less, all preferential drop AQM mechanisms outperform drop-tail and ARC. When the offered load goes beyond 1.2, drop-tail performs better than all ECN-based mechanisms, and SFG with drop-tail queue performs best.

From Figure 4.26 (top), after 1700 seconds, the SFG and SFA's turning on/off

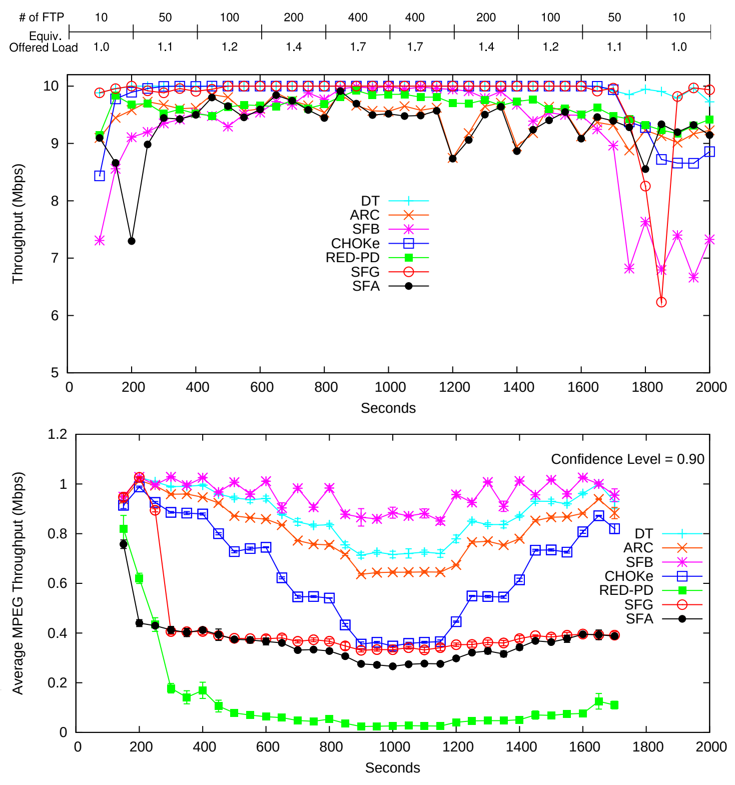


Figure 4.26: Multiple Unresponsive MPEG Video Streams - System Throughput and MPEG Throughput

mechanism in response to load is more accurate when used with ARC than with drop-tail. The turn on/off decisions are based on the estimated congestion notification rate (CNR) of the system which can be explicitly informed by ARC, whereas for the drop-tail queue, SFG has to measure the packet overflow rate to estimate the CNR.

In terms of fairness enforcement in Figure 4.26 (bottom), RED-PD, with its heavy-weight TCP-friendly target flow rate computation, performs best followed by SFG and SFA. For some additional complexity, the fairness control performance gap between RED-PD and SFG and SFA could be reduced by dynamically adjusting the target flow rate, as briefly discussed in Section 4.2.2 (and left as future work). However, even without this adjustment, the main goal of SFG and SFA has been met, and that is not to strictly enforce TCP-Friendly fairness but rather provide

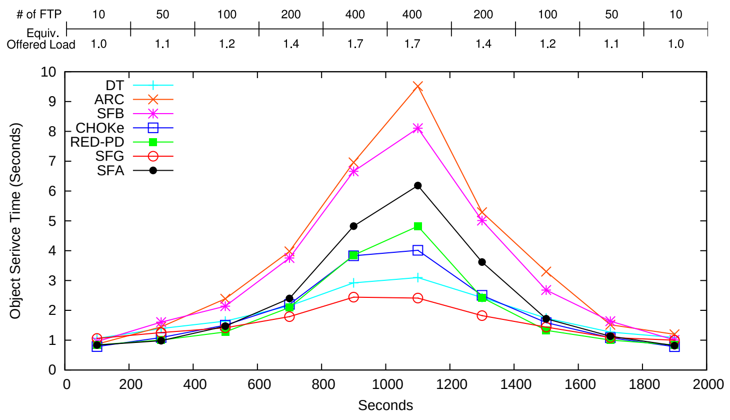


Figure 4.27: Multiple Unresponsive MPEG Video Streams - Average Web Object Service Time

reasonable protection from egregiously unresponsive flows.

4.2.4 Summary

In this section, we present a novel statistical packet filtering mechanism, Stochastic Fairness Guardian (SFG), which protects flows that respond to congestion from unresponsive flows through preferential packet drops. SFG, the Crimson Bandwidth Controller, is a lightweight and general packet filter that can be used in conjuction with a drop-tail queue or with an AQM to improve efficiency as well as provide protection. We also provide practical SFG configuration guidelines through performance bottleneck analysis and the false positive rate analysis.

In evaluation, SFG is integrated with a drop-tail queue and with ARC forming Stochastic Fair ARC (SFA) that is the building block the Crimson Internet. SFG and SFA are evaluated through simulations and compared with other preferential drop mechanisms including SFB [36], CHOKe [91] and RED-PD [85], and also compared with approaches with no filtering mechanisms including ARC and drop-tail. Performance metrics include queue dynamics, throughput, fairness, byte loss rate, packet drop rate and Web object service time.

Considering overall performance and design complexity, SFA outperforms other preferential dropping mechanisms as well as drop-tail and ARC over a wide, practical range of traffic loads. We also show that Congestion Controllers using ECN, while improving Web performance under light or moderate congestion, can degrade small Web object service times versus drop-tail for offered loads of 1.4 or higher. In such a high traffic load, SFG with drop-tail queue management shows the best Web performance.

4.3 Crimson Conclusions

In this chapter, we presented the design, configuration and evaluation of the two practical Crimson AQM mechanisms, ARC and SFG, and the integration referred to as SFA. Through simulations, we showed that SFA efficiently handles congestion, minimizes queuing delay, and provides affordable fairness protection from misbehaving flows over a wide range of realistic traffic conditions.

There is a concern that loading an IP router with more functionality like AQM may further slow down the packet processing speed of the router. This statement is true in general, since router access speed is an order of magnitude slower than the link speed, and the bottleneck may often be at the network processor rather at the link. The packet processing speed issue can be addressed using parallelism and pipelining in the process architecture, which is an approach already taken in some network processor designs today.

For Crimson AQM, computing the congestion notification probability (p) is the most computationally intensive function in the ARC design. However, since ARC computes p every 1 or 2 seconds (the recommended measurement epoch length), there will be little slow down in the packet processing speed. In case of SFG, the multi-hashing is the most computation intensive function. With integer hashing, SFG can perform hash through pipelined and parallel processing units each of which performs a number of simple bitwise operations. This will speed up the hashing process to the speed of performing a set of bitwise operations.

An related argument is that bandwidth over-provisioning without AQM would be enough to make a low delay network and be free from the threats of misbehaving traffic. This argument may be generally true for core network routers, yet may not hold for edge routers of 2nd or 3rd tier Internet service providers (ISP) and end-user Internet connections. Customers' demand for bandwidth grows as the provided Internet connection speed increases, which may quickly saturate the routers of lower tier ISPs. If not the lower tier ISPs, the end-user Internet connection links shall often face congestion. In addition, with the popular use of streaming media and peer-to-peer (P2P) file sharing applications, lower tier ISPs or end-users may want to control the network usage of specific traffic to achieve a certain performance goal. Even homes and small companies share an Internet connection link with multiple users and applications. End-users may often want to use P2P file transfers as they speak on the Internet phone, in which case the users may experience increased delays and degrade quality in the voice conversation. The Crimson AQM at the edge routers will help meeting quality of service (QoS) requirements of the delay sensitive applications.

Chapter 5

End-System Support for Streaming Media

This chapter first presents an Internet video streaming measurement study in Section 5.1. The study measures network layer and application layer performance of a commercial Internet video stream over TCP and UDP, and characterizes media streaming requirements. During the measurement study, we identify shortcomings of TCP as a streaming transport protocol: 1) TCP hides network information, such as packet loss rate and round-trip delay, needed for efficient available bandwidth estimation for media scaling. 2) The penalty of overestimating available bandwidth using TCP is severe due to the large TCP sender buffer, put in place to maximize throughput. As an effort to enhance transport protocol support for media streaming based on the findings, we modify TCP to provide Multimedia Transport Protocol (MTP) for multimedia streaming. Section 5.2 present the design and evaluation of MTP.

5.1 Understanding Streaming Requirements

The growth in power and connectivity of today's computers has enabled streaming video across the Internet to the desktop. For example, in 2001 an estimated of 350,000 hours of online entertainment was broadcast each week over the Internet [110], with countless more hours downloaded on-demand. Web sites today offer streaming videos of news broadcasts, music television, live sporting events and more.

While voice quality audio typically operates over a low and narrow range of bitrates (32-64 Kbps), video operates over a much higher and wider range of bitrates. Video conferences and Internet videos stream at about 0.1 Mbps[1], VCR quality videos at about 1.2 Mbps[2], broadcast quality videos at about 2-4 Mbps[3], studio quality videos at about 3-6 Mbps[3], and HDTV quality videos at about 25-34 Mbps[3]. Thus, video applications have the potential to demand enormous bitrates, often greater than the available network capacity, but also have the potential to reduce their data rates when available capacity is constrained.

Unlike typical Internet traffic, streaming video (or streaming media in general) is sensitive to delay and jitter, but can tolerate some data loss. Thus, streaming applications often prefer UDP over TCP as their transport protocol, since TCP's reliable transmission service can cause potentially large delay during congestion. Streaming over UDP may achieve better stream quality than streaming over TCP. However, UDP streams can be a threat for the well-being of the public network since streaming applications may not implement a proper congestion control mechanism. Streaming applications perform media scaling to achieve a best stream quality for the available network capacity, and thus can reduce UDP stream bitrates in response

[1]H.261 and MPEG-4
[2]MPEG-1
[3]MPEG-2

to congestion. Yet, with the use of repair techniques [11, 82, 95, 102], UDP packet losses can be partially or fully concealed, reducing the impact of loss on the quality of the video by the user, and thus reducing the incentive for multimedia applications to lower their bitrate in the presence of packet loss during congestion.

Potentially high-bitrate video over UDP using repair techniques suggest that video flows may not be *TCP-friendly* or, even worse, that video flows may be unresponsive to network congestion. In the absence of end-to-end congestion control, TCP flows competing with video flows using UDP reduce their sending rates in response to congestion, leaving the unresponsive UDP flows to expand to use the vacant capacity, or, worse, contribute to congestion collapse of the Internet [43]. The threat of UDP streams becomes even more prominent with the growth in broadband Internet connection capacity offered by Internet Service Providers (up to 3 Mbps for typical cable modem services), since high-bitrate video streaming becomes available for more end-users.

Recent research has proposed TCP-Friendly streaming transport protocols in the hope that streaming media applications will use them [44, 114, 113, 131]. However, such protocols are not yet widely part of most operating system distributions. Moreover, it is not clear whether the proposed protocols are indeed practical for streaming applications, and whether the protocols give enough incentives for streaming applications to use the new protocols in place of TCP or UDP. While there have been some studies characterizing streaming traffic [19, 73, 80, 90, 124, 130], few studies have been made to characterize the media scaling performance of Internet video streams and to understand the streaming end-system protocol requirements.

This study characterizes video streams on the current Internet by measuring and analyzing the application and network layer statistics of Internet video streams from Real Networks, one of the most widely used commercial streaming media product.

In particular, this study, also published in [26], characterizes the degree of media scaling supported by RealVideo streams on the Internet, measures media scaling performances of the TCP and UDP video streams, such as congestion responsiveness of UDP streams and media scaling effectiveness of TCP streams. By comparing the application and network layer performance of video streams over TCP and UDP, two drastically different transport protocols, this study characterizes streaming protocol requirements.

We set up a network testbed where two clients, one using UDP and the other using TCP, streamed video through a network router we control and connected to the Internet via a broadband connection. We varied the bottleneck bitrate to the clients by limiting the capacity of the router's outgoing connection, allowing us to explore a range of congestion situations. Over the course of 2 months from February 2002, the two clients streamed hundreds of videos selected with a variety of content and encoding formats from a diverse set of Web servers, while measuring packet loss rates and round-trip times as well as application level statistics such as encoded bitrates and frame rates.

Following is the summary of our key findings: 1) Most streaming RealVideo clips on the Internet are not capacity-constrained for a typical broadband connection. 2) In cases of congestion at the local Internet connection link, streaming RealVideo over UDP often gets a larger share of the bandwidth than the competing streaming RealVideo over TCP. However, UDP streams that respond to congestion by reducing the application layer encoding rate often meet TCP-Friendly criteria. 3) The bitrates of UDP RealVideo streams during buffering are up to 5 times the steady state playout rate, making the UDP streams bandwidth greedy during the periods. 4) About 35% of Internet RealVideo clip samples were single-bitrate congestion unresponsive clips that are unresponsive to congestion by changing levels. Moreover, about 40%

of the unresponsive clips were high-quality videos. 5) Media scaling over TCP is difficult since TCP hides network information such as packet loss rate and round-trip time needed for efficient media scaling decisions. Moreover, the huge penalty of overestimating available TCP bandwidth, i.e. the large queuing delay at the TCP sender caused by transmission of previous high quality video frames, contributes to the inefficient media adaption and causes frequent media re-buffering events at the receiver.

The rest of this paper is organized as follows: Section 5.1.1 presents background on RealPlayer to help understand our results; Section 5.1.2 describes our approach to obtain a wide-range of Internet measurements; Sections 5.1.3 and 5.1.4 present and analyze, respectively, the measurement data obtained; Section 5.1.5 discusses our findings; Section 5.1.6 summarizes our conclusions and presents possible future work.

5.1.1 RealVideo Background

RealPlayer provided by RealNetworks,[4] is one of the most popular streaming media players on the US Internet, with over 47% of the commercial market share in April 2001 [68]. RealVideo content providers create streaming videos using a variety of possible video codecs, convert it to RealNetworks' proprietary format and place it on an Internet host running RealServer. During creation, content providers select encoding bitrates appropriate for their target audience and specify other encoding parameters, such as frame size and frame rate, appropriate for their content. The RealServer then streams the video to a user's RealPlayer client upon request.

RealServer and players primarily use the Real Time Streaming Protocol[5] (RTSP)

[4]http://www.real.com/
[5]http://www.rtsp.org/

for the session layer protocol. Occasionally, RealServer will use HTTP for metafiles or HTML pages, and HTTP may also be used to deliver clips to RealPlayers that are located behind firewalls. Older versions of RealServer used the Progressive Networks Audio (PNA) protocol and, for backward compatibility, newer RealServers and players still support this protocol. For this measurement study, all the video clips selected used RTSP, as described in Section 5.1.2.1.

At the transport layer, RealServer uses both TCP and UDP for sending data. The initial connection is often in TCP, with control information then being sent along a two-way TCP connection. The video data itself is sent using either TCP or UDP. By default, the actual choice of transport protocol used is determined automatically by the RealPlayer and RealServer, resulting in UDP about 1/2 the time and TCP the other 1/2 [130]. The decision making process RealPlayer uses to choose either UDP or TCP is not publicly documented, and may be interesting future work. The choice of UDP or TCP can also be manually specified by the user [111]. For our study, we specifically set RealPlayer to use UDP in some cases and TCP in others, as described in Section 5.1.2.2.

RealSystem supports an application level media scaling technology called *SureStream* in which a RealVideo clip is encoded for multiple target bitrates [30, 112]. When streaming a SureStream RealVideo clip, RealServer determines which encoded stream to use based on feedback from the RealPlayer regarding the client end-host network conditions. With SureStream, the actual video stream served can be varied in mid-playout, with the server switching to a lower bitrate stream during network congestion and then back to a higher bitrate stream when congestion clears. We study the flexibility of SureStream scaling in Section 5.1.4.4.

For each video clip, RealPlayer keeps a buffer to smooth out the video stream because of changes in capacity, lost packets or jitter. Data enters the buffer as it streams to RealPlayer, and leaves the buffer as RealPlayer plays the video clip. If network congestion reduces available capacities for a few seconds, for example, RealPlayer can keep the clip playing with the buffered data. If the buffer empties completely, RealPlayer halts the clip playback for up to 20 seconds while the buffer is filled again. We measure the rate at which RealPlayer fills the buffer in Section 5.1.4.3.

5.1.2 Measurement Approach

Media scaling technologies adapt media encoding to the available bitrate in an effort to provide acceptable media quality over a range of available bitrates [9, 128]. In times of congestion, media scaling benefits both the network, by reducing offered load, and also the user, by providing graceful degradation in perceived quality [123]. As mentioned in Section 5.1.1, RealSystems provide SureStream media scaling at the application level that can select an adequate quality version of a video to fit into the current conditions of available network bitrate.

In order to empirically measure the media scaling performances of RealVideo over UDP and TCP, we employed the following methodology:

- Select RealVideo URLs that use the Real Time Streaming Protocol (RTSP) using well-known Web search engines (see Section 5.1.2.1).
- Construct an environment for measuring the responsiveness of RealVideo over UDP by comparing it to TCP under the same network conditions (see Section 5.1.2.2).
- Construct an environment for measuring the effectiveness of media scaling over TCP by comparing the application layer behavior of non-competing RealVideo over UDP or TCP (see Section 5.1.2.3).

- Iteratively play the selected RealVideo clips in both environments with different bottleneck capacities and analyze the results (see Section 5.1.3 and Section 5.1.4).

5.1.2.1 RealVideo Clip Playlist

We desired a relatively realistic environment in which we could measure and compare the media scaling performances of RealVideo over UDP and TCP. If we had chosen a stand-alone environment where we could precisely control the network conditions from the server to the client, the encoded content and server platform chosen might impact performance more than the network, resulting in inaccurate conclusions about the Internet at large. Thus, we decided to use publicly available Internet RealVideo servers and clips as the traffic sources.

To form a clip playlist, we searched for RealVideo clips (URLs) accessible through Web pages using well-known search engines, such as Yahoo and Google, and selected 100 valid RTSP RealVideo URLs from the first 100 search results returned. Of the selected URLs, 76 were from the United States, 9 from Canada, 8 from the United Kingdom, 6 from Italy, and 1 from Germany. While our selection method of using US/English based commercial search engines likely influenced the predominance of North American URLs, our RealPlayer clients also ran from North America, and it is likely that there is typically a similarly strong locality of access for most streaming players. Of the original set of 100 video clips, 21 clips became unavailable sometime after the initial selection and before the experiments were completed. Thus streaming results of the remaining 79 clips were used for analysis.

For the clips selected, the median clip length was about 3 minutes, while the shortest and longest clips played out in 20 seconds and 30 minutes, respectively. Other statistics on the selected RealVideo clips are available in Section 5.1.3, Sec-

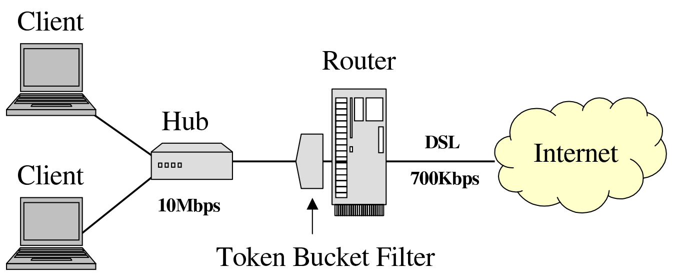


Figure 5.1: Testbed Network Setup: Environment to Measure the Responsiveness of RealVideo

tion 5.1.4.4 and [27].

5.1.2.2 Responsiveness of RealVideo over UDP Measurement Environment

Ideally, we sought an environment in which to measure the network layer responsiveness of RealVideo over UDP by comparing it to that of long-lived TCP flows under the same network conditions. We had two RealPlayers, one using UDP and the other using TCP, simultaneously stream a video clip from the same RealServer along the same network path, while we captured network and application statistics. As depicted in Figure 5.1, the two RealPlayers ran on separate PCs attached to the same 10 Mbps hub. Both PCs were equipped with a Pentium III 700 MHz processor, 128 MB RAM and a UDMA-66 15 GB hard disk, and were running Linux kernel version 2.4. Both PCs ran RealPlayer version 8.0.3, with one RealPlayer configured to use UDP and the other RealPlayer configured to use TCP.

The hub facilitated capturing network layer performance data since packets destined to either PC were broadcasted to both PCs. We ran `tcpdump`[6], a well-known network packet sniffer, on one PC to filter and log the video stream packets. As

[6]http://www.tcpdump.org/

the RealVideo packet format is proprietary, we were unable to obtain sequence numbers and, hence, loss information, from the packet traces themselves. We did run `tcptrace`[7] on the `tcpdump` data, but it only provides statistics on the very sparse amount of RTSP control traffic from the client to the server and not statistics on the data stream itself. Instead, during the playout of each clip, we ran a `ping` at 1 second intervals to the server to obtain samples of the round-trip time (RTT) and packet loss rate. During pilot studies, we confirmed that the RTTs and loss rates obtained via the `ping` samples were comparable to those obtained via `tcptrace`. Also, we verified that the packet filtering and logging did not induce significant CPU or disk load and did not interfere with the video playout. At the end of each RealVideo stream, information such as the IP packet size and arrival time were extracted from the `tcpdump` log using `ethereal`[8] and processed to obtain network layer statistics, such as throughput.

In order to control network congestion, we considered adding background traffic along the path from the client to the servers. However, as discussed, most RealServers do not simultaneously provide other file services making it difficult to add congestion-causing traffic to servers in a controlled manner. Instead, to consistently control the incoming available bitrate, we set up a private Linux router connected to a commercial 700 Kbps DSL network to enable us to create constrained bitrate situations. The router was configured to use network address translation (NAT) to eliminate the possibility that packets from the competing TCP and UDP streams would be routed differently. We attached a Linux token bucket filter (TBF)[9] module to the Ethernet card at the internal network of the router. The TBF queue size was

[7]http://irg.cs.ohiou.edu/tcptrace/tcptrace_new/

[8]http://www.ethereal.com/

[9]Recent work measuring broadband access links suggest some ISP providers similarly use TBFs to limit capacities [76].

set to 10 Kbytes and the burst allowed (the maximum number of tokens available during idle times) was set to 1600 bytes, slightly larger than a typical 1500 byte MTU. The *token rate* (available bitrate) was set to 600 Kbps, 300 Kbps, 150 Kbps and 75 Kbps. Note, since we have two streaming flows, one TCP and one UDP, competing, their fair capacity share is approximately half of each bottleneck capacity.

Although it is conventional wisdom that over-provisioning in core network routers has moved Internet performance bottlenecks to network access points [1], it is still possible that network bottlenecks may occur elsewhere. However, for our study, the location of the bottleneck, whether at the access link or further upstream, does not impact the competition between the TCP and UDP streams since the streams have the same NAT-translated IP address and thus share the same network path. Even if the network path is altered mid-stream due to a routing change, the change will be applied to both streams.

For each DSL-TBF configuration, we carried out two sets of measurements, where each set consecutively played all video clips in the playlist.

5.1.2.3 Media Scaling Measurement Environment

To study media scaling in RealPlayer and its effectivenss on TCP streaming we used *RealTracer*,[10] developed for a previous study [130], which plays RealVideo streams and records application level statistics, including encoding rate. One of the client machines was booted to run Microsoft Windows ME and equipped with RealPlayer 8 Basic version 6.0.9 and RealTracer version 1.0. We then ran a non-competing, single UDP or TCP stream for each URL in the playlist, while limiting the TBF incoming bitrate to 35 Kbps,[11] since the highest encoded bitrate for all clips that did media

[10]http://perform.wpi.edu/real-tracer/

[11]The queue was set to 5 Kbytes for the 35 Kbps DSL-TBF configuration.

scaling was above 35 Kbps. We tried other TBF rates such as 25 Kbps, 150 Kbps and 300 Kbps to verify we measured all possible scale levels (or encoded bitrates) used for clip playouts. However, only 2 sets of measurements, TCP for the entire playlist and UDP for the entire playlist, on the 35 Kbps DSL-TBF configuration was used to characterize the responsiveness of RealVideo media scaling (see Section 5.1.4.4).

5.1.3 Result

Over the course of 2 months from February 2002, we streamed a total of over 200 hours of video from a cumulative total of over 2000 video clips. Of the total 79 clips in the playlist, 24 (about 30%) of their servers did not respond to `ping` packets, making them unavailable for loss and round-trip time (RTT) analysis. For all RTT, loss and TCP-Friendly analysis in this report, we removed the data from these clips, leaving a total of 110 clips for each protocol type at each bottleneck capacity (55 clips $\times$ 8 $\times$ 2 sets of experiments = 880 total clips). However, we did use the other data recorded on the set of 148 clips for each protocol type at each bottleneck capacity (1184 clips total) for analysis that did not require use of the `ping` data.

Comparing the average RTTs obtained via `ping` probes for each bottleneck capacity, the 75 Kbps connection had the highest round-trip times. The median RTTs for the 75, 150, 300 and 600 Kbps configurations were 450, 340, 130 and 100 ms respectively. For the 150-600 Kbps configurations, about 33% of the clips had about the same RTT regardless of the bottleneck capacity since these clips streamed at less than 150 Kbps, and therefore do not suffer additional queuing delays at the router. For the remaining 67% of the clips, the lower the bottleneck capacity the higher the queuing delays, caused primarily by the 10 Kbyte buffer at the bottleneck router.

Summarizing the loss rates obtained via `ping` probes for each bottleneck capacity, the median loss rate for any configuration was less than 2%. About 37% of the clips played with low bottleneck capacities had no loss, while about 50% of the clips played at higher bottleneck capacities had no loss. Overall loss rates increased about 1% for each decrease in bottleneck capacity from 600 Kbps to 300 Kbps to 150 Kbps to 75 Kbps. The low loss rates, even at low capacities, implies that most of the RealVideo UDP streams adapted to the available bitrate, and is investigated in depth in Section 5.1.4.

Summarizing statistics on packet sizes, the TCP streams used larger packets than the UDP streams with a median UDP packet size of about 640 Kbytes, and a median TCP packet size of about 1100 Kbytes. Moreover, more than 30% of the TCP packets were equal to the typical network MTU, 1500 Bytes. A possible reason for the larger packet sizes over TCP is that while RealServers can control the application frame sizes to send, with TCP, those frames are often grouped and sent based on the current TCP window sizes.

We do not present further details on the results here, but refer the interested reader to [27].

5.1.4 Analysis

In analyzing the media scaling performance of RealVideo over UDP and TCP, we first analyze bitrates aggregated over all clips and then analyze bitrates for individual clip pairs (Section 5.1.4.1). We next analyze the TCP-Friendliness of RealVideo over UDP (Section 5.1.4.2). Then, we characterize the initial buffering rate compared with the steady playout rate (Section 5.1.4.3). Moving to the application layer, we analyze the application scaling behavior and the effectiveness of media scaling over TCP (Section 5.1.4.4). Lastly, we examine the smoothness of the network data rate of TCP and UDP streams (Section 5.1.4.5).

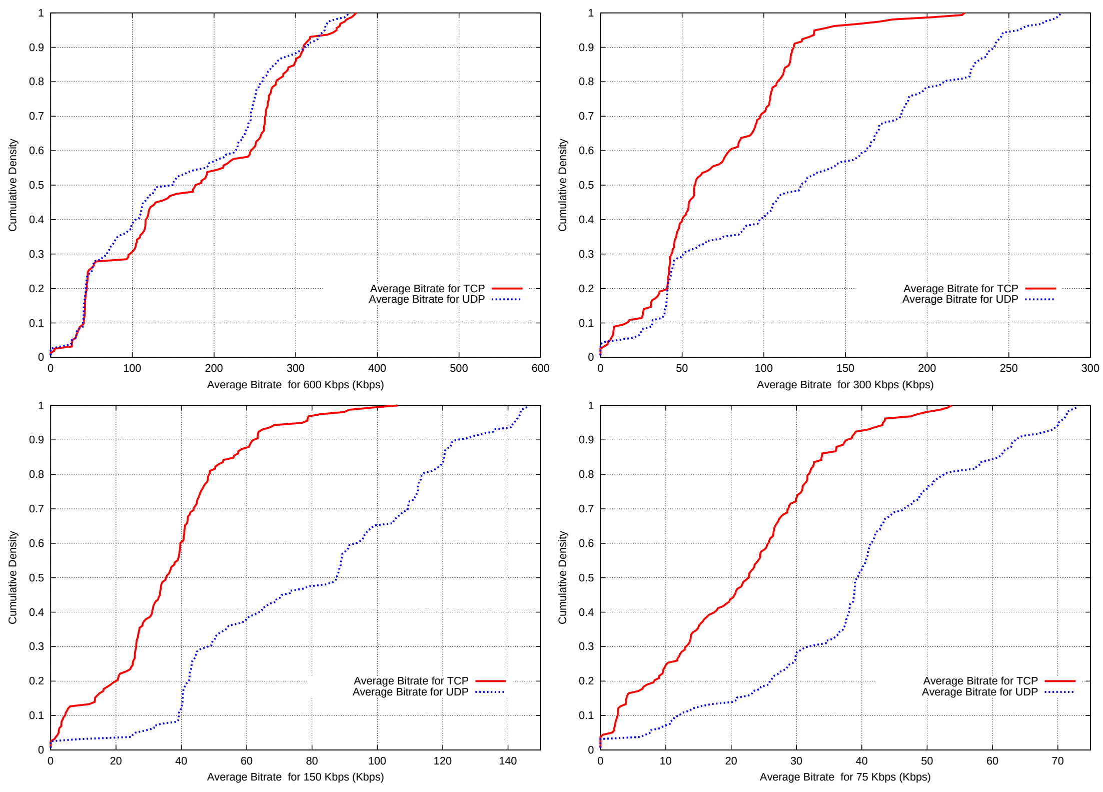


Figure 5.2: CDFs of Average Bitrates for Bottleneck Capacities of 600, 300, 150, and 75 Kbps

5.1.4.1 Bitrates

Figure 5.2 depicts Cumulative Density Functions (CDFs) of the per-clip average bitrate used by TCP and UDP for bottleneck capacities of 600, 300, 150 and 75 Kbps. The TCP and UDP distributions are nearly the same for the 600 Kbps bottleneck capacity. However, as capacity becomes more constrained, the distributions separate, with UDP having a consistently higher distribution of bitrates than TCP, as evidenced by UDP distributions being lower and to the right of the corresponding TCP distributions.

We next analyze the head-to-head bitrate for each pair of (TCP, UDP) clips. For each clip pair, in Figure 5.3 we plot an (x,y) point where x is the average bitrate used

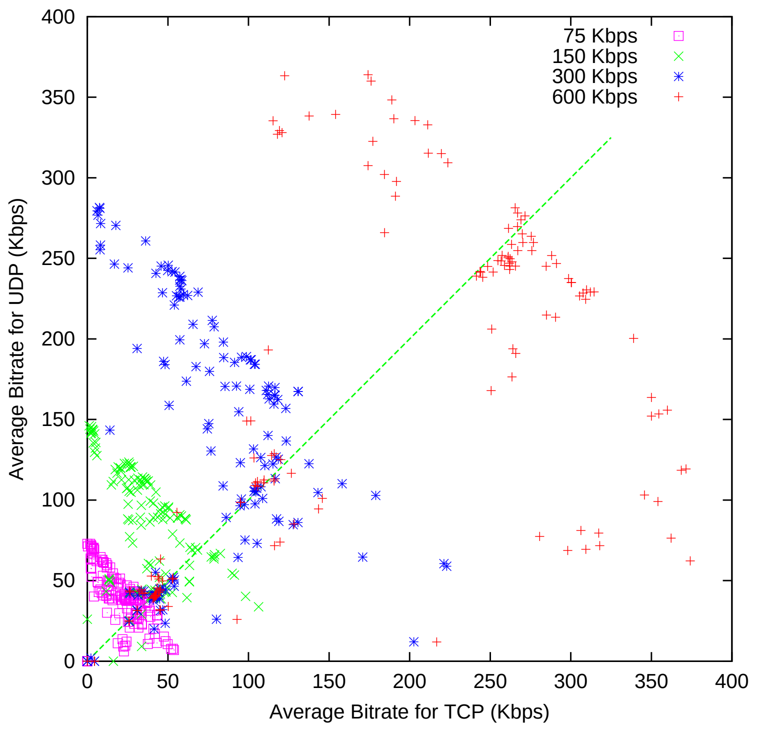


Figure 5.3: Head-to-Head Average Bitrate (all runs)

by the TCP stream and y is the average bitrate used by the UDP stream. The points for each bottleneck capacity are depicted by a different point style. The dashed 45 degree line provides a reference for equal bitrates for both TCP and UDP. Points above the line (top left of the graph) indicate UDP had a higher average bitrate while points below the line (bottom right of the graph) indicate TCP had a higher average bitrate. The distance from the line indicates the magnitude of the average bitrate difference.

From Figure 5.3, while there are some points that lie along the equal bitrate line, there are many cases of bitrate disparity. The highest bitrate playouts for the 600 Kbps bottleneck capacity had the greatest bitrate disparities. For the 600 Kbps bottleneck capacities, there are visually as many points below the equal bitrate line where TCP had a higher average bitrate as there are above the equal bitrate line

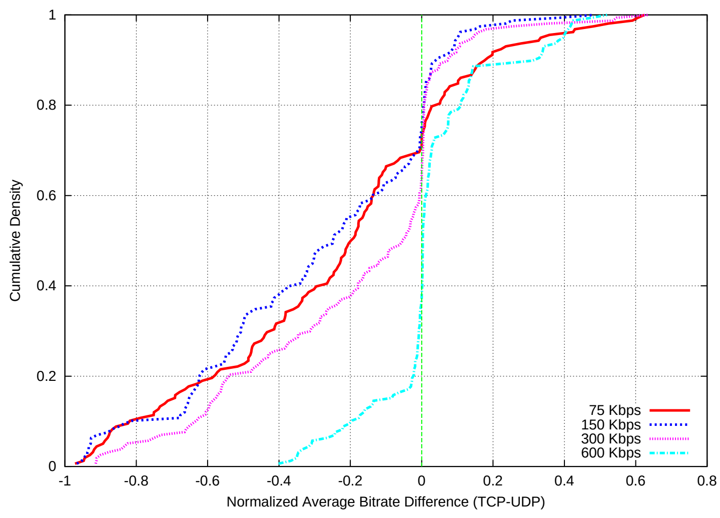


Figure 5.4: CDF of the Difference (TCP - UDP) in the Average Bitrate, Normalized by the Bottleneck Capacity (all runs)

where UDP had a higher average bitrate. For the lower bottleneck capacities, there are visually considerably more points above the equal bitrate line, indicating UDP had a higher average bitrate than did TCP.

We next analyze the bitrate disparity relative to the bottleneck capacity. For each clip pair, we subtract the UDP average bitrate from the TCP average bitrate and divide the difference by the bottleneck capacity. Thus, equal sharing of capacity has a value of 0, a value of -1 indicates UDP got the entire bottleneck capacity, and a value of +1 indicates TCP got the entire bottleneck capacity. Figure 5.4 depicts CDFs of the normalized bitrate differences for each bottleneck capacity.

For the 600 Kbps bottleneck capacity, about 40% of the clips shared the capacity equally. As indicated by the region in the top right, about 30% of TCP clips had a higher bitrate than their counterpart UDP clips while about 20% of the UDP flows had a higher bitrate than their counterpart TCP clips, as indicated by the region in the bottom left. For the 600 Kbps bottleneck capacity, the greatest bitrate disparity was approximately half the bottleneck capacity.

For the lower bottleneck capacities, there were increasingly fewer clips with equal bitrates. Many UDP clips had substantially higher bitrates than did their TCP counterparts, as indicated by the large areas under the distributions on the bottom left. For the 300 Kbps bottleneck capacity, about 60% of the UDP clips had higher bitrates than their TCP counterparts, and for the 150 Kbps and 75 Kbps bottleneck capacities, about 70% of the UDP clips had higher bitrates than their TCP counterparts. For the 300, 150 and 75 Kbps bottleneck capacities, about 20% of the UDP clips got twice the normalized bitrate of their TCP counterparts. For the 150 and 75 Kbps bottleneck capacities, about 20% of the UDP clips had more than 80% more of the normalized bitrate than their TCP counterparts. However, even for the lowest bottleneck capacities, there were still cases where the TCP clips had a higher bitrate than their UDP counterparts, as depicted by the areas above the distributions in the upper right.

In general, as bitrates become constrained, streaming RealVideo clips over UDP receive relatively more capacity than do streaming RealVideo clips over TCP. However, further limiting capacity does not significantly change the UDP vs. TCP bitrate allocation ratio. A significantly large number of the UDP video streams are able to adapt to reduced capacities without causing increased congestion. Moreover, in all cases, streaming RealVideo clips over UDP sometimes have lower bitrates than do competing TCP flows, especially for higher bottleneck capacities.

We next analyze the impact of round-trip time and loss rate on the normalized bitrate disparity. The data rate of TCP is paced by acknowledgments and is limited by packet loss rate, so a higher round-trip time or loss rate directly results in a lower maximum throughput. However, the data rate of UDP is not similarly constrained. Our analysis of round-trip times and loss rates obtained by the `ping` samples show modest correlations for both round-trip times and normalized bitrate disparity and

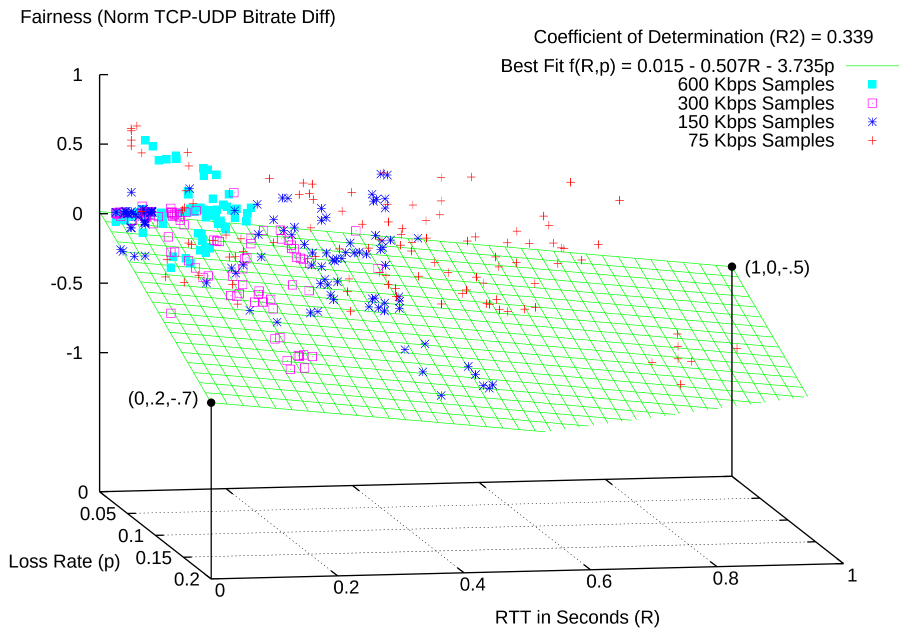


Figure 5.5: Loss Rate, Round-Trip Time and Fairness (Normalized TCP-UDP Average Bitrate Difference)

loss rates and normalized bitrate disparity. Figure 5.5 plots the normalized bitrate differences as a function of round-trip time and loss rate, and draws the best-fit (least square) plane for the samples. The coefficient of determination (R^2) of 0.339 indicates that the regression plane explains about one-third of the variation in the normalized bitrate disparity. The correlation of -0.51 for the round-trip time in seconds and -3.7 for the loss rate indicates that as round-trip times and loss rates increase, streaming RealVideo clips over UDP receive relatively more of the available bitrate than do streaming RealVideo clips over TCP.

5.1.4.2 Discussion of TCP-(Un)Friendliness

Although RealVideo over UDP may receive a disproportionate share of available bitrate than do their TCP counterparts, this may be because RealVideo TCP clips transmit at less than their maximum rate. A more serious test of unfairness is whether RealVideo over UDP is *TCP-Friendly* in that its data rate does not exceed the maximum rate of a conformant TCP connection under the same network conditions. The TCP-Friendly rate, T Bps, for a connection is given by [43]:

$$T \leq \frac{1.5 \times \sqrt{2/3} \times s}{R \times \sqrt{p}} \tag{5.1}$$

with packet size s, round-trip time R and packet drop rate p. For each clip for each run, we use Equation (5.1) to compute the TCP-Friendly rate (T), using a packet size (s) of 1500 bytes[12] and the loss rate (p) and RTT (R) obtained from the corresponding `ping` samples. We then compare T to the average bitrate used by the UDP clip. For each bottleneck capacity, we record the count of the number of times the UDP clip was not TCP-Friendly.

Bottleneck Capacity	Total Unfriendly	$min > fair$	$max < fair$	Effective Unfriendly
75 Kbps	8/110 (7%)	22	30	8/58 (14%)
150 Kbps	7/110 (6%)	12	42	5/56 (9%)
300 Kbps	9/110 (8%)	12	48	7/50 (14%)
Total	24/330 (7%)	46	120	20/164 (14%)

Table 5.1: Number (and percent) of Non-TCP-Friendly Flows

The TCP-Friendly results are shown in Table 5.1[13]. The "Unfriendly" columns indicate a count of the UDP clips that were not TCP-Friendly. The "$min > fair$" column indicates the count of clips that had a minimum encoded bitrate greater than the fair share of network capacity; these clips were not encoded to be able to properly respond to congestion. The "$max < fair$" column indicates the count of clips that had a maximum encoded capacity less than the fair share of the available bitrate; these clips, in general, had no need to respond to congestion. Removing

[12]The maximum packet size recorded. See [27] for more details on packet sizes.

[13]Since the 600 Kbps bottleneck capacity clips had very low loss rates, we do not include the 600 Kbps data in our analysis to avoid data skew from "unlucky" sampling.

the clips counted in these last two columns provides a base count for the non-TCP-Friendly clips, presented in the column “Effective Unfriendly”. This last analysis is useful as it exactly represents the percentage of RealVideo clips that must respond to congestion because of available bitrate constraints and have been encoded to allow the RealServer server to do so.

Overall, 36% (120/330) of the UDP streams had a maximum bitrate less than their fair share and thus were unconstrained by the network conditions. On the other hand, 14% (46/330) of the UDP streams were constrained by the network conditions but had not been encoded so as to allow them to respond to congestion. This latter set, while problematic from the congestion control point of view, can be readily addressed by content providers selecting multiple encoded bitrates when creating streaming video content for their Web sites. Of the remaining UDP streams that were constrained by the network and had been encoded to allow a congestion response, 14% were not TCP-Friendly.

The TCP-Friendly formula in Equation (5.1) is conservative in that it computes the maximum bitrate an aggressive TCP connection would receive. Thus, connections that achieve a higher bitrate than computed in Equation (5.1) are clearly not TCP-Friendly. In general, there is evidence to suggest many cases where streaming RealVideo over UDP is, in principle, TCP-Friendly, and there is also evidence to suggest that streaming RealVideo clips over UDP can sometimes be non-TCP-Friendly, particularly for capacity-constrained conditions.

5.1.4.3 Buffering Data Rate

As suggested in [80], RealPlayer buffers data at an accelerated rate for the first part of a clip. Confirming and analyzing the rate of this buffering rate versus steady playout rate may help to characterize the bursty nature of RealVideo streams.

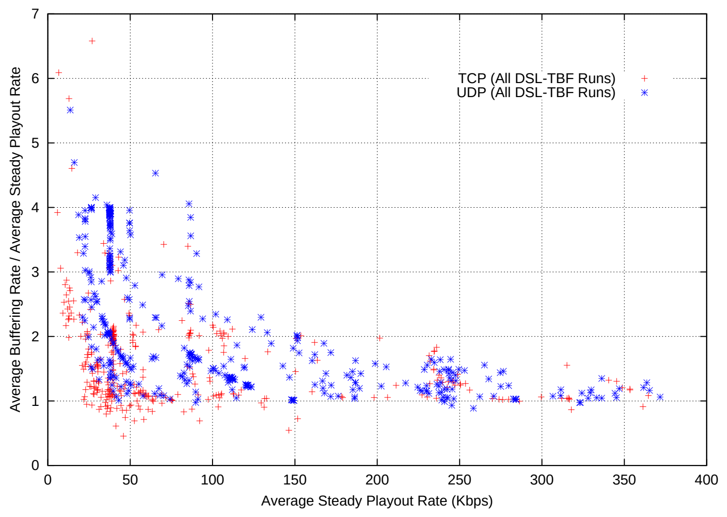


Figure 5.6: Ratio of Average Buffering Rate to Average Steady Playout Rate versus Average Steady Playout Rate (all runs)

For each clip, we compute the maximum bitrate averaged over 10 second intervals taken over the first 80 seconds (calling this the *buffering data rate*) and compare this to the average bitrate over the time from 100 seconds until the clip ends (calling this the *steady playout rate*).

Figure 5.6 depicts the ratio of (average buffering data rate / average steady playout rate) for different steady playout rates. For reference, a ratio of 1 indicates that the buffering data rate was equivalent to the steady playout rate. Low bitrate clips buffered at up to 6 times their average playout rates. Higher bitrate clips buffered at relatively lower rates, possibly because capacity restrictions limited them from buffering at a higher rate. Although not shown in this section, under capacity constrained conditions, UDP streams in buffering phase were often greedy limiting bitrates of the competing TCP streams also in buffering phase. The buffering / steady playout ratios less than 1 in the 0-150 Kbps range for some TCP streams are caused by TCP retransmission timeouts during buffering.

In order to determine if capacity restrictions limit buffering rates, we ran a set

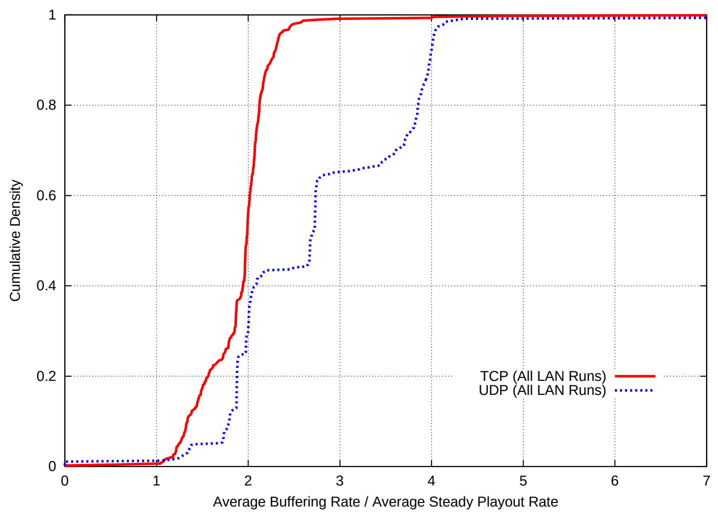


Figure 5.7: CDF of Ratio of Average Buffering Rate to Average Steady Playout Rate (LAN)

of experiments with the bottleneck capacity being the campus LAN attached to the Internet via a 15 Mbps link.[14] In this setup, the LAN environment was relatively unconstrained, having a bottleneck capacity which was typically at least three times that of our 600 Kbps bottleneck capacity.

Figure 5.7 depicts a CDF of the ratio of the average buffering data rate to the average steady playout rate. The buffering rate to steady rate ratio for UDP was nearly the same as that of TCP for 40% of the clips. For 60% of the clips, however, the ratio of buffering rate to steady rate for UDP was significantly higher than that of TCP. For UDP, the vertical "steps" in the CDF are at typical RealVideo encoding rates, where the buffering rate was a fixed multiple of these rates. For TCP, the steep slope in the CDF at around 2 suggests TCP streams typically buffered at a rate twice that of the steady playout rate.

In general, both RealVideo clips over UDP and RealVideo clips over TCP buffer data at a significantly higher rate than the steady playout rate. Due to this prominent buffering period, RealVideo cannot be modeled as a simple CBR flow, as is common

[14]http://www.wpi.edu/Admin/Netops/MRTG/

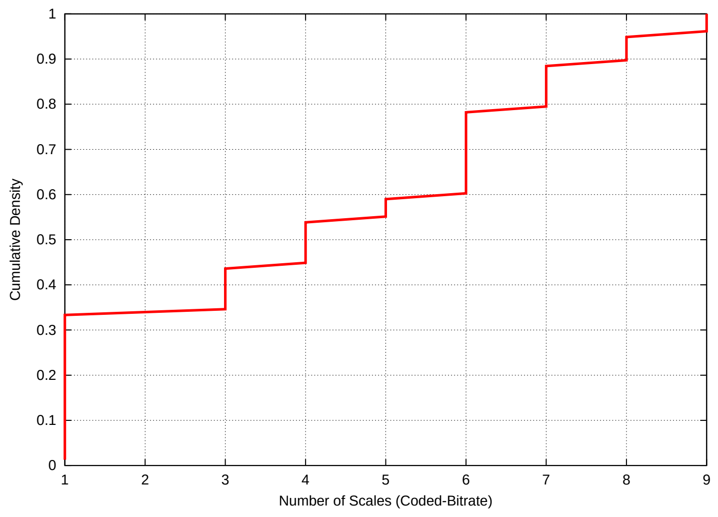


Figure 5.8: CDF of Media Scales (all runs)

in many network simulations that include streaming media. In fact, looking at a simple average bitrate over the length of the entire clip will also not reveal the true nature of RealVideo since it will miss the buffering period. An accurate bitrate distribution for RealVideo must include a buffering stage, whereby the sending data rate is typically from 2-5 times the steady-state playout rate and a post-buffering stage whereby the actual bitrate is dependent on the encoding bitrate of the content and the network conditions.

5.1.4.4 Media Scaling

In Section 5.1.4.2, we showed that even if using media scaling, RealVideo streaming over UDP can still be TCP-unfriendly. This section analyzes data from the media scaling measurement experiments, as described in Section 5.1.2.3, to characterize the degree of media scaling supported in Internet RealVideo streams and measure the effectiveness of media scaling over TCP. The application layer media scaling statistics also helps explaining factors that caused the TCP-unfriendly bitrates for some of the UDP RealVideo streams.

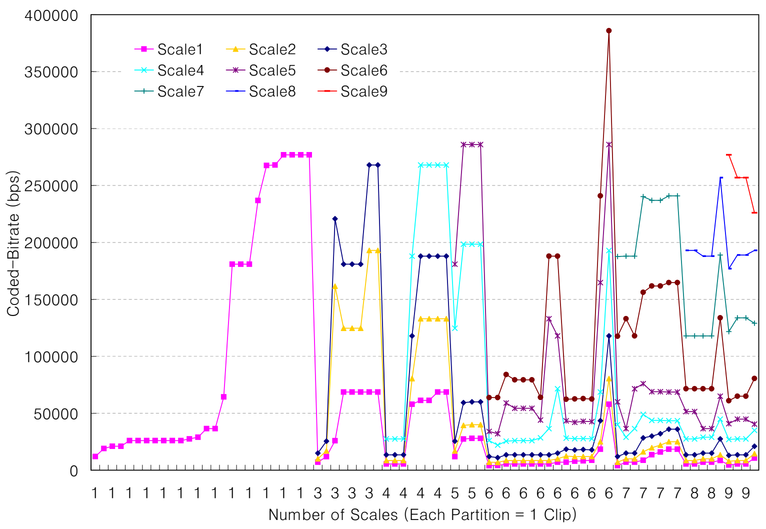
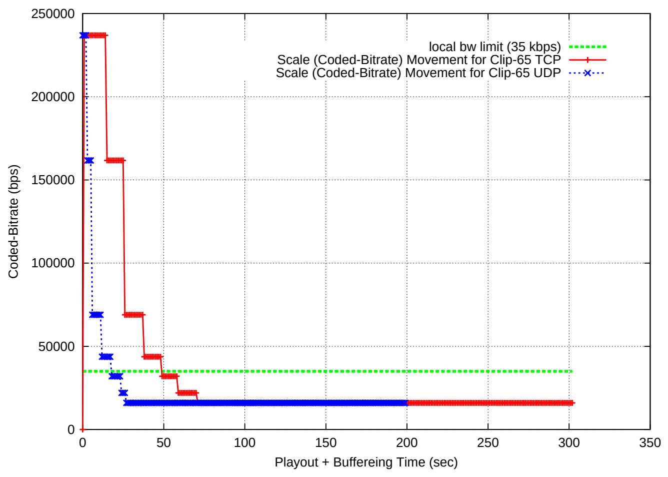


Figure 5.9: Media Scales and Encoded-Bandwidth (all clips). The horizontal-axis represents the number of different media scaling levels the clip can provide while the vertical axis represents the encoded bitrate for each scale level. The clips are sorted from the fewest scales on the left to the most scales on the right. For ties, the clips with the lowest encoded bitrate appear first.

Figure 5.8 shows a CDF of the number of distinct encoded bitrate levels seen in each clip for all runs. About 35% of the clips were not using media scaling at all, and therefore over UDP, these clips have difficulty responding to network congestion. Less than 50% of the clips were using more than 4 levels of scaling and so could only adjust to the available bitrate coarsely.

Figure 5.9 shows the scale levels and corresponding bitrates for each clip, sorted first by number of levels, and second by the lowest encoded bitrate. For the unresponsive clips (those with only 1 scale level), 40% were high-quality video clips that had a bitrate higher than 150 Kbps. Also, over 50% of the clips with 3 to 5 scale levels were targeted primarily for broadband connections and could not adapt to capacities below 50 Kbps. Streaming these clips on capacity-constrained links using UDP would cause unfairness to any competing TCP flows. RealVideo clips with more than 5 scale levels were designed to adapt more readily to low capacity



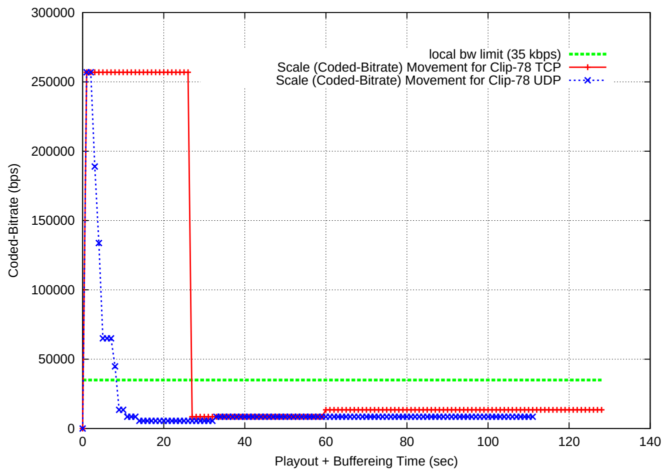


Figure 5.10: Media Scaling Dynamics: Clip-65 (top) and Clip-78 (bottom) (DSL: BR=35 Kbps, Q=5 Kbytes)

conditions, evidenced by the number of scale levels with low bitrates, but may still be unfair at higher capacities.

When available capacity is reduced during congestion, real-time streaming servers must employ media scaling in order to preserve timing, whether streaming over UDP or TCP. Figure 5.10 shows the media scaling behavior of two sample RealVideo clips streaming over UDP and TCP, where the available inbound bitrate was 35 Kbps. For both clips and both streams, the initial encoded bitrate was significantly higher than the available capacity, depicted by the horizontal line at 35 Kbps. Each horizontal "step" represents an application layer scaling of bitrate. The final playout bitrates achieved show the conservative adaptation of the RealServers since they stabilize at a bitrate less than what is available. This conservative media scaling behavior may result in less than optimal video quality but often helps the UDP RealVideo streams to achieve TCP-Friendly rates as supported by our TCP-Friendliness analysis. In the top graph of Figure 5.10, both TCP and UDP scaled their application data rate 6 times before the encoded rate settled below the available bitrate. However, UDP was able to obtain this application data rate much more quickly than did TCP. In the bottom graph of Figure 5.10, UDP quickly used 7 scale levels to adjust the application's data rate to the available bitrate, while TCP, on the other hand, took more than 20 seconds to adjust the rate, and then it did so in one, large encoding rate change.

We believe the difficulty RealPlayer over TCP has in adjusting the application data rate to the available network bitrate is because TCP does not expose network information to the application layer. Streaming applications over TCP can only measure application level goodput and not information on packet drop rates or round-trip times. Streaming applications over UDP, on the other hand, can more easily detect packet losses and measure round-trip times, allowing them to more quickly adjust the application data rate to the available network bitrate.

Moreover, for high-quality, high-bitrate videos, the inability to detect network congestion when using TCP is critical. As evidenced by the TCP stream in the bottom graph of Figure 5.10, the server fills the available TCP buffers with high quality video frames that must be delivered by the transport layer before it is able to scale down. For the user, this results in a large delay before frame playout begins as the high-quality frames are buffered over a low-capacity connection. Quantitatively, by looking at the end-time of transmission, the top graph of Figure 5.10 shows that to play 3 minutes (i.e., 180 seconds) of video, streaming over UDP took about 200

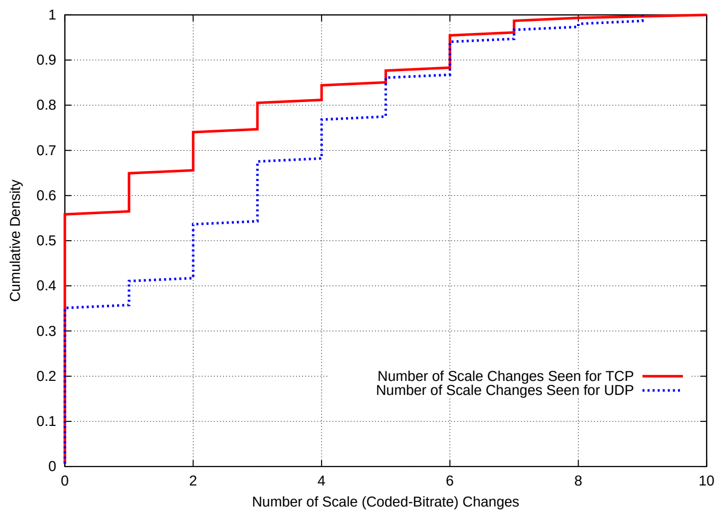


Figure 5.11: CDF of Media Scale Changes (DSL: BR=35 Kbps, Q=5 Kbytes)

seconds while streaming over TCP made frequent media re-buffering events and took more than 300 seconds. In other words, streaming over UDP required 20 seconds of buffering to play a 3 minute video clip, while streaming over TCP required more than 2 minutes of buffering to play the same 3 minute clip. This illustrates the effect of TCP send buffer queuing latency on streaming in [48].

In Figure 5.11, the CDFs depict the number of media scale changes seen for each video clip, and summarize the relative responsiveness of RealVideos in scaling the application data rate to below the available network bitrate. Overall, UDP streams had more scale changes than did TCP streams. Also, Figure 5.11 shows that about 20% (55% - 35%) of the streams that scaled when streamed over UDP did not scale at all when streamed over TCP.

Figure 5.12 summarizes the responsiveness of RealVideo media scaling based on how quickly the video stream adapted to the available bitrate after streaming started. Specifically, for the successfully adapted streams, we measure the time taken for the encoded bitrate to drop under the inbound capacity limit, depicted as the first point under the 35 Kbps limit for each stream in Figure 5.10. Figure 5.12 shows that about

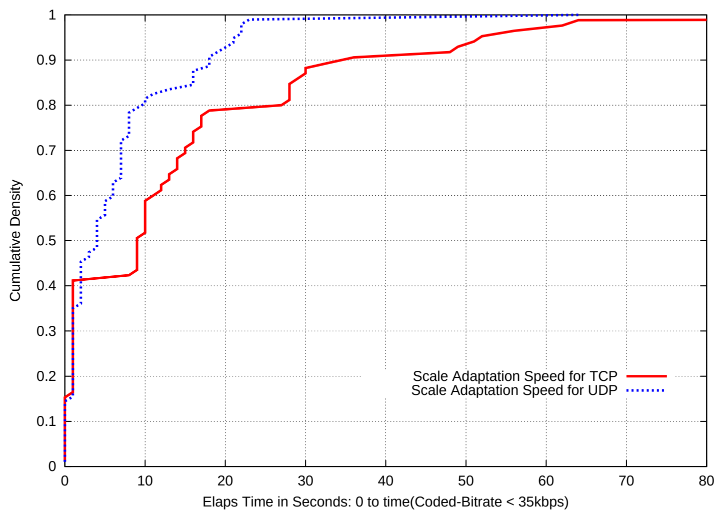


Figure 5.12: CDF of Media Scale Adaptation Speed (DSL: BR=35 Kbps, Q=5 Kbytes)

15% of the video clips were low-quality and always required less than 35 Kbps. Also, 25% (40% - 15%) of the video clips were able to adapt to the available bitrate within a couple of seconds, independently of the transport protocol used. However, for the remaining 60% of the clips, the TCP video streams took significantly more time to adapt their scales to the available bitrate. For example, 80% of the UDP video streams adapted to the available bitrate within 10 seconds, while it took more than 25 seconds for the same percentage of the TCP video streams to adapt.

In general, majority of RealVideo clips have ability to, and do, scale their application data rates to the available network capacity. However, a significant fraction of RealVideo clips are unable to adapt their application data rates to the available network capacity, causing UDP streaming to be unfair under capacity-constrained conditions. Also, the comparison of the media scaling performances of UDP and TCP streams reveals that media scaling over TCP is difficult due mainly to the streaming-unfriendly TCP API. RealVideo streams over UDP can adjust their application data rates to the available bitrate more efficiently than can RealVideo over

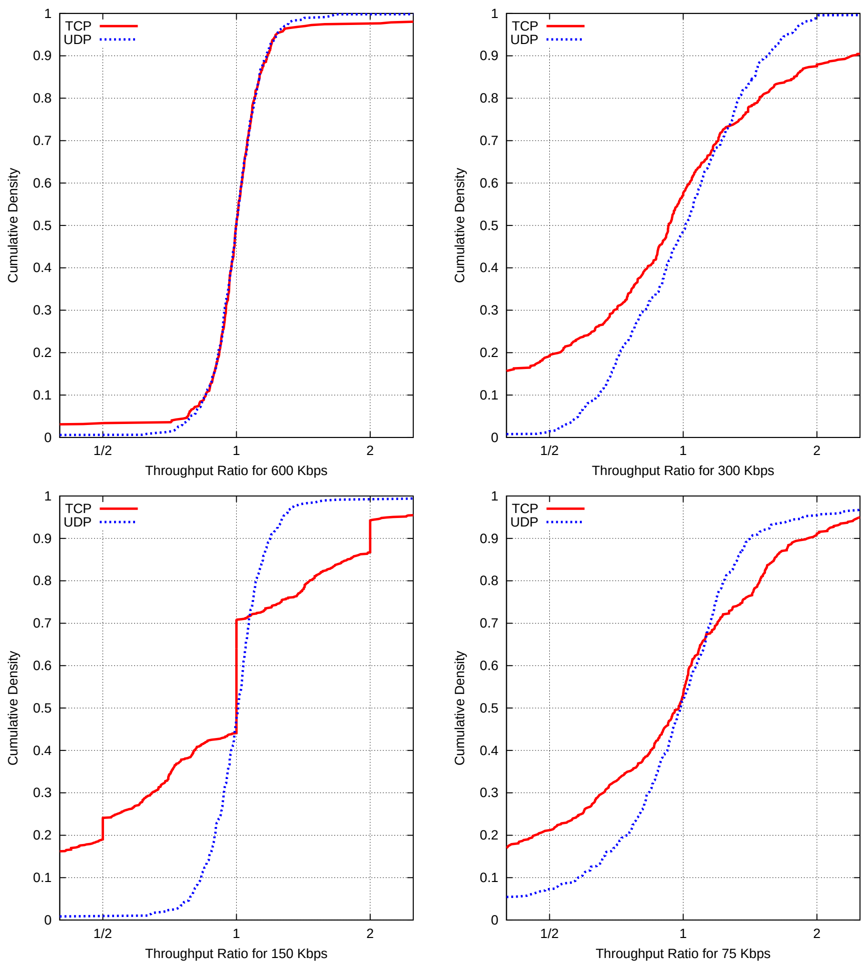


Figure 5.13: Smoothness Ratio for Bottleneck Capacities of 600, 300, 150, and 75 Kbps

TCP.

5.1.4.5 Smoothness

Streaming video requires not only a moderate to high bitrate but also a smooth data rate. TCP's acknowledgment-based window advancement can result in a bursty data rate, thus requiring a significantly larger receiver buffer at the application level for a smooth media playout than is required for UDP streams. Streaming media applications, especially real-time applications, sometimes cite these reasons for choosing

UDP as their primary transport protocol [48].

For each clip, we calculate the "smoothness" of the network data rate every 500 ms by taking the ratio of consecutive bitrates measured over each interval. For example, if the data rate is 200 Kbps for one time interval and 400 Kbps the next time interval, the smoothness of the interval would be 2. If the data rate then dropped by half back to 200 Kbps, it would be 0.5 for the next interval. Figure 5.13 depicts CDFs of smoothness for each network bottleneck bandwidth, with the x-axis drawn in log-scale so as to make a smoothness of 0.5 and 2 visually equal. Both TCP and UDP were smooth for a bottleneck capacity of 600 Kbps. With bottleneck capacities of 300, 150 and 75 Kbps, both TCP and UDP became noticeably less smooth, with TCP often far less smooth than UDP.

In general, as for other streaming applications, streaming RealVideo clips over UDP receive a smoother playout rate than do streaming RealVideo clips over TCP for capacity-constrained conditions.

5.1.5 Discussion of Results

In the current Internet, there are no concrete incentives for applications that use UDP to initiate end-to-end congestion control. In fact, at the network level, unresponsive applications may be "rewarded" by receiving more than their fair share of available bitrate. As seen in Section 5.1.4, streaming media over UDP can sometimes result in a higher average bitrate than streaming media over TCP, primarily because competing TCP sources are forced to transmit at a reduced rate. Plus, as seen in Section 5.1.4.4, it is more difficult for the application layer to adjust the encoding rate to the available bitrate when using TCP, because the streaming-unfriendly TCP API hides network information required for efficient media scaling and does not provide methods to control the TCP sender buffer. Thus, there are strong application-oriented reasons for streaming media to use UDP rather than TCP, suggesting potentially high-bitrate video over UDP may contribute to congestion collapse.

However, an unresponsive "fire-hose" application, such as high-quality video using UDP over a congested link, is ineffective from the application standpoint primarily because having a congested router randomly drop packets can cause the more important data packets to be dropped [13]. Instead, applications can significantly benefit by using media scaling, as illustrated by RealPlayer in Section 5.1.4.4, to make intelligent decisions about which packets not to send beforehand, making low quality video over the same congested link quite effective.

As shown in Section 5.1.4.2, media scaling, a streaming QoS control mechanism, can also be an effective means of responding to network congestion. While scaling the application data rate to meet the available bitrate, RealVideo over UDP often achieves a TCP-Friendly transmission rate. However, these results, obtained in an experimental environment that induces contention at the low-capacity last-mile link, may or may not hold for contention on high-capacity backbone links. Typically, the packet drop rate of a high-capacity link may be affected little by the data rate of a single video stream, causing a weaker control relation between media scaling and network contention (packet loss rate). Under such a condition, the sensitivity of the media scaling will dominate the congestion responsiveness of the UDP video stream. While we were unable to measure the scaling sensitivity of RealVideo streams in this study, in the worst case, a RealVideo may very coarsely react to the network regardless of the scale levels supported by the clip, streaming either at the highest or at the lowest quality level.

In addition, although media scaling, when coupled with properly scale-encoded RealVideo clips, may be effective it is not always guaranteed as media scaling is an optional encoding feature provided to content providers as a means to enhance streaming media quality rather than as a proper congestion control mechanism. Section 5.1.4.4 shows that about 30% of RealVideo streams could not do application media scaling at all, being unresponsive to network congestion when streaming over UDP.

Moreover, the higher buffering rate seen in Section 5.1.4.3, being beneficial for users, can possibly be harmful to the network. A higher buffering rate either allows the player to build up a larger buffer before beginning frame playback and thus better avoids any un-smoothness caused by network jitter or transient congestion, or allows the frame playback to begin earlier. However, the increased buffering rate makes the streaming traffic more bursty and, with UDP, it can cause even more unfairness versus any competing TCP flows.

Thus, despite some positive congestion responsiveness results of RealVideo UDP streams, end-to-end congestion control that relies solely on media scaling may not be a suitable solution for the well-being of the Internet. Instead, a streaming-friendly transport protocol with a proper congestion control mechanism should be provided. Such a protocol would ideally provide a streaming-friendly API that gives applications transmission state information as well as control over the transmission buffer management for efficient media scaling. In addition, the protocol should give streaming applications incentives to use the new protocol instead of UDP, such as support for ECN and/or TCP compatible firewall friendliness (i.e. can go through existing firewalls as does TCP).

5.1.6 Summary

In this work, we evaluated the network-level and application-level media scaling performance of UDP and TCP RealVideo streams under the same network conditions. We set up a testbed that allowed us to stream two RealVideo clips, one over TCP and one over UDP, along the same network path. Our testbed also let us control the network bottleneck capacity, thus allowing us to evaluate the responsiveness to congestion of the UDP streams. Using our testbed, we streamed over 600 hours of videos from over 2000 video clips with a variety of content and encoding bandwidths selected from across the Internet.

Media scaling directly determines the congestion responsiveness of UDP streams and can be an effective means of responding to congestion when paired with video clips encoded with multiple scale levels. Overall, we find RealVideo over UDP typically receives TCP-Friendly bitrates under normal network conditions and even during periods of packet loss. However, under capacity-constrained conditions, RealVideo over UDP can have a higher bitrate than TCP and the bitrate gets increasingly unfair with an increase in packet loss rate and round-trip time. Moreover, properly scale-encoded video clips are not guaranteed as they are an optional encoding feature provided as a means to enhance streaming media quality rather than as a proper congestion control mechanism. In addition, the user-beneficial rapid buffering of traffic over UDP can cause considerable congestion and can make RealVideo network traffic more difficult to manage.

This study concludes that while not threatening the well-being of the Internet as is commonly feared under normal network conditions, RealVideo UDP streams may also not necessarily be good Internet citizens. Furthermore, since our observations apply only to RealVideo streams with the congestion responsiveness of other popular streaming applications still unknown, we see the need for a streaming friendly network protocol that supports end-to-end congestion control.

This study also analyzed the suitability and/or shortcomings of using TCP as a streaming transport protocol. Adjusting the application data rate to the available network bitrate is more difficult when streaming over TCP versus UDP, because application streams over TCP do not have as much information about the current network state as do the applications when streaming over UDP, and the penalty of overestimating the available capacity is severe for TCP. This suggests that streaming friendly application programming interface (API) design is one of the most important factors for successful deployment of a future streaming transport protocol.

5.2 Multimedia Transport Protocol

The two characteristics of continuous media that distinguish streaming media applications from other traditional Internet applications are the continuity and the repairability of the media. As an effort to improve stream quality and avoid media play at interruption, streaming applications use media scaling to select or re-select an encoded medium appropriate for the available bandwidth of the network. In addition, streaming applications use a media buffer at the receiver to prevent media playout jitter due to network transmission delays. Media buffer contents are measured in seconds of media play time, not bytes or packets.

Streaming applications today choose either TCP or UDP as their transport protocol, depending on individual needs. It has been a myth that UDP is the dominant choice for streaming media on the Internet, believing that it is difficult to achieve acceptable streaming performance over TCP. However, Merwe, Sen and Kalmanek [125] report that video on demand (VoD) and live broadcasting applications predominantly use TCP over UDP, with TCP used for about 72% to 75% of all bytes transferred.

Streaming over UDP is undesirable when firewalls block UDP to limit the penetration of streaming traffic. This restriction for UDP also occurs when network address translation (NAT) is employed at end-user routers in home and small corporation networks. Furthermore, the unresponsiveness of UDP streams can lead to excessive congestion at the bottlenecked router. While media scaling can be effectively employed by streaming applications to respond to network congestion, it is often not deployed in an appropriate congestion avoidance fashion as shown in Section 5.1 and [93]. Using streaming repair techniques [11, 82, 95, 102], can partially or fully conceal UDP packet losses thus reducing the incentive for UDP flows to be responsive to congestion.

Recent research has proposed TCP-Friendly streaming transport protocols [44, 114, 113, 131] in the hope they will be used by streaming media applications. However, most TCP-Friendly protocols focus on achieving smooth and TCP-Friendly transmission rates, but with little concern for an application programming interface (API) to meet media scaling performance requirements. Thus, most proposed TCP-Friendly protocols are not streaming-friendly in that they make it difficult to acquire network information and effectively perform media scaling over the protocols. Additionally, very little progress has been made on deployment of TCP-Friendly protocols because they do not have stability equivalent to that of TCP and cannot obtain support from current firewalls. Obviously, TCP-Friendly protocols without support from current firewalls do not appeal to streaming media that choose TCP for its availability in the presence of firewalls. Such protocols do not appeal to UDP-based streaming media as well, since they provide little benefit to the streaming applications in return for restricting their network usage.

Taking into account the problems with both UDP and TCP-Friendly protocols for streaming applications, recent research analyzes cases where TCP streaming provides satisfactory performance [129], and suggests ways to improve streaming performance over TCP [117, 72, 31]. Nevertheless, TCP streaming remains problematic due to following reasons: 1) Performing media scaling over TCP is difficult, since TCP's API hides network information such as packet loss rate and round-trip times that are essential for making efficient media scaling decisions; 2) Overestimating the available TCP data rate leads to stream quality degradation due to large queuing delays at the TCP sender as shown in Section 5.1; 3) TCP's reliable in-order packet delivery often induces large frame reception jitter that can interrupt streaming media playout.

When a TCP sender's transmission rate is less than the streaming bitrate, media frames are queued and delayed at the sender's protocol buffer. Since high-quality media frames can block transmission of lower-quality media frames in the sender buffer, the delay added by the protocols sender buffer can significantly degrade stream quality when the streaming system is downscaling the media quality. While delay-aware TCP input queue adaptation, such as in [48], can reduce media frame reception jitter, the queue length adaptation alone does not resolve the difficulties in obtaining network state information over TCP nor reduce retransmission-induced jitter.

This section presents Multimedia Transport Protocol (MTP), a modified form of TCP for streaming that favors prompt and timely datagram delivery service over reliable in-order transmission service. By removing retransmissions from the TCP protocol, MTP instead sends the packet with the highest sequence number in place of a retransmission. By removing both retransmissions and ordered packet delivery from TCP, MTP reduces the high delay and jitter characteristics that make TCP impractical for interactive applications, and makes network information such as packet loss and round-trip time transparent for making media scaling decisions.

MTP offers two modes of transmission at the API: 1) non-blocking transmission mode offering UDP API semantics, and 2) the block-on-full-queue mode of default TCP. It is a common practice that streaming media servers, particularly ones over UDP, operate in a rate-based frame transmission mode, a so called “fire-and-forget” mode, receiving no indications of packet delivery [105]. This often is true even if RTCP [116] is used to get receiver information. The non-blocking transmission mode of MTP supports the contemporary rate-based streaming applications by requiring little modification to switch to MTP. Offering UDP transmission semantics requires MTP senders to use a best-effort queue management mechanism to drop packets from applications when the sender buffer is full. For implementation of the non-blocking transmission API, MTP uses a simple drop-front queue management that works well with streaming media. On the other hand, the block-on-full-queue mode of MTP offers prospective streaming applications an advanced control over frame transmissions. For both transmission modes, MTP may additionally use a dynamic queue length adaptation mechanism introduced in [48] to support interactive streaming applications.

MTP has advantages over other TCP-Friendly transport protocols: 1) MTP has the proven stability of TCP since it has the exact congestion avoidance mechanism of TCP; 2) MTP can be implemented as a mode for TCP in the deployment phase as well as a separate protocol to get full support from the existing firewalls; 3) MTP can be easily made available for all operating system distributions, since an MTP implementation can reuse most of an existing TCP implementation; 4) Existing UDP streaming applications can easily switch to MTP with a minimum change using the non-blocking transfer mode, while new streaming applications may use the block-on-full-queue option as well. In addition, the Internet community proposes to build an unreliable transport protocol incorporating end-to-end congestion control, called Datagram Congestion Control Protocol (DCCP) [71]. DCCP proposes to support a TCP-like window-based congestion control mechanism (Congestion Control ID 2) similar to MTP and to support TFRC [44] (Congestion Control ID 3), a rate-based end-to-end mechanism. The design and evaluation of MTP for streaming media is a valuable contribution toward the design and evaluation of the DCCP ID 2 congestion control mechanism.

MTP evaluation requires a realistic streaming application that performs media scaling and simulates media buffering and playout. While implementing and evaluating MTP on Linux was considered, this approach was dropped when we were unable to find a reasonable open-source streaming application that could be modified to use MTP. Thus the alternative path of evaluating MTP using NS [127] simulations was selected. This required building MTP into NS based on the existing TCP Reno implementation and designing and implementing a video streaming system, called *Goddard*, into NS based on streaming application behavior observed in Section 5.1 and [93].

The simulations show that MTP video streams adapt as quickly as UDP streams to available bandwidth and significantly reduce rebuffering events that are common in TCP streams while maintaining other media qualities such as frame rate and picture resolution at TCP stream levels. The results also show that existing UDP streaming application can use MTP with little modification to their media scaling mechanisms to achieve better quality than TCP streams.

The following sections are organized as follow: Section 5.2.1 presents the design of MTP; Section 5.2.2 presents the design of the Goddard streaming server and client; Section 5.2.3 evaluates MTP in comparison with TCP and UDP using Goddard; and Section 5.2.4 summarizes our conclusions and lists possible future work.

5.2.1 Design of MTP

Multimedia Transport Protocol (MTP) is a TCP modification that disables retransmissions, while preserving the transmission timings and congestion responsiveness characteristics of TCP. MTP performs slow start, congestion avoidance, fast retransmission and fast recovery as does TCP, yet offers a UDP-like transparent API and provides UDP packet delivery semantics. Namely, MTP does not offer guaranteed or in-order packet delivery.

MTP retains the same loss detection and recovery mechanisms of TCP. Instead of a retransmission, the MTP sender temporally inflates its transmission window and sends a new packet.[15] The inflated transmission window is deflated back when an acknowledgment for the retransmission-replacement packet is received. This tempo-

[15]Note, for this discussion the terms segment and packet are used interchangeably.

rary transmission window inflation has the effect of not counting the retransmission-replacement packet sent as a new transmission when making the next new packet transmission decision and is required for MTP to have the same transmission behavior at the network layer as that of TCP.

In MTP, a retransmission replacement packet is marked in the TCP header by the MTP sender to distinguish it from a normal packet at the MTP receiver. On reception of a retransmission-replacement packet, the MTP receiver marks as received the oldest outstanding packet in its packet reception window and advances the receiver window in the same manner as a TCP receiver would upon reception of a retransmitted packet. Additionally, the MTP receiver records the sequence number of the replaced new packet, updates the receiver window again and sends an acknowledgment for the highest consecutively received packet sequence number. An MTP receiver does not hold any received packets in the receiver window for in-order delivery, but provides packets to the application as soon as they are received. Thus, the receiver window contains only sequence numbers of received packets for management of acknowledgment packets.

In summary, an MTP implementation requires a mechanism to keep track of replacement packets in the transmission window at the MTP sender and a bit in the TCP header, referred to as the *replacement bit*, to distinguish retransmission replacement packets from new packets. We implemented MTP in NS by extending the built-in TCP Reno code. Thus, the subsequent discussion describes the behavior of MTP built upon TCP Reno.

5.2.1.1 Duplicate Acknowledgment Management

The general description of MTP behavior given in the previous section is a brief summary of MTP's behavior upon receiving three duplicate acknowledgments. When encountering duplicate acknowledgments, the MTP source performs congestion avoidance, fast retransmission and fast recovery as does TCP, and yields identical congestion window movement and packet transmission timings. However, unlike TCP, on reception of a triple duplicate acknowledgment, MTP inflates the transmission window size by the number of not-yet-acknowledged retransmission-replacement packets including the current one, advances the highest sequence number sent thus far, and transmits a new data packet in the place of the retransmission with the highest sequence number and the replacement bit set in the TCP header. In the case there is no new data available in the input buffer on receiving a triple duplicate acknowledgment, the MTP sender transmits the acknowledged lost packet in the transmission window. Retransmission-replacement packets received by a MTP receiver are handled as described in the previous section.

Figure 5.14 provides an example of MTP duplicate acknowledgment management behavior as compared to TCP Reno. In this example, both the TCP and MTP senders detect a single packet loss while they have the congestion window size (*cwnd*) of 6 and are in congestion avoidance mode indicated by *cwnd* equal to the slow start threshold (*ssthresh*). TCP retransmits the lost packet 11 when getting triple duplicate acknowledgment for packet 10 and halves *cwnd* to 3. On the other hand, MTP transmits the retransmission replacement packet 19 when detecting loss of packet 11 and halves *cwnd* to 3. Thus, although the packet sequence number advancements are different, the congestion window movement and packet transmission timings at the network packet level of TCP and MTP are identical for a single packet loss in a window. This is also true for multiple packet losses in a window and for acknowledgment packet losses as long as the sender does not timeout due to lack of duplicate acknowledgments or loss of a retransmitted (or retransmission-replacement) packet during fast recovery.

starting *ssthresh*=6		Sender State				Receiver State
		Ack	Seq	Wnd	Note	Recv Window
TCP		10	16	6		10
		10	17	6+1	dup1	___ 12
		10	18	6+2	dup2	___ 12 13
	fast-retransmit	10	11	3+3	dup3	___ 12 13 14
	fast-recovery	10		3+4	dup4	___ 12 ... 15
		10		3+5	dup5	___ 12 ... 16
		10	19	3+6	dup6	___ 12 ... 17
		10	20	3+7	dup7	___ 12 ... 18
		18	21	3		18
MTP		10	16	6		10
		10	17	6+1	dup1	___ 12
		10	18	6+2	dup2	___ 12 13
	fast-retransmit	10	19*	3+3+1	dup3	___ 12 13 14
	fast-recovery	10		3+4+1	dup4	___ 12 ... 15
		10		3+5+1	dup5	___ 12 ... 16
		10	20	3+6+1	dup6	___ 12 ... 17
		10	21	3+7+1	dup7	___ 12 ... 18
		19	22	3		19

Figure 5.14: An Example Duplicate Acknowledgment Management of TCP (Reno) and MTP: Wnd = congestion window + the number of duplicate acknowledgments (+ the retransmission replacement inflation for MTP), and the notation n^* for MTP represents packet n with the retransmission replacement bit set in the TCP header.

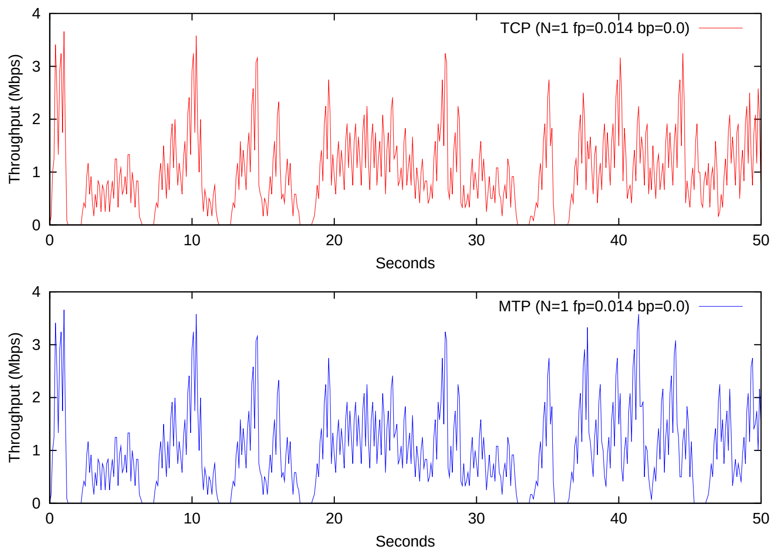


Figure 5.15: TCP versus MTP Throughput: The forward network packet loss rate $p_f = 0.014$, the backward packet loss rate $p_b = 0.0$, the round-trip time $RTT = 60$ ms and the bottleneck capacity $C = 100$ Mbps.

Figure 5.15 compares the throughput (measured in 100 ms intervals) of a single TCP and a single MTP flow on a simulated network with capacity of 100 Mbps, round-trip link delay of 60 ms and forward direction packet loss rate $p_f = 0.014$, where the same simulation was run once with a TCP flow and then again with an MTP flow with packet drops generated by the same random seed. The two lines on top of each other before the timeout around 37 seconds show that MTP packet transmission characteristics are identical to that of TCP as long as the senders can effectively detect lost packets via the triple duplicate acknowledgment mechanism. In addition, the TCP and MTP throughputs are identical even for the five timeouts before 37 seconds indicating that TCP and MTP transmission characteristics are equivalent even for some timeout recovery situations. However, the timeout recovery behavior of MTP can be slightly different from that of TCP under some conditions, as discussed in detail in the next section.

5.2.1.2 Retransmission Timeout Recovery

Retransmission timeout in TCP and MTP occurs in two cases: 1) When there are not enough outstanding packets to generates three duplicate acknowledgments for all the lost packets in a window, or there is a failure to deliver the acknowledgments to the sender due to lost acknowledgment packets; 2) When a retransmitted or retransmission-replacement packet generated during fast retransmit is also lost, making the sender unable to return to the congestion avoidance mode after fast recovery.

On a retransmission timeout, the MTP sender tries to recover from the timeout in a manner similar to that of a TCP sender. That is, MTP performs slow start until the congestion window ($cwnd$) reaches the slow start threshold ($ssthresh$), and returns to congestion avoidance mode. Yet, unlike TCP which restarts the transmission from one below the highest consecutively acknowledged packet sequence number (*hi_ack*), MTP restarts by transmitting a new packet[16] with the replacement bit set and advances the highest sequence number sent so far (*max_seq*) by one.[17].

At the beginning of timeout recovery, *cwnd* is set to one. Thus, if the size of the sender's transmission window is computed as *max_seq* minus *hi_ack* as in TCP, MTP cannot transmit a new packet. To resolve this situation, MTP uses a new sender state variable called *transmission window base* (*wnd_base*) as the lower bound of the the transmission window for computing the transmission window size. When MTP is not in a timeout recovery, *wnd_base* is set to *hi_ack* whenever *hi_ack* is updated upon the arrival of a new acknowledgment packet. This yields the same transmission window size as in TCP. However, at the start of a timeout recovery, the MTP sender sets *wnd_base* to *max_seq* before sending out a new packet and declares timeout recovery until *hi_ack* is less than or equal to *wnd_base*.

During a timeout recovery, all packets are marked as retransmission-replacement packets by the MTP sender before transmission. However, since the sender does not know the state of the receiver's window upon restart, the sender cannot determine whether a new packet sent is indeed used as a retransmission replacement packet or not. Thus, the MTP sender does not inflate the transmission window on the transmission of an intended retransmission-replacement packet. Instead, the sender monitors acknowledgment packets during timeout recovery and inflates the transmission window when a new acknowledgment packet with the acknowledgment number less than or equal to *wnd_base* is received. Otherwise, since the updated *hi_ack* (with the new acknowledgment number) is greater than the value of *wnd_base*, MTP comes

[16]In case there is no new application data to send, the MTP sender waits until new data is available.

[17]In an OS implementation stack, the packet sequence number and window advancement should use bytes instead of packets.

starting	Sender State				Receiver State
$ssthresh$=2	Ack	Seq	Wnd	Note	Recv Window
		11	1		___ 12 13 ___
TCP	13	14,15	2		13 ___
	14	16	2		14
	15	17,18	3		15
		15^*	1	wnd_base	___ 12 13 ___
MTP	13	16^*17^*	2+0+1	= 14	___ 15
	16	18	2	wnd_base	16
	17	19,20	3	$= hi\_ack$	17

Figure 5.16: Example Recovery of TCP and MTP from a Timeout Due to Lack of Duplicate Acknowledgments: Packets 11 and 14 are lost in the previous transmissions. Notions (Wnd and *) are the same as in Figure 5.14.

out of the timeout recovery, sets the wnd_base to the updated hi_ack and continues transmission in the normal transmission mode.

Figure 5.16 provides an examples that illustrates the timeout recovery transmission behavior of MTP in comparison with TCP. In the example, the loss of packets 11 and 14 in the previous transmission window when $cwnd = 4$ cause the retransmission timeout and $ssthresh$ before the timeout is 4. When the retransmission timer expires, TCP reduces $cwnd = 1$ and $ssthresh = 2$ and retransmits the packet hi_ack+1 (packet 11). When the acknowledgment comes back for packet 13, TCP increases $cwnd = 2$ and retransmits packet 14 and transmits a new packet 15. Since in this example, packet 14 was also lost in the previous transmission, the retransmitted packet 14 was useful.

When the timeout occurs, MTP sets wnd_base to max_seq, which is 14 in the example, and transmits packet 15 setting the replacement bit in the TCP header. The receiver, upon reception of the retransmission-replacement packet 15, sends acknowledgment for packet 13. On receiving the acknowledgment, the sender updates $hi\_ack = 15$ and $cwnd = 2$. Then, the MTP sender compares hi_ack with wnd_base, and inflates the transmission window by one as $hi\_ack \leq wnd\_base$. The sender

starting	Sender State				Receiver State
ssthresh=2	Ack	Seq	Wnd	Note	Recv Window
TCP		11	1		___ 12 ___ 14
	12	$13,14^1$	2		___ 14
	14	15,16	2		14
	14^1	17	2+1	dup1	14
	15	18	3		15
MTP		15^*	1	wnd_base	___ 12 ___ 14
	12	16^*17^*	2+0+1	= 14	___ 14 15
	16	18	2	wnd_base	16
	17	19,20	3	$= hi\_ack$	17
	18	21	3		18

Figure 5.17: Example Recovery of TCP and MTP from a Timeout Due to Lack of Duplicate Acknowledgments: Packets 11 and 13 are lost in the previous transmissions. Notions (Wnd and *) are the same as in Figure 5.14.

transmits packet 16 and 17 as it can transmit up to $max\_seq - wnd\_base +$ the retransmission replacement inflation (rrp_inf). Additionally, packets 16 and 17 are marked as able to replace retransmissions as the MTP sender is still in timeout recovery mode. Upon receiving the acknowledgment for packet 16 ($> wnd\_base$), the sender comes out of the timeout recovery by setting $wnd\_base = hi\_ack = 16$ and returns to congestion avoidance mode.

The traffic scenario in Figure 5.16 is a special case of dual packet loss that yields the identical packet transmission timings for TCP and MTP. In general, TCP and MTP transmission timings are identical for timeouts due to a single packet loss for a small *cwnd* or due to two packet losses in a window where the most recently sent packet is lost as in Figure 5.16. In other cases, MTP behaves slightly more efficiently than TCP.

Figure 5.17 illustrates the behavior of TCP and MTP under the transmission scenario that is identical to the scenario of Figure 5.16 except that packet 13 is lost instead of packet 14. In TCP, the retransmitted packet 14 is wasted, since the receiver already has packet 14 in the receiver buffer. This also generates a duplicate acknowledgment that delays advancement of *cwnd* at the sender. On the other hand, by transmitting new packets each time, MTP avoids duplicate acknowledgments during timeout recovery and may achieve a higher goodput than TCP. Although TCP and MTP transmission timings may be slightly different after a timeout in general, this makes little difference on their throughput as long as the TCP sender does not face three duplicate acknowledgments by unwisely retransmitting packets that the TCP receiver already has in the buffer. In such a case, the TCP sender unnecessarily reduces its congestion window. The next example in Figure 5.18 illustrates this point.

Figure 5.18 shows TCP and MTP transmissions on a timeout due to loss of retransmitted (or retransmission replacement) packet. The example assumes that both the original packet 12 and its retransmission (or the replacement) during fast recovery are lost, and the retransmission timer expires when there are 4 outstanding packets. In addition, it is also assumed that both *cwnd* and *ssthresh* are reduced from 8 to 4 due to a triple duplicate acknowledgment event prior to the timeout. As the timeout occurs, both the TCP and MTP senders reduce $cwnd = 1$ and $ssthresh = 2$, and transmit their first recovery packet, packet 12 for TCP and packet 32 for MTP. After the first timeout recovery transmission, each sender gets four duplicate acknowledgments generated by the outstanding packets before the timeout. As a result, the TCP sender retransmits packets 13, 14, 12, 15 and 16 before it receives the acknowledgment generated by the first packet 12 retransmitted at the timeout, and falsely reduces its *cwnd*. These packets again cause 5 duplicated acknowledgments after the TCP sender receives acknowledgment for packet 30, and force the second unnecessary *cwnd* reduction.

The MTP sender also receives the four duplicate acknowledgments generated by the outstanding packets before the timeout, and falsely reduces its *cwnd* for the first time. However, the MTP sender does not cause further duplicate acknowledgments,

		Sender State				Receiver State
starting *ssthresh*=2		Ack	Seq	Wnd	Note	Recv Window
TCP			12	1		___ 13 ... 26
		11	13^1	1+1	dup1	___ 13 ... 26 27
		11	14^2	1+2	dup2	___ 13 ... 26 27 28
		11	12^3	1+3	dup3	___ 13 ... 26 27 28 29
		11	$15^4 16^5$	1+4	dup4	___ 13 ... 26 27 28 29 30
		30	31,32	2		30
		30^1	33	2+1	dup1	30
		30^2	34	2+2	dup2	30
		30^3	31^6	1+3	dup3	30
		30^4	35	1+4	dup4	30
		30^5	36	1+5	dup5	30
		31		2		31
		32		2		32
		33		3		33
		34	37	3		34
		34^6	38	3+1	dup1	34
		35		3		35
MTP	*wnd_base* = 31		32*	1		___ 13 ... 23 ___ 25 26 27
		11	33*34*	1+1+1	dup1	___ 13 ... 23 ___ 25 ... 28
		11	35*36*	1+2+2	dup2	___ 13 ... 23 ___ 25 ... 29
		11	37*	1+3+3	dup3	___ 13 ... 23 ___ 25 ... 30
		11	38*39*40*	1+4+4	dup4	___ 13 ... 23 ___ 25 ... 31
		23		2+0+5		___ 25 ... 32
	wnd_base = *hi_ack*	33		2		33
		34		2		34
		35		3		35
		36		3		36
		37		3		37
		38	41,42	4		38

Figure 5.18: Example Recovery of TCP and MTP from a Timeout Due to Loss of a Retransmitted Packet: This example assumes 4 outstanding packets at the time of the retransmission timeout. Notions (Wnd and *) are the same as in Figure 5.14.

since it transmits new packets with the advanced sequence number, and quickly returns from the timeout recovery. In the beginning of the timeout recovery, MTP is confused by the four duplicate acknowledgments and overestimates the number of replacement packets, inflating the transmission window to the replacement packets plus the number of duplicated acknowledgments. However, the overestimation has little affect on the overall throughput of MTP, since the inflated amounts of the window are deflated as soon as the sender receives a new acknowledgment and returns from the timeout recovery. The sender has to defer the next new packet transmission until the transmission window is reduced below *cwnd*.

5.2.1.3 TCP-Friendliness

MTP may improve goodput over TCP by avoiding retransmission during a timeout recovery. However, the fact that TCP may falsely back off due to unnecessary retransmissions while MTP does not may bring up a concern about the TCP-Friendliness of MTP. In practice, TCP may rarely get triple duplicate acknowledgments during a timeout recovery, and even so, it does not significantly reduces the TCP throughput. Moreover, newer versions of TCP have an option to use selective acknowledgment (SACK) [86], which remedies this drawback of Reno TCP.

Figure 5.19 compares the aggregate throughput of 25 MTP flows with that of 25 TCP flows for simulated networks with different packet loss rates. This simulation uses a dumbbell topology with a 100 Mbps links, where the round-trip delay of 25 source-edge node pairs are randomly distributed over the range [140,480] ms. Random packet drop modules are installed at the forward and backward backbone links and the backward packet drop rate is set to $p_b = 0.10$ in order to create a fair amount of acknowledgment compression. Then, the forward packet loss rate (or ECN marking rate for ECN traffic) p_f is varied over 0.01, 0.05 and 0.10. The same simulation is run with same random seed twice for each p_f, once with 25 TCP flows and once with 25 MTP flows.

Figure 5.19 (top) shows that the aggregate throughput of 25 TCP flows and 25 MTP flows are nearly identical to each other for all forward packet drop rates verifying the TCP-Friendliness of MTP flows. Figure 5.19 (bottom) demonstrates that

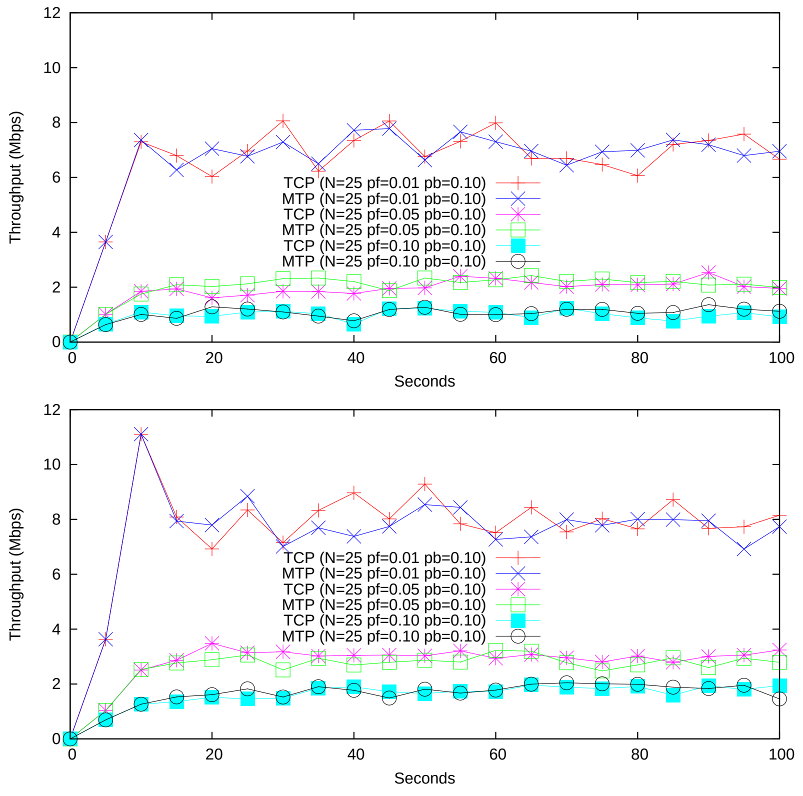


Figure 5.19: Aggregate Throughput of 25 TCP and 25 MTP Flows: Drop (Top) and ECN (Bottom)

ECN enabled MTP flows are also TCP-Friendly, achieving competitive throughput to that of ECN enabled TCP flows. In addition, by avoiding packet losses, the ECN flows achieve a little higher throughput than the non-ECN counterparts under the same p_f. MTP with ECN should further improve streaming media quality since media frame losses due to network congestion can be minimized.

5.2.1.4 Application Programming Interface

MTP offers two modes of transmission at the application programming interface (API): block-on-full-queue mode offering TCP transmission API semantics, and non-blocking transmission mode offering UDP transmission API semantics.

When the block-on-full-queue transmission mode is used, a user process sending a packet is blocked while the MTP sender buffer is full. The block-on-full-queue mode is for prospective streaming applications or existing TCP streaming applications that use non-blocking socket write system calls in a polling manner to detect whether the transport sender queue is full and use the information to make media scaling decisions. For the block-on-full-queue transmission mode, an MTP sender may also use the dynamic queue length adaptation scheme introduced in [48] to reduce MTP queuing delays. The block-on-full-queue transmission mode is not discussed or evaluated further in this dissertation report.

The non-blocking transmission mode, which can also be used with dynamic queue length adaptation for delay critical interactive streaming, is designed for existing UDP streaming applications, so that they can easily adapt to MTP with little modification. In order to implement the non-blocking transmission API, MTP uses drop-front queue management on the MTP sender input queue. Drop-front queue management works better with delay sensitive streaming media transmissions than does drop-tail queue management, because a prompt notification of packet loss helps streaming applications to timely perform media scaling or selective retransmission. Especially when scaling down, dropping packets from high quality media frames in the queue helps timely transmission of the lower quality media frames. Thus, drop-front queue management helps prevent the media buffer at the application receiver from running out of frames avoiding an interrupt in the media playout due to re-buffering. As in the block-on-full-queue mode, the non-blocking transmission mode can also be used with a dynamic queue length adaptation mechanism to enhance the support for delay-critical interactive streaming. Evaluation of such a combination is left as future work.

Currently, the MTP implementation in NS does not provide packet loss rate or round-trip time (RTT) information at the API. However, applications can measure packet (or frame) loss rate or RTT themselves over MTP. Thus, MTP without any specific API changes from TCP still provides a transparent API that allows applications to estimate network information of their interest. Future work include modifying MTP API to provide network information such as packet loss rate and RTT. Packet loss rate and RTT together can be used by the application to estimate the available MTP bitrate that is necessary to make informed media scaling decision. In addition, packet loss rate information is required by streaming servers to determine the amount of forward error correction (FEC) needed to maximize the stream quality. Also, the RTT of the path should be measured to make efficient selective retransmission decisions by streaming applications. Yet, the exact network information, computation and format that should be provided to be useful for media scaling and FEC requires further study.

5.2.1.5 Discussion on TCP Options

TCP has an option to use selective acknowledgment (SACK) [86]. While the TCP SACK option is beneficial when retransmitting packets, it is not particularly useful for MTP. Since MTP transmits a new segment in place of a retransmission, it does not need to know exactly which packet is lost in the network. When MTP API is extended to provide applications of network packet loss rate, the SACK option could help more accurately compute network packet loss rate at the MTP sender. However, it is usually sufficient for MTP senders to measure packet loss event rates without SACK as an estimate of the packet loss rate.

TCP's delayed Acknowledgment (ACK) serves to give the application an opportunity to send an immediate response, in which case the ACK can be piggybacked with the packet carrying the response. This saves the network bandwidth and reduces the protocol processing overhead. However, delayed ACK option does not improve MTP performance in most cases since most MTP application communications are one-way streaming. Even for two-way interactive streaming, delayed ACKs do little to improve MTP performance, since conversation streams in each direction take place in turn. Delayed ACKs may even have a negative impact on low bitrate interactive streaming such as voice over IP (VoIP), since the transmission of voice samples, each of which fits into single MTP packet, can be further delayed in slow start.

5.2.2 Goddard Streaming Client and Server

We design and implement in NS a streaming system (client and server) called *Goddard*.[18] Goddard is designed based on the behaviors observed in the RealVideo streaming measurement study in Chapter 5.1 and Windows Media Player streaming measurement study in [93]. The Goddard streaming client and server use packet-pairs [10, 65, 70] to estimate the bottleneck capacity and select an appropriate media encoding level before streaming. During streaming, the Goddard client and server re-select the media to stream (i.e., perform media scaling) in response to network packet losses or re-buffering events that occur when the client playout buffer empties. Goddard also simulates frame playout of the received media at the client, allowing frame rate and jitter to be measured for performance evaluation. To the best of our knowledge, Goddard is the first and only realistic streaming system available in NS.[19]

As in commercial systems, the Goddard server supports multiple levels of encoded media that are configured by giving the frame size and the frame rate for each scale level. A sample media scale configuration that simulates multiple level encoding of

[18]Our streaming system is named after Robert Goddard, the “Father of Modern Rocketry” and a WPI alumnus.

[19]The Goddard code is available at http://perform.wpi.edu/downloads#goddard.

Level	Frame Size	Frame Rate	Bitrate
0	1 KB	10 FPS	80 Kbps
1	1 KB	15 FPS	120 Kbps
2	2 KB	15 FPS	240 Kbps
3	2 KB	20 FPS	320 Kbps
4	4 KB	20 FPS	640 Kbps
5	4 KB	30 FPS	960 Kbps
6	8 KB	30 FPS	1.92 Mbps

Table 5.2: A Sample Media Scale Levels

Parameter	Default Value	Description
pkp_timeout_interval	2 seconds	Packet-pair timeout interval
buf_factor	1.5	Buffering rate factor
play_buf_thresh	5 Seconds	Media buffer threshold to start playout
loss_monitor_interval	5 seconds	Loss monitoring interval
downscale_frame_loss_rate	0.05	Down-scale frame loss rate
upscale_interval	60 seconds	Up-scale decision interval
upscale_frame_loss_rate	0.01	Up-scale frame loss rate
upscale_limit_time_factor	3	Up-scale limit time factor

Table 5.3: Goddard Client (Gplayer) Parameters and The Default Values

a high quality Internet video is shown in Table 5.2. In addition, the Goddard server has an option for setting the maximum fragment size for fragmenting large media frames before transmission. Typically, the maximum fragment size would be set to the maximum transmission unit (MTU) of the underlying network. The Goddard client, also called *Gplayer* for Goddard Player, has the configuration parameters shown in Table 5.3. The default parameter values are set based on the observations in Chapter 5.1 and [93].

Similar to commercial streaming systems, the Goddard client and server use three communication channels for a streaming session: a control channel using a TCP connection, a UDP packet-pair channel, and a media streaming channel that can be TCP, UDP or MTP. When setting up a streaming session, the Goddard server sends the list of supported media scale levels to the Gplayer using the control channel. Then, Gplayer sets a timer with *pkp_timeout_interval* and requests the server to send a pair of UDP packets to estimate the capacity of the network path. If any one of the packet-pairs is lost, the packet-pair timer expires and Gplayer will send a request for another packet-pair to the Goddard server. On successful reception of a packet-pair, the capacity of the network path is computed by dividing the packet size by the dispersion [108]. Then, Gplayer selects the largest media scale level with a bitrate less than the computed capacity and notifies the server.

Gplayer also notifies the server of the buf_factor before starting streaming to determine how much the server should increase the transmission rate during media buffering periods. The Goddard client and server operate in two modes: *buffering* or *streaming*. During buffering, the Goddard server transmits the chosen media frames at the rate of buf_factor times the streaming bitrate, where buf_factor used for commercial streams typically ranges from 1.5 to 4 as shown in Chapter 5.1. Gplayer maintains a media playout buffer and a playout threshold ($play\_buf\_thresh$). When the Goddard server starts media transmission in buffering mode, the Gplayer buffers the frames received in the media buffer. When the media buffer size (given in playout time) reaches the $play\_buf\_thresh$, Gplayer tells the server to switch to streaming mode and starts playing the media according to the timing described for the current media scale level. If the media buffer runs out of frames, Gplayer stops media playout and switches back to buffering mode. At this time, Gplayer re-selects the largest media scale level with a bitrate less than the average received throughput for the previous control interval. Then, Gplayer tells the server to transmit frames of the new scale level at the buffering rate, that is the streaming bitrate times buf_factor.

When a Goddard streaming session uses UDP or MTP for the media channel, Gplayer can also use frame loss information to make media scaling decisions. In this case, Gplayer monitors the frame loss rate each time it receives a media frame. When it is at least $loss\_monitor\_interval$ since the last scale adjustment decision was made and the frame loss rate is greater than $downscale\_frame\_loss\_rate$, Gplayer scales the media down one level if the current scale level is not already at the minimum. If the current scale is at the minimum, Gplayer maintains the current scale level. The default value for $downscale\_frame\_loss\_rate$ is set to 0.05 according to [93].

Gplayer also makes decisions to scale the media up to a higher level, but does so slowly and gently. Gplayer increases the scale level by one if the frame loss rate of the stream is less than $upscale\_frame\_loss\_rate$ for $upscale\_interval$ since the last time scaling decision was made and the bitrate of the stream after the increase is less than or equal to the estimated network capacity. Also, in order to reduce the chance of playout interruption, Gplayer limits scaling up to one below the last scale level that caused media re-buffering. This limit on scaling up is heuristically relaxed by one scale level if the stream maintains good quality (i.e., no scale down events) for $upscale\_limit\_time\_factor$ times the $upscale\_interval$. The default for $upscale\_interval$ is set to 60 seconds, a value from the observed range (30 to 90 seconds) during the streaming measurement studies in Chapter 5.1 and [93].

Thus, Goddard simulates a realistic streaming video application that performs media scaling, buffering and playout. Implementation of support for video frame dependencies, selective retransmission or other media repair mechanisms are left as future work.

5.2.3 Evaluation under Drop-Tail Network

This section evaluates MTP by comparing the performance of Goddard streaming video over TCP, UDP and MTP through detailed simulations. The simulations use an extended dumbbell network topology that adds an intermediate node (i_1 to i_k) for each of the end-user node (e_1 to e_k) as shown in Figure 5.20. The intermediate nodes simulate a 750 Kbps DSL modem (symmetric up- and down-link capacity) for

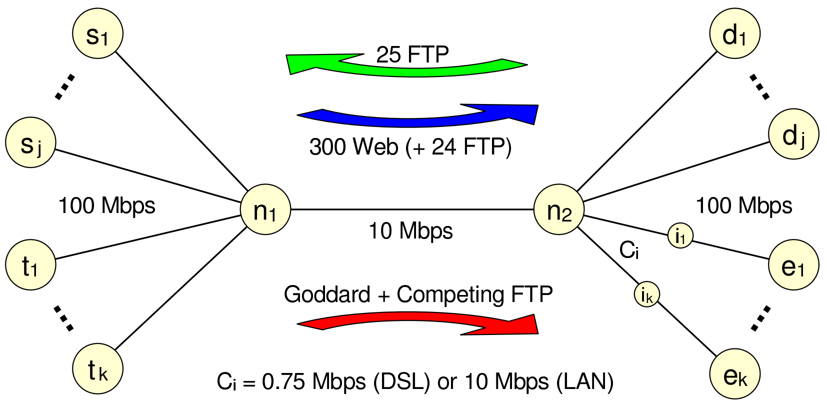


Figure 5.20: Network Topology

the limited local bandwidth study in Section 5.2.3.1, and a 10 Mbps router for the rest of the simulations. The backbone link ($n_1 \leftrightarrow n_2$) capacity is set to 10 Mbps and all the other network link capacities are set to 100 Mbps. The network uses a maximum packet size of 1000 bytes. Round-trip link delays (RTLD) are randomly uniformly distributed over the range [60:1000] ms based on measurements in [67], except for the streaming and the competing FTP paths ($t_k \leftrightarrow e_k$). The RTLD of these paths are set to 140 ms.

All the routers in the simulations use drop-tail queues. The physical queue limit for the drop-tail queues is set to 500 Kbytes for the 10 Mbps links[20] and 20 Kbytes for the 0.75 Mbps DSL simulation link, approximately equal to the bandwidth-delay product for the mean round-trip time.

On the normal dumbbell paths ($s_k \leftrightarrow d_k$), each simulation has 25 backward direction bulk transfer FTP flows and 300 forward direction background Web sessions (using the Webtraf code built into NS) that start evenly distributed during the first 30 seconds. Based on settings from [4, 55], each Web session requests pages with 2 objects drawn from a Pareto distribution with a shape parameter of 1.2 and an

[20]We also set the queue limit for 100 Mbps links to 500 Kbytes. This queue limit has little effects on the traffic, since 100 Mbps links are not saturated.

average size of 5 Kbytes. The Web sessions have an exponentially distributed think time with a mean of 7 seconds, which results in an average utilization of about 2.5 to 3 Mbps of the 10 Mbps capacity, a fraction typical of some Internet links, such as in [115]. Except for the simulations in Section 5.2.3.1 that test the media adaptation behavior of different Goddard streams on the limited local connection link capacity (0.75 Mbps), 24 forward direction bulk transfer FTP flows on the normal dumbbell path are used to congest the backbone link.

The Goddard server uses configuration with the seven media scale levels shown in Figure 5.2 and the fragment threshold of 1 Kbyte. The Goddard client uses the default configuration parameters shown in Figure 5.3.

We evaluate MTP by analyzing the network layer and application layer performances of Goddard video streams over TCP, UDP and MTP. The streaming performance is measured in terms of average sustained scale level, frame rate, durations of both initial buffering and media play, re-buffering event count, TCP-Friendliness, TCP-Friendly rate adaptation time, and frame reception jitter. For lost frames, the frame reception jitter is computed by taking the inter-frame arrival time of two consecutively received frames divided by the number of lost frames between the two received frames plus one. For example, if frame 6 is lost, the inter-frame arrival time between frame 5 and 7 is computed as the reception time difference between frame 5 and frame 7 divided by 2.

5.2.3.1 Limited Local Bandwidth

This experiment shows the media scale adaptation of the Goddard streams over TCP, UDP and MTP on the network where the end-host connection link capacity is less than the maximum stream capability. For this study, the intermediate link capacity (C_i) is set to 0.75 Mbps, a typical capacity of a DSL modem. This creates

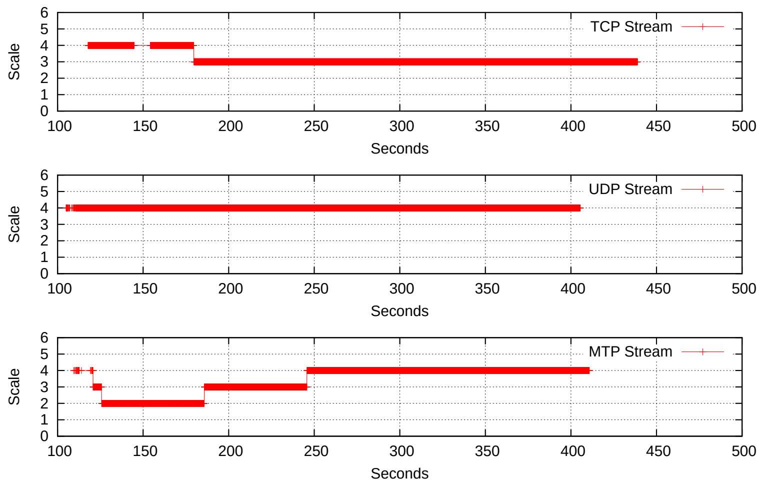


Figure 5.21: Example Media Scale Dynamics (Run 0)

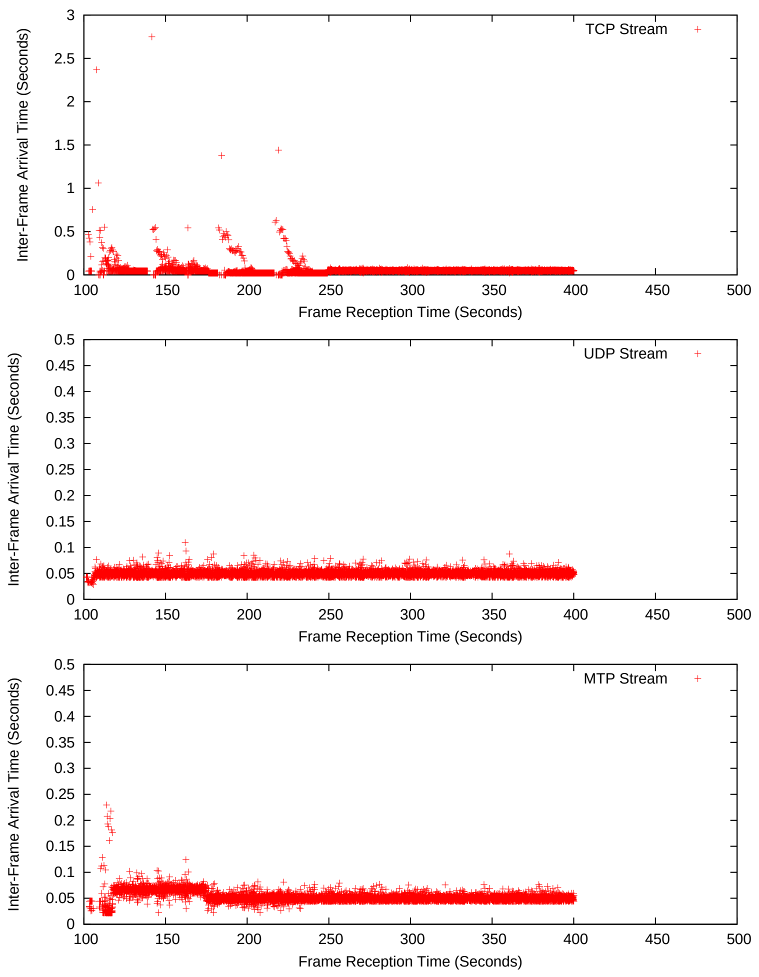


Figure 5.22: Example Media Frame Reception Jitter (Run 0)

a capacity limited streaming condition since the Goddard streams are configured to support the maximum streaming bitrate of 1.92 Mbps. On the capacity limited paths ($t_k \rightarrow e_k$), 3 Goddard streams (TCP, UDP and MTP) and 4 bulk FTP flows are simultaneously started at 100 seconds and stopped at 400 seconds, where the background Web traffic and reverse FTP traffic are running from the beginning of the simulation. This traffic aggregate does not cause congestion at the backbone link, since the total maximum expected traffic rate is well under the backbone link capacity (10 Mbps). The maximum expected traffic rate is 8.75 Mbps (7$\times$0.75 Mbps + 3 Mbps of Web and acknowledgments traffic). However, the traffic is to simulate a utilized backbone link for realistic transmission timing variations to the Goddard streams. We run the simulation 7 times with different random seeds to avoid biases in any one measurement.

Figure 5.21 shows the media scaling dynamics of the TCP, UDP and MTP streams from one of the simulations, and Figure 5.22 shows frame reception jitter for the corresponding streams. Figure 5.21 show that for all three streams the packet-pairs estimation of capacity choose the initial scale level of 4 providing streaming bitrate of 640 Kbps. The streams start transmission in the buffering mode at the rate of 960 Kbps, 1.5 times faster than the streaming bitrate and larger than the local connection capacity of 750 Kbps.

The high buffering bitrate causes TCP to probe for available bandwidth, incurring packet losses at the local link. The TCP stream experiences an unfortunate sequence of packet drops at about 140 seconds, which causes the large frame jitter and results in a media re-buffering event. TCP scales down a media level and stays at that level for the life of the stream. The smooth TCP stream jitter after 240 seconds in Figure 5.22 indicates that the TCP sender does not have enough packets for bandwidth probing as the stream scales down and switches to the streaming mode.

The high buffering bitrate also causes frame losses for the UDP stream in the buffering period. However, the UDP stream tolerates the initial frame losses and maintains its initial scale level, since it can tolerate the packet loss rate for the initial measurement interval which is less than 5%. In addition, the UDP stream maintains low frame reception jitter at the mean (μ) of 50 ms that is the playout interval of the 20 frames-per-second (FPS) video, and a standard deviation (θ) of 6 ms.

The MTP stream scales down the media bitrate like the TCP stream but for a different reason. The small number of packet losses (or frame losses) during MTP bandwidth probing scarcely affect the performance of the stream, since MTP immediately delivers received packets to the application and does not increase the frame reception jitter at the client. However, scale-down decisions are made due to the frame losses that occur during the initial buffering period at the input buffer of the slow-starting MTP sender. Figure 5.21 shows that large frame loss bursts at the start of the MTP stream cause it to scale down two levels consecutively in 10 seconds. The MTP stream does scale back up to the initially selected scale level, but slowly and conservatively due the large up-scale decision interval (60 seconds). These inefficient scaling decisions made by the MTP stream at the initial buffering period can be avoided by using selective retransmission, which is supported by most commercial streaming products [92]. In addition, Figure 5.22 shows that the MTP frame jitter (μ = 52 ms, θ = 11 ms) is as comparably low as the UDP jitter throughout the stream lifetime for the local connection bandwidth constrained condition.

Figure 5.23 and Figure 5.24 provide cumulative distribution function of the streamed media scale level dynamics and the inter-frame reception time (jitter) of all seven simulations. Figure 5.23 shows that the UDP streams stay at level 4, the highest level allowed by the local link, all the time, and the TCP streams stay at level 3 about 85% of the times. The MTP streams achieve the scale level of the UDP

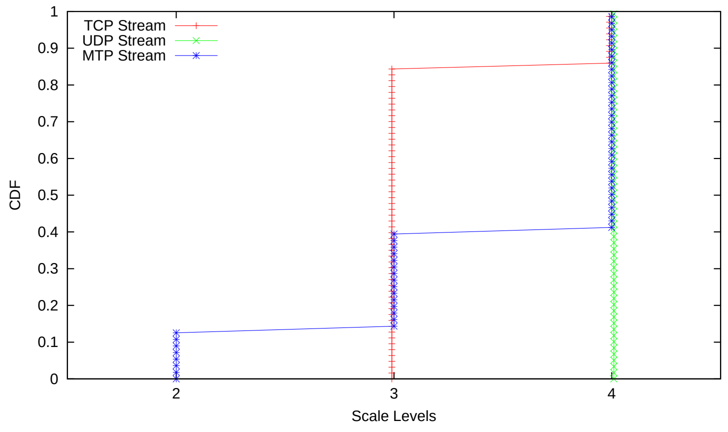


Figure 5.23: CDF of Streamed Media Scale Levels

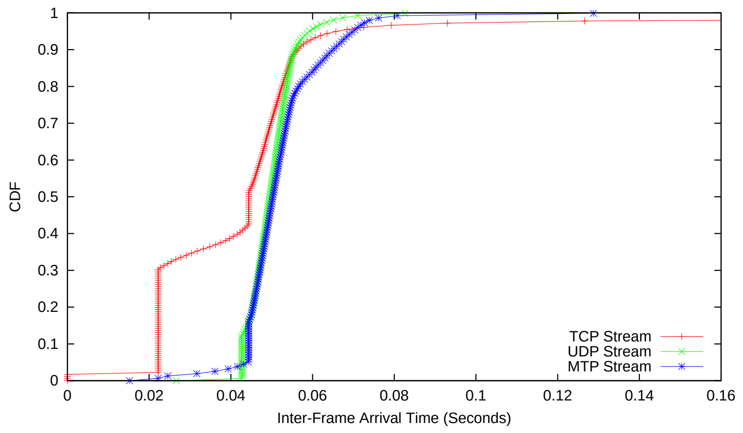


Figure 5.24: CDF of Media Frame Reception Jitter

streams (level 4) about 60% of times, although about 12% of the times they stay at the level 2 due to the inefficient scaling decisions made during the initial buffering periods.

Figure 5.24 confirms that that the frame reception jitter of the MTP streams is as smooth as that of the UDP streams under a normal condition where the local connection link is the only bottleneck of the network path. In addition, Figure 5.24 reveals the ineffectiveness of media scaling over TCP as well as the effect of TCP's reliable in-order packet delivery on frame reception jitter. The TCP streams' inter-

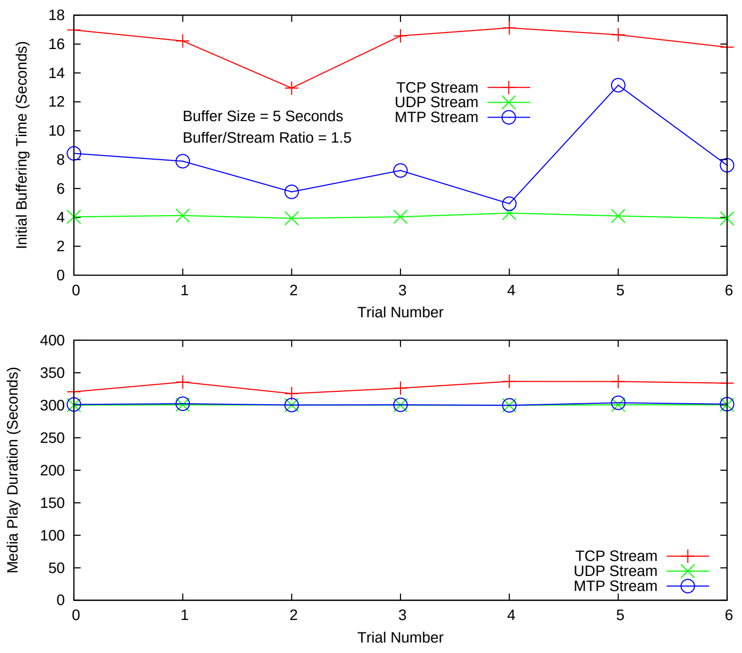


Figure 5.25: Initial Buffering Time, Video Play Duration (including intermediate buffering times)

frame arrival times less than 50 ms, the playout interval of the 20 FPS video (level 3 and 4), indicate that about 40% the frames are sent in the buffering mode and/or delayed in the TCP receiver buffer due to packet losses. Especially, the inter-frame arrival time of zero in Figure 5.24 indicate that two or more consecutive frames are delayed in the TCP receiver buffer and delivered to the Gplayer at once.

Figure 5.25 shows the initial buffering time (top), the streamed video play duration including re-buffering time (bottom) for the TCP, UDP and MTP streams. Figure 5.25 (top) shows that the TCP streams need significantly more initial buffering time than the UDP streams. The MTP streams also require more time for initial buffering than the UDP streams, however MTP streams require only a half as much as the time taken by the TCP streams in average. Figure 5.25 (bottom) shows that the media play durations of the UDP and MTP streams are bounded almost exactly by the playout length of the video. TCP streams do take a little more time to finish

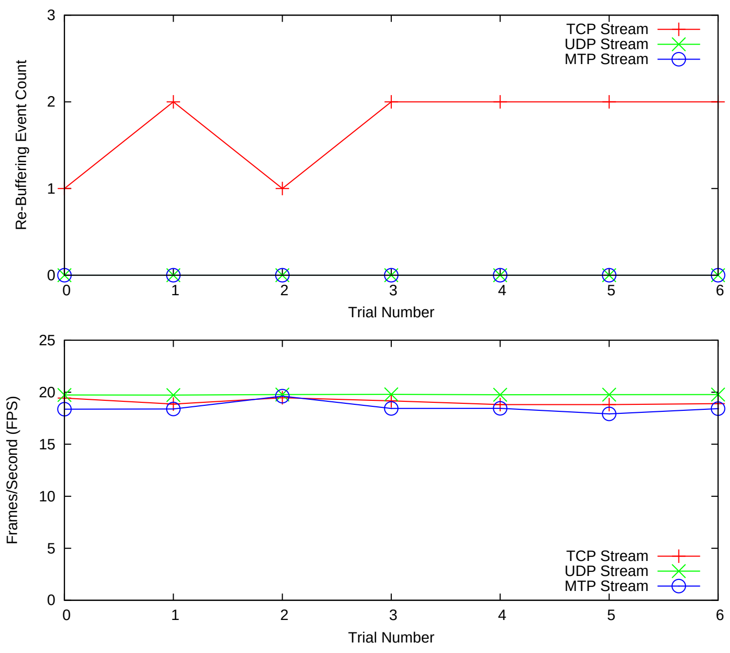


Figure 5.26: Re-buffering Event Counts and Average Frame Playout Rate

playing the video due to re-buffering events.

Lastly, Figure 5.26 shows the re-buffering event count (top) and the average frame playout rate (bottom) computed as the number of played frames divided by the play durations including the play-halted re-buffering periods. Figure 5.26 (top) shows that all the TCP streams incur one or two re-buffering events, while the UDP and MTP streams have none. Figure 5.26 (bottom) shows that the UDP streams achieved frame rate close to the maximum, 20 FPS. The TCP streams achieve about 19 FPS on average although they experience couple of long pauses in media playout due to media re-buffering. The MTP streams achieved between about 18 to 19 FPS, since the MTP streams often scale down to the 15 FPS level-2 encoded media in the beginning. Thus, all three types of stream perform similarly to one another in terms of frame playout rate, although the TCP streams have a couple of pauses during the playout.

This section showed that streaming over TCP, UDP and MTP can achieve comparable performances under a lightly-loaded network condition where the network usage is mostly limited by the local connection capacity. In addition, it is shown that MTP eliminates media play interruptions caused by TCP, and works well with a streaming application designed for UDP with little modification.

5.2.3.2 Backbone Link Congestion

This experiment shows the performance of the Goddard streams over TCP, UDP and MTP on a network path congested with many flows. For this study, the intermediate link capacity (C_i) is set to 10 Mbps, not to be a bottleneck in the network path. In order to congest the backbone link, each simulation uses 24 forward direction bulk FTP flows on the normal dumbbell path ($s_j \leftarrow d_j$) in addition to the forward direction Web traffic and the backward direction FTP traffic. On the other paths ($t_k \rightarrow e_k$), each simulation runs a Goddard stream and a competing bulk FTP flow from 100 seconds to 400 seconds. A simulation set is composed of three simulations with the same random seed but different streaming transport protocols, TCP, UDP and MTP. The set of simulations is repeated 7 times with different random seeds to eliminate skews in the measurement.

Figure 5.27 shows the media scaling dynamics of the TCP, UDP and MTP streams from one of the simulation sets, and Figure 5.28 shows the frame reception jitter of the corresponding streams. During streaming (100 to 400$^+$ seconds), a fair bandwidth share with this traffic load is about 270 Kbps ((10 Mbps $-$ background_traffic) / 26 flows), where the background traffic takes about 2.5 to 3 Mbps. Thus, a TCP-Friendly stream will chose a media scale level of 2 (a 240 Kbps stream) given the traffic conditions.

Figure 5.27 shows that the TCP stream suffers from overestimating the available network bitrate in the beginning. The TCP stream makes a scale-down decision

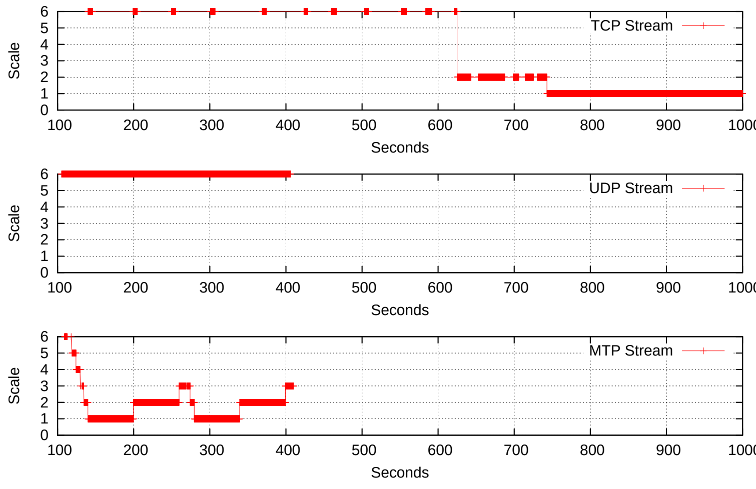


Figure 5.27: Example Media Scale Dynamics (Run 0)

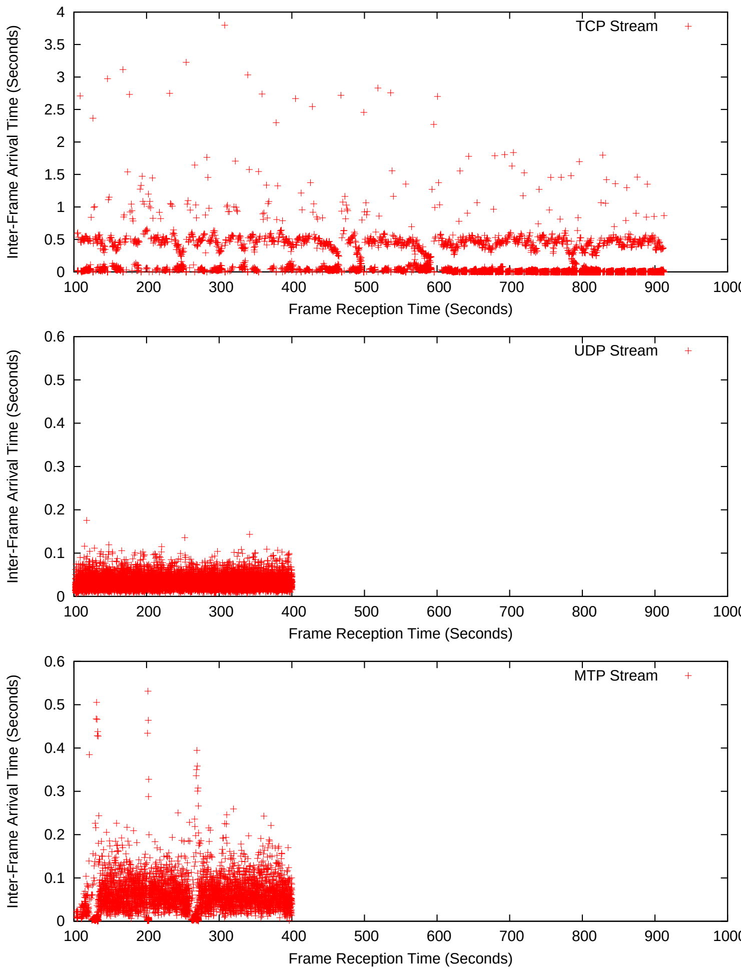


Figure 5.28: Example Media Frame Reception Jitter (Run 0)

about 42 seconds after it starts streaming. During this period, the Goddard server is in buffering mode and pushes about 15 Mbytes of frames into the underlying TCP buffer, taking about 450 seconds to transmit all the high quality frames based on the 270 Kbps fair share assumption. In fact, the TCP stream takes about 500 seconds to drain the frames from the TCP sender queue. This huge TCP sender queue size is due to the unrealistic NS TCP implementation that does not simulate a limited input buffer size. For most current Unix kernels, TCP uses a send buffer of at least 64 Kbytes [48]. Although exaggerated, the TCP streaming simulation results show the difficulty in media scaling over TCP.

Unlike the TCP stream, the Goddard server and client over UDP finishes streaming and playout of the media with no delay. However, it streams the highest quality media that has a streaming bitrate much higher than the fair share, since the streamed media quality is not significantly degraded by small network packet losses. The measured packet loss rate at the backbone link during streaming is about 0.005. This illustrates how a UDP stream can be TCP-unfriendly during congestion, although the stream can adapt to the limited local connection capacity as shown in Section 5.2.3.1.

The MTP stream shown in Figure 5.27 inherits the good characteristics of both the TCP and UDP streams. That is, the MTP stream quickly finds an appropriate scale level in response to network congestion and achieves both uninterrupted stream playout and a TCP-Friendly streaming bitrate. In the beginning, the MTP stream quickly scales down to the level-1 media, and then at 200 seconds, scales up to level-2 media. When the MTP stream tries level-3 media at around 260 seconds, the buffer in the MTP sender overflows and the Goddard client and server scale back.

Figure 5.28 shows that the TCP stream has the highest frame reception jitter (μ = 82 ms, θ = 213 ms), followed by the MTP stream (μ = 64 ms, θ = 60 ms) and the

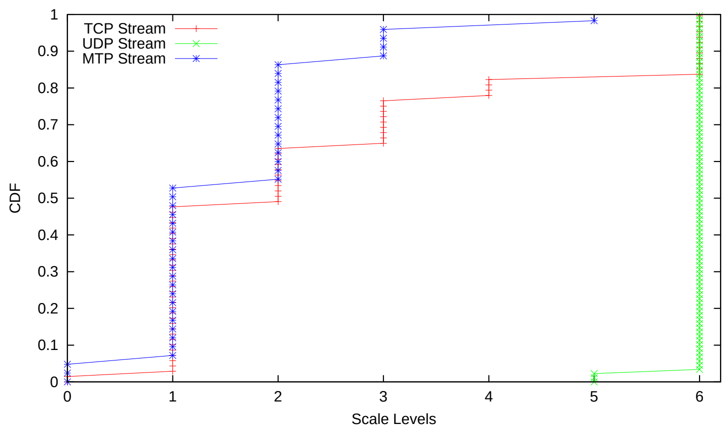


Figure 5.29: CDF of Streamed Media Scale Levels

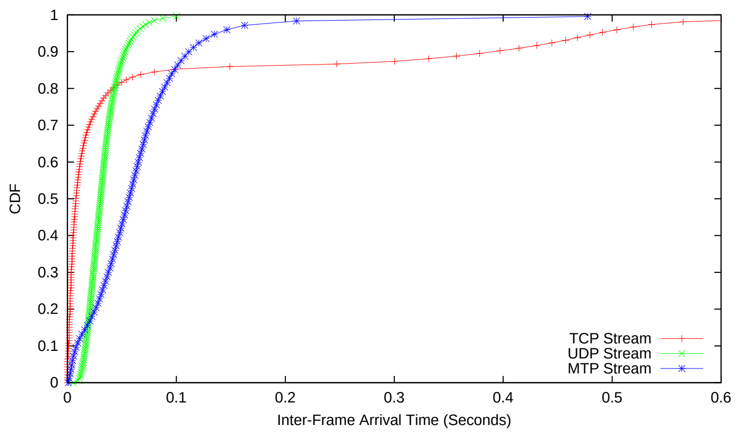


Figure 5.30: CDF of Media Frame Reception Jitter

UDP stream ($\mu = 33$ ms, $\theta = 16$ ms). Comparing the MTP and UDP streams, the UDP jitter is lower than that of MTP. The mean UDP inter-frame arrival time of about a half that of the MTP stream is due to the differences in the scale level. The UDP stream uses the highest quality media with playout interval of 33 ms, while the MTP stream's media playout interval is about 67 ms. Comparing the θ/μ ratio shows UDP is a little smoother than, but comparable to MTP. However, the jitter of the TCP stream is an order of magnitude higher than UDP or MTP, even after the TCP sender clears the highest quality frames from its buffer at 600 seconds. This confirms that the main source of the TCP's streaming unfriendly delay and jitter is the retransmission mechanism that provides reliable in-order packet delivery.

Figure 5.29 and Figure 5.30 summarize the streamed media scale level dynamics and the inter-frame reception time (jitter) of all seven sets of simulations in cumulative distribution functions. Figure 5.29 confirms that Goddard over MTP streams chooses the correct media scale level (it has a media TCP-Friendly bitrate so it must choose a scale level 2 or lower) most of the time. In addition, Figure 5.30 shows that the MTP frame reception jitter is consistently low. About 95% of MTP's inter-frame arrival times are under 134 ms, two times the playout interval (67 ms) of the streamed media. This means that Gplayer can play 95% of the received video frames after a buffering delay of only 67 ms when the one-way delay of the stream path is about 100 ms and the average packet loss rate is 0.44%. Given that streamed video bitrate is about 120 to 240 Kbps, typical of Internet videoconferencing, these results suggest the potential for MTP to be used as a streaming transport protocol for interactive applications as well as non-interactive applications.

Figure 5.31 shows the initial buffering time of the streamed video (top) and the playout duration including re-buffering time (bottom), and Figure 5.32 shows the re-buffering event count (top) and the average frame playout rate (bottom) for the TCP, UDP and MTP streams. Figure 5.31 (top) shows that the TCP streams need significantly more initial buffering time than the UDP and MTP streams. Figure 5.31 (bottom) shows that the media play durations of the UDP and MTP streams are bounded almost exactly by the playout length of the video, while the TCP streams take two to three times longer to playout. Figure 5.32 (top) shows that the TCP streams incur frequent re-buffering events, while the UDP and MTP streams have none or at most one. Figure 5.32 (bottom) shows that the TCP streams achieve a low average frame playout rate due to frequent re-buffering pauses, while the UDP

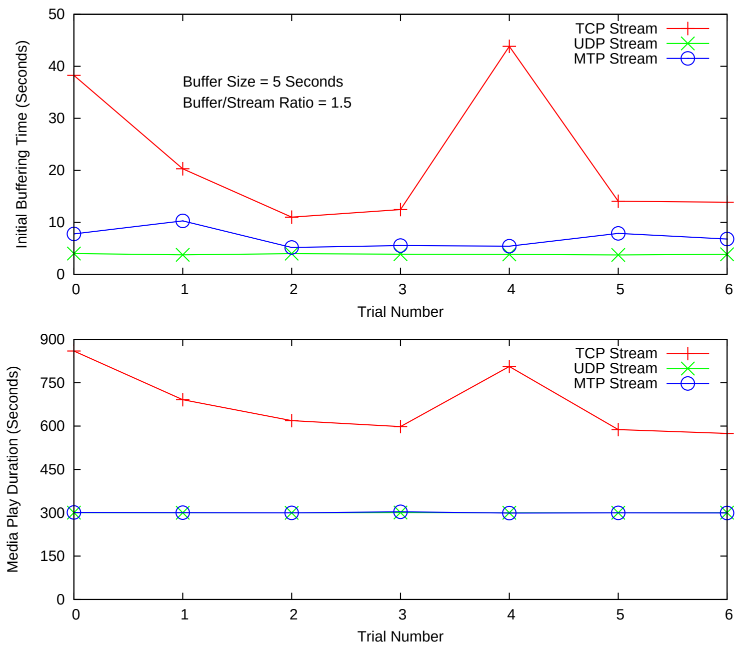


Figure 5.31: Initial Buffering Time and Video Play Duration (including intermediate buffering times)

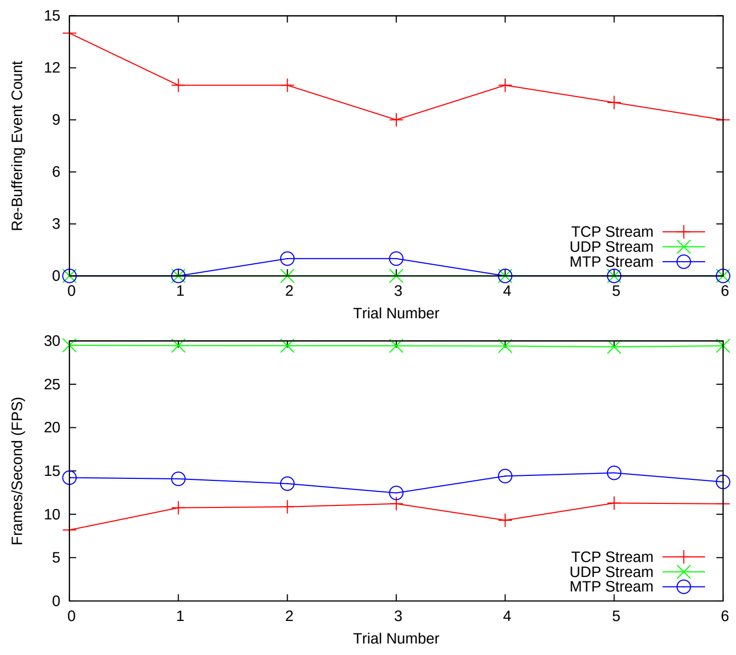


Figure 5.32: Re-buffering Event Counts and Average Frame Playout Rate

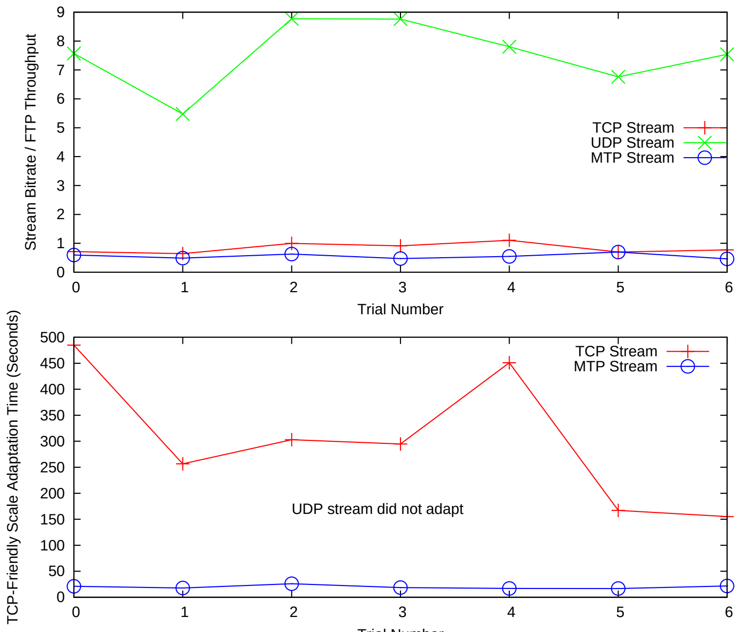


Figure 5.33: TCP-Friendliness of Media Streams and TCP-Friendly Rate Adaptation Time

and MTP streams achieve frame playout rates close to the encoded frame rates.

Lastly, Figure 5.33 (top) shows the average throughput ratio of the media streams in comparison to the competing bulk FTP flow, and Figure 5.33 (bottom) shows the time each stream takes to adapt to the TCP-Friendly rate. The UDP streams are extremely TCP-unfriendly, taking about 6 to 9 times the throughput of the competing bulk FTP, and never adapt to a TCP-Friendly rate. Both the TCP and MTP streams have throughput less than that of the bulk FTP flow, since the Goddard servers do not always have frames available when MTP or TCP can send. The MTP streams quickly adapt their media scale bitrate to the TCP-Friendly rate, while the TCP streams take two to three minutes to adapt their media scale bitrate to the TCP-Friendly rate.

5.2.4 Summary

This section presents the design and evaluation of Multimedia Transport Protocol (MTP). MTP is an alternate to UDP for streaming and other delay sensitive Internet applications that favor prompt and timely datagram delivery service over the reliable transmission service of TCP. Removing retransmissions from TCP removes delays due to retransmission at the TCP sender and packet ordered preservation at the TCP receiver, and allows MTP to reveal network information essential for media scaling. MTP supports non-blocking transmission using input queue management as well as block-on-full-queue transmission to help existing UDP-based streaming applications to switch to MTP with little modification. MTP has the exact congestion avoidance mechanism and proven stability of TCP, and can be implemented as a mode of TCP during incremental deployment to take advantage of firewall support for TCP traffic.

MTP is implemented in NS by modifying the built-in Reno TCP implementation. In order to evaluate MTP, the Goddard streaming client and server are designed and implemented. Goddard estimates the bottleneck capacity, selects the media level to stream, performs media scaling during streaming, and simulates playout of the received media at the client. To the best of our knowledge, Goddard, which also simulates playout of the received media at the client, is the only realistic streaming application in NS.

Our simulation results show that MTP video streams inherit the good characteristics of both TCP and UDP streams. MTP streams are TCP-Friendly, but can still quickly and effectively perform media scaling and adapt to the available TCP-Friendly bitrate. Thus, most of the time, MTP streams avoid interruptions to the streamed media playout. Existing UDP streaming applications can use MTP with little modification to their media scaling mechanisms, achieving better quality streams than TCP streaming applications. Additionally, our simulation results show that

MTP dramatically reduces media frame reception jitter from TCP's reliable in-order packet delivery mechanism, and illustrates the potential of MTP as a streaming transport protocol for interactive as well as non-interactive applications.

When MTP is used with explicit congestion notification (ECN), MTP timeouts can be reduced since network packet losses are reduced and the MTP sender does not need to wait for duplicate acknowledgments to detect network congestion. Therefore, MTP with ECN will further enhance the support for interactive applications. Chapter 6 evaluates ECN-enabled MTP streams under Aggregate Rate Controller (ARC) and Stochastic Fairness Guardian with ARC (SFA).

Chapter 6

Streaming MTP over Crimson

In Chapter 4 and Chapter 5, we presented the design and evaluation of each of the building blocks of the Crimson network that reduces congestion and improves support for delay sensitive multimedia traffic on the Internet: Aggregate Rate Controller (ARC) to manage congestion and minimize network queuing delay, Stochastic Fairness Guardian (SFG) to punish congestion unresponsive and misbehaving traffic, and Multimedia Transport Protocol (MTP) to provide a practical, congestion responsive transport protocol for multimedia streaming. This chapter presents evaluation of media streaming under Crimson. This chapter compares streaming performance over TCP, UDP and MTP under drop-tail queue management, ARC, and the combination of SFG and ARC (SFA).

6.1 Simulation Setup

For the evaluation of streaming over MTP on Crimson, we use the same network topology shown in Figure 5.20, and the same parameter and traffic settings used to evaluate MTP streaming for the backbone link congestion with drop-tail queue

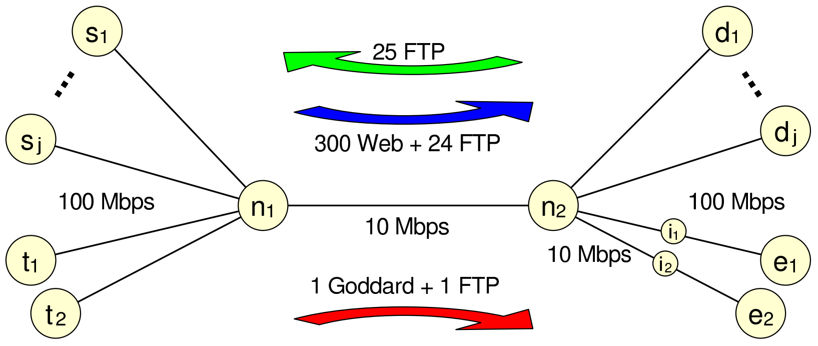


Figure 6.1: Network Topology

management as in Section 5.2.3.2. This time, we replace the drop-tail queue once first with ARC and then with SFA (the combination of SFG and ARC), run the same sets of simulations, and compare the results with those of drop-tail simulations.

The capacity of the backbone link ($n_1 \leftrightarrow n_2$) and the intermediate links ($n_2 \leftrightarrow i_1$ and $n_2 \leftrightarrow i_2$) are set to 10 Mbps while other link capacities are set to 100 Mbps. The physical queue limit is set to 500 Kbytes for all the routers. The link delays of the streaming path ($n_2 \rightarrow i_1$) and the competing FTP path ($n_2 \rightarrow i_2$) are both set to 70 ms giving a 140 ms of round-trip link delay. The round-trip link delays of the other normal dumbbell paths are randomly uniformly selected over the range [60:1000] ms, based on measurements in [67].

For ARC and SFG, the configuration parameter settings described in Section 4.1.3 and Section 4.2.3 are used. Table 6.1 summarizes the parameters and their values.

As each simulation run starts, the backbone link is loaded with 300 Web sessions and 24 FTP flows in the forward direction dumbbell path ($s_j \rightarrow d_j$), and acknowledgments of 25 backward FTP flows ($s_j \leftarrow d_j$). As before, each Web session requests pages with 2 objects drawn from a Pareto distribution with a shape parameter of 1.2 and an average size 5 Kbytes. The Web sessions have an exponentially distributed think time with a mean of 7 seconds, which results in an average utilization of about

	Parameter	Value	Description
SFG	L	3	Number of hash filter levels
	N	20	Number of bins in a level
	d_s	2 seconds	SFG control interval
	m_h	0.02	SFG turn-on CNP threshold
	m_l	0.01	SFG turn-off CNP threshold
ARC	α	1.42×10^{-5}	ARC control parameter
	γ	0.98	Target link utilization
	q_0	0	Target queue length
	d_a	1 second	ARC control interval
	ecn	true	Enable ECN marking

Table 6.1: SFG and ARC Parameters

2.5 to 3 Mbps on the 10 Mbps backbone link. Then, a Goddard stream and a bulk FTP flow are started at 100 seconds on the competition path ($t_1 \rightarrow e_1$ and $t_2 \rightarrow e_2$) and stopped at 400 seconds. The Goddard server uses the configuration with seven media scales shown in Table 5.2 and the fragment threshold of 1 Kbyte. The Goddard client (Gplayer) uses the default configuration parameters shown in Table 5.3.

Goddard streams over TCP, UDP or MTP, where TCP and MTP support ECN. For each of the transport protocols, the same simulations are conducted 7 times with different random seeds to eliminate skews in the measurement. We run this set of 21 simulations for ARC and then for SFA (the combination of SFG and ARC), and compare the results with that of drop-tail queue management presented in Section 5.2.3.2.

6.2 Analysis

Figure 6.2 and Figure 6.3 present the play time and scale levels of media frames for a set of TCP, UDP and MTP streaming simulations under ARC and SFA. Comparing Figure 6.2 and Figure 5.27 in Section 5.2.3.2, it is shown that the TCP, UDP and MTP streams behaves similarly in general with drop-tail queue management and

ARC. That is, Goddard streams experience difficulty in media scaling over TCP and do not scale down to be TCP-Friendly media over UDP, while the MTP streams achieve both uninterrupted media play and TCP-Friendliness. However, a closer comparison shows the effect of the ECN marking of ARC on the TCP and MTP streams, and the effect of uniform random packet-dropping congestion notification of ARC on the UDP stream. The TCP and MTP streams operate at slightly higher media scale levels under ARC than under the drop-tail queue management due to the effectiveness of ECN. Yet, Goddard over UDP with ARC scales down a level and streams the level-5 media (frame size = 4 KB, frame rate = 30 FPS) while the UDP stream with drop-tail stays at the level-6 media (frame size = 8 KB, frame rate = 30 FPS).

For the UDP streams without selective-retransmission and/or media repair, a frame is considered lost even if a single packet of the frame gets lost. With drop-tail queue management, it is likely that consecutively transmitted packets are dropped in a burst when the network queue is full. Thus, the 8 KB media frames fragmented into 1 KB network packets have a high chance of not being dropped when the network queue has room for 8 packets. However, the 8 KB media frames have less chance to survive under ARC, since ARC uniform randomly drops packets of the media frames for congestion notification regardless of room in the queue. The UDP stream with ARC experiences a higher frame drop rate than the sustainable rate for the level-6 media, and picks the level-5 media that reduces the size of media frames in half. Yet, the level-5 media with a streaming bitrate of 960 Kbps is still well over the fair share.

Figure 6.3 shows that SFA (SFG in combination with ARC) can effectively force the UDP stream to scale down, with little affect on MTP streaming performance. Goddard over UDP under SFA streams the level-6 media and gets punished severely by SFG in the beginning. Then, it scales down to and stays at the level-2 and

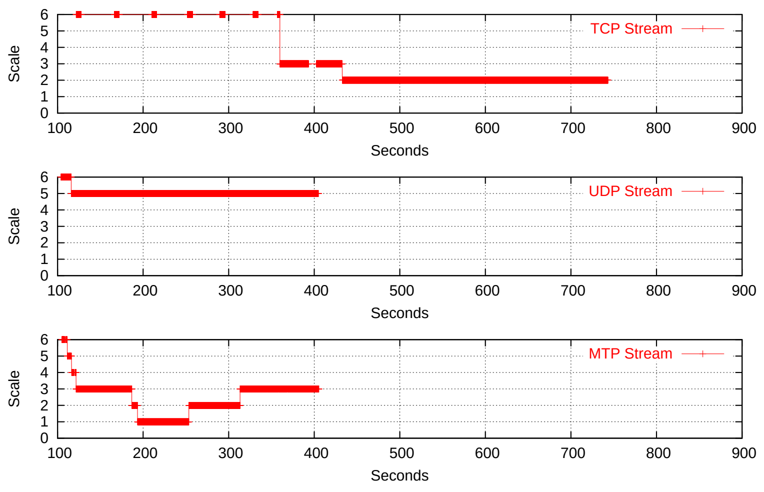


Figure 6.2: ARC: Example Media Scale Dynamics (Run 0)

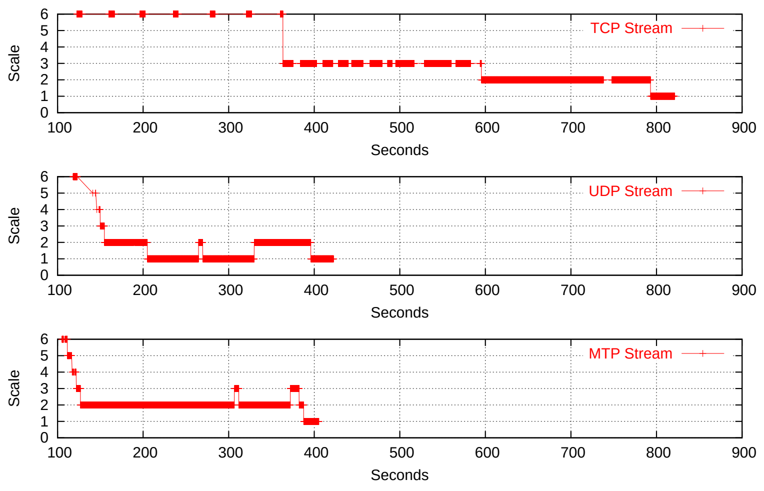


Figure 6.3: SFA: Example Media Scale Dynamics (Run 0)

level-1 media thereafter. The TCP and MTP streams occasionally experience packet drops imposed by the SFG. For the TCP stream, the packet drops cause TCP to retransmit the lost packets, which delays the frame deliveries, and cause the frequent re-buffering events similar to the TCP stream under the drop-tail queue management in Figure 5.27. In contrast, the MTP stream does not experience a re-buffering event, since MTP does not retransmit or delay the media frame deliveries.

Figure 6.4 through Figure 6.9 compares the average streaming performance for the seven runs of TCP, UDP and MTP streams under drop-tail queue management (labeled as DT in the figures), ARC and SFA. Figure 6.4 shows the average media scaling levels of frames played, Figure 6.5 presents the average streaming bitrates and Figure 6.6 graphs the average frame playout rate. Figure 6.7 and Figure 6.8 provide the average length of the initial buffering periods and the media play duration of the streams respectively, and Figure 6.9 depicts the average re-buffering event count for each type of streams.

Figure 6.4 and Figure 6.5 present that Goddard over UDP, on average, streams the highest quality media and consumes 1.87 Mbps of the 10 Mbps backbone link capacity under drop-tail, and streams the level-5 media and consumes about 0.94 Mbps under ARC. Under the supervision of SFA, the UDP streams achieve average scaling level of about 1.4 and consume 140 Kbps that is even below the bitrate of average TCP streams. However, the UDP streams under SFA still achieve an average frame playout rate close to the frame encoding rate of the level-1 and level-2 media (15 FPS), as shown in Figure 6.6, while the average TCP streams achieve only 10 FPS. This is because the UDP streams finish the video playout with very few interruptions as shown in Figure 6.8 and Figure 6.9. The UDP streams experience a significantly longer time to buffer the initial 5 seconds of media frames, as shown in Figure 6.7, since they scarcely stream the highest quality media frames under the punishment of SFG.

Figure 6.4 and Figure 6.5 show the Goddard over TCP achieves a slightly higher average scale level and streamed more bits under ARC and SFA than under the drop-tail queue management, due to ECN, which also improves goodput. However, Figure 6.6 presents that the TCP streams under SFA achieve a low average frame playout rate of about 10 FPS, since they experience a higher number of re-buffering events for the increased streamed media quality than under drop-tail or ARC as

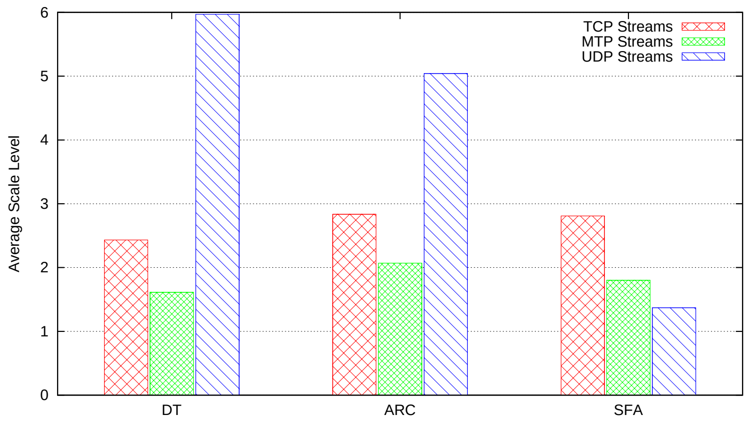


Figure 6.4: Average Media Scale Level

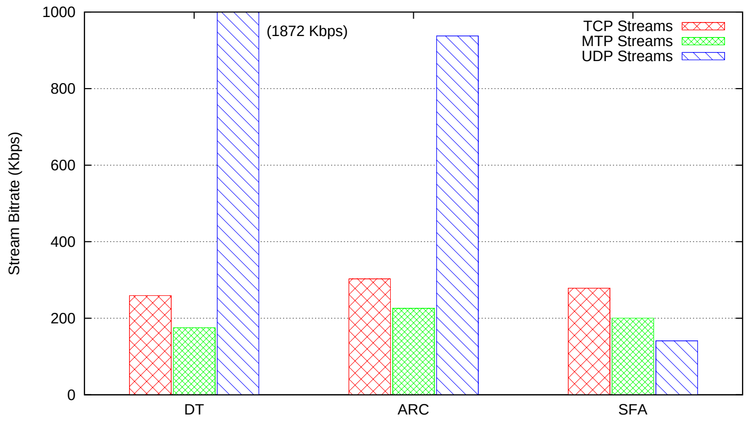


Figure 6.5: Average Streaming Bitrate

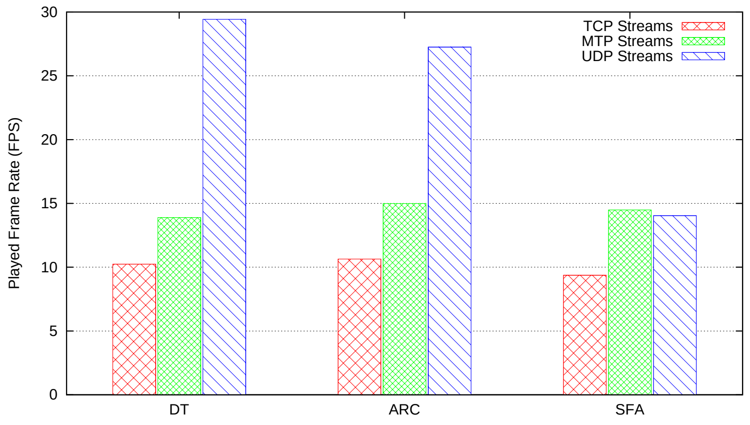


Figure 6.6: Average Played Frame Rate (Frames per Seconds)

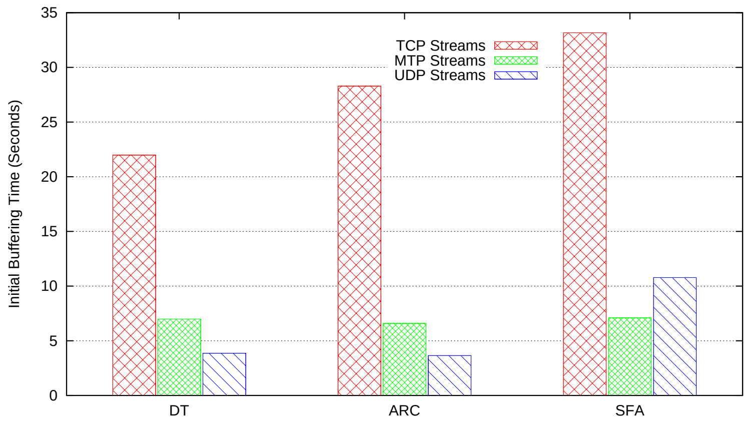


Figure 6.7: Initial Buffering Time

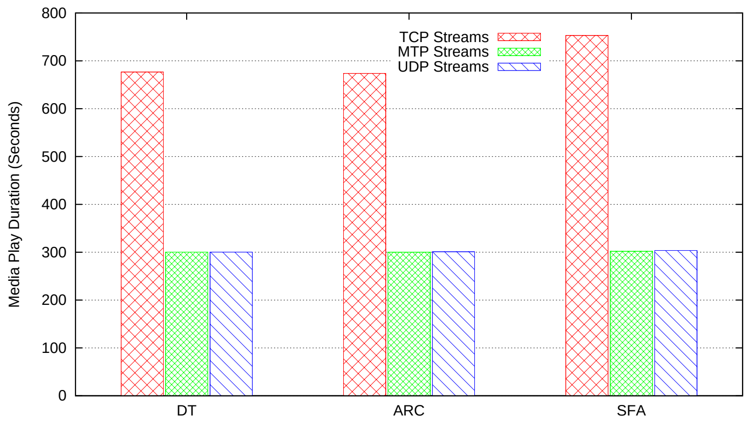


Figure 6.8: Media Playout Duration

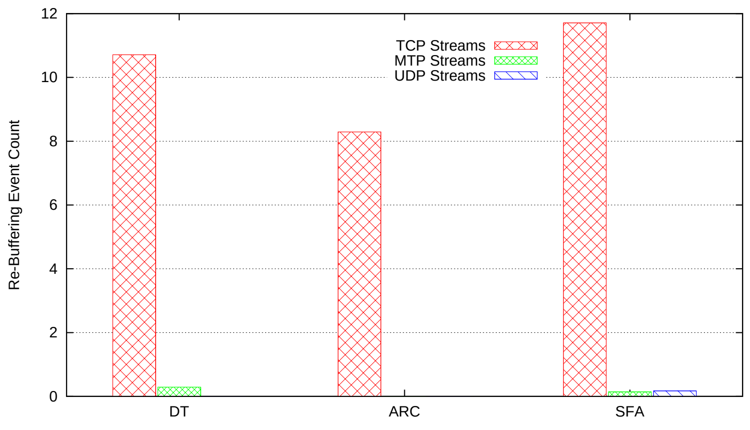


Figure 6.9: Re-Buffering Event Count

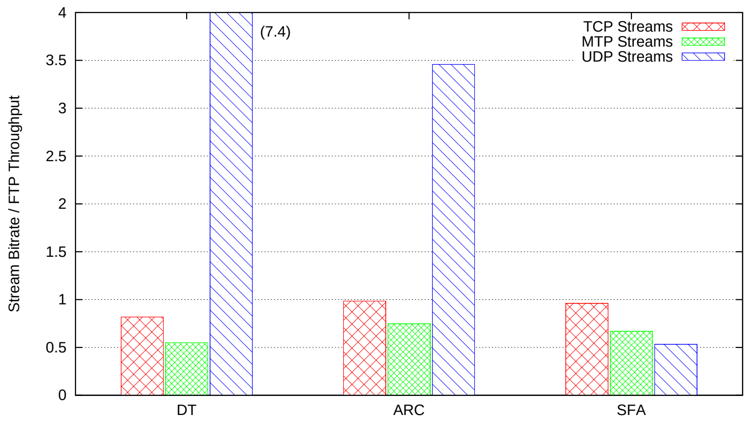


Figure 6.10: Streaming Bitrate / Competing Bulk FTP Throughput

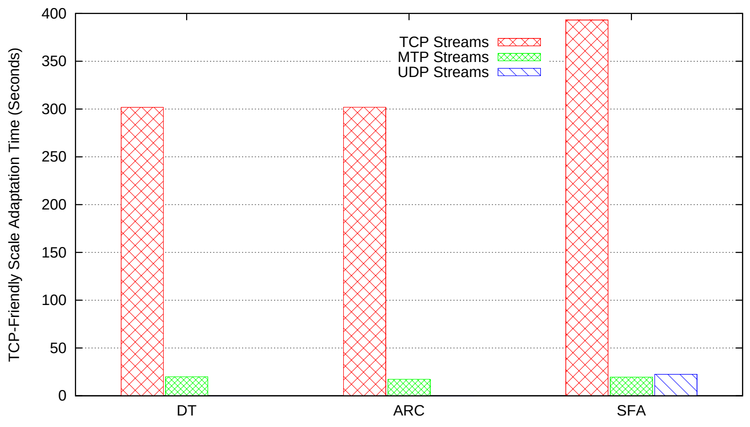


Figure 6.11: TCP-Friendly Rate Adaptation Time

shown in Figure 6.9. More importantly, Figure 6.8 shows that TCP streams in all cases take much longer to finish streaming than the duration of the video, since they stop playing and re-buffer frequently due to the frequently delayed frame deliveries by TCP.

The Goddard clients and servers over MTP are TCP-Friendly in contrast to the UDP streams, and achieve much better media scaling performance and higher frame playout rate than the TCP streams with all three queue management schemes. Figure 6.4 and Figure 6.5 show that Goddard over MTP streams a lower quality media on average and consumes less average bandwidth than the TCP streams, while the UDP streams consume much higher average bandwidth than the TCP streams. Nevertheless, Figure 6.6 through Figure 6.9 show that the MTP streams achieve much better media scaling performance and higher frame playout rate than the TCP streams. The MTP streams have the media frame playout rate close to the encoded frame rate, a short average initial buffering periods close to that of the UDP streams, and very few re-buffering events and little delay in the media playout.

In addition, it is shown in Figure 6.4 and Figure 6.6 that ARC and SFA with ECN improve the quality of MTP streams. Goddard over MTP with ARC and SFA streams a higher average quality media than with drop-tail while maintaining the average frame playout rate close to the encoded frame rate of the level-1 and level-2 media. The streaming performance gain obtained through ECN is slightly decreased with SFA, as SFG incurs some packet drops for the ECN enabled flows while performing traffic policing. However, the negative effect of SFG on MTP streaming performance is negligible considering the protection SFG offers from the misbehaving UDP streams.

Figure 6.10 quantifies the TCP-Friendliness of the TCP, UDP and MTP streams by comparing the average bitrate of the media streams to the throughput of the competing bulk FTP flow, and Figure 6.11 visualizes how quickly the streams adapt to a media scale level that has a TCP-Friendly streaming bitrate. Figure 6.10 shows that the average UDP stream consumes 740% and 350% of the competing bulk FTP throughput under drop-tail and ARC respectively. Yet, under SFA, the UDP streams are forced to consume only 60% of the competing bulk FTP throughput. The average TCP stream bitrate is less than, but close to, the throughput of the bulk FTP flow, since the TCP sender buffer is filled with media frames most of the time as Goddard overestimates the available TCP bandwidth. The average MTP

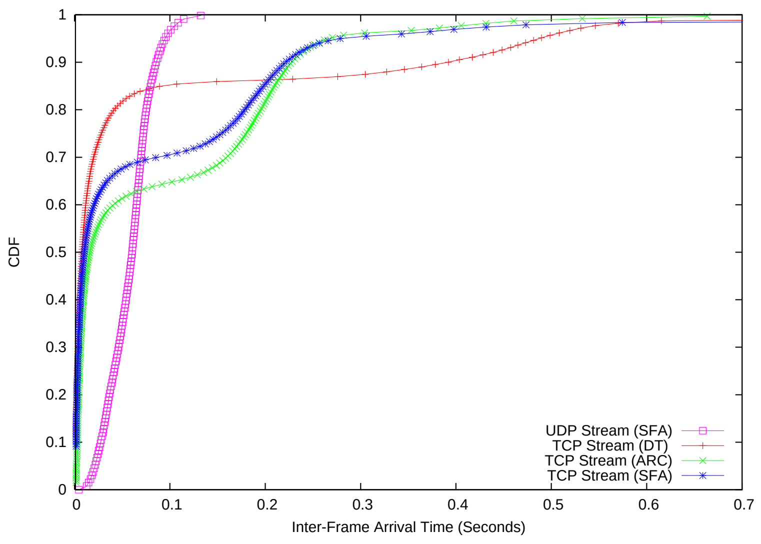


Figure 6.12: Inter-Frame Arrival Times of TCP Streams

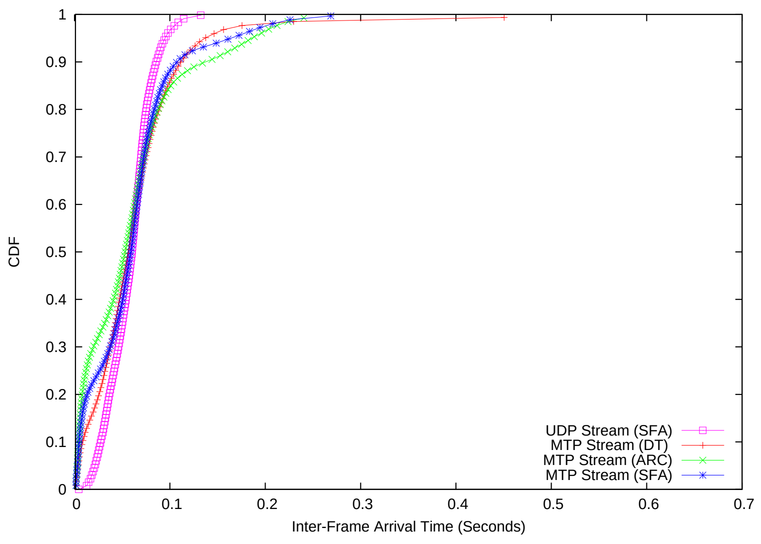


Figure 6.13: Inter-Frame Arrival Times of MTP Streams

stream consumes about 60% to 70% of the bulk FTP throughput in all cases. In addition, Figure 6.11 shows that the MTP streams quickly adapt to a TCP-Friendly media scale level all the time, while TCP stream adaptation is much delayed at the TCP sender queues. Lastly, Figure 6.11 illustrates that SFG can quickly regulate the bitrate of misbehaving UDP streams under the target bitrate.

Next, we analyze the effect of uniform random ECN marking on media frame reception jitter for the TCP and MTP streams. Figure 6.12 and Figure 6.13 show the cumulative distribution function (CDF) of TCP and MTP inter-frame arrive times under drop-tail, ARC and SFA, while showing the inter-arrival times of UDP streams under SFA as a desirable target.

Figure 6.12 shows that ECN improves the tail of the TCP frame jitter distribution, since ECN marking significantly reduces transmission delays for the media frames that are retransmitted under drop-tail queue management. However, Figure 6.12 also shows that the midrange frame reception jitter over TCP increases with ARC and SFA compared to drop-tail. This is due to the artificial side effect of measuring jitter in terms of inter-frame arrival time. Under drop-tail queue management, packets that are transmitted after a lost packet are delayed in the TCP receiver buffer for an in-order delivery. When the lost packet is successfully retransmitted, all the packets in the TCP receiver buffer are delivered to the application at once. This gives artificial inter-frame arrival time of zero, in case the delivered packets constitute multiple media frames. Use of ECN avoids both the retransmission delays at the TCP sender and the in-order data delivery delays at the TCP receiver, and makes the high inter-frame arrival times and the low inter-frame arrival times merge into middle values.

The TCP streams have 90% of their inter-frame arrival times less than 230 ms under ARC and SFA, whereas 90% of the inter-frame arrival times under drop-tail are less than 400 ms. Thus, ECN dramatically improves jitter for the TCP streams by reducing TCP retransmissions. Yet, compared to the UDP frame jitter under SFA, the TCP streams' inter-frame arrival times under ARC and SFA are not smooth enough to support interactive streaming applications. However, considering that Goddard does not efficiently perform media scaling over TCP and often saturates the TCP input buffer with high quality media frames, the TCP frame jitter shown in Figure 6.12 does not accurately represent the frame reception jitter of a well adapted TCP media stream. Assuming ECN effectively avoids all packet drops, TCP transmissions are identical to those of MTP under ECN. Therefore, jitter of a well-adapted TCP media stream should be identical to that of an equally well-adapted MTP media stream. Indeed, we observed that the frame reception jitter of TCP streams that are forced to stream level-2 media are very similar to the MTP jitter under ARC and SFA.

Figure 6.13 shows that the MTP streams under drop-tail, ARC and SFA have smooth inter-frame arrival times close to those of the UDP streams. The small extra MTP frame jitter at the high- and low-end of the CDF illustrates the window-based transmission timings of MTP have an insignificant effect on frame jitter compared to TCP retransmissions. In addition, Figure 6.13 shows that ECN shortens the tail of the frame jitter by reducing the MTP transmission timeouts. However, ECN does little to improve the overall MTP frame reception jitter over drop-tail, since MTP does not retransmit lost packets. It is shown in Figure 6.13 that the MTP frame jitter under ARC and SFA are even slightly increased compared to that under drop-tail. Yet, this is not because ECN has a negative effect on MTP frame jitter, but Goddard under ARC and SFA streamed higher quality media than under drop-tail on average, as shown in Figure 6.4.

The MTP streams have 90% of their inter-frame arrival times less than 140 ms under all queue management, where about 50 ms to 67 ms is due to the encoded frame interval of the level-1 and level-2 Goddard streams. This means the Goddard clients having 80 ms (140 - 60 ms) of a playout buffer at the client can render about 90% of the frames received, even under fairly congested network conditions. Knowing that the level-1 and level-2 Goddard streams simulate Internet videoconference quality

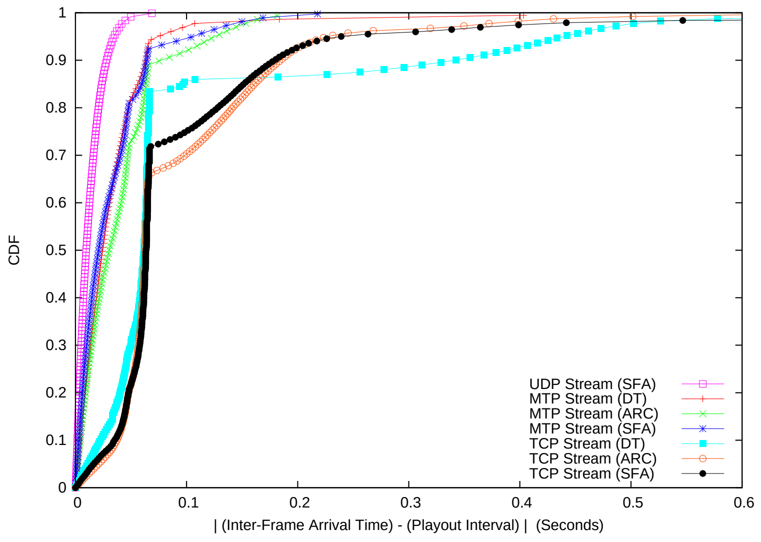


Figure 6.14: Absolute Difference between Inter-Frame Arrival Time and Playout Interval of TCP and MTP Streams

streams, our results demonstrate the potential of MTP as an effective interactive video streaming transport protocol as well as a non-interactive streaming protocol.

Summarizing the media frame reception jitter analysis, Figure 6.14 compares TCP and MTP stream jitter in terms of absolute difference between inter-frame arrival time and the corresponding playout interval of the streams to eliminate the artificial inter-frame arrival time of zero for TCP streams. Figure 6.14 confirms that that MTP streams have a low jitter close to those of the UDP streams, while TCP streams experience significantly degraded jitter for all three queue management schemes.

Figure 6.15 and Figure 6.16 summarize queue length seen by each packet arrival and utilization of the 10 Mbps backbone link respectively for all simulations. Figure 6.15 shows that ARC and SFA dramatically reduce the queuing delay. Under ARC and SFC, 50% of the incoming packets wait less than 36 ms (45.32 Kbps / 10 Mbps) and 90% of the incoming packets wait less than 100 ms (123 Kbytes / 10 Mbps) in the queue, whereas 50% of incoming packets entering the drop-tail queue

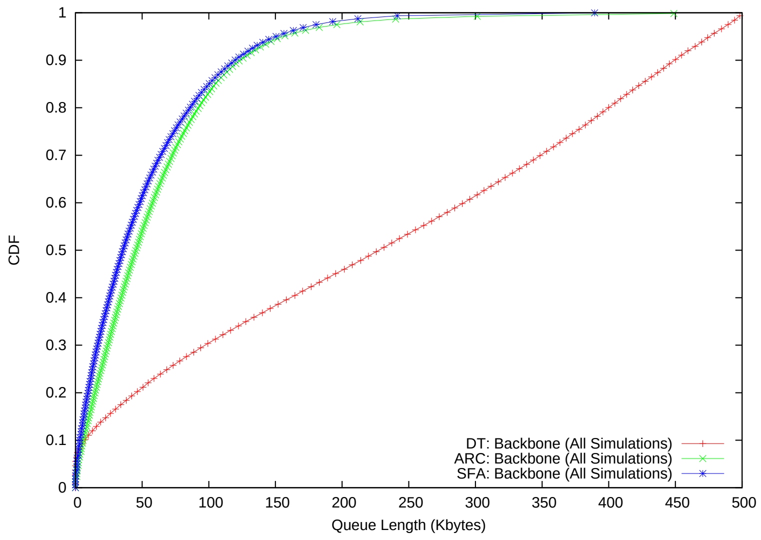


Figure 6.15: Backbone Queue Length (CDF)

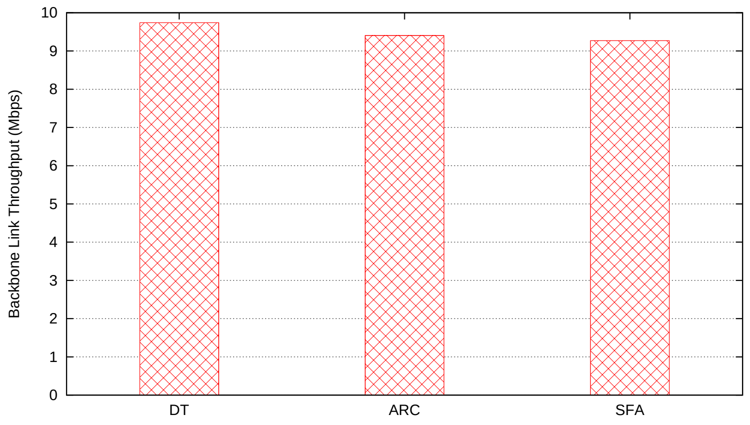


Figure 6.16: Backbone Link Utilization

wait more than 182 ms (228 Kbytes / 10 Mbps). Figure 6.16 shows that 9.4 Mbps and 9.3 Mbps of the 10 Mbps backbone links are utilized under ARC and SFC respectively, while the drop-tail queue utilizes 9.7 Mbps. Thus, our results confirm that ARC guarantees a low queuing delay under persistent congestion while maintaining high link utilization. In addition, SFA that uses SFG in combination with ARC can prevent misbehaving traffic from dominating the congested link while little affecting the queue dynamics or the link utilization.

	Network Packet Transmission Delay	App Frame (UDP) Transmission Delay
DT	275 ms	329 ms
ARC	107 ms	111 ms
SFA	101 ms	101 ms

Table 6.2: Average Transmission Delays (link delay of the path = 70 ms)

Lastly, Table 6.2 depicts the impact of ARC and SFA on end-to-end transmission delays measured at the network and application layers for the streaming traffic (path: $t_1 \rightarrow e_1$ in Figure 6.1). For the application layer statistics, we only show the average UDP frame transmission delays, since queuing delays at an IP router little affect the end-to-end transmission delays of our MTP streams that simulate non-interactive Internet video streams with a relatively large MTP input buffer (64 KBytes). Table 6.2 confirms that ARC and SFA enhance the support for interactive streaming by significantly reducing the end-to-end transmission delays. When the link delay of the path is 70 ms, ARC and SFA achieve both the network and application layer end-to-end delays close to 100 ms, that is about 1/3 of the average transmission delays provided by the drop-tail system.

6.3 Summary

This chapter evaluated streaming media over Multimedia Transport Protocol (MTP) with the Crimson network by comparing performance of TCP, UDP and MTP Goddard streams with drop-tail queue management, ARC, and the combination of SFG and ARC (SFA).

Our results show that MTP is TCP-Friendly and provides most of the streaming performances that UDP can offer to a well-behaving streaming application regardless of the queue managements used in the network. MTP provides an application programming interface (API) that allows streaming applications to efficiently perform media scaling. In addition, MTP offers smooth frame playout jitter comparable to that of UDP streams by removing extra delays caused by TCP retransmissions under drop-tail queue management. The results demonstrate that MTP with smooth frame jitter can even be considered as transport protocol for interactive streaming applications such as videoconferencing under the current Internet or under the Crimson network.

Performing media scaling over TCP is difficult. Nevertheless, under a network that uses ARC, streaming media that adapts to network conditions over ECN-enabled TCP can have as low a frame playout jitter as over MTP, since ECN avoids retransmissions. In addition, ARC guarantees a low queuing delay and high link utilization, thus benefitting delay sensitive applications over drop-tail queue with little degradation of the performance of throughput sensitive applications.

Lastly, our simulation results show that a complete Crimson network that uses SFG along with ARC protects public networks from the threat of unresponsive or misbehaving UDP traffic with little affect on the performance of traffic that responds to congestion or on overall system performance.

Chapter 7

Conclusions and Future Work

The dissertation provides an IP network solution, called *Crimson*, that can be seamlessly deployed in the current Internet to: 1) protect well-behaving traffic from misbehaving high-bandwidth flows; and 2) minimize network delays to support quality of service (QoS) requirements of delay sensitive multimedia applications such as streaming media and network games. Furthermore, to enhance Internet support for media streaming, this dissertation proposes and evaluates a TCP-Friendly and streaming-friendly transport protocol called the *Multimedia Transport Protocol* (MTP). Section 7.1 summarizes this dissertation research and Section 7.2 lists possible future work.

7.1 Summary

At the core of the Crimson network are *Aggregate Rate Controller* (ARC) and *Stochastic Fairness Guardian* (SFG). These are independent IP router traffic/queue management mechanisms that can be deployed in the current Internet to minimize network delays and protect well-behaved traffic from misbehaved traffic.

ARC is a Congestion Controller designed to minimize queuing delay through efficient congestion management. ARC detects network congestion and computes the congestion notification probability (CNP) before the queue grows and overflows by monitoring the CNP and link utilization. ARC takes a rate-based congestion monitoring approach that significantly reduces congestion estimation noise over the typical queue sampling based approach. ARC notifies congestion responsive traffic sources of the CNP using either explicit congestion notification (ECN) to improve congestion signaling efficiency for ECN-enabled traffic sources or implicit packet-dropping congestion notification for traffic sources that do not support ECN. ARC is a Proportional Integral (PI) controller with reduced parameters to provide easy configuration in a time-delayed system (i.e., the Internet). Unlike other PI-based Congestion Controllers such as the PI controller [58], AVQ [75] and REM [5], ARC provides complete configuration guidelines that are essential for practical deployment.

SFG is a Bandwidth Guardian that uses a simple traffic filter to statistically regulate the bitrates of misbehaving high-bandwidth flows without requiring per-flow information. When the outbound link is congested, SFG enables a multi-level hash filter to regulate the bitrate of high-bandwidth flows until congestion abates. Unlike other statistical flow management mechanisms such as SFB [36] and RED-PD [85], SFG does not require identifying misbehaving flows, since such identification is computationally intensive and error prone, and thus may not be affordable for broadband network routers. Yet, SFG offers filtering performance comparable to that provided by more complicated, per-flow mechanisms such as RED-PD [85]. In addition, as in ARC, SFG provides practical configuration guidelines.

Through a simulation study using NS [127], this dissertation demonstrates that by complying with the configuration guidelines, ARC can efficiently support a wide-range of traffic conditions, dampening queue oscillations, keeping queue sizes low and throughput high, even when the configuration is not optimized for the current traffic. ARC can provide significantly improved performance over the PI controller for lightly loaded conditions, the norm for many Internet routers, as well as for sudden traffic load changes, owing to ARC's low frequency rate-based control data acquisition. In addition, this dissertation shows that considering overall performance and design complexity, SFG integrated with ARC (SFA) outperforms other preferential dropping mechanisms, such as SFB [36], RED-PD [85] and CHOKe [91], as well as drop-tail and ARC over a practical range of traffic loads. Thus, by providing a best-delay effort service with misbehaving flow management Crimson can be gradually deployed in the Internet to effectively support various Internet application domains with diverse QoS requirements.

In addition to Crimson, this dissertation provides the design and evaluation of the Multimedia Transport Protocol (MTP) to enhance end-host support for multimedia streaming. In order to identify properties of a streaming-friendly transport protocol, a measurement study was conducted to evaluate the network-level and application-level performance of UDP and TCP RealVideo streams. Based on the study, the shortcomings of using TCP as a streaming transport protocol were: 1) TCP's application programming interface (API) hides network information, such as packet loss rate and round-trip delay, needed for efficient available bandwidth estimation for media scaling; 2) The penalty for overestimating available bandwidth using TCP is severe due to the large TCP sender buffer, put in place to maximize throughput; 3) TCP's reliable in-order packet delivery is harmful for streaming media under congestion, since retransmission not only delays delivery of a lost packet but delays transmission of all the following packets.

Based on the findings, this dissertation proposes MTP, an alternate to UDP (and TCP) for streaming and other delay sensitive Internet applications that favor prompt and timely datagram delivery service over the reliable transmission service of TCP. By removing retransmissions from TCP, MTP reveals network information essential for media scaling and removes delays due to retransmission at the TCP sender and packet ordered preservation at the TCP receiver. MTP has the exact congestion avoidance mechanism and proven stability of TCP, and can be implemented as a mode for TCP in the deployment phase to take advantage of firewall support for TCP traffic.

MTP supports non-blocking transmission mode at the API using drop-front input queue management as well as block-on-full-queue transmission mode to help existing UDP-based streaming applications to switch to MTP with little modification. This dissertation evaluates the non-blocking transmission mode of MTP. Evaluation of MTP requires a realistic streaming application that performs media scaling and simulates media buffering and playout. We designed and implemented a video streaming system, called *Goddard*, in NS based on streaming application behavior observed in our streaming measurement study and [93], and evaluated MTP built based on the TCP Reno implementation in NS.

Our simulations show that MTP is a TCP-Friendly and streaming-friendly transport protocol. Video streams using MTP can quickly and effectively perform media scaling and adapt to the available TCP-Friendly bitrate offered by MTP, avoiding media re-buffering events that are common with video over TCP. Additionally, our simulation results show that MTP dramatically reduces the media frame reception/playout jitter that appear with TCP's reliable in-order packet delivery mechanism, and illustrates the potential of MTP as a streaming transport protocol for interactive applications as well as non-interactive applications.

Finally, this dissertation evaluates streaming media over MTP with the Crimson network by comparing performance of TCP, UDP and MTP Goddard streams with drop-tail queue management, ARC, and the combination of SFG and ARC. Our results confirm that MTP helps protecting the public network from potentially misbehaving UDP streams by offering them an alternative TCP-Friendly and streaming-friendly transport service. It is also shown that a complete Crimson network with SFG secures the network from the threat of high-bandwidth misbehaving UDP traffic with little affect on the performance of traffic that responds to congestion or on overall system performance. In addition, Crimson's best-delay-effort service guarantees a low queuing delay and high link utilization, benefitting delay sensitive applications over drop-tail queue with little degradation of the performance of throughput sensitive applications. Our simulations demonstrate that that MTP with Crimson can be used as transport protocol for interactive streaming applications such as video-conferencing.

7.2 Future Work

This section lists future work that can be extended from this dissertation research. Future work that applies both to the Crimson and MTP researches is to implement the Crimson active queue management mechanism and MTP under Linux, and evaluate them using a realistic network traffic. In particular, an overhead analysis including processing time and memory use is required for both SFG and ARC. Additionally, it is desirable to evaluate performance of commercial streaming applications over MTP. Sections 7.2.1 to 7.2.4 list additional future work for each of the four sub-researches that constitute this dissertation.

7.2.1 Extension to Aggregate Rate Controller

Future work includes extending ARC to dynamically adapt the controller parameters to the current traffic conditions. Although ARC provides a robust congestion control over a wide-range of network conditions, it would be desirable to fine-tune the parameters to best serve the evolving traffic load and mix. One possible design may be to slowly change the value of the control parameter α based on the displacement of the average traffic rate from the target capacity within the initially configured α range. While this approach may sound promising, it introduces another controller (a low pass filter) into the system. Therefore, any dynamic control parameter adaptation would require additional stability and implementation analysis.

Another area of future work includes extensive investigation of incremental deployment issues. In particular, analysis of a congested ARC router in conjunction with other congested drop-tail routers and vice versa for multiple heterogeneous bottleneck network is required before adopting ARC into the current Internet architecture.

In addition to TCP support, ARC can be easily extended to concurrently support FAST [99] traffic as shown in Section 4.1.4. Stability analysis and evaluation of the FAST-extended ARC system is left as future work.

7.2.2 Extension to Stochastic Fairness Guardian

Currently, SFG uses a simple on/off traffic filtering approach that enables the multi-level hash filter that has the static number of levels (L) and bins per level (N) only when the outbound link is congested. A more sophisticated approach is to dynamically adjust N every control/measurement epoch using a TCP-friendly rate estimator similar to that used in RED-PD [85]. This dynamic configuration approach is elegant but has increased complexity because the SFG hash functions will have to be adjusted frequently as N changes. Evaluation of such a dynamic bin adjustment is left as future work.

7.2.3 Streaming Characterization

The streaming characterization work in Section 5.1 is only another step in the analysis of streaming multimedia traffic on the Internet, leaving many areas for future work. In our study, pre-recorded video clips are intentionally selected to help ensure consistency in the videos played out during each set of experiments. Live content, captured and served directly from a video camera or television, typically has different characteristics than does pre-recorded content [126]. Future work could be to measure the performance of live RealVideo content on the Internet and compare it to that of the pre-recorded RealVideo content in our study.

In addition, this work did not explore the relationship between perceptual quality of the video, influenced by application level performance such as frame rate and jitter, and network metrics. A better understanding of the impact on perceptual quality on video streaming over UDP versus TCP might further aid development of more effective ways to use a TCP-Friendly share of the available bitrate. In particular, future work includes investigating how to use the available TCP-Friendly bandwidth for media scaling and forward error correction (FEC) to maximize perceptual quality of the streamed media.

7.2.4 Extension to MTP

Currently, MTP offers an API that transparently provides underlying network information such as packet losses or round-trip time to applications above. Future work includes enhancing the MTP API to explicitly provide useful network information, such as effective MTP transmission rate, that can be utilized to improve media scaling and repair performance. The exact network information, computation and format that should be provided to be useful for media scaling requires further study.

Another area of future work includes evaluating live and interactive streaming over MTP, since live and interactive streams may have different media scaling characteristics than archived streams due to limited computing resources and delay constraints. Related future work includes evaluating MTP with dynamic queue length adaptation in [48] for delay critical interactive streaming.

Another future work is to implement and evaluate MTP under Linux to validate the performance of MTP as a streaming transport protocol. In particular, evaluating MTP under Linux shall reveal some of the MTP API issues that are hard to identify and evaluate using NS. Additionally, the publicly available MTP in Linux will help the network community to further extend streaming transport protocol studies and/or to conduct other streaming related researches.

7.2.5 Evaluation of Goddard

Although Goddard streaming application built into NS models most of the essential streaming features of commercial streaming applications, Goddard requires further evaluation before it can be widely used by the NS research community. Such an evaluation should compare behaviors of Goddard streams with those of commercial streams under a controlled environment to see how realistically Goddard stream react to network conditions.

Bibliography

[1] Akella, A., Seshan, S., and Shaikh, A., "An Empirical Evaluation of Wide-Area Internet Bottlenecks," in *Proceedings of the ACM Internet Measurement Conference (IMC)*, (Miami, FL, USA), October 2003. 148

[2] Allman, M., Paxson, V., and Stevens, W., "Congestion Control." RFC-2581, April 1999. 55, 56

[3] Appenzeller, G., Keslassy, I., and McKeown, N., "Sizing Router Buffers," in *Proceedings of ACM SIGCOMM*, (Portland, OR, USA), September 2004. 124

[4] Arlitt, M. and Jin, T., "Workload Characterization of the 1998 World Cup Web Site," Tech. Rep. HPL-1999-35R1, Hewlett-Packard Laboratories, October 1999. 65, 92, 99, 121, 194

[5] Athuraliya, S., Li, V. H., Low, S. H., and Yin, Q., "REM: Active Queue Management," *IEEE Network*, vol. 15, pp. 48–53, May/June 2001. 2, 27, 30, 106, 107, 229

[6] Berners-Lee, T., Fielding, R., and Frystyk, H., "Hypertext Transfer Protocol – HTTP/1.0." RFC-1945, May 1996. 64

[7] Bernet, Y., Grossman, S. B. D., and Smith, A., "An Informal Management Model for Diffserv Routers." RFC-3290, May 2002. 40

[8] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W., "An Architecture for Differentiated Services." RFC-2475, December 1998. 4, 34, 39, 64, 69

[9] Bocheck, P., Campbell, A. T., Chang, S.-F., and Liao, R. R.-F., "Utility-based Network Adaptation for MPEG4 Systems," in *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Basking Ridge, NJ, USA), June 1993. 66, 144

[10] Bolot, J. C., "Characterizing End-to-End Packet Delay and Loss in the Internet," *Journal of High Speed Networks*, vol. 2, pp. 289–298, September 1993. 14, 190

[11] Bolot, J.-C., Fosse-Parisis, S., and Towsley, D., "Adaptive FEC-Based Error Control for Internet Telephony," in *Proceedings of IEEE INFOCOM*, March 1999. 6, 66, 140, 172

[12] Bonald, T., May, M., and Bolot, J.-C., "Analytic Evaluation of RED Performance," in *Proceedings of IEEE INFOCOM*, pp. 1415–1424, 2000. 26, 60

[13] Boyce, J. and Gaglianello, R., "Packet Loss Effects on MPEG Video sent over the Public Internet," in *Proceedings of ACM Multimedia*, (Bristol, U.K.), pp. 181–190, September 1998. 168

[14] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and Zhang, L., "Recommendations on Queue Management and Congestion Avoidance in the Internet." RFC-2309, April 1998. 2, 16, 109

[15] Braden, R., "Requirements for Internet Hosts – Communication Layers." RFC-1122, October 1989. 55

[16] Brakmo, L. and Peterson, L., "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communication*, vol. 13, pp. 1465–1480, October 1995. 105

[17] Cao, Z., Wang, Z., and Zegura, E. W., "Rainbow Fair Queueing: Fair Bandwidth Sharing Without Per-Flow State," in *Proceedings of IEEE INFOCOM*, pp. 922–931, 2000. 3, 33

[18] Casetti, C., Gerla, M., Lee, S., Mascolo, S., and Sanadidi, M., "TCP with Faster Recovery," in *Proceedings of MILCOM*, (Los Angeles, CA, USA), October 2000. 62

[19] Chesire, M., Wolman, A., Voelker, G., and Levy, H., "Measurement and Analysis of a Streaming Media Workload," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, (San Francisco, CA, USA), pp. 1–12, March 2001. 140

[20] Choi, B.-Y., Moon, S., Zhang, Z.-L., Papagiannaki, K., and Diot, C., "Analysis of Point-To-Point Packet Delay in an Operational Network," in *Proceedings of IEEE INFOCOM*, (Hong Kong, China), March 2004. 86, 118, 119

[21] Christiansen, M., Jeffay, K., Ott, D., and Smith, F. D., "Tuning RED for Web Traffic," in *Proceedings of ACM SIGCOMM*, (Stockholm, Sweden), pp. 139–150, August-September 2000. 26, 61

[22] Chung, J. and Claypool, M., "Better-Behaved, Better-Performing Multimedia Networking," in *Proceedings of SCS Euromedia*, (Antwerp, Belgium), May 2000. 132

[23] Chung, J. and Claypool, M., "Dynamic-CBT and ChIPS - Router Support for Improved Multimedia Performance on the Internet," in *Proceedings of the ACM Multimedia Conference*, (Los Angeles, CA, USA), November 2000. 3, 32

[24] Chung, J. and Claypool, M., "Rate-Based Active Queue Management with Priority Classes for Better Video Transmission," in *Proceedings of the Seventh IEEE Symposium on Computers and Communications (ISCC)*, (Taormina/Giardini Naxos, Italy), July 2002. 41

[25] Chung, J. and Claypool, M., "Analysis of Active Queue Management," in *Proceedings of The 2nd IEEE International Symposium on Network Computing and Applications (NCA)*, (Cambridge, MA, USA), April 2003. 3, 61, 62, 105

[26] Chung, J., Claypool, M., and Zhu, Y., "Measurement of the Congestion Responsiveness of RealPlayer Streaming Video Over UDP," in *Proceedings of the Packet Video Workshop (PV)*, (Nantes, France), April 2003. 141

[27] Chung, J., Zhu, Y., and Claypool, M., "FairPlayer or FoulPlayer? - Head to Head Performance of RealPlayer Streaming Video Over UDP versus TCP," Tech. Rep. WPI-CS-TR-02-17, CS Department, Worcester Polytechnic Institute, May 2002. 146, 150, 156

[28] Clark, D. and Fang, W., "Explicit Allocation of Best-Effort Service," *IEEE/ACM Transactions on Networking*, vol. 6, August 1998. 41

[29] Claypool, M., LaPoint, D., and Winslow, J., "Network Analysis of Counter-strike and Starcraft," in *Proceedings of the 22nd IEEE International Performance, Computing, and Communications Conference (IPCCC)*, (Phoenix, AZ, USA), April 2003. 67

[30] Conklin, G. J., Greenbaum, G. S., Lillevold, K. O., Lippman, A. F., and Reznik, Y. A., "Video Coding for Streaming Media Delivery on the Internet," *IEEE Transactions on Circuits and Systems*, pp. 269 – 281, March 2001. 143

[31] de Cuetos, P. and Ross, K. W., "Adaptive Rate Control for Streaming Stored Fine-Grained Scalable Video," in *Proceedings of the 12th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Miami, Florida, USA), May 2002. 173

[32] Delgrossi, L., Halstrick, C., Hehmann, D., Herrtwich, R. G., Krone, O., Sandvoss, J., and Vogt, C., "Media Scaling for Audiovisual Communication with the Heidelberg Transport System," in *Proceedings of the First ACM International Conference on Multimedia*, (Anaheim, CA, USA), pp. 99–104, 1993. 66

[33] Demers, A., Keshav, S., and Shenker, S., "Analysis and Simulation of a Fair Queuing Algorithm," in *Proceedings of ACM SIGCOMM*, (Austin, TX, USA), pp. 1–12, September 1989. 3, 32

[34] Diot, C., Iannaccone, G., and May, M., "Aggregate Traffic Performance with Active Queue Management and Drop from Tail," Tech. Rep. TR01-ATL-012501, Sprint Advanced Technology Laboratories, July 2001. 26

[35] Feng, W., Kandlur, D., Saha, D., and Shin, K. G., "Blue: An Alternative Approach to Active Queue Management," in *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, June 2001. 3, 27, 36, 114

[36] Feng, W., Kandlur, D. D., Saha, D., and Shin, K. G., "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness," in *Proceedings of IEEE INFOCOM*, pp. 1520–1529, 2001. 3, 12, 35, 63, 111, 113, 119, 135, 229, 230

[37] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and Berners-Lee, T., "Hypertext Transfer Protocol – HTTP/1.1." RFC-2068, January 1997. 64

[38] Firoiu, V. and Borden, M., "A Study of Active Queue Management for Congestion Control," in *Proceedings of IEEE INFOCOM*, pp. 1435–1444, 2000. 22, 58, 61

[39] Floyd, S. and Henderson, T., "The NewReno Modification to TCP's Fast Recovery Algorithm." RFC-2582, April 1999. 57

[40] Floyd, S., "HighSpeed TCP (Web page)." http://www.icir.org/floyd/hstcp.html. 63

[41] Floyd, S., "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, vol. 24, October 1994. 3, 20

[42] Floyd, S. and Fall, K., "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *Computer Communication Review*, vol. 26, July 1996. 56, 57

[43] Floyd, S. and Fall, K., "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, 1999. 2, 3, 36, 58, 116, 117, 140, 156

[44] Floyd, S., Handley, M., Padhye, J., and Widmer, J., "Equation-Based Congestion Control for Unicast Applications," in *Proceedings of ACM SIGCOMM*, (Stockholm, Sweden), pp. 43–56, August-September 2000. 3, 6, 14, 46, 58, 63, 105, 140, 173, 175

[45] Floyd, S. and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993. 3, 7, 26, 59, 60

[46] Floyd, S. and Jacobson, V., "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, 1995. 3, 31, 41

[47] Gao, Y. and Hou, J., "A State Feedback Control Approach to Stabilizing Queues for ECN-Enabled TCP Connections," in *Proceedings of IEEE INFOCOM*, (San Francisco, CA, USA), April 2003. 3, 8, 12, 26, 71, 72, 74, 91, 92, 97

[48] Goel, A., Krasic, C., Li, K., and Walpole, J., "Supporting Low Latency TCP-Based Media Streams," in *Proceedings of International Workshop on Quality of Service (IWQoS)*, (Miami Beach, FL, USA), May 2002. 47, 164, 167, 174, 175, 188, 204, 235

[49] Guo, L. and Matta, I., "The War Between Mice and Elephants," in *Proceedings of the 9th International Conference on Network Protocols (ICNP)*, (Riverside, CA, USA), November 2001. 65

[50] Hardman, V., Sasse, M. A., Handley, M., and Watson, A., "Reliable Audio for Use over the Internet," in *Proceedings of Internet Society's International Networking Conference (INET)*, (Ohahu, Hawaii, USA), 1995. 66

[51] Hashem, E., "Analysis of Random Drop for Gateway Congestion Control," Tech. Rep. LCS TR-465, Laboratory for Computer Science, MIT, Cambridge, MA, 1989. 59

[52] Heinanen, J., Baker, F., Weiss, W., and Wroclawski, J., "Assured Forwarding PHB Group." RFC-2597, June 1999. 40

[53] Hemy, M., Hengartner, U., Steenkiste, P., and Gross, T., "MPEG System Streams in Best-Effort Networks," in *Proceedings of the Packet Video Workshop (PV)*, (New York, NY, USA), April 1999. 66

[54] Henderson, T., Sahouria, E., McCanne, S., and Katz, R., "On Improving the Fairness of TCP Congestion Avoidance," in *Proceedings of IEEE Globecom*, (Sydney, Australia), pp. 539–544, 1998. 62

[55] Hernandez-Campos, F., Jeffay, K., and Smith, F., "Tracing the Evolution of the Web Traffic: 1995-2003," in *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, (Orlando, FL, USA), October 2003. 92, 121, 194

[56] Heying, Z., Baohong, L., and Wenhua, D., "Design of a Robust Active Queue Management Algorithm Based on Feedback Compensation," in *Proceedings of ACM SIGCOMM*, (Karlsruhe, Germany), August 2003. 3

[57] Hollot, C. V., Misra, V., Towsley, D., and Gong, W.-B., "A Control Theoretic Analysis of RED," in *Proceedings of IEEE INFOCOM*, pp. 1510–1519, 2001. 75, 76

[58] Hollot, C. V., Misra, V., Towsley, D. F., and Gong, W., "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," in *Proceedings of IEEE INFOCOM*, pp. 1726–1734, 2001. 3, 8, 12, 27, 31, 58, 61, 71, 72, 73, 74, 75, 81, 85, 91, 92, 229

[59] Huffaker, B., Fomenkov, M., Moore, D., and kc claffy, "Macroscopic Analyses of the Infrastructure: Measurement and Visualization of Internet Connectivity and Performance," in *Proceedings of the 2nd Passive and Active Measurement Workshop (PAM)*, (Amsterdam), April 2001. 48

[60] Hurley, P., Boudec, J., Thiran, P., and Kara, M., "ABE: Providing a Low-Delay Service within Best-Effort," *IEEE Network*, vol. 15, no. 3, pp. 60–69, 2001. 4, 38

[61] International Telecommunication Union (ITU), "Transmission Systems and Media, General Recommendation on the Transmission Quality for an Entire International Telephone Connection; One-Way Transmission Time." Recommendation G.114, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, March 1993. 48

[62] Internet2 QoS working group, "Internet2 QoS Working Group." http://www.internet2.edu/qos/wg/. 41

[63] Internet2 QoS working group, "QBone." http://qbone.internet2.edu/. 41

[64] Jacobson, V., Nichols, K., and Poduri, K., "An Expedited Forwarding PHB." RFC-2598, June 1999. 40

[65] Jacobson, V., "Congestion Avoidance and Control," in *Proceedings of ACM SIGCOMM*, (Stanford, CA, USA), August 1988. 2, 14, 51, 52, 53, 56, 190

[66] Jain, R., Ramakrishnan, K., and Chiu, D. M., "Congestion Avoidance in Computer Networks with a Connectionless Network Layer," Tech. Rep. DEC-TR-506, Digital Equipment Corporation, August 1987. 54

[67] Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., and Towsley, D., "Inferring TCP Connection Characteristics Through Passive Measurements," in *Proceedings of IEEE INFOCOM*, (Hong Kong, China), March 2004. 86, 88, 92, 118, 119, 120, 194, 212

[68] Jupiter Media Metrix, "Users of Media Player Applications Increased 33 Percent Since Last Year," April 2001. Press Release. http://www.jup.com/company/pressrelease-.jsp?doc=pr01040. 142

[69] Katabi, D., Handley, M., and Rohrs, C., "Congestion Control for High Bandwidth-Delay Product Networks," in *Proceedings of ACM SIGCOMM*, (Pittsburgh, PA, USA), August 2002. 2, 22, 29, 58, 60, 73, 104

[70] Keshav, S., "A Control-Theoretic Approach to Flow Control," in *Proceedings of the conference on Communications Architecture & Protocols*, pp. 3–15, 1991. 14, 190

[71] Kohler, E., Handley, M., and Floyd, S., "Datagram Congestion Control Protocol (DCCP)." IETF Internet-Draft: draft-ietf-dccp-spec-11, March 2005. 175

[72] Krasic, C. and Walpole, J., "Priority-Progress Streaming for Quality-Adaptive Multimedia," in *Proceedings of the Ninth ACM International Conference on Multimedia*, (Ottawa, Canada), October 2001. 173

[73] Kuang, T. and Williamson, C., "A Measurement Study of RealMedia Audio/Video Streaming Traffic," in *Proceedings of ITCOM*, (Boston, MA, USA), pp. 68–79, July 2002. 140

[74] Kuhmiinch, C., Kiihne, G., Schremmer, C., and Haenselmann, T., "A Video-Scaling Algorithm Based on Human Perception for Spario-Temporal Stimuli," in *Proceedings of IS&T/SPIE/ACM Multimedia Computing and Networking (MMCN)*, (San Jose, California, USA), January 2001. 66

[75] Kunniyur, S. and Srikant, R., "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," in *Proceedings of ACM SIGCOMM*, (San Diego, CA, USA), August 2001. 3, 8, 12, 28, 71, 72, 74, 75, 91, 92, 229

[76] Lakshminarayanan, K. and Padmanabhan, V., "Some Findings on the Network Performance of Broadband Hosts," in *Proceedings of the Internet Measurement Conference (IMC)*, (Miami, FL, USA), October 2003. 147

[77] Le, L., Aikat, J., Jeffay, K., and Smith, F. D., "The Effects of Active Queue Management on Web Performance," in *Proceedings of ACM SIGCOMM*, (Karlsruhe, Germany), August 2003. 126

[78] Le, L., Aikat, J., Jeffay, K., and Smith, F. D., "Differential Congestion Notification: Taming the Elephants," in *Proceedings of the 12th International Conference on Network Protocols (ICNP)*, (Berlin, Germany), October 2004. 31

[79] Lee, C.-S., Claypool, M., and Kinicki, R., "Chablis - Achieving Fair Bandwidth Allocation with Priority Dropping Based on Round Trip Time," Tech. Rep. WPI-CS-TR-02-19, Computer Science Department, WPI, Worcester, MA, May 2002. 63

[80] Li, M., Claypool, M., and Kinicki, R., "MediaPlayer versus RealPlayer – A Comparison of Network Turbulence," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW)*, (Marseille, France), November 2002. 11, 44, 140, 157

[81] Lin, D. and Morris, R., "Dynamics of Random Early Detection," in *Proceedings of ACM SIGCOMM*, (Cannes, France), pp. 127–137, September 1997. 3, 33

[82] Liu, Y. and Claypool, M., "Using Redundancy to Repair Video Damaged by Network Data Loss," in *Proceedings of IS&T/SPIE/ACM Multimedia Computing and Networking (MMCN)*, (San Jose, California, USA), January 2000. 6, 66, 140, 172

[83] Loguinov, D. and Radha, H., "Measurement Study of Low-bitrate Internet Video Streaming," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW)*, (San Francisco CA, USA), November 2001. 42

[84] Low, S. H., Paganini, F., Wang, J., Adlakha, S., and Doyle, J. C., "Dynamics of TCP/AQM and a Scalable Control," in *Proceedings of IEEE INFOCOM*, 2002. 60, 63

[85] Mahajan, R., Floyd, S., and Wetherall, D., "Controlling High-Bandwidth Flows at the Congested Router," in *Proceedings of the 9th International Conference on Network Protocols (ICNP)*, (Riverside, CA, USA), November 2001. 3, 12, 35, 37, 109, 111, 119, 135, 229, 230, 233

[86] Mathis, M., Mahdavi, J., Floyd, S., and Romanow, A., "TCP Selective Acknowledgment Options." RFC-2018, October 1996. 57, 186, 189

[87] McCreary, D., Li, K., Watterson, S. A., and Lowenthal, D. K., "TCP-RC: A Receiver-Centered TCP Protocol for Delay-Sensitive Applications," in *Proceedings of the 12th SPIE Multimedia Computing and Networking Conference (MMCN)*, (San Jose, CA, USA), January 2005. 48

[88] McCreary, S. and Claffy, K., "Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange," in *13th ITC Specialist Seminar*, pp. 1–11, September 2000. xi, 42, 67, 68, 118

[89] McKenny, P., "Stochastic Fairness Queueing," in *Proceedings of IEEE INFOCOM*, (San Francisco, CA, USA), June 1990. 32

[90] Mena, A. and Heidemann, J., "An Emprical Study of Real Audio Traffic," in *Proceedings of IEEE INFOCOM*, (Tel-Aviv, Israel), pp. 101–110, March 2000. 43, 140

[91] Mitra, D., Stanley, K., Pan, R., Prabhakar, B., and Psounis, K., "CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," in *Proceedings of IEEE INFOCOM*, (Tel-Aviv, Israel), March 2000. 3, 12, 35, 119, 135, 230

[92] Nichols, J., "Measurement of Windows Streaming Media," Master's thesis, Worcester Polytechnic Institute, February 2004. Advisor: Mark Claypool and Robert Kinicki. 198

[93] Nichols, J., Claypool, M., Kinicki, R., and Li, M., "Measurements of the Congestion Responsiveness of Windows Streaming Media," in *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2004. 11, 172, 176, 190, 191, 193, 231

[94] Ogata, K., *Modern Control Engineering, Fourth Edition*. Upper Saddle River, New Jersey, USA: Prentice Hall, 2002. 79

[95] Padhye, C., Christensen, K., and Moreno, W., "A New Adaptive FEC Loss Control Algorithm for Voice Over IP Applications," in *Proceedings of IEEE International Performance, Computing and Communication Conference*, Feburary 2000. 6, 66, 140, 172

[96] Padhye, J., Firoiu, V., Towsley, D., and Krusoe, J., "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proceedings of ACM SIGCOMM*, (Vancouver, Canada), pp. 303–314, September 1998. 46, 58, 59

[97] Padhye, J. and Floyd, S., "Identifying the TCP Behavior of Web Servers," Tech. Rep. 01-002, ICSI, 2001. 57

[98] Padhye, J. and Floyd, S., "On Inferring TCP Behavior," in *Proceedings of ACM SIGCOMM*, (San Diego, CA, USA), August 2001. 58

[99] Paganini, F., Wang, Z., Low, S. H., and Doyle, J. C., "A New TCP/AQM for Stable Operation in Fast Networks," in *Proceedings of IEEE INFOCOM*, (San Francisco, CA, USA), April 2003. 2, 29, 63, 73, 104, 105, 108, 233

[100] Palm, W. J., *Modeling, Analysis, and Control of Dynamic Systems, Second Edition*. New York, New York, USA: John Wiley and Sons, Inc., July 1999. 22, 23

[101] Papagiannaki, K., Taft, N., and Diot, C., "Impact of Flow Dynamics on Traffic Engineering Design Principles," in *Proceedings of IEEE INFOCOM*, (Hong Kong, China), March 2004. 35

[102] Park, K. and Wang, W., "QoS-Sensitive Transport of Real-Time MPEG Video Using Adaptive Forward Error Correction," in *Proceedings of IEEE Multimedia Systems*, pp. 426–432, June 1999. 6, 66, 140, 172

[103] Parris, M., Jeffay, K., and Smith, F., "Lightweight Active Router-Queue Management for Multimedia Networking," in *Proceedings of SPIE conference on Multimedia Computing and Networking*, January 1999. 3, 31

[104] Perkins, C., Hodson, O., and Hardman, V., "A Survey of Packet-Loss Recovery Techniques for Streaming Audio," *IEEE Network Magazine*, September/October 1998. 66

[105] Phelan, T., "Datagram Congestion Control Protocol (DCCP) User Guide." IETF Internet-Draft: draft-ietf-dccp-user-guide-02, July 2004. 174

[106] Phirke, V., Claypool, M., and Kinicki, R., "Traffic Sensitive Active Queue Management for Improved Multimedia Streaming," in *Proceedings of the International Workshop on QoS in Multiservice IP Networks (QoS-IP)*, (Milano, Italy), February 2003. 4, 38

[107] Postel, J. (ed.), "Transmission Control Protocol – DARPA Internet Program Protocol Specification." RFC-793, September 1981. 51

[108] Prasad, R., Dovrolis, C., Murray, M., and Claffy, K., "Bandwidth Estimation: Metrics, Measurement Techniques, and Tools," *IEEE Network*, vol. 17, November/December 2003. 192

[109] Ramakrishnan, K., Floyd, S., and Black, D., "The Addition of Explicit Congestion Notification (ECN) to IP." RFC-3168, September 2001. 1, 3, 61, 62, 71

[110] Real Networks Incorporated, "RealNetworks Facts," 2001. URL: http://www.reanetworks.com/gcompany/index.html. 139

[111] Real Networks Incorporated, "RealPlayer 8 User Manual," copyright 2000. URL: http://service.real.com/help/player/plus_manual.8-/rppmanual.htm. 143

[112] Real Networks Incorporated, "RealProducer User's Guide," copyright 2000. URL: http://www.service.real.com/help/library/guides-/producerplus85/producer.htm. 143

[113] Rejaie, R., Handley, M., and Estrin, D., "RAP: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet," in *Proceedings of IEEE INFOCOM*, pp. 1337–1345, March 1999. 3, 6, 46, 140, 173

[114] Rhee, I., Ozdemir, V., and Yi, Y., "TEAR: TCP Emulation at Receivers - Flow Control for Multimedia Streaming," tech. rep., Department of Computer Science, NCSU, Raleigh, NC, April 2000. 3, 6, 47, 63, 140, 173

[115] Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D., and Levy, H. M., "An Analysis of Internet Content Delivery Systems," in *Usenix Operating Systems Design and Implementation (OSDI)*, (Boston, MA, USA), pp. 315 – 327, October 2002. 93, 99, 121, 195

[116] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications." RFC-3550, July 2003. 174

[117] Seelam, N., Sethi, P., and chi Feng, W., "A Hysteresis Based Approach for Quality, Frame Rate, and Buffer Management for Video Streaming Using TCP," in *Proceedings of 4th IFIP/IEEE International Conference on Management of Multimedia Networks and Services*, (Chicago, IL, USA), October 2001. 173

[118] Silva, G., Datta, A., and Bhattacharyya, S. P., "PI Stabilization of First-Order Systems with Time Delay," *Automatica*, December 2001. 8, 71, 74, 81

[119] Stoica, I., Shenker, S., and Zhang, H., "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," in *Proceedings of ACM SIGCOMM*, (Stanford, CA, USA), pp. 118–130, August 1998. 3, 33

[120] Tanenbaum, A. S., *Computer Networks, Third Edition*. Upper Saddle River, New Jersey, USA: Prentice Hall PTR, 1996. 49, 51

[121] Teitelbaum, B., "Internet2 QoS Testbed: the QBone." Quality of Service Breakout Sessions, Monday September 28 Internet2 Project Meeting 1998 San Francisco, CA, http://www.internet2.edu/qos/wg/calendar/Sep98GeneralMtg/breakouts.html. v, 40

[122] Thompson, K., Miller, G., and Wilder, R., "Wide-Area Internet Traffic Patterns and Characteristics," *IEEE Network*, vol. 11, pp. 10–23, November/December 2000. 58

[123] Tripathi, A. and Claypool, M., "Improving Multimedia Streaming with Content-Aware Video Scaling," in *Workshop on Intelligent Multimedia Computing and Networking (IMMCN)*, March 2002. 144

[124] van der Merwe, J., Caceres, R., hua Chu, Y., and Sreenan, C., "mmdump - A Tool for Monitoring Internet Multimedia Traffic," *ACM Computer Communication Review*, vol. 30, pp. 48–59, October 2000. 140

[125] van der Merwe, J., Sen, S., and Kalmanek, C., "Streaming Video Traffic: Characterization and Network Impact," in *Proceedings of the 7th International Workshop on Web Content Caching and Distribution*, (Boulder CO, USA), August 2002. 43, 172

[126] Veloso, E., Almeida, V., Meira, W., Bestavros, A., and Jin, S., "A Hierarchical Characterization of a Live Streaming Media Workload," in *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, (Marseille, France), pp. 117 – 130, November 2002. 234

[127] VINT, "Virtual InterNetwork Testbed, A Collaboration among USC/ISI, Xerox PARC, LBNL, and UCB." http://www.isi.edu/nsnam/vint/. 11, 57, 83, 175, 229

[128] Walpole, J., Koster, R., Cen, S., Cowan, C., Maier, D., McNamee, D., Pu, C., Steere, D., and Yu, L., "A Player for Adaptive MPEG Video Streaming Over The Internet," in *Proceedings of the SPIE Applied Imagery Pattern Recognition Workshop*, October 1997. 144

[129] Wang, B., Kurose, J., Shenoy, P., and Towsley, D., "Streaming via TCP: An Analytic Performance Study," in *Proceedings of ACM Multimedia*, (New York, NY, USA), October 2004. 173

[130] Wang, Y., Claypool, M., and Zuo, Z., "An Empirical Study of RealVideo Performance Across the Internet," in *Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW)*, (San Francisco, California, USA), pp. 295 – 309, November 2001. 6, 11, 44, 140, 143, 148

[131] Yang, Y. R. and Lam, S. S., "General AIMD Congestion Control," in *Proceedings of the 8th International Conference on Network Protocols (ICNP)*, (Osaka, Japan), November 2000. 3, 6, 45, 63, 140, 173

[132] Zhang, Y., Breslau, L., Paxon, V., and Shenker, S., "On the Characteristics and Origins of Internet Flow Rates," in *Proceedings of ACM SIGCOMM*, (Pittsburgh, PA, USA), August 2002. 31