RICE UNIVERSITY

Edge-Based Inference, Control, and DoS Resilience for the Internet

by

Aleksandar Kuzmanovic

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:

Dr. Edward W. Knightly, Chair
Associate Professor
Electrical and Computer Engineering

Dr. Peter Druschel
Professor
Computer Science

Dr. Rudolf H. Riedi
Assistant Professor
Statistics

Houston, Texas

August, 2004

Edge-Based Inference, Control, and DoS Resilience for the Internet

Aleksandar Kuzmanovic

Abstract

Realizing new services on the Internet ultimately requires edge-based solutions for both deployability and scalability. Each such solution has two fundamental aspects. The first is the ability to accurately infer critical network parameters and processes such as Quality of Service (QoS) mechanisms, end-to-end available bandwidth, or the existence of a Denial of Service (DoS) activity; and the second is the ability to effectively utilize this knowledge to build endpoint services. This thesis presents the design, implementation, and evaluation of a series of edge-based algorithms and protocols for efficient inference, control, and DoS resilience of the Internet from its endpoints. The proposed solutions together form a new foundation for a robust quality-of-service communication via a scalable edge-based architecture where the novel functionality is added strictly at either edge routers or end hosts. In particular, this thesis develops techniques for multi-class service inference, active probing for available bandwidth, and end-point-based protection against DoS attacks.

The proposed multi-class service inference techniques reveal the sophisticated multi-class network components such as service disciplines and rate limiters using solely passive packet monitoring at the network edges. These inferences significantly enhance the network monitoring and service validation capabilities and provide vital information for making efficient use of resources. The proposed active probing

scheme infers and utilizes only the available network bandwidth and aims to realize a low-priority service from the network endpoints, a functionality that would otherwise require a multi-priority or separate network. Finally, this thesis discovers and explores two deterrent vulnerabilities of the Transmission Control Protocol (TCP), the dominant transport protocol in today's Internet. The first is TCP's vulnerability to low-rate periodic attacks that can be as harmful as the high-rate ones, yet much harder to detect, due to their low-rate nature; the second is an extreme vulnerability of the class of receiver-driven TCP stacks to the misbehaviors launched at the receiving endpoints which may temper with the congestion control algorithm for their own benefit. The proposed end-point schemes significantly outperform the state-of-the-art core-based solutions and demonstrate that counter-DoS mechanisms should be implemented not only in the network core, as conventionally done, but also at the network edge. More importantly, the thesis demonstrates that protocol performance on one hand, and vulnerability to misbehaviors on the other, are quite often fundamentally coupled such that both cannot be maximized simultaneously.

Acknowledgments

I have always believed that writing acknowledgments is a risky job for two reasons. First, because there is always a chance to unpurposely miss somebody; and second, because such texts can sometimes be overwhelmingly pathetic. With the hope that I did not miss anybody, and without bounding the pathetic part, I would like to thank the following people.

I am deeply grateful to my advisor Edward W. Knightly. First, because he provided me with the opportunity to come to Ph.D. studies on Rice University. Second, for being open-minded to give me a freedom that is so necessary for research. Finally, over years, I realized that I adopted and started sharing the same values with my advisor for the networking research, and that is what I am grateful for the most.

I am also thankful to my thesis committee members- Professor Peter Druschel and Professor Rudolf H. Riedi for their help and guidance. Rolf has always been a dear friend. His patience, enthusiasm, and ability to listen are truly remarkable.

I would like to thank all the members of the Rice Networks Group (RNG) that I worked with during the last five years. In particular, I would especially like to thank my colleague Violeta for her friendship and help in both research and course work. My confusion in the first year on Rice was endless, and I am not sure if I would have survived all the difficulties without her help. Next, I would like to thank Soups for his friendship, especially during our internship in Fall 2003. He helped me to better understand the beauty of the city of San Francisco and Bay Area in general. Also, a special thanks go to the RNG postdocs, with whom I always had a great

communication: Chengzhi, Huirung, and Roger. Finally, I would like to thank other RNG members: Vikram, Bahar, Yonghe, Coscun, Ashu, Ping, Rahul, Hasan, Joshua, and Joseph for their help and support.

I would certainly like to thank other members of the Rice community: Shri and Vinay for helpful research discussions; Professor Rich Baraniuk for spreading joy and enthusiasm wherever he appears; Professor Michael Orchard for a set of very deep philosophical conversations about various topics during the years of 2000 and 2001; Predrag for discussions and analysis of the Serbian and world politics; and Farbod for sharing his inspiring philosophical thoughts;

I would like to thank to my mother Slavka and my brother Dejan for their help and support during my studies. I know very well how much they are proud on me, and that makes me very happy.

I would like to thank my wife Goca. Knowing how much she is allergic to pathetic sentences, I am doing my best to make the long story short: none of this would be the same without you - I love you.

Finally, I dedicate this thesis to my father Zivko Kuzmanovic, let him rest in peace.

Contents

Tables

Illustrations

Chapter 1

Introduction

The Internet has evolved from a small, well controlled, trusted network into a gigantic, loosely controlled, and highly uncooperative infrastructure of astonishing scale and complexity. Hence, the development and deployment of advanced services on it has reached a crossroads: efforts to add new services in the network core have quickly encountered scalability problems, yet new services are in critical demand and must be rapidly and widely deployed.

Consequently, there is a quest to step back toward the original design principles of the Internet [1] and push the advanced functionality to the network edge while implementing minimum functionality at the network core. And while the design principles and constraints are left unchanged, the expectations from the future Internet significantly overcome a moderate single best-effort datagram service of the past: the users demand multiple traffic classes, service differentiation, security, and QoS guarantees. To achieve these goals, network researchers are challenged to devise sophisticated new algorithms or improve the existing ones, yet using solely measurements obtained at the network edge.

To implement a QoS or a transmission control from the network edge, it is essential to make solid *inferences* of both static and dynamic internal network properties. For example, to make successful capacity planning decisions, a network engineer needs estimates of both lower (or guaranteed) and upper service bounds. Unfortunately, the Internet's large scale and the uncooperative nature of its hosts and networks

make direct monitoring infeasible. Moreover, Internet Service Providers (ISP) are usually reluctant to divulge information about the internal mechanisms that they implemented (if any) to shape or differentiate clients' traffic.

On the other hand, to perform a solid transmission control in a wide-area network that does not provide any explicit information about the network path, a transmission protocol should form its own estimates of current network conditions, and then to use them to adapt as efficiently as possible. A classic example of such estimation and control is how TCP infers the presence of congestion along an Internet path by observing packet losses, and either cuts its sending rate in the presence of congestion, or increases it in the absence.

Another problem arises from the fact that the Internet is designed with an assumption of a global cooperation that is increasingly invalid today: not only that different hosts or networks of the Internet have divergent functional or economical interests, but much worse, the Internet has become a "playground" for malicious denial-of-service attackers of all kinds. A typical attack scenario is the one where a malicious client disobeys congestion control rules and floods the network with traffic, thereby denying service to well-behaving TCP-controlled flows. Hence, it is becoming *a must* to develop robust end-point protocols that would have not just to target certain functionality, scalability, and efficiency, but also to explore the "security holes", anticipate possible DoS attacks, and consequently define counter-measures in presence of malicious behavior. It is essential to apply the above counter-DoS measures *a priori*, during the protocol design process, and before the new services become deployed. Otherwise, possible vulnerabilities may soon be revealed and exploited by unscrupulous hosts, and the overall effects may have devastating consequences.

The goal of this thesis is to design, implement, and evaluate a series of edge-based algorithms and protocols for efficient inference and control of the Internet from its

endpoints. The proposed solutions together form a new foundation for a DoS-resilient quality-of-service communication via a scalable edge-based architecture where the novel functionality is added strictly at either edge routers or end hosts. In particular, this thesis focuses on three parts of the above edge-based inference and control problem: developing algorithms and methodologies for accurate *inference, characterization, and classification of QoS-enhanced systems*, providing globally-available *low-priority service via end-point congestion control*, and *building robust DoS-resilient end-point protocols*.

Inference, Characterization, and Classification of QoS-Enhanced Systems

This thesis develops a framework for monitoring, validation, and inference of multi-class services. It shows how passive monitoring of system arrivals and departures can be used to detect if a class has a minimum guaranteed rate and/or a rate limiter. Moreover, if such elements exist, this work shows how to compute their maximum likelihood parameters. Beyond a single class, it also shows how inter-class relationships can be assessed. For example, this research devises tests which infer not only whether a service discipline is work-conserving or non-work-conserving, but also the relationship among classes, such as weighted fair or strict priority.

In particular, the methodology uses empirical arrival and service rates measured across multiple time scales and devises hypothesis tests for determining the most likely service discipline among Earliest Deadline First (EDF), Weighted Fair Queuing (WFQ,) and Strict Priority (SP). Beyond inferring the scheduler type, the methodology further finds the maximum likelihood estimates of the unknown scheduler parameters such as EDF delay bounds, WFQ weights, and relative SP class priorities.

This research considers a general system model that encompasses a broad class of multi-service elements ranging from routers to web servers, yet it necessarily forgo modeling of many of the intricacies of realistic systems (e.g., the discussion is limited to a single bottleneck node). For inferences of the system's multi-class characteristics, it considers the case where internal system information, such as buffer size or link capacity, is not available. Moreover, it is assumed that arrivals and departures of other classes (i.e., "cross traffic") cannot be explicitly observed and measured at the network edge. Thus, an integral part of the technique is to first assess and statistically characterize the service available to the traffic aggregate that is explicitly measured at the network edge, and then determine mutual relationships among classes within the aggregate.

Low-Priority Service via End-Point Congestion Control

This thesis further devises TCP Low Priority (TCP-LP), an end-point protocol that achieves two-class service prioritization without any support from the network. The key observation is that end-to-end differentiation can be achieved by having different end-host applications employ different congestion control algorithms as dictated by their performance objectives. Since TCP is the dominant protocol for best-effort traffic, TCP-LP is designed to realize a low-priority service as compared to the existing best effort service. Namely, its objective is for TCP-LP flows to utilize the bandwidth left unused by TCP flows in a non-intrusive, or TCP-transparent, fashion. Moreover, TCP-LP is a distributed algorithm that is realized as a sender-side modification of the TCP protocol.

The methodology for developing TCP-LP is as follows. First, this thesis develops a reference model to formalize the two design objectives: TCP-LP transparency to TCP, and (TCP-like) fairness among multiple TCP-LP flows competing to share the

excess bandwidth. The reference model consists of a two level hierarchical scheduler in which the first level provides TCP packets with strict priority over TCP-LP packets and the second level provides fairness among microflows within each class. TCP-LP aims to achieve this behavior in networks with non-differentiated (first-come-first-serve) service.

Next, to approximate the reference model from a distributed end-point protocol, TCP-LP employs two new mechanisms. First, in order to provide TCP-transparent low-priority service, TCP-LP flows must detect oncoming congestion prior to TCP flows. Consequently, TCP-LP uses inferences of one-way packet delays as early indications of network congestion rather than packet losses as used by TCP. A desirable consequence of early congestion inferences via *one-way* delay measurements is that they detect congestion only on the forward path (from the source to the destination) and prevent false early congestion indications from reverse cross-traffic.

TCP-LP's second mechanism is a novel congestion avoidance policy with three objectives: (1) quickly back off in the presence of congestion from TCP flows, (2) quickly utilize the available excess bandwidth in the absence of sufficient TCP traffic, and (3) achieve fairness among TCP-LP flows. To achieve these objectives, TCP-LP's congestion avoidance policy modifies the additive-increase multiplicative-decrease policy of TCP via the addition of an inference phase and use of a modified back-off policy.

Building Robust End-Point Protocols

Finally, this thesis analyzes TCP in presence of misbehaving flows, which are one of the major threats to adequate QoS. It first analyzes TCP's vulnerability to low-rate periodic attacks, and then the vulnerability of receiver-driven TCP stacks. In both scenarios, the thesis develops counter-DoS mechanisms and analyzes their efficiency.

Low-rate TCP-targeted DoS Attacks and Counter Strategies

While TCP's congestion control algorithm is highly robust to diverse network conditions, its implicit assumption of end-system cooperation results in a well-known vulnerability to attack by high-rate non-responsive flows. However, little is known about *low-rate* denial of service attacks. This work discovers and presents low-rate attacks that can be as harmful as the high-rate ones, yet even more dangerous due to the fact that they are difficult for routers and counter-DoS mechanisms to detect.

In particular, the low-rate attack (named the *shrew* attack) consists of short, maliciously-chosen-duration bursts of packets that repeat with a fixed, maliciously chosen, slow-time-scale frequency. This traffic pattern is carefully designed to exploit TCP's deterministic retransmission timeout (RTO) mechanism: (1) the initial short-time-scale burst creates an outage that forces TCP flows to simultaneously backoff on the RTO time-scales; (2) the repeated RTO-periodic bursts manage to deny service to TCP flows who continually incur loss as they try to exit the timeout state, fail to exit timeout, and obtain near zero throughput. Hence the foundation of the shrew attack is a null frequency at the relatively slow time-scale of approximately RTO enabling a low average rate attack that is difficult to detect.

The thesis first explores and analytically analyzes the impact of the shrew attack on a single TCP flow, and then generalizes the scenario and explores the impact of aggregation and heterogeneity on the effectiveness of the attack. It shows that even under aggregate flows with heterogeneous RTT's, heterogeneous file sizes, different TCP variants (New Reno, Sack, etc.), and different buffer management schemes (drop tail, RED, etc.), similar behavior occurs albeit with different severity for different flows and scenarios. The reason for this is that once the first brief outage occurs, all flows will simultaneously timeout. If their RTOs are nearly identical, they

synchronize to the attacker's period and enter a cycle identical to the single-flow case, even with heterogeneous RTTs and aggregation. However, with highly variable RTTs, the success of the shrew attack is weighted such that small RTT flows will degrade far worse than large RTT flows, so that the attack has the effect of a high-RTT-pass filter.

Finally, this thesis explores potential solutions to shrew attacks. While it may appear attractive to remove the RTO mechanism all together or choose very small RTO values, the thesis does not pursue this avenue as timeout mechanisms are fundamentally required to achieve high performance during periods of heavy congestion [2]. Instead, it considers a class of randomization techniques in which flows randomly select a value of minRTO such that they have random null frequencies. The thesis then develops a combination of analytical modeling and simulation to show that such strategies can only distort and slightly mitigate TCP's frequency response to the shrew attack. Moreover, it devises an optimal DoS attack given that flows are randomizing their RTOs and shows that such an attack is still quite severe.

The Vulnerability of Receiver-Driven TCP Stacks and Counter Strategies

Further, this thesis explores the vulnerability of the class of receiver-centric TCP stacks (e.g., [3–10]) in environments with untrusted receivers which would have both means (faster web browsing and file downloads) and incentive (open source operating systems) to temper with the congestion control algorithm for their own benefit. First, this research anticipates and analyzes a set of possible receiver misbehaviors, ranging from classical denial-of-service attacks, e.g., receiver request flooding, to more moderate and consequently harder-to-detect misbehaviors. The misbehaviors are divided into two classes: the first is long-time-scale misbehaviors that manipulate the additive-increase-multiplicative-decrease (AIMD) or retransmission timeout (RTO)

parameters such that flows steal bandwidth over longer time-scales; the second class is short-time-scale misbehaviors that forge parameters such as the initial congestion-window size, such that these flows improve the short-file response times at the expense of well-behaved flows. The thesis develops analytical models for both classes of misbehaviors, and shows that the receiver modifications can lead to uncooperative flows achieving dramatically higher bandwidths and reduced latency as compared to behaving flows.

Second, this thesis evaluates a set of state-of-the-art router- and edge-based mechanisms designed to detect and thwart denial-of-service attacks and other flow misbehaviors. Unfortunately, the finding is that some of the schemes, e.g., [11], are completely unable to detect any receiver misbehavior, whereas others, e.g., [12], are fundamentally limited in their ability to detect even very severe end-point misbehaviors. The key reason is the lack of knowledge of flows' round-trip times, which forces such schemes to penalize flows based on their absolute throughput, which in a heterogeneous-RTT environment typically results in punishing short-RTT flows.

Finally, the thesis proposes and evaluates a set of sender-side mechanisms designed to detect and thwart receiver misbehavior, yet without *any* help from a potentially misbehaving receiver. The initial focus is on the long time-scales and the thesis develops a TFRC-based scheme in which senders (i) independently estimate RTT and loss rate without any cooperation from a potentially misbehaving receiver, (ii) dynamically compute the TCP-friendly rate, and (iii) detect out-of-profile behavior. While this end-point approach at the sender-side is able to accurately detect even slight receiver misbehaviors and strictly enforce TCP-friendliness, this research shows that a fundamental tradeoff arises from the fact that in the absence of trust between the sender and receiver, it becomes problematic for the sender to infer whether the receiver is misbehaving or legitimately trying to optimize its performance with an

enhanced protocol stack. Hence, there is a need to strike a balance between performance and trust – fostering innovation and deployment of enhanced TCP stacks while also providing counter-measures against severe abuse.

1.1 Contributions

This work makes the following contributions:

- *A novel methodology for inference and characterization of QoS-enhanced systems from edges.*

  The methodology enables the network [13] and Web-server [14] clients to accurately detect the service differentiation discipline employed by the provider, and to estimate the maximum-likelihood parameters of the class differentiation scheme using solely passive packet monitoring at network edges. The first aspect of thesis' contribution here lies in the discovery that the time-scales play a key role in inferring the internal scheduler type: short time-scales measurements are crucial for detecting and analyzing non work-conserving elements such as rate limiters; in contrast, long time scale measurements best reveal "link sharing" and weights. The second contribution comes from the fact that the methodology treats all these phenomena occurring at different time scales in a uniform and methodical way by using a unified abstraction of envelopes, hypothesis testing, and maximum likelihood estimations.

- *A solution to the service differentiation problem by introducing the concept of low-priority service.*

  Conventional approaches to solving the service differentiation problem consider the existing best-effort class as the low-priority class, and attempt to develop

mechanisms that provide "better-than-best-effort" service. In contrast, the contribution of this thesis lies in the fact that it explores the opposite approach and devises TCP-LP [15], a new algorithm to realize a *low priority* service (as compared to the existing best effort). For example, such a differentiation scheme can significantly reduce the response times of web connections (by up to 90%), when long-lived bulk data transfers use TCP-LP rather than TCP. The second contribution is the fact that TCP-LP is *a distributed* algorithm that uses the concept of delay-based congestion control to approximate the low-priority service from the network endpoints. Hence, TCP-LP requires no support from the network, which makes it feasible, scalable, and easy to deploy.

- *TCP-LP implementation and experimental investigation in the Internet.*

  The key aspect of thesis' contribution here lies in the development and implementation [16] of a practically applicable protocol. The source code of TCP-LP in Linux is available at *http://www.ece.rice.edu/networks/TCP-LP/*. TCP-LP is only a sender-side modification of TCP which makes it easy to deploy. Moreover, due to its low-priority nature, it is incrementally deployable and could be successfully used by *any* subset of users in the Internet or enterprise networks. The foundation of the experimental investigation is that significant amount of excess capacity is available in the Internet today. More importantly, despite their low-priority nature, even longer-RTT TCP-LP flows are able to utilize substantial amounts of spare available bandwidth in a wide area network environment. This foundation strongly supports TCP-LP's candidacy for the leading bulk-data transfer protocol of the future Internet. Moreover, this thesis develops the *high-speed* TCP-LP (HSTCP-LP), a variant of the protocol suitable for low-priority bulk data transfer in high-speed high-RTT networks

[17].

- *Discovery and analysis of the vulnerability of end-point congestion control protocols.*

  Here, the thesis makes the following contributions. First, it develops the low-rate periodic attacks against TCP flows. These attacks can be as harmful as the high-rate ones, yet much harder to detect, due to their low-rate nature [18]. This finding is supported with analytical modeling and simulations. Moreover, to demonstrate the ubiquity of the low-rate attacks, the thesis presents the effects of the limited-scale attacks performed in the parts of the Internet. Second, this thesis discovers an extreme vulnerability of the class of receiver-driven TCP stacks, analytically quantifies the substantial benefits that a malicious client can achieve in terms of stolen bandwidth over long time-scales (e.g., in file- or streaming-server scenarios) and response time improvements for short-files in HTTP scenarios, and studies the deployability of such protocols in environments with untrusted receivers, such as the Internet of today [19]. Finally, on a more general level, the contribution of this thesis lies in promoting and fostering a *proactive* approach in detecting possible system vulnerabilities. Such an approach enables us to discover and prevent new classes of denial-of-service attacks – before they become widely exploited.

- *Fundamental limitations of core-based counter-DoS solutions.*

  This thesis studies state-of-the-art solutions designed to detect and throttle malicious flows in the network (e.g., [11, 12, 20–27]), and the contribution is the finding of the following fundamental limitations. First, while many of the schemes are relatively successful in detecting high-rate attacks, the core-based schemes are highly unsuccessful in their ability to detect low-rate attacks

that operate on short time-scales. Moreover, any attempt to re-engineer these schemes to operate on shorter time-scales would necessarily increase the false alarm probability, i.e., many legitimate bursty flows would be incorrectly detected as malicious [28]. Second, common to all core-based solutions is their fundamental limitation to *precisely* and accurately detect the endpoint misbehaviors due to their lack of knowledge of the flows' round-trip times. Hence, the endpoints can maliciously re-tune their parameters without adverse effects, and yet they can achieve substantial performance benefits, particularly in the receiver-driven TCP scenarios.

- *Development of end-point counter-DoS solutions.*

  This thesis develops several counter-DoS schemes at the network endpoints. The first is the randomization of the RTO parameter used in an attempt to thwart the shrew attack; the second is a sender-side scheme developed to verify the receivers' TCP-friendliness in the receiver-driven TCP scenarios. The first contribution here lies in reaching the understanding that counter-DoS protection mechanisms need not to be implemented exclusively in the network core, as conventionally done, but that such mechanisms need to be included in the end-point protocol design as well. Moreover, this thesis demonstrates that the end-point solutions can significantly outperform the core-based solutions. However, the key contribution of this part of the thesis is the finding that protocol performance on one hand, and vulnerability to misbehaviors on the other, are quite often coupled such that both cannot be maximized simultaneously. Hence, to completely defend the system in presence of such misbehaviors and attacks, it is necessary to sacrifice system performance in their absence.

1.2 Dissertation Overview and Summary of Representative Experimental Results

Chapter 2 reviews the fundamentals of the inter-class resource sharing theory and provides background on TCP congestion control. In particular, it first gives a background on TCP's timeout mechanism and then reviews the class of receiver-based transport protocols.

Chapter 3 presents the framework and methodology for inference and characterization of QoS-enhanced systems. It first explains targeted Web server and network scenarios, defines the measurement and inference problem, and describes the system model. Next, it devises the maximum likelihood estimates for the system parameters and hypothesis tests for inference of the service discipline. It further presents a large set of simulation experiments in both networking and web-server scenarios and finds that the technique is practically applicable. For example, in the networking experiments with the majority-rule hypothesis test performed across multiple time scales, multi-class EDF scheduling was correctly inferred 100% of the time when the class delay bounds were sufficiently differentiated, and class-based fair queuing was correctly inferred 94% of the time. Once the service discipline is known, the algorithm estimated class WFQ weights within 1.4% of the correct value with 95% confidence. In the web-server experiments, the scheme correctly classified the scheduling discipline in more than 90% of the cases.

Chapter 4 first develops the reference model to describe TCP-LP's design objectives, and then presents the TCP-LP protocol. In the experimental part, it presents an extensive set of *ns-2* simulation experiments to study TCP-LP's characteristics in a variety of scenarios (single and multiple bottlenecks, short- and long-lived TCP flows, etc.). First, in the experiments with greedy TCP flows (FTP downloads), the

results show that TCP-LP is largely non-intrusive to TCP traffic, and that TCP flows achieve approximately the same throughput whether or not TCP-LP flows are present. Second, the thesis explores TCP-LP's dynamic behavior using experiments with artificial "square-wave" background traffic. It shows that single and aggregate TCP-LP flows can successfully track and utilize the excess network bandwidth. Finally, in the experiments with HTTP background traffic, the thesis shows that flows in the best-effort class can benefit significantly from the two-class service prioritization scheme. For example, the response times of web connections in the best-effort class decrease by up to 90% when long-lived bulk data transfers use TCP-LP rather than TCP.

Furthermore, Chapter 4 presents an implementation of TCP-LP in Linux and evaluates it both in a testbed as well as on the Internet. In the testbed, this work presents experiments with many TCP and TCP-LP flows and shows that TCP-LP remains its TCP-transparent property even in such large-aggregation regimes. Likewise, the Internet experiments show that TCP-LP remains non-intrusive in a wide-area network environment, while being able to utilize substantial amounts of the available spare network bandwidth. For example, when compared to TCP, TCP-LP is able to utilize approximately 45% of the TCP throughput on average during working-hours (8 a.m. to 5 p.m.), and as much as 75% outside this interval.

Chapter 5 presents the low-rate denial of service attacks against TCP flows. It first explains and models origins of the vulnerability of the timeout mechanism to low-rate attacks. Next, it explores the impact of TCP flow aggregation and heterogeneity on the effectiveness of the shrew attack and finally proposes and evaluates a set of core- and end-point-based counter-DoS mechanisms intended to mitigate the effects of the shrew attacks. In the experimental part, it presents the results of the Internet experiments performed both in a local- and a wide-area environment. While necessarily

small scale experiments (given that the expected outcome is to reduce TCP throughput to near zero), the experiments validate the basic findings and show that even a remote attacker (across a WAN) can dramatically reduce TCP throughput. For example, in the WAN experiments, a remote 909 kb/sec average-rate attack consisting of 100 ms bursts at the victim's RTO time-scale reduced the victim's throughput from 9.8 Mb/sec to 1.2 Mb/sec.

Chapter 6 first analyzes the limitations of existing network-based solutions and then proposes and evaluates a set of sender-based solutions targeted to protect the system over long- and short-time-scales, respectively. The thesis analytically quantifies the substantial benefits that a malicious client can achieve in terms of stolen bandwidth over long time-scales and response time improvements for short-files in HTTP scenarios, and confirms these results through simulation.

Chapter 7 concludes the dissertation and discusses some of the future work.

Chapter 2

Background

This chapter provides a brief overview of inter-class resource sharing, which is a starting point upon which the multi-class inference techniques are developed. Next, it reviews the main TCP congestion control phases with an emphasis on those which are of interest for available bandwidth inference and DoS research.

2.1 Inter-Class Resource Sharing

In a multi-class system, the service discipline defines the interclass relationships of the service when different classes compete for resources. For example, an SP scheduler consists of one queue per traffic class with packets from the highest priority non-empty class serviced first. A packet in level i is serviced only if no packets are backlogged in levels $1, \cdots, i-1$. Thus, with an SP scheduler, the highest priority class receives all demanded service up to the available link capacity, and in that way is completely isolated from other classes' demands. In contrast, lower priority classes utilize only *remaining* capacity from higher priority classes and their performance is strongly dependent on these classes' demands.

For WFQ, each traffic class i is allocated a guaranteed capacity $\phi_i C$ such that whenever packets from class i are backlogged, the class receives service at a rate of at least $\phi_i C$. Unused capacity of non-backlogged classes is distributed in a weighted fair manner among backlogged classes. For EDF, each class has an associated delay bound so that packet j of class i arriving at time a_j^i has deadline a_j^i plus its delay

bound, and the scheduler selects the packet with the smallest (earliest) deadline for service.

The following section provides a theoretical description of such interclass relationships via statistical service envelopes which provide a unifying abstraction for both arrivals and services and incorporate a system's behavior across time scales.

2.1.1 Statistical Service Envelopes

To characterize a flow's rate, an associated interval length must also be specified. However, the arrival workload (expressed in number of bits) varies in time over intervals of the same length, simply due to variable source behavior. Thus, to accurately characterize randomness of flow arrivals, this thesis applies the concept of *statistical* arrival envelopes to capture a flow's variability over intervals of different length. Denote class i's statistical arrival envelope as $B^i(t)$, which is a sequence of random variables that characterizes arrivals from class i over time intervals of duration t.[1] It is assumed that each class arrival process is stationary and that $B^i(t)$ and $B^j(t)$ are statistically independent when $i \neq j$.

In [29], statistical admission control tests are developed for several multi-class schedulers. The key technique for exploiting inter-class resource sharing is to characterize a class' available service beyond its worst-case allocation. For example, in a WFQ server, a class with weight ϕ_i receives service at rate no less than $\phi_i C$ whenever it is backlogged ($\sum_j \phi_j = 1$). However, due to statistically varying demands of other classes, the service received can be far greater than this lower bound. A statistical *service* envelope $S^i(t)$ is therefore a general characterization of the service received by class i over intervals of length t for which the class is continually backlogged.

[1] For a particular time scale $t = t_1$, $B^i(t_1)$ is a random variable.

Equations (2.1), (2.2) and (2.3) show the statistical service envelopes for SP, WFQ, and EDF schedulers, respectively.

$$S^i_{SP}(t) = \left(Ct - \sum_{n=1}^{i-1} B^n(t) \right)^+ \tag{2.1}$$

$$S^i_{WFQ}(t) = \phi_i Ct + \left((1-\phi_i)Ct - \sum_{n \neq i} B^n(t) \right)^+ \tag{2.2}$$

$$S^i_{EDF}(t) = \left(Ct + CD_i - \sum_{n \neq i} B^n(t - \delta_n + \delta_i) \right)^+ \tag{2.3}$$

The envelopes are a function of the link capacity C, and as described above, the other class' input traffic, described by the arrival envelope $B^i(t)$. For SP, observe that class i's service is only a function of the workload in classes $1, 2, \cdots, i-1$. In contrast, for WFQ class i's service is a function of all other classes' traffic but is upper bounded by Ct if all other classes are always idle and lower bounded by $\phi_i C$ if all other classes are continuously backlogged. Finally, with EDF class i's service again depends on all other class' inputs as well as the delay bound of class i denoted by δ_i. Chapter 3 shows how theoretical concepts described here can be applied in practice to classify and infer the parameters of multiclass systems.

2.2 TCP Congestion Control

This section provides a review of TCP congestion control. It first explains basic phases of the transmission control, and then briefly discusses the class of delay-based congestion control protocols that are relevant for the work presented in Chapter 4. Next, it presents background on TCP's retransmission timeout (RTO) mechanism, which is the key source of vulnerability to low-rate denial-of-service attacks, which are explained and explored in Chapter 5. Finally, it provides a background on the

class of receiver-driven TCP protocols, whose vulnerabilities are studied in detail in Chapter 6.

2.2.1 TCP Phases and Delay-Based Congestion Control

Figure 2.1 shows a temporal view of the TCP congestion window behavior at different stages with points on the top indicating packet losses.[2] Data transfer begins with the *slow-start* phase in which TCP increases its sending rate exponentially until it encounters the first loss or maximum window size. From this point on, TCP enters the *congestion-avoidance* phase and uses an additive-increase multiplicative-decrease policy to adapt to congestion. Losses are detected via either time-out from non-receipt of an acknowledgment, or by receipt of a triple-duplicate acknowledgment. If loss occurs and less than three duplicate ACKs are received, TCP reduces its congestion window to one segment and waits for a period of retransmission time out (RTO), after which the packet is resent. In the case that another time out occurs before successfully retransmitting the packet, TCP enters the *exponential-backoff* phase and doubles RTO until the packet is successfully acknowledged.

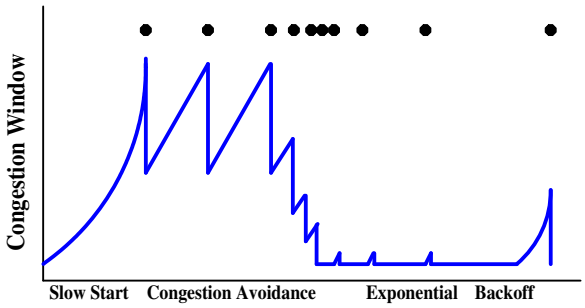


Figure 2.1 : Behavior of TCP Congestion Control

[2] A detailed description of TCP can be found in [30].

While packet losses are almost exclusively used as indications of the network congestion, there are proposals to apply *delay-based* congestion control and the representative protocols are Jain's delay-based congestion avoidance protocol [31], Wang *et al.'s* TCP/Dual [32], and Brakmo *et al.'s* TCP/Vegas [33]. Common to all of the protocols is the use of roundtrip times for congestion indications (instead of packet losses) in an effort to increase TCP throughput due to a reduced number of packet losses and timeouts. Chapter 4 demonstrates how delay-based congestion control mechanisms can be used to effectively implement a protocol that utilizes only the excess network bandwidth, hence provides a strict low-priority service without any support from the network.

2.2.2 TCP's Timeout Mechanism

This section presents background on TCP's retransmission timeout (RTO) mechanism [34]. TCP detects loss via either timeout from non-receipt of ACKs, or by receipt of a triple-duplicate ACK. If loss occurs and less than three duplicate ACKs are received, TCP waits for a period of retransmission timeout to expire, reduces its congestion window to one packet and resends the packet.

Selection of the timeout value requires a balance among two extremes: if set too low, spurious retransmissions will occur when packets are incorrectly assumed lost when in fact the data or ACKs are merely delayed. Similarly, if set too high, flows will wait unnecessarily long to infer and recover from congestion.

To address the former factor, Allman and Paxson experimentally showed that TCP achieves near-maximal throughput if there exists a lower bound for RTO of one second [2]. While potentially conservative for small-RTT flows, the study found that *all flows* should have a timeout value of at least 1 second in order to ensure that congestion is cleared, thereby achieving the best performance.

To address the latter factor, a TCP sender maintains two state variables, SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation). According to [34], the rules governing the computation of SRTT, RTTVAR, and RTO are as follows. Until a RTT measurement has been made for a packet sent between the sender and receiver, the sender sets RTO to three seconds. When the first RTT measurement R' is made, the host sets SRTT = R', RTTVAR = R'/2 and RTO = SRTT + max(G, 4RTTVAR), where G denotes the clock granularity (typically $\leq$ 100 ms). When a subsequent RTT measurement R' is made, a host sets

$$RTTVAR = (1 - \beta)\, RTTVAR + \beta\, |SRTT - R'|$$

and

$$SRTT = (1 - \alpha) SRTT + \alpha\, R'$$

where $\alpha = 1/8$ and $\beta = 1/4$, as recommended in [35].

Thus, combining the two parts, a TCP sender sets its value of RTO according to

$$RTO = \max(minRTO,\, SRTT + \max(G, 4\, RTTVAR)). \tag{2.4}$$

Finally, the RTO management is illustrated via a *retransmission-timer* timeline in Figure 2.2. Assume that a packet with sequence number n is sent by a TCP sender at reference time $t = 0$, and that a retransmission timer of 1 second is initiated upon its transmission. If packet n is lost and fewer than three duplicate ACKs are received by the sender, the flow "times out" when the timer expires at $t = 1$ sec. At this moment, the sender enters the exponential backoff phase: it reduces the congestion window to one, doubles the RTO value to 2 seconds, retransmits the unacknowledged packet with sequence number n, and resets the retransmission timer with this new RTO value.

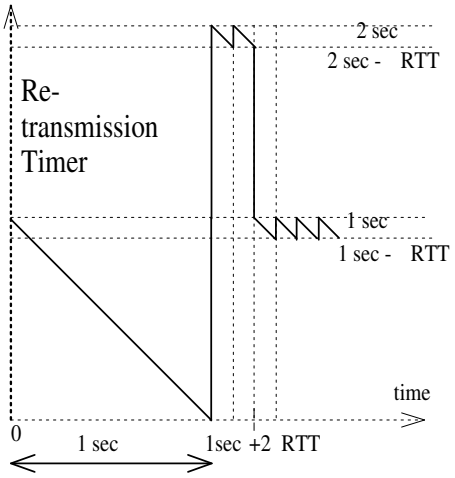


Figure 2.2 : Behavior of the TCP retransmission timer

If the packet is lost again (*not* shown in Figure 2.2), exponential backoff continues as the sender waits for the 2 sec-long retransmission timer to expire. At $t = 3$ sec, the sender doubles the RTO value to 4 seconds and repeats the process.

Alternately, if packet n is successfully retransmitted at time $t = 1$ sec as illustrated in Figure 2.2, its ACK will arrive to the sender at time $t = 1 + \text{RTT}$. At this time, the TCP sender exits the exponential backoff phase and enters slow start, doubling the window size to two, transmitting two new packets $n + 1$ and $n + 2$, and resetting the retransmission timer with the current RTO value of 2 sec. If the two packets are not lost, they are acknowledged at time $t = 1 + 2\,\text{RTT}$, and SRTT, RTTVAR and RTO are recomputed as described above. Provided that minRTO $>$ SRTT + max(G, 4 RTTVAR), RTO is again set to 1 sec. Thus, in this scenario in which timeouts occur but exponential backoff does not, the value of RTO deviates by no more than RTT from minRTO for $t >$ minRTO + 2 RTT.

The above timeout mechanism, while essential for robust congestion control, provides an opportunity for low-rate DoS attacks that exploit the slow-time-scale dy-

namics of retransmission timers. In particular, an attacker can provoke a TCP flow to repeatedly enter a retransmission timeout state by sending high-rate, but short-duration bursts having RTT-scale burst length, and repeating periodically at slower RTO time-scales. The victim will be throttled to near-zero throughput while the attacker will have low average rate making it difficult for counter-DoS mechanisms to detect. This issue will be studied in detail in Chapter 5.

2.2.3 Receiver-Based TCP

This section reviews transport protocols that delegate some or all control functions to receivers. For scenarios ranging from web browsing to wireless networks, the key advantages of receiver-driven protocols are improved response times and throughput due to exploitation of information available at the receiver. However, the fact that some or all control functions are delegated to receivers raises a fundamental security concern that will be studied in detail in Chapter 6.

Delegating Control Functions to Receivers

One of the first transport protocols that exploits increased receiver functionality is Clark *et al.*'s NETBLT [3], which makes error recovery more efficient by placing the data retransmission timer at the receiver. In later work, an increased set of control functions appear at the receiver, either for performance or practical reasons (e.g., to decrease the computation burden at the sender). For example, Sinha *et al.*'s WTCP [6] calculates the sending rate at the receiver; Floyd *et al.*'s TFRC [4] maintains the loss history and computes the TCP-friendly rate at the receiver; Tsaoussidis and Zhang's TCP-Real [8] tracks loss events and determines the data delivery rate at the receiver; Spring *et al.* [7] and Mehra *et al.* [5] add functionality to the receiver to control the bandwidth shares of incoming TCP flows, i.e., by adapting the receiver's

advertised window and delay in transmitting *ack* messages, the receiver is able to control the bandwidth share on the access link according to the client's needs.

Fully Receiver-Driven Transport Protocols

In contrast to the above protocols, *all* control functions are delegated to receivers in Web Transport Protocol (WebTP) [9] and Reception Control Protocol (RCP) [10]. Hsieh *et al.* [10] argue that the key advantage of fully receiver-centric transport protocols is that the *receiver* controls *how much data can be sent*, and *which data should be sent* by the sender. The section below summarizes some of the performance and functionality gains that a fully receiver-centric protocol can achieve in the large-scale server scenarios as well as in a scenario where the mobile host acts as the receiver for traffic from a wireline sender.

Performance Gains

This section elaborates on two of the sources of performance gains explored in [10]. The first is improved loss recovery. In particular, while TCP *ack* packets are resilient to losses due to their cumulative nature, they provide little information that the sender can use to effectively recover from losses. While TCP Sack [36] is able to recover from losses by using three "Sack blocks", the effectiveness of recovery is limited to the extend to which the sender can accurately construct the receiver buffer in a timely fashion. Hence, heavy losses on the forward path, coupled with a lossy reverse path (typical for wireless environments), may prevent the TCP Sack sender from accurately constructing the receiver's buffer state. On the contrary, the receiver has direct access to the receive buffer, and hence can always recover from losses in an effective fashion, without incurring the overhead inaccuracies of TCP Sack.

The second source of performance gains arise via wireless-aware congestion control.

Namely, the wireless link typically plays a defining role in determining the characteristics of an end-to-end path. Hence, "wireless-aware" congestion control algorithms exploit information about the characteristics of the wireless link (e.g., loss classification, RTT sample filtering or reasons for non-congestion related outages (handoffs or channel blackouts)). Since the receiver is adjacent to the wireless last-hop, it has first-hand knowledge about the above information.

Functionality Gains

The fact that receiver-centric protocol design delegates the entire protocol "intelligence" to the receiver yields a number of functionality improvements. First, sender-based TCP places all protocol state on the servers which must handle large scale workloads (e.g., web servers). On the other hand, receiver-driven transport protocols can significantly reduce the complexity of the server implementation since they distribute the state management across the large number of clients.

Second, during periods of mobility, a mobile host with heterogeneous wireless interfaces can benefit from the fact that the transport protocol functionality is concentrated at the receiver. For example, when a mobile host performs a handoff from one access network to another, it can avoid connection disruptions due to temporary link outage by using both interfaces simultaneously if the transport layer is able to use multiple interfaces without suffering from performance degradation due to persistent packet reordering. Furthermore, the mobile user can benefit from migrating to a replicated server, either because the new interface has no access to the old server, or for performance considerations.

RCP Protocol

This section provides a brief overview of RCP, variants of which we further consider in Chapter 6.

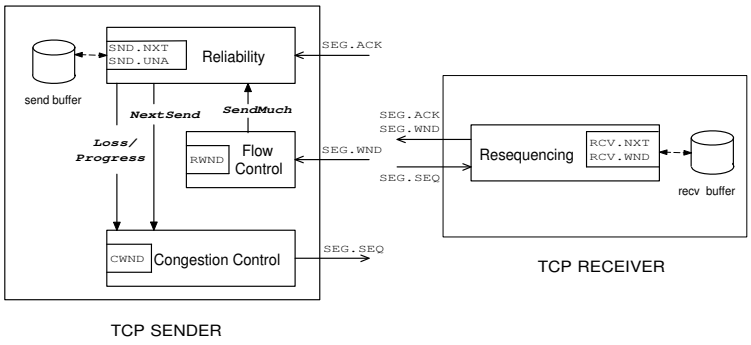


Figure 2.3 : TCP functionalities at the sender and receiver

All TCP variants provide reliable in-sequence data delivery to the application, with protocol operations consisting mainly of four mechanisms: connection management, flow control, congestion control, and reliability. Figure 2.3 depicts a schematic view of the interaction between sender and receiver in TCP, together with several state variables.

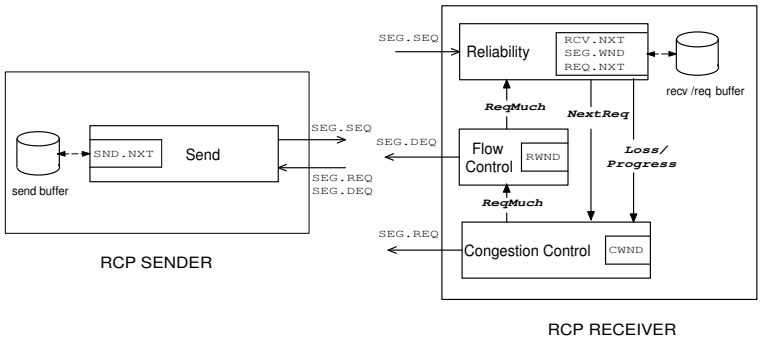


Figure 2.4 : RCP functionalities at the sender and receiver

Observe that except for connection management, which needs to be implemented at both ends, Figure 2.4 indicates that RCP delegates all other control functions to the receiver. Thus, either the sender or receiver can initiate connection setup, after which the receiver becomes fully responsible for reliability, flow control, and congestion control, using the same window-based mechanisms employed in sender-driven TCP. Since RCP shifts the control of data transfer from the sender to receiver, the *data-ack* style of message exchange in TCP is no longer applicable. Instead, to achieve the self-clocking characteristics of TCP, RCP uses *req-data* exchange for data transfer, where any data transfer from the sender is preceded with an explicit request (*req*) from the receiver. Equivalently, the RCP receiver uses incoming *data* packets to clock the requests for new data. In summary, RCP represents a clone of sender-side TCP which simply transfers *all* important control functionalities to the receiver.

However, the fact that *all* control functions are delegated to receivers raises a fundamental security concern for misbehaving receivers that will manipulate protocol parameters (all available at the receiver) and gain significant performance benefits. This concern is amplified by the fact that receivers would have the opportunity (open source operating systems requiring a minor change), and incentive (faster web browsing and file downloads) to perform such activities. Chapter 6 treats this issue in depth.

Chapter 3

Edge-Based Inference, Characterization, and Classification of QoS-Enhanced Systems

Both research and commercial networks and web servers are increasingly able to provide minimum quality-of-service levels to traffic and application classes, e.g., [37]. Example components of such networks include QoS schedulers [38, 39], diffserv-style service level agreements [40–43], edge-based traffic shaping and prioritizing devices, and novel architectures and algorithms for scalable QoS management [44–46]. Similar resource management mechanisms, request scheduling policies and algorithms are also developed for quality-of-service web servers [47–50]. However, even as both the network's and web-server's infrastructure and services become increasingly sophisticated, the network's and web-server's *clients* lack reciprocal tools for validation and monitoring of the system's QoS capabilities, and the available tools allow only the inferences of parameters such as bottleneck link speeds or available bandwidth [51–54]. Clients of Service Level Agreements (SLAs) will have monitoring requirements ranging from basic validation of the SLA's raw bandwidth to more sophisticated inference of multi-class functionalities. For example, is a class rate limited (policed)? If so, what are the rate limiter's parameters and what is necessary to detect this? In a multi-class environment with multiple classes within or among SLAs, what is the inter-class relationship? Fair, weighted fair, strict priority, and with what parameters? Is resource "borrowing" across classes fully allowed or only allowed within certain limits?

Similar issues occur in a web server scenario. The requirements of a client of a web hosting service range from the ability to track, assess and *quantify* basic service capabilities, such as minimum rate at which user's requests are serviced, to the ability to assess mechanisms and parameters by which capacity is allocated to various hosted sites. Besides clients, web hosting *providers* will have similar objectives in a larger scale web-hosting environment in which a number of front-end servers use resources of back-end servers and when different QoS mechanisms are simultaneously implemented in the system. In such an environment, a need to quantify service and assess the inter-class relationship arises.

Obtaining "off-line" answers to such questions can be quite trivial. In particular, consider a system with an unknown service (suppose the system is a single router for simplicity). To assess whether classes are rate limited, one could probe each class, one at a time, with a high rate test sequence: the output of the system would yield the policing parameters. Similarly, simultaneously probing at a high rate in all classes would yield the inter-class relationships: if one class receives all of the service, the system is strict priority (at least for that class); if weighted service is received, the system performs a variant of weighted fair queuing.

In contrast, the "on-line" case, in which one cannot force all other traffic classes to remain idle while experiments are performed, is quite different. Even for classes which are under the control of the client, it may be highly undesirable to disrupt the class with experiments such as above. For example, sending at a high rate to detect rate-limiters may cause excessive packet losses for established sessions.

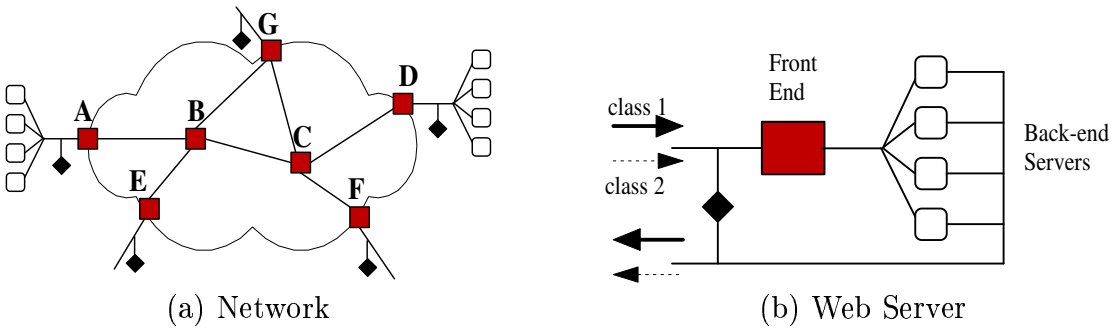
3.1 Targeted Systems and Problem Statement

This section first describes the targeted networking and web server scenarios, and then formally defines the problem and the system model.

3.1.1 Targeted Systems

Network Scenario

Figure 3.1(a) depicts the targeted networking scenario. In this case, measurement modules are placed at the periphery of the network. The goal is to use passive edge-based client measurements to infer the multi-class QoS mechanisms and parameters employed by the network operator. With an improved understanding of the way traffic is internally serviced, clients can better manage their use of multi-class networks. Also, network clients can use the framework to *quantatively* estimate their service when only relative performance guarantees are provided or when end-to-end service is provided through more than one ISP. For example, if the provider guarantees that class X will have higher priority than class Y, the framework can determine maximum likelihood lower and upper service bounds of both classes and infer actual inter-class relationship. Such inferences can be used by network clients to better utilize their available bandwidth, i.e., for capacity planning. Similarly, operators or third parties can employ the methodology to test and validate the performance and potential performance of multiple service classes.



(a) Network (b) Web Server

Figure 3.1 : Targeted Systems

QoS Web Servers

Figure 3.1(b) depicts a two-class distributed web server, where a passive measurement module is depicted by a diamond. QoS functionalities in the server may include prioritized scheduling of incoming requests at the front-end, prioritized distribution of jobs to back-end nodes, and operating-system mechanisms such as prioritized scheduling of CPU, memory, and disk access [49]. In any case, the goal is to provide an application-layer characterization of the system's multi-class QoS mechanisms. For example, weighted share of CPU resources does not guarantee the same level of differentiation for the application, since the actual response times also depend on the file type (static or dynamic), file size and its caching state. Also, if several QoS mechanisms are simultaneously employed with the goal of providing weighted fair service among different classes, the technique will estimate a class' net "guaranteed rate", i.e., it's minimum serviced request throughput. Such inferences have important implications for both performance monitoring and resource management.

3.1.2 Problem Formulation

For inferences of the system's multi-class characteristics, this thesis considers the case where internal system information is *not* available, i.e., neither static configuration information (such as the scheduler's parameters) nor empirical information (such as mean buffer length). Instead, the available information consists of the external observations from passive monitoring of requests, namely request arrival and departure times along with request class labels and sequence numbers. In the case of web servers (both single node and distributed), both arrivals and departures are directly observable from the system's front end (see [47, 49] for a detailed description of such an architecture). In the case of networks, packet time stamping at ingress nodes provides a mechanism to observe both arrival and departure times at the departure node

[55]. In particular, for low speed links (e.g., up to 100 Mb/sec), *tcpdump* can capture and record header information at line rate [55]. For higher speed implementations, this functionality would best be achieved with hardware support. Otherwise, the measurement modules can communicate their collected information off-line. Below, the multi-class service inference problem is formally defined.

Problem statement: Consider a multi-class system with an unknown scheduling discipline fed by requests from N classes. Denote with G the number of observable or explicitly measured classes and assume that classes $1, \cdots, G$ are observable while classes $G+1, \cdots, N$ are not. Denote the arrival and departure times of request j from class i as a_j^i and d_j^i respectively. Given $a_1^i, a_2^i, \cdots$ and $d_1^i, d_2^i, \cdots$ for $i = 1, \cdots, G$,

(a) Estimate the available service of the aggregate consisting of G classes.

(b) Assess the most likely service discipline among SP, WFQ [56], and EDF.

(c) Estimate the maximum likelihood values of the class parameters for each of G measured traffic classes: "guaranteed rate" (ϕ_i) in WFQ, delay bound (δ_i) in EDF and rate limiters (r_i) in non-work conserving servers.

3.1.3 System Model

The general system model considered in this chapter is depicted in Figure 3.2. As in the basic abstraction of service disciplines described in [57], it consists of two stages: non-work-conserving elements which limit a class' rate and a work-conserving packet or request scheduler. For rate limiters, this thesis considers single-level leaky bucket regulators, and for the packet scheduler, it considers SP, WFQ, and EDF, as explained in Chapter 2.

This formulation covers a broad set of class-based scheduling elements, including minimum guaranteed rates, maximum policed rates, weighted fairness, sorted priority, and strict priority. While necessarily not comprehensive, it incorporates both work-

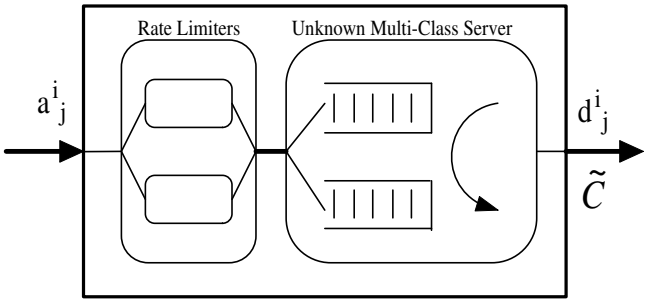


Figure 3.2 : System Model for Multi-Service Measurement

conserving and non-work-conserving service disciplines and a number of mechanisms for inter-class resource sharing and quality-of-service differentiation. The choice of SP, WFQ and EDF, which belong to rate and delay-based classes of schedulers, is made since these schedulers are both well studied and implemented in practice. Also, it should be noted that the multi-class inference framework developed in this chapter can be applied to any other scheduler for which one can derive a statistical service envelope, the key inference tool that is explained in detail in the following section.

This work considers that the capacity of a multi-class system is not known and can vary over time. In the networking scenario, this formulation covers the problem of unknown cross-traffic, while it applies equivavalently to the web-server inference problem, where the capacity is non-constant as the service times for different requests vary due to different CPU service times, disk service times and variable file sizes. The first step in the inference methodology is to assess and statistically characterize the service available to an aggregate of all measured classes, and then determine inter-class relationships within the aggregate.

A special case of the general system model that is considered throughout the chapter is a single bottleneck multi-class networking router with fixed service capacity C, in which all classes' arrivals and departures are known. This thesis studies this

special case service model for two reasons: first, for simplicity of presentation and secondly, as a reasonable and intuitive checkpoint of the inference methodologies applicable to a general system model with variable capacity.

3.2 Service Measurements and Concept of Envelopes

As described above, the goal is to infer the elements and parameters of the multi-class system. In such a system, the request service discipline defines the inter-class relationships or the service received when different classes compete for resources. Parts 3.2.1 and 3.2.2 describe empirical arrival and service models, i.e., they explain how the theoretical concepts described in Chapter 2 can be applied in practice.

3.2.1 Empirical Arrival Model

Here the thesis shows how statistical arrival envelopes $B^i(t)$ defined in Chapter 2 can be measured over multiple time scales using class i's arrival request sequence. Measurement at multiple time scales is important in this context as different system components are most accurately detected at different time scales.

Focusing on a single class for illustration, denote the total arrivals in the interval $[s, s+t]$ by $A[s, s+t]$. A traffic envelope refers to a time invariant characterization of the arrivals as a function of interval length t (see [58] for examples of deterministic envelopes). For a measurement window $[s, s+T]$ and a particular interval length I_k beginning at time $s+(j-1)I_k$, class i's arrival *rate* is given by

$$R_{k,j}^{i,A} = \frac{A^i[s+(j-1)I_k,\ s+jI_k]}{I_k}$$

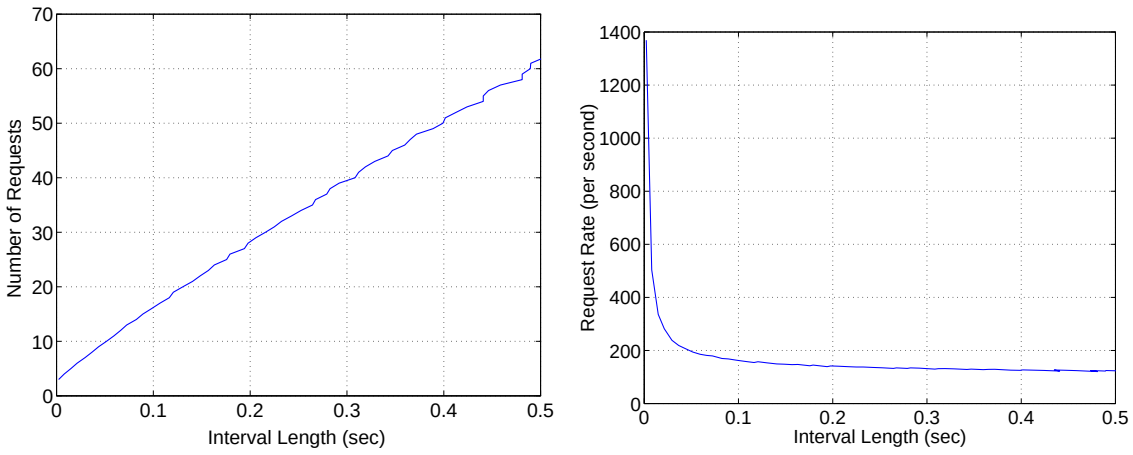
for $j = 1, \cdots, N_k$, where $N_k = \lfloor T/I_k \rfloor$ is the number of successive intervals of length I_k in the measurement window $[s, s+T]$.

Using measured rates over different sub-intervals within the window T, the mean and variance of the empirical rate envelope of class i for intervals of length I_k can be computed as

$$\bar{R}_k^{i,A} = \frac{1}{N_k} \sum_{j=1}^{N_k} R_{k,j}^{i,A} \tag{3.1}$$

and

$$RV_k^{i,A} = \frac{1}{N_k} \sum_{j=1}^{N_k} (R_{k,j}^{i,A} - \bar{R}_k^{i,A})^2 \tag{3.2}$$



(a) Statistical Arrival Envelope (b) Statistical Arrival Rate Envelope

Figure 3.3 : Arrival Envelopes ($mean + 1.6\, deviation$)

Observe that the first two moments of the *rate* arrival envelope (e.g., $\bar{R}_k^{i,A}$ and $RV_k^{i,A}$) are simply empirical and *normalized* versions of the first two moments of class i's arrival envelope $B^i(I_k)$ at time scale I_k. As an example envelope, Figure 3.3(a) shows the representation of the arrival envelope $B^i(t)$ for the Rice University CS department trace described in Section 5.4, while Figure 3.3(b) shows the reciprocate *rate* envelope normalized to the interval length I_k so that the y-axis is rate. Specifically, the Figure 3.3(b) depicts $\bar{R}_k^{i,A} + 1.6\sqrt{RV_k^{i,A}}$ for 50 time scales, $I_k = 0.01, 0.02 \cdots, 0.5$. It is clear that over short interval lengths, significantly more requests than the mean

100 per second (as can be seen from the rate to which the curve in Figure b converges) can arrive. It will be shown that such characteristics of the request workload, i.e., its variability over time scales, is the key input for obtaining accurate scheduler inferences.

Section 3.3 describes how this empirical class-based arrival rate envelope is incorporated into the above multi-class inference problems, and Section 5.4 experimentally investigates applications of this traffic characterization.

3.2.2 Empirical Service Model

This section describes a general mechanism for measuring and characterizing a service rate. Analogous to the traffic envelope, the service rate envelope is not simply a single service bandwidth, but a statistical characterization of service across time scales. There are two types of service rate envelopes: *aggregate* and *class* envelopes.

Aggregate service rate envelope characterizes a service rate available to an aggregate of explicitly measured classes. It captures the effects of non measured cross traffic in networks, or reveals and characterizes non-constant service capacity of a web sever. Similarly, a *class* service rate envelope characterizes a service rate available to each traffic class *within* the aggregate and across time scales. In both cases, this multiple-time-scale characterization is critical to inference of diverse service components such as maximum policed bandwidth, minimum service, and analysis of inter-class resource sharing relationships. Moreover, its statistical nature reflects the fact that a class' service can fluctuate according to the varying demands of other classes and the mechanism by which the scheduler arbitrates this demand.

Backlogging Condition

The empirical service rate envelope characterizes the service rate received by the flow (either a class' or an aggregate's flow) as a function of the interval length over which the flow is backlogged, where a flow is said to be backlogged whenever it has at least one packet in the system. A traffic flow is continuously backlogged for k packet transmissions in the interval $[a_j, d_{j+k-1}]$ if

$$d_{j+m} > a_{j+m+1}, \text{ for all } 0 \leq m < k-2,$$

for $k \geq 2$. Note that all packet transmissions are backlogged for $k = 1$ in the interval $[a_j, d_j]$ (see reference [13] for an illustration of a backlogging condition).

Thus, denoting $U[s, s+t]$ as number of flow's bits[1] serviced in the backlogged interval $[s, s+t]$, the service *rate* received in $[s, s+t]$ is simply

$$M^S(t) = \frac{U[s, s+t]}{t}. \tag{3.3}$$

Finally, the measurement for each backlogged interval is included in the measurement $\vec{M}_k^S$ if

$$(k - 0.5)I_1 < t \leq (k + 0.5)I_1. \tag{3.4}$$

Measured service envelope samples $\vec{M}_k^S$, both per class and aggregate, are used in inferring the scheduling discipline as will be in detail explained in the following section.[2]

---

[1] The actual units of monitoring are packets or requests. However, since these can be of different size, the workload is represented in bits.

[2] Notice that for convenience, the arrival envelope is discretized in time and the service envelope is discretized in bits. However, to perform the comparative computations of Section 3.3, both are expressed in discrete time rates with service interpolated.

Empirical Aggregate Service Model

Analogous to arrival traffic envelope, the *aggregate* service rate envelope is determined with first two moments of its service rates across the time scales. Denote the aggregate service rate measurements in time scale I_k as $\vec{C}_k = \vec{M}_k^S$, where $\vec{M}_k^S$ is measured as explained in Section 3.2.2 for the arrival-departure sequence of the aggregate flow. Then, using these measured rates over different sub-intervals within the window T, the mean and variance of the empirical aggregate rate for intervals of length I_k can be computed as

$$\bar{C}_k = \frac{1}{M_k} \sum_{j=1}^{M_k} C_{k,j} \tag{3.5}$$

and

$$CV_k = \frac{1}{M_k} \sum_{j=1}^{M_k} (C_{k,j} - \bar{C}_k)^2 \tag{3.6}$$

where M_k is the number of measured system rate samples in time scale I_k.

Empirical Class Service Model

In the case of the *class* service rate envelope, denote the service rate measurements for class i and time scale I_k as $\vec{R}_k^{i,S} = \vec{M}_k^S$, where $\vec{M}_k^S$ is measured as explained in Section 3.2.2 for the arrival-departure sequence of class-i flow. It should be noted that $\vec{R}_k^{i,S}$ contains normalized (on intervals of length I_k, i.e., $(\frac{\vec{R}_k^{i,S} I_k}{I_k})$) samples of the service envelope $S^i(I_k)$.

Further, note that according to 3.2.2, the measured class must be backlogged in order to infer its service rate. However, the measured class does not require *other* classes to be backlogged when monitoring its service, as this information is indirectly revealed by fluctuations in its own measurements. Finally, observe that in the case of the empirical *aggregate* service model, only the first two moments of the system

rate are computed, while in the case of the *class* service model, the measurement vector $\vec{R}_k^{i,S}$ is retained. This is due to specific inference methodology, as the first two moments of the available aggregate service rate, together with first two moments of each class arrival rates are used for obtaining expected class service rate distributions for different schedulers. On the other hand, empirical *class* service rate measurements are used for detecting the scheduling discipline itself, as will in detail be explained in the following section.

3.3 Service Inference

This section explains how to use both theoretically ideal and measured envelopes as described above to characterize elements and parameters of the multi-class system. Part 3.3.1 explains the concept of empirical service rate distributions, while Parts 3.3.2 and 3.3.3 present parameter estimation and scheduler inference methodologies.

Under a particular scheduler hypothesis, this research performs Maximum Likelihood Estimations (MLEs) of the scheduler's parameters, such as guaranteed rates in WFQ and deadlines in EDF. Using the envelope's ideal description of a class' service, this thesis then develops hypothesis tests to infer which service discipline is employed by the system via statistical analysis of the empirical inter-class sharing relationships. Finally, the MLEs of the unknown parameters under the inferred scheduler are selected.

3.3.1 Empirical Service Distributions

This section describes the expected distributions of service for a given arrival distribution under different service disciplines. For simplicity, it considers a two-class

system and aggregate traffic $A^i[s, s+t]$ with a Gaussian distribution.[3] Notice that even under Gaussian arrivals, the service envelopes will be non-Gaussian due to the non-linearities of the multi-class server. For simplicity, the section first describes the expected service distributions for constant aggregate service rate and then generalize the analysis for variable service rate.

Denote X_k^i as a Gaussian random variable with mean $CI_k - \sum_{n=1}^{i-1} \bar{R}_k^{n,A} I_k$, variance $\sum_{n=1}^{i-1} RV_k^{n,A} {I_k}^2$ and probability density function $p_{X_k^i}(x)$.

$$X_k^i \sim N\left(CI_k - \sum_{n=1}^{i-1} \bar{R}_k^{n,A} I_k, \sum_{n=1}^{i-1} RV_k^{n,A} {I_k}^2\right).$$

From Equation (2.1), the probability density function of the service envelope $S_k^i = S^i(I_k)$ under the hypothesis that the server is SP is given by

$$p_{S_k^i}^{SP}(x) = P(X_k^i \leq \phi_i CI_k)\delta(x - \phi_i CI_k) + p_{X_k^i}(x) I\left(\phi_i CI_k \leq x \leq CI_k\right) + P(X_k^i \geq CI_k)\delta(x - CI_k), \tag{3.7}$$

where $I(\cdot)$ is an indicator function and $\delta(\cdot)$ is a delta function.

Similarly, denote Y_k^i as a Gaussian random variable with mean $CI_k - \sum_{n \neq i} \bar{R}_k^{n,A} I_k$, variance $\sum_{n \neq i} RV_k^{n,A} {I_k}^2$ and probability density function $p_{Y_k^i}(y)$,

$$Y_k^i \sim N\left(CI_k - \sum_{n \neq i} \bar{R}_k^{n,A} I_k, \sum_{n \neq i} RV_k^{n,A} {I_k}^2\right).$$

[3] The motivation behind the Gaussian traffic characterization is that it is very simple and accurate when a large number of sources are multiplexed (via the Central Limit Theorem). In fact, it has been shown in [26] that aggregation of even a fairly small number of traffic streams is usually sufficient for the Gaussian characterization of the input process to accurately predict queue performance. However, note that the Gaussian assumption is not necessary for traffic envelopes; see [59] for example. Regardless, this chapter makes this assumption as it makes the solution more computationally efficient while also retaining a high degree of accuracy.

From Equation (2.2), the probability density function of the service envelope $S_k^i = S^i(I_k)$ under the hypothesis that the server is WFQ is given by

$$p_{S_k^i}^{WFQ}(y) = P(Y_k^i \leq \phi_i CI_k)\delta(y-\phi_i CI_k)+p_{Y_k^i}(y)I\left(\phi_i CI_k \leq y \leq CI_k\right)+P(Y_k^i \geq CI_k)\delta(y-CI_k). \tag{3.8}$$

Finally, define the random variable Z_k^i such that

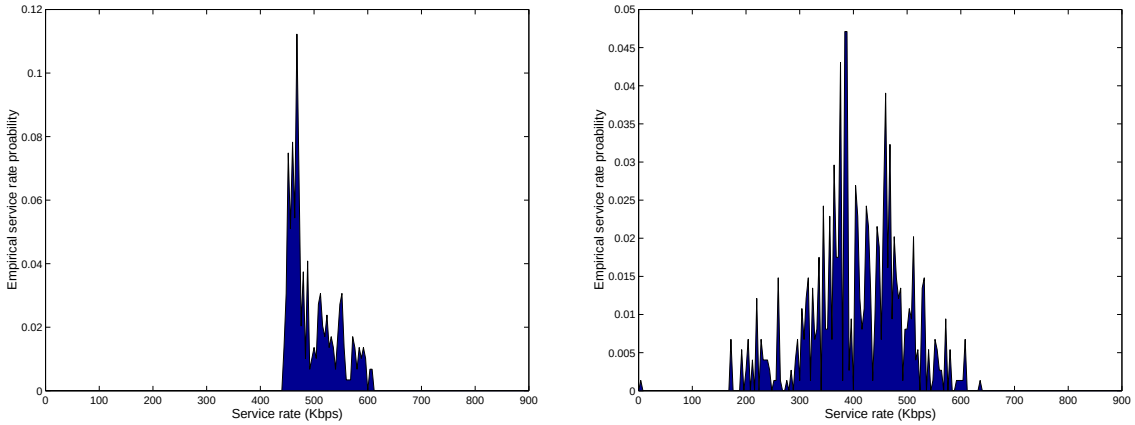
$$Z_k^i \sim N\left(CI_k + C\bar{D}_i - \sum_{n\neq i} \bar{R}_{l_n}^{n,A} I_{l_n}, \sum_{n\neq i} RV_{l_n}^{n,A} I_{l_n}^2\right).$$

Further denote the probability density function of Z_k^i by $p_{Z_k^i}(z)$, where $l_n = k-\lfloor \delta_n - \delta_i \rfloor$ and $\bar{D}_i$ is empirical *mean* delay. From the EDF service envelope of Equation (2.3), it follows that the probability density function of S_k^i under the EDF hypothesis is given by

$$p_{S_k^i}^{EDF}(z) = P(Z_k^i \leq 0)\delta(z) + p_{Z_k^i}(z)I\left(0 \leq z \leq CI_k\right) + P(Z_k^i \geq CI_k)\delta(z-CI_k). \tag{3.9}$$

Examples of empirical class service rate distributions for WFQ and SP servers are presented in Figures 3.4(a) and 3.4(b). The interval length I_k is 400 ms and additional parameters such as traffic load and statistical workload characterization are given in Section 5.4.

The observations about the figures are the following. First, the service distribution of WFQ visibly exhibits the truncated behavior defined by Equation (3.8): this is due to WFQ's guaranteed rate which lower bounds the service. Second, observe that no such "hard" lower border exists for SP without strict rate limiters on all higher priority traffic classes. Finally, notice that upper limits on the density functions are not evident here, as in this case, neither class reached its upper limits due to statistical fluctuations in the demand of the other class. Also, it should be noted



(a) WFQ Service Rate Histogram (b) SP Service Rate Histogram

Figure 3.4 : Service Rate Histograms for WFQ and SP

that the variance of arrival traffic plays a key role in revealing the scheduler type. For example, as the variance of arrivals becomes larger, according to Equation (3.8), the probability of clipping lower service bound increases. Likewise, the probability of detecting scheduler correctly increases since the service distributions for WFQ and EDF schedulers become statistically more differentiated.

Next, this thesis describes the expected distributions of a class' service for a given arrival distribution and for a given aggregate distribution. Without loss of generality, this thesis assumes Gaussian distribution[4] for C_k. Denote the probability density function of the aggregate service envelope in time scale I_k with $p_{C_k I_k}(y)$. Next, denote the probability density function of a class service envelope in the same scale, but for a given aggregate service (e.g., Equation (3.8) for WFQ), with $p_{S_k^i}^{SCH}(x|C_k I_k = y)$, where SCH denotes scheduling discipline which can be SP, WFQ or EDF. Then, the probability density function of the class service envelope is given by

[4]Observe that if the distribution is non-Gaussian, the pdf of the aggregate service could be estimated and used in Equation (3.10).

$$\tilde{p}^{SCH}_{S^i_k}(x) = \int_0^\infty p_{C_k I_k}(y) p^{SCH}_{S^i_k}(x|C_k I_k = y) dy \tag{3.10}$$

Observe that when the aggregate service rate is constant, i.e., when $p_{C_k I_k}(y) = \delta(y - CI_k)$, then $\tilde{p}^{SCH}_{S^i_k}(x) = p^{SCH}_{S^i_k}(x|y = CI_k)$, which is a special case treated above.

3.3.2 Parameter Estimation Under Scheduler Hypothesis

This section describe how a scheduler's parameters such as weights in WFQ and deadlines in EDF can be estimated under the hypothesis of a particular scheduler EDF, WFQ, or SP. THis research employs the Generalized Likelihood Ratio Test by first obtaining Maximum Likelihood Estimates of unknown parameters under each hypothesis, and then using the likelihood ratio test. Next, it is shown how the scheduling mechanism itself can be inferred by choosing the more likely hypothesis as the true one. Finally, the MLEs of unknown parameters under the chosen hypothesis become the final estimates.

SP Relative Priority Estimation

The first problem is to determine unknown class' priorities under the hypothesis that server is SP. Given G classes, there are $G!$ combinations of relative class' priritizations and the goal is to find the most probable one. Thus, for $j = 1, \cdots, G!$, denote $\vec{\epsilon}_j$ as a jth priority vector corresponding to jth priority combination, e.g., $\vec{\epsilon}_1 = (1, \cdots, G)$. Given the observations of each class' service in intervals of length I_k, the MLE is used to determine the most likely priority vector $\vec{\epsilon}_j$ as

$$\hat{\vec{\epsilon}}_{j,k} = \underset{\vec{\epsilon}_{j,k}}{\operatorname{argmax}} \tilde{p}^{SP}(\vec{R}^{1,S}_k I_k, \vec{R}^{2,S}_k I_k, \cdots, \vec{R}^{G,S}_k I_k | \vec{\epsilon}_{j,k}), \tag{3.11}$$

where

$$\tilde{p}^{SP}(\vec{R}_k^{1,S} I_k, \vec{R}_k^{2,S} I_k, \cdots, \vec{R}_k^{G,S} I_k | \vec{\epsilon}_{j,k}) =$$
$$\prod_{m=1}^{M} \tilde{p}^{SP}_{S_k^1}(x = \vec{R}_{k,m}^{1,S} I_k) \prod_{n=1}^{N} \tilde{p}^{SP}_{S_k^2}(y = \vec{R}_{k,n}^{2,S} I_k) \cdots \prod_{l=1}^{L} \tilde{p}^{SP}_{S_k^G}(z = \vec{R}_{k,l}^{G,S} I_k),$$

and M, N and L denote the respective sizes of $\vec{R}_k^{1,S}$, $\vec{R}_k^{2,S}$ and $\vec{R}_k^{G,S}$. Thus, this methodology employs a numerical search over all possible priority combinations, and find the most likely one for each time scale I_k. The final solution $\vec{\epsilon}_j$ is obtained by using the majority rule over all time scales.

WFQ Relative Weight Estimation

The next problem is to determine each class' unknown weight parameter under the hypothesis that the server is WFQ. Given the observations of each class' service in intervals of length I_k, the MLE is used to estimate the unknown parameters ϕ_i as

$$(\hat{\phi}_{1,k}, \hat{\phi}_{2,k}, \cdots, \hat{\phi}_{G,k}) = \underset{(\phi_{1,k}, \phi_{2,k}, \cdots, \phi_{G,k})}{\text{argmax}} \tilde{p}^{WFQ}(\vec{R}_k^{1,S} I_k, \vec{R}_k^{2,S} I_k, \cdots \vec{R}_k^{G,S} I_k, | \phi_{1,k}, \phi_{2,k}, \cdots, \phi_{G,k}) \tag{3.12}$$

where $\tilde{p}^{WFQ}(\vec{R}_k^{1,S} I_k, \vec{R}_k^{2,S} I_k, \cdots, \vec{R}_k^{G,S} I_k | \phi_{1,k})$ is computed similarly as for the SP scenario explained above. Since a closed form expression cannot be found for the MLE in Equation (3.12), a numerical grid search is employed by maximizing the likelihood function with respect to the unknown parameters $\phi_{i,k}$ in the interval $[0, 1]$, such that $\sum_{i=1}^{G} \phi_{i,k} = 1$. (Notice that the unknown values have known and closed borders so that the grid numerical search is justified.) The estimate is obtained for each interval I_k independently, and the final estimate of $\hat{\phi}_i$ is computed by averaging the estimates for different time scales.

The physical interpretation of Equation (3.12) is as follows. The relative class weight estimation can be performed only over time intervals when all classes are backlogged since it is only during such intervals that all classes incur their lower

bounds in service. Such intervals cause peaks at the lower clipping of the service rate distribution and also maximize the joint distribution of Equation (3.12).

For EDF, similar expressions can be derived by applying the same methodology of using the EDF service envelopes to compute the MLE expressions for the class delay bounds, and performing a grid search to estimate $\hat{\delta}_i$.

Rate-Limiter Parameter Estimation

Thus far, this thesis considered work-conserving service disciplines. Here, the thesis develops a measurement methodology applicable to rate-limiters, i.e., non-work-conserving elements which limit a flow's arrivals to within a pre-specified constraint. For a single token bucket with a bucket depth of one packet, the rate limiter for class i is characterized by an unknown rate r_i. The key problem is to distinguish such a limit on class i's service from throughput limits due to the workloads of other traffic classes and other mechanisms in the multi-class scheduler.

Thus, the goal is to find the maximum likelihood estimation of r_i under the hypothesis of a particular scheduler (inferred as above). With rate limiters, the service envelopes of Equations (2.1), (2.2) and (2.3) have r^i in place of C as the maximum service rate. Thus considering the EDF hypothesis as an example, the maximum likelihood estimation of r_i can be computed as

$$(\hat{r}_k^i, \hat{\delta}_i) = \underset{r_k^i, \delta_i}{\operatorname{argmax}} \tilde{p}^{EDF}(\vec{R}_k^{1,S} I_k, \vec{R}_k^{2,S} I_k, \cdots, \vec{R}_k^{G,S} I_k | r_k^i, \delta_i). \tag{3.13}$$

Estimation of rate limiter parameters highlights the importance of time scales. This is illustrated in Figure 3.5, which depicts the probability that a class transmits at the rate limiter's bound as a function of interval length. The scenario is a two-class class-based fair queuing scheduler with class weights of 0.5. The classes have 60 and 40 exponential on-off flows with peak rate 32 kb/s. The figure shows the empirical

probability that the aggregate traffic of class 1 transmits at its rate limit of 1 Mb/s as a function of interval length. As shown, for short time scales this occurs quite frequently whereas it is increasingly rare over longer time scales. While this property is an inherent characteristic of any variable rate flow, the key point is that inference of rate limiter parameters at long time scales is inhibited by flows becoming less and less likely to send at peak rates for sustained periods. As a consequence, measurement of multi-level leaky buckets, which *require* longer time scale measurements due to traffic constraint functions which shape the traffic differently at different time scales (see [60] for example) will incur higher measurement errors.

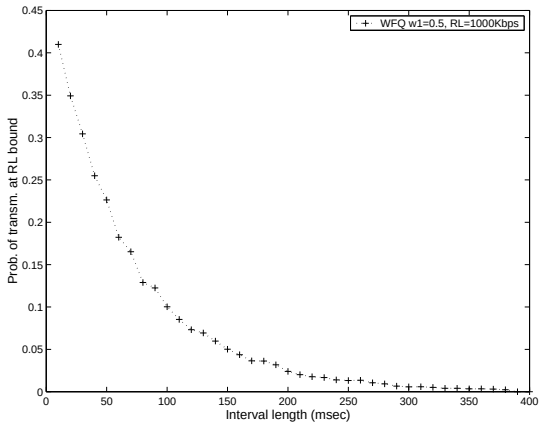


Figure 3.5 : Probability of Transmitting at Rate Limiter Bound

3.3.3 Scheduler Inference

The above technique allows estimation of a scheduler's parameters under the hypothesis of particular scheduler. Here, the thesis shows how the scheduling policy itself can be inferred. The key technique is to choose the hypothesis that makes the measured service observation most likely. It highlights the importance of variability of both class arrival and aggregate service envelopes and the crucial role of time scales.

To infer which service discipline is the most likely under the observations, this work applies Generalized Likelihood Ratio Test (GLRT), a detection method in which estimated unknown parameters are used in the likelihood ratio test. Thus, for each time scale I_k, the scheduler hypothesis test is given by

$$\frac{\tilde{p}^{EDF}(\vec{R}_k^{1,S} I_k)\tilde{p}^{EDF}(\vec{R}_k^{2,S} I_k)\cdots\tilde{p}^{EDF}(\vec{R}_k^{G,S} I_k)}{\tilde{p}^{WFQ}(\vec{R}_k^{1,S} I_k)\tilde{p}^{WFQ}(\vec{R}_k^{2,S} I_k)\cdots\tilde{p}^{WFQ}(\vec{R}_k^{G,S} I_k)} \underset{WFQ}{\overset{EDF}{\gtrless}} 1 \tag{3.14}$$

for EDF and WFQ hypothesis. If there are more than 2 hypothesis, then similar tests are used for finding the most likely one. Since the GLRT is applied for *all* time scales I_k, and the method should provide only one final decision about the scheduler hypothesis, the next problem is to determine which time scales to consider in determining the most likely scheduling policy. As explained above, increased variability of arrivals makes the service rate distributions more statistically differentiated. In contrast, increased variability of the aggregate available service has opposite effect. An example is given in Figure 3.6, which depicts the service rate distributions in a QoS enabled web server with FCFS and WFQ scheduling policies implemented in the listen queue. The curves shown in Figure 3.6 are numerically computed using Equations (3.8) and (3.9). The interval length is 200 ms and additional simulation parameters are given in Section 5.4. Observe that due to large variance of the aggregate service rate compared to the variance of the class arrival rate (the actual ratio in the experiment is 2.1 for 200 ms time scale), no hard lower bound is observable in Figure 3.6 for WFQ. This is because the high variability of the *aggregate* service envelope directly affects the variability of *class* service envelopes according to Equation (3.10). Also, observe that the curves in Figure 3.6 are almost the same, except for the slight difference in the rates around 230 req/sec. Thus, increased variability of the *aggregate* service rate makes the inference problem harder, as the service rate distributions become statistically closer.

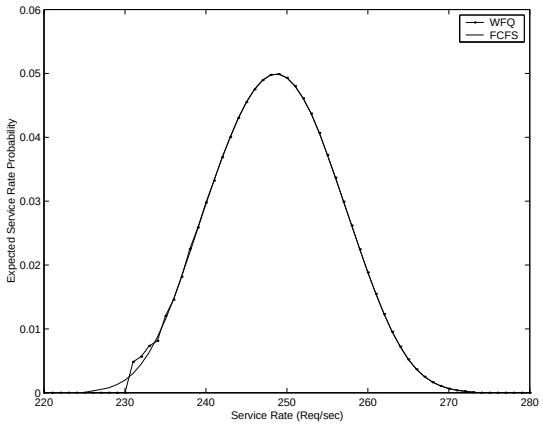


Figure 3.6 : WFQ and FCFS Expected Probability Density Functions

To include this effect in the inference procedure, define a rate variance ratio $\gamma_{k,i} = \frac{\sum_{n \neq i} RV_k^{n,A}}{CV_k}$ for each class i and time scale I_k as the measure of detection accuracy. A decision from the particular time scale I_k is included in determining the final decision if the following rate-variance condition

$$\gamma_{k,i} > \gamma^* \tag{3.15}$$

is met for a certain threshold γ^*. Thus, the methodology chooses only those time scales that have larger probability of correct service inference, i.e., time scales for which service rate distributions are statistically more differentiated. The final decision is obtained using majority rule over time scales and classes that satisfied the rate variance condition. While analytical calculation of a threshold γ^* for $\gamma_{k,i}$ that guarantees a desired probability of correct detection is intractable (because of non-linearities in expected class service distributions), this work experimentally finds the relationship between probability of correct decision and threshold using trace driven simulation in Section 5.4. This relationship can serve as a guideline for setting the threshold γ^* in practice.

The physical interpretation of the rate variance condition is as follows: in the network case, CV_k is the measure of the variability of unknown cross traffic. If the variability of the cross traffic increases, the probability to correctly detect the scheduling policy decreases. Likewise, in the web server case, if the variability of application layer service increases (e.g., due to file size distribution or caching), the probability to correctly detect the differentiated policy implemented either in the listen queue or CPU decreases, because class service measurement will be "blurred" due to this effect. Further, this illustrate challenges in providing strong capacity guarantees in systems in which it is not possible to control all the elements of the system that influence service times (i.e., file sizes in this particular case).

Observe that when $CV_k = 0$ for all k, all time-scales are included in measurements, which is exactly the case when aggregate capacity is constant. Another extreme case is when $RV_k^{n,A} = 0$, i.e., it is not possible to infer the scheduling discipline when there is no variability in arrivals.

3.3.4 The Algorithm Summary and Discussion

Table 3.7 summarizes the proposed methodology, which is divided into measurement, parameter estimation and scheduler-inference procedures.

Since most of the statistical inference techniques proposed in the parameter-estimation procedure are iterative, it may become a computational bottleneck when the number of classes G increases. However, note that the overall algorithm can be implemented in a computationally efficient way by decoupling measurement procedure on one side from parameter-estimation and scheduler-inference procedures on the other. For example, data can be collected within measurement windows of the length of several seconds (see [55] for implementation details), followed by a duration of several tens of seconds to allow computationally intensive parameter-estimation

procedure to converge.

Also, note that the Gaussian traffic characterization substantially contributes to computational efficiency. This is because Gaussian processes are completely specified by their first two moments, which makes the Gaussian traffic characterization ideal from a measurement point of view, since measuring statistics beyond the second moment is often impractical.

3.4 Experimental Investigations

This section performs a set of simulation experiments to evaluate the effectiveness of the multi-class inference tools described above. It studies WFQ weight estimation, inference of the service discipline for EDF, SP, and WFQ as well as "measurable regions", the conditions necessary to obtain accurate estimates of WFQ weights. Experiments are performed for both QoS network routers and QoS enabled web servers.

All networking simulations are performed with the *ns-2* simulator with a single router and various numbers of hosts in the topology of Figure 3.2. The link capacity is 1.5 Mb/s and packet sizes are 500 and 100 Bytes, as specified in the various experiments. The minimum interval length for measuring arrival and service envelopes is I_1 = 10 msec and the maximum interval-length for measurement is 0.5 sec for a 50-point arrival envelope. For these experiments, the measurement window T is varied in the experiments from 2 to 10 sec as indicated. The simulations consider two traffic classes and EDF, WFQ, and SP scheduling.

For the web server simulations, the simulator described in [47] (that was developed to closely approximate the behavior of OS management or CPU, memory and caching/disk storage) is modified. A simplified model of a distributed web server is depicted in Figure 3.8. The simulated server has a listen queue in which all incoming requests are queued before being serviced. Upon arrival, each request is queued onto

A. Measurement:

Denote θ as the set of all measurement time scales. For $I_k \in \theta$ compute:

1. $\bar{R}_k^{i,A}$ *and* $RV_k^{i,A}$ *for* $i = 1, \cdots, G$, *(Eq. (3.1) and Eq. (3.2)).*

2. $\vec{R}_k^{i,S}$ *for* $i = 1, \cdots, G$, *(Eq. (3.3) and Eq. (3.4)).*

3. $\bar{C}_k$ *and* CV_k *(Eq. (3.5) and Eq. (3.6)).*

B. Parameter Estimation:

4. *Determine a subset of time scales $\psi_i \subseteq \theta$ for which the rate-variance condition (Eq. (3.15)) holds, for $i = 1, \cdots, G$.*

5. SP server: *for $I_k \in \psi_i, i = 1, \cdots, G$,*

(a) *Determine $\hat{\vec{\epsilon}}_{j,k}$ (Eq. (3.11), using Eq. (3.7) and Eq. (3.10)).*

(b) *Compute* $\tilde{p}^{SP}(\vec{R}_k^{1,S} I_k, \vec{R}_k^{2,S} I_k, \cdots, \vec{R}_k^{G,S} I_k | \vec{\epsilon}_{j,k} = \hat{\vec{\epsilon}}_{j,k})$

6. WFQ server: *for $I_k \in \psi_i, i = 1, \cdots, G$,*

(a) *Determine $\hat{\phi}_{1,k}, \cdots, \hat{\phi}_{G,k}$ (Eq. (3.12), using Eq. (3.8) and Eq. (3.10)).*

(b) *Compute* $\tilde{p}^{WFQ}(\vec{R}_k^{1,S} I_k, \vec{R}_k^{2,S} I_k, \cdots, \vec{R}_k^{G,S} I_k | (\phi_{1,k}, \cdots, \phi_{G,k}) = (\hat{\phi}_{1,k}, \cdots, \hat{\phi}_{G,k}))$

7. EDF server: *for $I_k \in \psi_i, i = 1, \cdots, G$,*

(a) *Determine $\hat{\delta}_{1,k}, \cdots, \hat{\delta}_{G,k}$ (using Eq. (3.9) and Eq. (3.10)).*

(b) *Compute* $\tilde{p}^{EDF}(\vec{R}_k^{1,S} I_k, \vec{R}_k^{2,S} I_k, \cdots, \vec{R}_k^{G,S} I_k | (\delta_{1,k}, \cdots, \delta_{G,k}) = (\hat{\delta}_{1,k}, \cdots, \hat{\delta}_{G,k}))$

C. Scheduler Inference:

8. *Determine the most likely scheduler by finding $max(\tilde{p}^{SP}(.), \tilde{p}^{WFQ}(.), \tilde{p}^{EDF}(.))$, where $\tilde{p}^{SP}(.)$, $\tilde{p}^{WFQ}(.)$ and $\tilde{p}^{EDF}(.)$ are computed in 5(b), 6(b) and 7(b), respectively.*

9. *For the most likely hypothesis, determine final parameter estimates, e.g., for WFQ server $\hat{\phi}_i = \frac{\sum_{I_k \in \psi_i} \hat{\phi}_{i,k}}{\sum_{I_k \in \psi_i} 1}$, for $i = 1, \cdots, G$.*

Figure 3.7 : Summary of the Measurement/Inference Algorithm

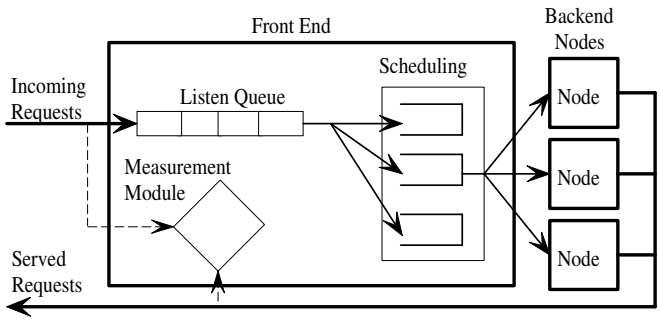


Figure 3.8 : Distributed QoS Web Server

the listen queue or dropped if the listen queue is full. Processing a request requires the following steps: dequeuing from the listen queue, connection establishment, disk reads (if needed), data transmission and finally connection tear down. A transfer time of 0.41 ms per 4 KB (resulting in the peak transfer rate of 10 MB/s) is set. WFQ scheduling in the server listen queue and CPU scheduling algorithm is implemented. CPU differentiation is implemented such that each traffic class is guaranteed its fixed share of CPU time as long as it is backlogged, while each request from the same class is given a fair share of the CPU time within that class. For example, if there are 2 requests from class 1 and two other requests from class 2, and WFQ weights are 0.7 and 0.3, then each request from class 1 is given 35% of CPU time, while each request from class 2 is given 15% of CPU time. Maximum CPU time per request is 100 ms. The trace used in simulations is generated from the CS department server log at Rice University. The inter-arrival times are simulated as exponential.

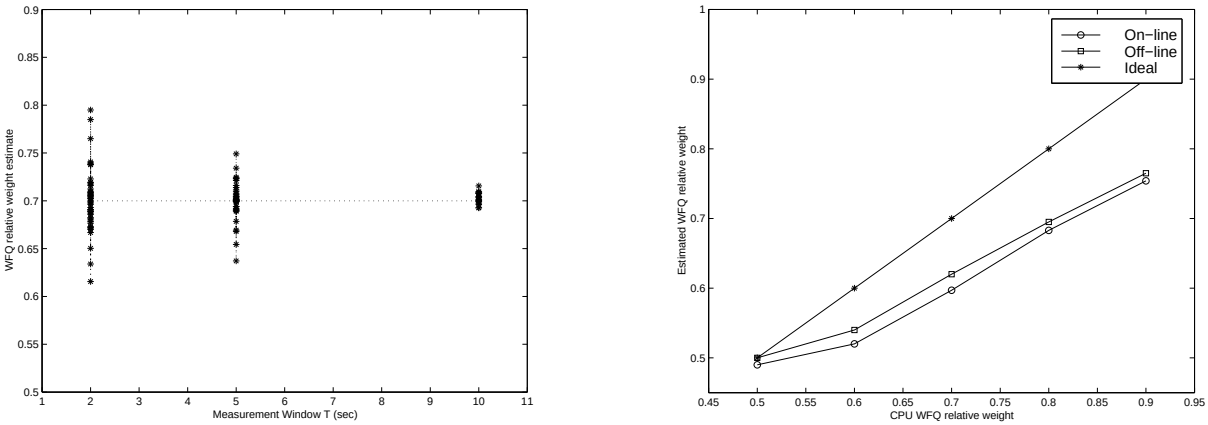
3.4.1 WFQ Weight Estimation

Network Router Experiment

This section experimentally investigates the statistical properties of the WFQ weight estimation algorithm. In this scenario, the system has from 65 to 68 exponential on-off sources with on-rate 32 kb/s and on and off periods of 0.36 sec. Moreover, there are from 25 to 28 sources of the same type for class 2. The number of flows in the system is varied to simulate flow-level arrivals and departures which are common in a real system. The true WFQ weights are $\phi_1 = 0.7$ and $\phi_2 = 0.3$. The packet size is 500 Bytes.

In the experiments, 50 simulation runs are performed corresponding to each data point in Figure 3.9(a). For a particular simulation, the measurement window T is set to 2, 5, or 10 sec as reported on the horizontal axis. Each point on the plot indicates the maximum likelihood estimation of ϕ_1, $\hat{\phi}_1$, using the methodology of Section 3.3.

First observe that the variance of the estimator reduces with increasing measurement period T, due simply to the fact that more sample points are available with larger T. This is because $\frac{T}{I_k}$ increases with T, where I_k is the length of a particular interval. For example, with $T = 2$ sec, 95% of the weight estimations are within 11% of the true value, whereas with $T = 10$ sec, 95% of the weight estimations are within 1.4% of the true value. However, T should not be set arbitrarily large, as longer-time-scale fluctuations due to flow arrivals and departures may introduce non-stationarities which would bias the tests. While the number of flows in the system did vary in these simulations, as defined above it is within a range of 5 to 10% of the system load.



(a) WFQ Weight Estimation vs. Meas. Window T (b) WFQ Weight Estimation vs. CPU Weight

Figure 3.9 : WFQ Weight Estimation in a Router and a Web-Server Scenarios

QoS Web Server Experiment

This section experimentally investigates the WFQ weight estimates in the QoS enabled web server. It estimates relative class weights and the simulation setup consists of two traffic classes for which CPU WFQ weights are varied from 0.5 to 0.9. This work further performs two types of experiments. In the first one, called on-line, the requests from two traffic classes entering and leaving the system are passively monitored. The total arrival rate is 1800 req/s, and the mean arrival rate of each class is proportional to its relative CPU weight. For the relative classes weight estimates, the inference methodology presented in this chapter is used. On the other side, this research performs another set of experiments, called off-line, where each class is artificially probed with the arrival rate of 2500 req/s, thus making the total request arrival rate as high as 5000 req/s and saturating the web server. Recall that the experiments probing at a high rate in all classes yield a true inter-class relationships - lower service bounds in this particular case. In the off-line experiments, the *mean* service rates for both classes (m_1 and m_2) is measured, and the appropriate estimator for relative

class-1 weight is $\hat{\phi}_1 = \frac{m_1}{m_1+m_2}$.

In the experiments, 10 simulation runs are performed for each WFQ weight shown on the x-axis of Figure 3.9(b) and the averaged WFQ estimates are computed using both on-line and off-line estimation procedures. First, observe that the results for the on-line experiments are just slightly biased when compared to off-line case. However, the overall results confirm the accuracy of the passive monitoring estimation methodology developed in this chapter.

Second, note that the ideal class' CPU relative weights are not reached in the on-line nor off-line experiments. For example, a CPU weighted share of 0.7 is revealed as a weighted share of 0.59 in the on-line case (consider a point with coordinates (0.7; 0.59) in the x-y plane of Figure 3.9(b)). This effect is due to a preemptive nature of CPU scheduling, i.e., the fact that the request service time does not depend only on CPU scheduling weight, but also on the number of requests present in the system and the CPU time required by the request. This example emphasizes an important feature of the inference methodology developed in this thesis - it estimates net service class parameters, as seen by users from the system edge.

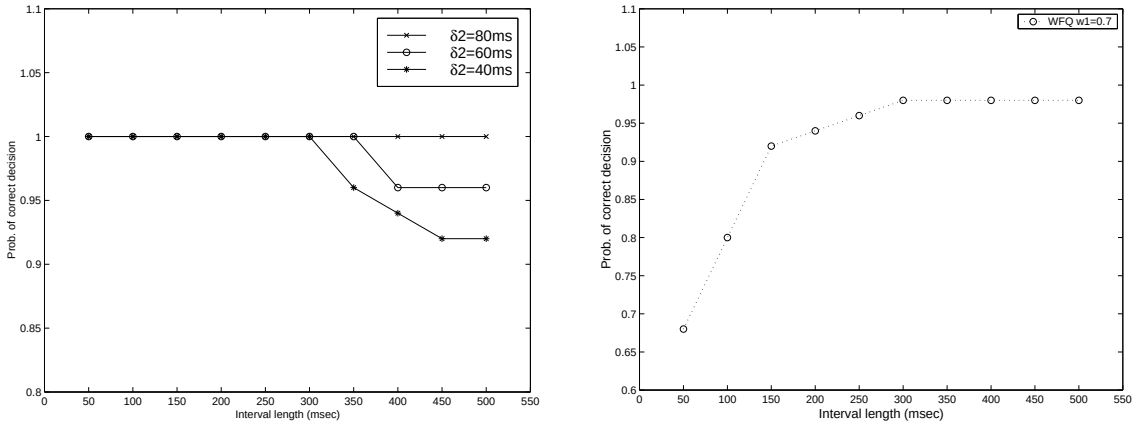
3.4.2 Scheduler Inference

As described in Section 3.3, the above WFQ weight estimations can only be performed under the hypothesis that the server is WFQ. Thus, statistical tests are necessary to infer the scheduling mechanism itself.

Constant System Capacity

This section describes simulation experiments for scheduler inference using the same number of sources for each class and the same packet size as in the previous network router experiment. Figure 3.10 depicts the experimental probability of correct

decision vs. time scale for the respective correct hypothesis of EDF and WFQ. In both cases, 50 simulations are performed and the probability of correct decision is computed as the number of correct decisions versus total number of tests for each time scale (recall the final decision is performed by majority rule).



(a) EDF (b) WFQ

Figure 3.10 : Probability of Correct Decision vs. Time Scale

For the experiments of Figure 3.10(a), the correct hypothesis is EDF with delay bounds of $\delta_1 = 20\,\text{ms}$ for class 1 and $\delta_2 = 40$, 60 and 80 ms for the three curves for class 2. As indicated in the figure, EDF is correctly inferred 100% of the time at short time scales (I_k up to 300 ms) while less frequently for longer interval lengths, especially as $\delta_2 - \delta_1$ decreases. Yet in all cases the probability of correct decision is no less than 92%. The reason that the probability of correct decision decreases as $\delta_2 - \delta_1$ decreases, is that there is less and less differentiation provided by the scheduler, making the service envelopes statistically closer, and the inference problem more difficult. Indeed, if $\delta_2 = \delta_1$, the scheduler is actually performing FCFS, as is also evident from the service envelope in Equation (2.3).

Regardless, in all cases, the correct *final* decision is made as majority rule is performed over different time scales, and incorrect decisions at a particular time scale

are never frequent enough to form a majority. Also, observe that *all* time scales are included in determining final decision, as $CV_k = 0$, i.e., rate variance condition is fulfilled for all k.

Figure 3.10(b) depicts the experimental results for WFQ. Observe that in this case, the correctness ratio is quite poor on shorter time scales. This is due to the mismatch between the fluid approximation used in the analytical model and the packet-layer simulations. In particular, over short time intervals, the fluid approximation does not hold and not every packet gets serviced at rate $\phi_i C$ (indeed, see [61] for a detailed discussion of such short-time-scale unfairness). In this case, such errors impact the final decision and the overall correctness probability is 0.94 (less than the correctness of 1 achieved in the EDF case) as the short-time-scale errors form a majority in 6% of the cases.

Finally notice that the relationship of the probability of correct decision and time scale are reversed for WFQ as compared to EDF. The reason for this is that over longer time scales, WFQ overcomes packet level unfairness and, when flows are backlogged for long durations, it can become quite clear (statistically) that there is a minimum guaranteed service rate clipping the distribution of the service envelope. In contrast, for EDF, the differences are most pronounced for small interval lengths where the shifts in the arrival envelopes (cf. Equation (2.3)) are more prominent.

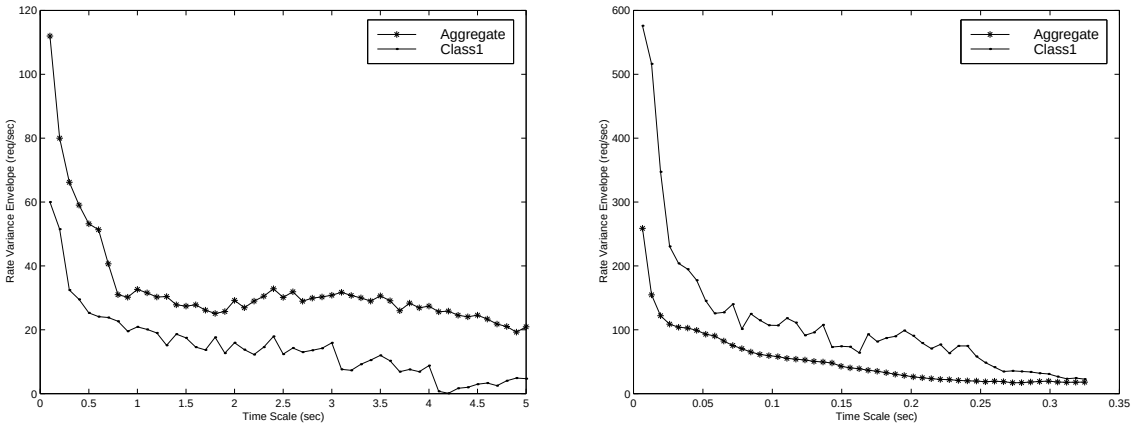
Variable System Capacity

The goal here is to explore and quantify the extent to which the system variability influences the probability of correct scheduler inference. THe thesis performs experiments with WFQ and FCFS scheduling policies[5] in a listen queue of a QoS enabled

[5]Recall that EDF scheduler with δ_i=0 performs FCFS.

web server. The simulations use traces generated from the CS department server log at Rice University. The Poisson arrival rate was 125 req/s, with arrival rate of 87 req/s and 38 req/s for classes 1 and 2 respectively, i.e., the ratio of means of arrival rates for traffic classes was 7:3. WFQ scheduler weights in the listen queue were set to 0.7 and 0.3.

Figure 3.11(a) depicts the class 1 arrival rate variance envelope and aggregate rate variance envelope for a single simulation run with the Rice CS trace. Observe that the aggregate rate variance is larger than the class arrival rate variance for all measured time scales. This is not surprising since the arrival process is Poisson, while heavy tailed file size distribution cause increased variability of serviced requests over longer time scales.



(a) Rice CS trace (b) 4K Rice CS trace

Figure 3.11 : Rate Variance Envelope vs. Measurement Interval

Recall that a rate variance ratio is defined as a measure of detection accuracy. High values of this ratio indicate high detection probability and vice versa. To validate this technique, the simulation is run 10 times for different random seeds, 5 times for each scheduler. Thus, the total number of GLRT tests is 500 (in each simulation run there are 50 GLRT tests corresponding to 50 time scales). The percentage of correct

scheduler detection averaged over all 500 tests is 0.53. Namely, it is 1.0 for FCFS (all 250 tests for FCFS scheduler were correct), while it is 0.06 for WFQ (only 15 out of 250 decisions were correct). This is because the aggregate rate variability is too high compared to class arrival variability (i.e., the rate variance ratio is too small, less than 0.6 in all cases). Thus, variations of service times due to variability of file sizes and caching dominate the inference tests, thereby overwhelming the scheduling policy implemented in the web server's listen queue and making the system to statistically appear closer to FCFS than WFQ when observed from the edge. An analogous networking example would be the one when highly bursty cross-traffic flow, which cannot be measured from the edge, interferes with the edge-measured traffic in the bottleneck router. In this case again, high variance of the aggregate service envelope would overwhelm the inference of bottleneck node's scheduling policy.

Next, to further explore and *quantify* the influence of the aggregate rate envelope variability on correct scheduler inference probability, and to be able to determine and experiment with different values for threshold γ^*, the distribution of file sizes of the trace is changed by replacing all files larger than 10 KBytes with files of 4 KBytes[6].

Figure 3.11(b) depicts the class 1 arrival rate and aggregate rate variance envelopes for a single simulation run with this changed trace. Observe that the rate arrival variance is now larger than the aggregate rate variance for all time scales, which is a direct consequence of the change of the file size distribution. For this setup, there are again 10 simulations, 5 for each scheduler.

Recall from Section 3.3 that if the rate variance ratio $\gamma_{k,i}$ is grater than the threshold γ^*, the decision from particular time scale I_k is included in determining final decision. For example, for threshold $\gamma^* = 1.0$ total number of GLRT tests is 500. Out

[6]Alternative approach in changing rate variance ratio was to increase variability of arrivals.

of these 500 tests, 492 fulfilled the condition that $\gamma_{k,i} > 1.0$ for both classes $i = 1, 2$. Further, in 305 out of this 492 tests the correct scheduler is detected. Thus, the probability of correct scheduler detection averaged over *all time scales* for which the rate variance condition is fulfilled is 0.62. However, the majority rule over those time scales that fulfilled the rate variance ratio condition gives final correctness detection probability of 0.9 (only once failed for WFQ scheduler).

The probability of correct detection increases when threshold γ^* increases. For example, when $\gamma^* = 2.0$, the per-time-scale correctness probability increases to 0.94, while the majority rule over time gives final correctness probability of 1.0. However, one should not use arbitrarily large threshold values γ^*, as the number of time scales for which the condition from Equation (3.15) is fulfilled decreases when γ^* increases. Finally, note that reduced variability of aggregate service envelope, as compared to the example from Figure 3.11(a), increases the probability of correct scheduler detection.

3.4.3 Measurable Region

The methodology presented in this chapter is based on *passive* measurements, i.e., no probing packets are transmitted to modify the system's workload. However, with passive monitoring, it is possible that other classes' particular workloads *prohibit* inference of certain network elements. For example, in the extreme case that all other classes are idle, it is impossible to detect a guaranteed minimum rate. Similarly, the multi-class nature of the scheduler itself would not be measurable, and only rate-limiter parameters could be obtained. The required workload to measure a particular network behavior is referred to as the *measurable region*.

Next, the thesis addresses the issue of the conditions necessary to infer lower and upper service limits for WFQ schedulers. For the simulations, each flow has on and off periods of 0.36 sec and on-rate 32 kb/s. The packet size is 100 Bytes.

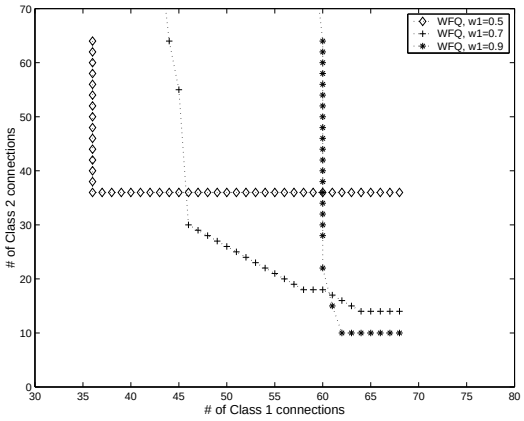


Figure 3.12 : Measurable Region for Lower Service Bounds

Figure 3.12 depicts the resulting measurable regions for WFQ weight estimates. Each point represents the minimum number of class 1 and class 2 flows needed such that the relative weights can be estimated within 5% of their correct value. In other words, these curves represent the borders between measurable and non-measurable regions. That is, if either class has fewer flows than indicated by this measurable region, then estimation is not possible, as the conditions required for weight estimation occur too rarely. Similarly, the scheduler inference correctness probability (not shown in the figure) sharply decreases when the number of flows in the system drops below the measurable region minimum.

Observe that as the weight of class 1 increases from ϕ_1 of 0.5 to 0.7 and 0.9 (corresponding to the three curves), the curves shift to the lower right indicating that a higher number of class 1 flows and lower number of class 2 flows are needed to infer ϕ_1. The reason for this is that as ϕ_1 becomes larger, a higher traffic load in class 1 is required to backlog class 1 sufficiently to estimate the guaranteed rate.

Finally, observe that a typical point on the curve refers to a relatively modest resource utilization. For example, under $\phi_1 = 0.7$, at least 30 class 2 flows are required

when 46 class 1 flows are present. This corresponds to an average system utilization of 62%, i.e., the mean utilization must be at least 62% to perform the measurements passively, otherwise active probing is required.

3.5 Summary

This chapter developed a scheme for clients to perform a series of hypothesis tests across multiple time scales in order to infer the request service discipline among class-based weighted fair queuing, earliest deadline first, and strict priority. These inferences significantly enhance the network monitoring and service validation capabilities and provide vital information for making efficient use of resources. The next chapter will show how end-point inferences of available bandwidth can be used to achieve a two-level service differentiation policy, yet without any support from the network.

Chapter 4

Low-Priority Service via End-Point Congestion Control

This chapter devises TCP-LP (Low Priority), an end-point protocol that achieves two-class service prioritization without any support from the network. The key observation is that end-to-end differentiation can be achieved by having different end-host applications employ different congestion control algorithms as dictated by their performance objectives. Since TCP is the dominant protocol for best-effort traffic, this thesis designs TCP-LP to realize a low-priority service as compared to the existing best effort service. Namely, the objective is for TCP-LP flows to utilize the bandwidth left unused by TCP flows in a non-intrusive, or TCP-transparent, fashion. Moreover, TCP-LP is a distributed algorithm that is realized as a sender-side modification of the TCP protocol.

One class of applications of TCP-LP is low-priority file transfer over the Internet. For network clients on low-speed access links, TCP-LP provides a mechanism to retain faster response times for interactive applications using TCP, while simultaneously making progress on background file transfers using TCP-LP. Similarly, in enterprise networks, TCP-LP enables large file backups to proceed without impeding interactive applications, a functionality that would otherwise require a multi-priority or separate network. Finally, institutions often rate-limit certain applications (e.g., peer-to-peer file sharing applications) such that they do not degrade the performance of other applications. In contrast, TCP-LP allows low priority applications to use all excess

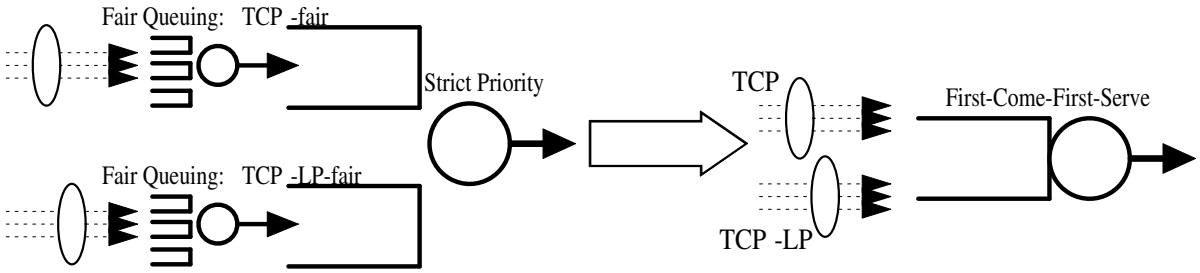
capacity while also remaining transparent to TCP flows.

A second class of applications of TCP-LP is inference of available bandwidth for network monitoring, end-point admission control [44], and performance optimization (e.g., to select a mirror server with the highest available bandwidth). Current techniques (e.g., [54, 62, 63]) estimate available bandwidth by making statistical inferences on measurements of the delay or loss characteristics of a sequence of transmitted probe packets. In contrast, TCP-LP is algorithmic with the goal of transmitting at the rate of the available bandwidth. Consequently, competing TCP-LP flows obtain their *fair share* of the available bandwidth, as opposed to probing flows which infer the *total* available bandwidth, overestimating the fraction actually available individually when many flows are simultaneously probing. Moreover, as the available bandwidth changes over time, TCP-LP provides a mechanism to continuously adapt to changing network conditions.

4.1 Reference Model and Design Objectives

The objective of TCP-LP is to use excess network bandwidth left unutilized by non TCP-LP flows thereby making TCP-LP flows transparent to TCP and UDP flows. This design objective is formalized in Figure 4.1(a) which depicts a two-class hierarchical scheduling model (see [64]) that achieves the idealized system functionality. In the reference system, there is a high-priority and low-priority class, with the former obtaining strict priority service over the latter. Within each class, service is fair among competing flow-controlled flows. As networks do not typically employ such scheduling mechanisms, the objective of TCP-LP is to obtain an approximation to the reference model's behavior via an end-point congestion control algorithm. As depicted in Figure 4.1(b), in the actual system, all flows (high and low priority) are multiplexed into a single first-come-first-serve queue and service approximating that

of the reference model is obtained via the use of two different congestion control protocols, TCP and TCP-LP. In other words, TCP flows should obtain strict priority service over TCP-LP flows, and competing TCP-LP flows should each obtain a fair bandwidth share compared to other TCP-LP flows.[1]



(a) Reference Model (b) TCP-LP Realization

Figure 4.1 : Reference Model and TCP-LP Realization

To further illustrate, consider again the system shown in Figure 4.1(b). Denote C as the link capacity, D as the aggregate rate demanded by all non-TCP-LP flows (high priority), and n as the number of TCP-LP flows in the system, with all TCP-LP flows having infinite demand and identical round trip times. Since the excess network bandwidth is $(C-D)^+$, the goal is for each TCP-LP flow to utilize bandwidth given by $(C-D)^+/n$.

4.2 TCP-LP Protocol: Mechanisms and Deployment

This section develops TCP-LP, a low-priority congestion control protocol that uses the excess bandwidth on an end-to-end path, versus the fair-rate utilized by TCP. It first devises a mechanism for early congestion indication via inferences of one-way packet

[1] As UDP flows are non-responsive, they would also be considered high priority and multiplexed with the TCP flows.

delays. Next, it presents TCP-LP's congestion avoidance policy to exploit available bandwidth while being sensitive to early congestion indicators. We then develop a simple queueing model to study the feasibility of TCP-transparent congestion control under heterogeneous round trip times. Finally, this section provides guidelines for TCP-LP parameter settings.

4.2.1 Early Congestion Indication

To achieve low priority service in the presence of TCP traffic, it is necessary for TCP-LP to infer congestion earlier than TCP. In principle, the network could provide such early congestion indicators. For example, TCP-LP flows could use a type-of-service bit to indicate low priority, and routers could use Early Congestion Notification (ECN) messages [65] to inform TCP-LP flows of lesser congestion levels than TCP flows. However, given the absence of such network support, this thesis devises an endpoint realization of this functionality by using packet delays as early indicators for TCP-LP, as compared to packet drops used by TCP. In this way, TCP-LP and TCP implicitly coordinate in a distributed manner to provide the desired priority levels.

Delay Threshold

TCP-LP measures one-way packet delays and employs a simple delay threshold-based method for early inference of congestion. Denote d_i as the one-way delay of the packet with sequence number i, and d_{min} and d_{max} as the minimum and maximum one-way packet delays experienced throughout the connection's lifetime.[2] Thus, d_{min} is an estimate of the one-way propagation delay and $d_{max} - d_{min}$ is an estimate of the maximum queueing delay.

[2]Minimum and maximum one-way packet delays are initially estimated during the slow-start phase and are used after the first packet loss, i.e., in the congestion avoidance phase.

Next, denote γ as the delay smoothing parameter, and sd_i as the smoothed one-way delay. An exponentially weighted moving average is computed as

$$sd_i = (1 - \gamma)sd_{i-1} + \gamma d_i. \tag{4.1}$$

An early indication of congestion is inferred by a TCP-LP flow whenever the smoothed one-way delay exceeds a threshold within the range of the minimum and maximum delay. In other words, the early congestion indication condition is

$$sd_i > d_{min} + (d_{max} - d_{min})\delta. \tag{4.2}$$

where $0 < \delta < 1$ denotes the threshold parameter (the setting of parameters δ and γ is discussed in detail in Section 4.2.4). Thus, analogous to the way ECN uses increasing queue sizes to alert flows of congestion before loss occurs, the above scheme infers forthcoming congestion from the end points' delay measurements so that TCP-LP flows can be non-intrusive to TCP flows.

Delay Measurement

TCP-LP obtains samples of one-way packet delays using the TCP timestamp option [66]. Each TCP packet carries two four-byte timestamp fields. A TCP-LP sender timestamps one of these fields with its current clock value when it sends a data packet. On the other side, the receiver echoes back this timestamp value and in addition timestamps the ACK packet with its own current time. In this way, the TCP-LP sender measures one-way packet delays. Note that the sender and receiver clocks do not have to be synchronized since only the relative time difference matters. Moreover, a drift between the two clocks is not significant here as resets of d_{min} and d_{max} on time-scales of minutes can be applied [67]. Finally, note that by using *one-way* packet delay measurements instead of round-trip times, cross-traffic in the reverse direction does not influence TCP-LP's inference of early congestion.

4.2.2 Congestion Avoidance Policy

Objectives

TCP-LP is an end-point algorithm that aims to emulate the functionality of the reference-scheduling model depicted in Figure 4.1. Consider for simplicity a scenario with one TCP-LP and one TCP flow. The reference strict priority scheduler serves TCP-LP packets only when there are no TCP packets in the system. However, whenever TCP packets arrive, the scheduler immediately begins service of higher priority TCP packets.

Similarly, after serving the last packet from the TCP class, the strict priority scheduler immediately starts serving TCP-LP packets. Note that it is impossible to exactly achieve this behavior from the network endpoints as TCP-LP operates on time-scales of round-trip times, while the reference scheduling model operates on time-scales of packet transmission times. Thus, the goal is to develop a congestion control policy that is able to *approximate* the desired dynamic behavior.

Reacting to Early Congestion Indicators

TCP-LP must react quickly to early congestion indicators to achieve TCP-transparency. However, simply decreasing the congestion window promptly to zero packets after the receipt of an early congestion indication (as implied by the reference scheduling model) unnecessarily inhibits the throughput of TCP-LP flows. This is because a single early congestion indication cannot be considered as a reliable indication of network congestion given the complex dynamics of cross traffic. On the other hand, halving the congestion window of TCP-LP flows upon the congestion indication, as recommended for ECN flows [68], would result in too slow a response to achieve TCP transparency.

To compromise between the two extremes, TCP-LP employs the following algorithm. After receipt of the initial early congestion indication, TCP-LP halves its congestion window and enters an *inference phase* by starting an *inference time-out timer*. During this inference period, TCP-LP only observes responses from the network, without increasing its congestion window. If it receives another early congestion indication before the inference timer expires, this indicates the activity of cross traffic, and TCP-LP decreases its congestion window to one packet. Thus, with persistent congestion, it takes two round-trip times for a TCP-LP flow to decrease its window to 1. Otherwise, after expiration of the inference timer, TCP-LP enters the additive-increase congestion avoidance phase and increases its congestion window by one per round-trip time (as with TCP flows in this phase).

As with router-assisted early congestion indication [68], consecutive packets from the same flow often experience similar network congestion state. Consequently, as suggested for ECN flows, TCP-LP also reacts to a congestion indication event at most once per round-trip time. Thus, in order to prevent TCP-LP from over-reacting to bursts of congestion indicated packets, TCP-LP ignores succeeding congestion indications if the source has reacted to a previous delay-based congestion indication or to a dropped packet in the last round-trip time.

Finally, the minimum congestion window for TCP-LP flows in the inference phase is set to 1. In this way, TCP-LP flows conservatively ensure that an excess bandwidth of at least one packet per round-trip time is available before probing for additional bandwidth.

Pseudo Code

Figure 4.2 shows the pseudo code for TCP-LP's congestion avoidance policy. Denote *cwnd* as congestion window size and *itti* as the inference time-out timer state indica-

Variables

new-ACK: indication that ACK packet has arrived

cong_ind: congestion indication

itti: inference time-out timer indication

cwnd: congestion window

Pseudocode

```
1.    if (new_ACK == 1)
2.      if (cong_ind == 1)
3.        if (itti == 1)
4.          cwnd = 1;
5.        else
6.          cwnd = cwnd/2;
7.        endif
8.        itt = 1;
9.      else
10.       if (itti != 1)
11.         cwnd += 1/cwnd;
12.       endif
13.     endif
14.   endif
```

Figure 4.2 : TCP-LP Congestion Avoidance Policy

tor. It is set to one when the timer is initiated and to zero when the timer expires. Further, Figure 4.3 illustrates a schematic view of TCP-LP's congestion window behavior at different stages, where points on the top mark early congestion indications and the inference timer period is labeled *itt*. For example, with the first early congestion indicator, this flow enters the inference phase. It later successfully exits the inference phase into additive increase as no further early congestion indicators occur. On the other hand, the second early congestion indicator is followed by a second indicator within the inference phase such that the congestion window is subsequently set to one.

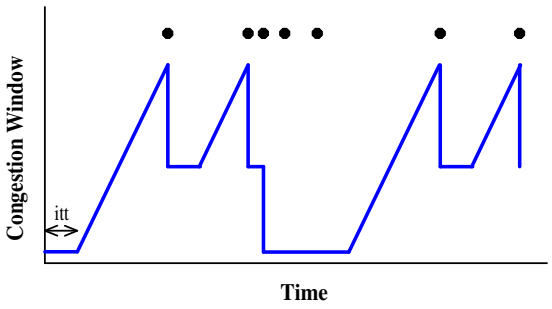


Figure 4.3 : Behavior of TCP-LP Congestion Avoidance Phase

Preserving TCP-Transparency in Large Aggregation Regimes

A key goal of TCP-LP is to achieve non-intrusiveness to TCP flows. Thus, as explained above, TCP-LP reduces its window size to one packet per RTT in the presence of TCP flows. However, in scenarios with many TCP-LP flows, it becomes increasingly possible for TCP-LP aggregates to impact TCP flows. For example, consider a scenario with a hundred TCP-LP flows competing with TCP flows on a 10 Mb/s link. If the round-trip time of the TCP-LP flows is 100 ms and the packet size is 1500 Bytes, then this TCP-LP aggregate utilizes 12% of the bandwidth, despite the

fact that each flow sends only a single packet per RTT.[3] To mitigate this problem, TCP-LP decreases the packet size to 64 Bytes whenever the window size drops below 5 packets. In this way, TCP-LP significantly decreases its impact on TCP flows in high-aggregation regimes, yet it is still able to quickly react (after RTT) to changes in congestion. In the above example, a hundred TCP-LP flows would then utilize only 0.5% of the bandwidth in the presence of TCP flows.

4.2.3 Modeling TCP and TCP-LP Interactions

As described above, TCP-LP must detect congestion earlier than TCP. However, in a heterogeneous networking environment, different flows can have different round-trip times ranging from several *msec* to several *sec*. The thesis here explores to what extent TCP-LP flows with large round-trip times can still infer congestion prior to TCP flows with smaller round-trip times. Such behavior is required such that TCP-LP flows with large round-trip times can still utilize excess network bandwidth without hindering TCP flows with small round-trip times.

The approach is to develop a simple queueing model that characterizes TCP-LP's non-intrusiveness in the presence of TCP cross-traffic, and quantifies it with respect to the threshold parameter δ. The model, illustrated in Figure 4.4, consists of a bottleneck queue with capacity C driven by traffic from one TCP-LP connection with round-trip time rtt_l. Moreover, the queue services (high priority) TCP cross traffic with round-trip time denoted by rtt_h. For simplicity, the cross traffic is also modeled as originating from a single TCP connection.

Denoting the queue's total buffer space by Q, the early congestion indication condition is satisfied whenever the queue length is greater than $Q\delta$ packets, which

[3]The effects of the exponential-backoff phase (that may actually decrease this percentage) are being disregarded.

is equivalent to condition (4.2) with $\gamma = 1$ in this idealistic scenario. Further consider that without congestion, the two flows are increasing their rates linearly with constants α_l and α_h packets per second respectively.[4]

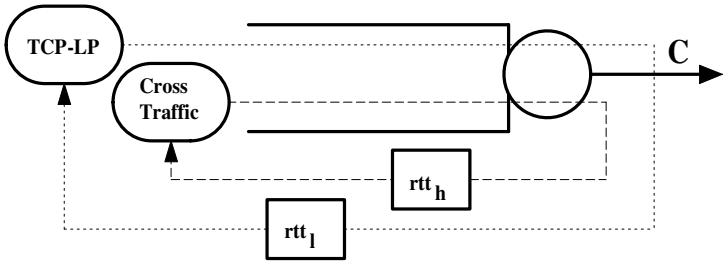


Figure 4.4 : Simplified Model of Heterogeneous RTT Effects

In such a scenario and under a fluid flow model, it is possible to quantify the conditions in which the TCP-LP flow will decrease its sending rate before the TCP cross-traffic will experience packet loss. It is assumed that the queue is initially empty and it is considered that the aggregate rate of the two flows is C at $t = 0$. Denote t_l and t_h as the respective times when the TCP-LP and TCP cross-traffic flow determine that the queue is congested. For TCP-LP, this time is given by the solution to

$$Q\delta = \int_0^{t_l} (C + (\alpha_l/rtt_l + \alpha_h/rtt_h)t - C)dt, \tag{4.3}$$

so that $t_l = \sqrt{\frac{2Q\delta}{\alpha_l/rtt_l+\alpha_h/rtt_h}}$. Similarly, $t_h = \sqrt{\frac{2Q}{\alpha_l/rtt_l+\alpha_h/rtt_h}}$. In Equation (4.3), the term $C + (\alpha_l/rtt_l + \alpha_h/rtt_h)t$ denotes the instantaneous arrival rate of the two flows at time t, whereas C denotes the service rate. For the TCP-LP flow to decrease its rate before the cross traffic experiences packet loss, it is necessary that $t_l + rtt_l < t_h$, which is equivalent to

[4]An increase in congestion window of α packets is considered to be equal to an increase in bandwidth of α packets per second.

$$rtt_l < \sqrt{\frac{2Q}{\alpha_l/rtt_l + \alpha_h/rtt_h}}(1 - \sqrt{\delta}). \tag{4.4}$$

To interpret this result, consider that $\alpha_l/rtt_l = n\alpha_h/rtt_h$. For $\alpha_l = \alpha_h$, this means that the TCP-LP flow's round-trip time is n times larger than the competing TCP flow's round-trip. In this case, the above condition is equivalent to

$$n(n+1) < \frac{\alpha_h}{\alpha_l^2 rtt_h} 2Q(1 - \sqrt{\delta})^2. \tag{4.5}$$

Inequality (4.5) gives an upper bound on n as a function of the cross traffic's round-trip time rtt_h, the queue size Q (in packets) and the delay threshold δ. To interpret this result, consider a typical queue size of $Q = 2.5Crtt_h$ and increase parameters $\alpha_l = \alpha_h = 1\,\text{packet/RTT}$. With the approximation that $n(n+1) \approx n^2$, it follows that

$$n < \sqrt{5C}(1 - \sqrt{\delta}). \tag{4.6}$$

Figure 4.5 depicts the relationship between the ratios of the round-trip times n and the delay threshold δ for capacity $C = 1.5\,\text{Mb/s}$ and average packet size of 1 kB. Observe that TCP-LP's responsiveness rapidly decreases with increasing delay threshold δ. Moreover, the figure indicates TCP-LP's potential to achieve TCP transparency. For example, the point (0.4, 11.25) shows that with delay threshold $\delta = 0.4$, a single TCP-LP connection infers congestion before the competing TCP incurs loss, even if the TCP-LP flow's round-trip time is 11 times larger than that of the TCP flow. Similar conclusions can be drawn from Equation (4.5) for $rtt_l = rtt_h$ and $\alpha_l \neq \alpha_h$.

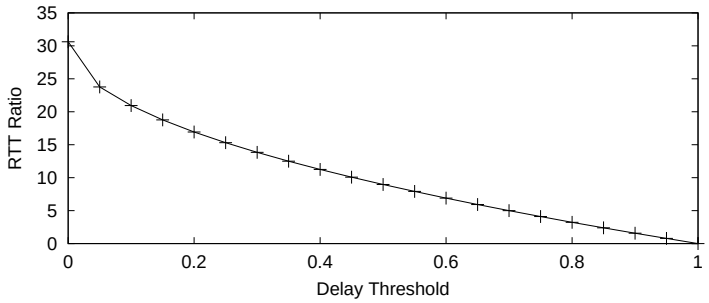


Figure 4.5 : Relationship between the RTT Ratio n and Threshold δ

4.2.4 Guidelines for Parameter Settings

This section proposes guidelines for setting TCP-LP's parameters given that the receipt of a single packet whose smoothed one-way delay is greater than a prespecified threshold serves as an early notification of congestion to a TCP-LP flow.

Delay Smoothing γ

First, consider the delay smoothing parameter γ of Equation (4.1). With large variations in network delay due to bursty cross traffic, smoothing one-way packet delays is essential for preventing false early congestion indications. On the other hand smoothing over excessively long time intervals (corresponding to small vales for γ) can substantially degrade TCP-LP's ability to detect congestion in its early stages. To balance these two requirements, TCP-LP uses smoothing parameter $\gamma = 1/8$, the value typically used for computing the smoothed round-trip time for TCP.

Delay Threshold δ

Next, consider the early-congestion-indication delay threshold δ of Equation (4.2). The example from Figure 4.5 illustrates the advantages of small values for the threshold δ as TCP-LP's responsiveness decreases when δ increases. However, the use of

very small thresholds can substantially degrade TCP-LP's throughput in realistic scenarios. This is because even very small (and frequent) bursts of cross-traffic can cause queueing delays on a bottleneck link. TCP-LP senses these delays from the edge, and if it uses small thresholds, frequent delay oscillations can be misinterpreted as congestion indications, even in a lightly loaded network. In turn, false early congestion indications would cause a TCP-LP flow to unnecessarily decrease its sending rate.

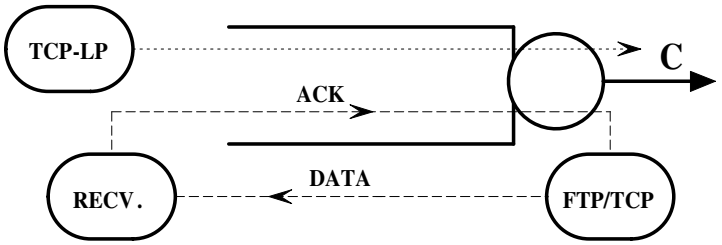


Figure 4.6 : Scenario with Reverse ACK Traffic

Thus, δ must be set to balance increased protocol responsiveness with avoiding false early congestion indications. To obtain the smallest value of δ capable of avoiding false indications, this research devises the following experiment with reverse traffic. Consider a single TCP-LP flow in a single-bottleneck scenario, where different numbers of long-lived FTP/TCP flows operate in the reverse direction, as depicted in Figure 4.6. Thus, the ACK packets of the TCP flow form a cross-traffic stream that multiplexes with TCP-LP's data traffic. The objective is to set the threshold δ such that TCP-LP's throughput does not degrade in the presence of this reference ACK stream.

Figure 4.7 depicts TCP-LP's normalized throughput for different values of the threshold parameter δ. Observe that even this low bit-rate cross-traffic reference stream, which consists solely of ACK packets, can degrade TCP-LP's throughput substantially if the threshold is set too low. For example, as depicted in Figure 4.7,

TCP-LP's throughput can drop to as low as 10% of the link bandwidth if the threshold δ is set to 0.01. However, the figure also indicates that the throughput improves with increasing δ, since for larger values of δ TCP-LP becomes non-sensitive to pure ACK bursts.

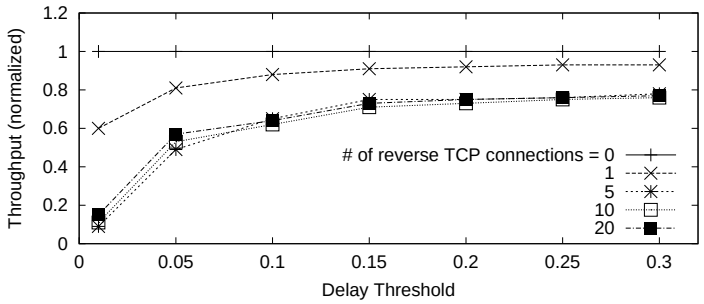


Figure 4.7 : Throughput vs. Threshold δ

Thus, while necessarily not comprehensive, the above experiment shows that setting the threshold δ to the value of 0.15 is able to accurately decouple the influence of ACK cross-traffic streams from data cross-traffic streams. In other words, while being robust in utilizing available bandwidth in the presence of pure ACK streams, TCP-LP retains its responsive nature in the presence of pure data or aggregation of data and ACK streams.[5]

Inference Time-out *itt*

Finally, a similar tradeoff between congestion-responsiveness and throughput-aggressiveness holds for the inference time-out timer parameter. With a longer inference time-out

[5]Numerous additional simulations (not shown) including scenarios with hundreds of flows, heterogeneous link capacities and multiple bottlenecks corroborate that this value represents a high performance compromise between TCP-LP's responsiveness and ability to prevent false congestion indications.

timer, TCP-LP becomes more responsive to congestion whereas a smaller inference time-out timer causes TCP-LP to switch sooner to the more aggressive additive-increase phase. To compromise between the two, this thesis sets *itt* to three round-trip times, thereby giving enough space for a TCP-LP flow to rapidly decrease its window size in periods of persistent congestion, while at the same time allowing TCP-LP to probe the network aggressively enough.

4.3 Simulation Preliminaries

This section describes TCL-LP/ECN, a benchmark algorithm that uses network ECN instead of end-point delay thresholds to infer congestion. This provides means to evaluate the early-congestion-inference aspect of TCP-LP separately from its congestion-control policy. It also presents the baseline simulation scenario and describes the "square-wave" and web-like background traffic patterns.

4.3.1 TCP-LP/ECN Benchmark Algorithm

This section describes TCP-LP/ECN, a variant of TCP-LP that uses ECN for detecting congestion instead of one-way packet delays. (Recall that one of the basic design goals is to develop an end-point protocol that is able to operate without any support from the network.) Use of router-supported early congestion indication allows us to study the effectiveness of inferences from one-way packet delay to provide early inference of congestion.

TCP-LP/ECN is simulated by modifying the implementation of RED [69] in *ns-2* as follows. First, the minimum and the maximum RED thresholds are set to the value of δQ packets. Second, the RED gateways are configured to set the ECN bit in the TCP-LP packet header when the average queue size exceeds δQ as an early indication of congestion. When a TCP-LP receiver receives a data packet with the ECN bit set

in the packet header, the receiver sets the ECN bit in the next outgoing ACK packet. On the other hand, packets belonging to TCP flows are neither marked nor dropped when the queue size exceeds δQ, and TCP packets are dropped only when the queue overflows. In this way, TCP-LP/ECN emulates the distributed TCP-LP protocol with the former using router queue measurements and the latter using end-point delay measurements.

4.3.2 Topology and Background Traffic

As a baseline topology, this research considers many flows sharing a single congested link as shown in Figure 4.8. The bandwidth of this link is either 1.5 Mb/s or 10 Mb/s and it has propagation delay 20 ms. The access links have capacity 100 Mb/s and delay 2 ms, so that the minimum round-trip time for flows is approximately 50 ms. The queue size is set to 2.5 times the delay-bandwidth product. For each data point, there are 50 simulation runs and averages are reported. Each simulation run lasts 1000 sec. The *ns-2* implementation of TCP-LP used here is derived by modifying TCP/Reno.

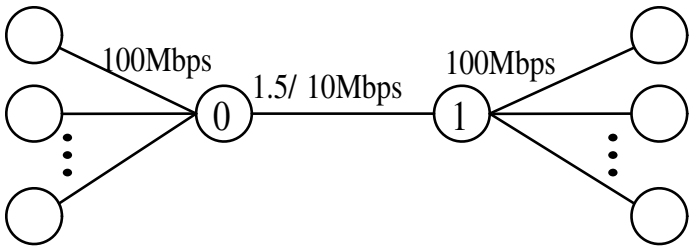


Figure 4.8 : Single Bottleneck Simulation Scenario

To explore the dynamics of TCP-LP, this work uses on-off constant-rate flows with equal on and off times, giving periodic "square-wave" patterns of available bandwidth as in reference [70]. While not representative of actual traffic patterns, this scenario is

motivated by the need to systematically explore TCP-LP's ability to utilize the excess bandwidth and to study its transparency and fairness properties in the presence of dynamic background traffic. In these experiments, the available bandwidth alternates between the full link capacity of 10 Mb/s and 3.3 Mb/s when the periodic source is idle and active respectively. The period of oscillations is changed from one to 1000 round-trip times, i.e., from 50 ms to 50 sec.

Next, to explore TCP-LP's behavior with web traffic, this research adopts the model developed in [71]. In this model, clients initiate sessions from randomly chosen web sites with several web pages downloaded from each site. Each page contains several objects, each of which requires a TCP connection for delivery (i.e., HTTP 1.0). The inter-page and inter-object time distributions are exponential with means of one sec and one msec, respectively. Each page consists of ten objects and the object size is distributed according to a Pareto distribution with shape parameter 1.2.

4.4 Simulation Experiments

This section uses simulation to evaluate the performance of TCP-LP in a variety of scenarios, including FTP, "square-wave", and HTTP background traffic patterns, with long and short-lived TCP flows and both single and multiple-bottleneck network topologies. The goal is to explore TCP-LP's behavior in both artificial and realistic network environments. This thesis evaluates TCP-LP's impact on both the throughput and delay characteristics of competing cross-traffic. Moreover, it explores TCP-LP's ability to utilize the excess network bandwidth and to achieve fairness among competing TCP-LP flows. The TCP-LP *ns* code and simulation scripts are available at *http://www.ece.rice.edu/networks/TCP-LP*.

Table 4.1 : Normalized Throughput (%)

scenario	TCP	TCP vs. TCP-LP	TCP vs. TCP-LP/ECN
no reverse TCP traffic	100	96.8 vs. 2.7	96.8 vs. 2.7
reverse TCP traffic	49.7	49.3 vs. 7.3	49.1 vs. 8

4.4.1 FTP and Reverse Background Traffic

This section first considers simultaneous FTP downloads, where one flow uses TCP-LP and the other uses TCP. The objectives are to examine to what extent TCP-LP can utilize excess bandwidth in the presence of greedy long-lived TCP traffic, and to investigate the extent to which TCP-LP flows perturb TCP traffic. In addition to this scenario, this section also presents measurements of throughput in simulations without TCP-LP consisting of one and two TCP flows. The results are summarized in the first row of Table 4.1. In this scenario, there is no excess capacity available for TCP-LP, and TCP-LP slightly perturbs the TCP flows and receives a throughput of 2.7% of the link capacity for both TCP-LP and TCP-LP/ECN.

With ten FTP/TCP flows in the reverse direction, the ACKs of the forward-direction TCP flows are delayed thereby increasing their round-trip time and ACK losses, and decreasing their throughput. Thus, excess capacity is indeed available for TCP-LP flows. In particular, the second row of Table 4.1 illustrates that the throughput of the (forward) TCP flow in this case is 49.7%. With the presence of a TCP-LP flow, the TCP flow's throughput is only marginally reduced to 49.3%, indicating that TCP-LP achieves nearly perfect TCP transparency while achieving 7.3% throughput.

Figure 4.9 depicts the temporal dynamics of this scenario and illustrates that

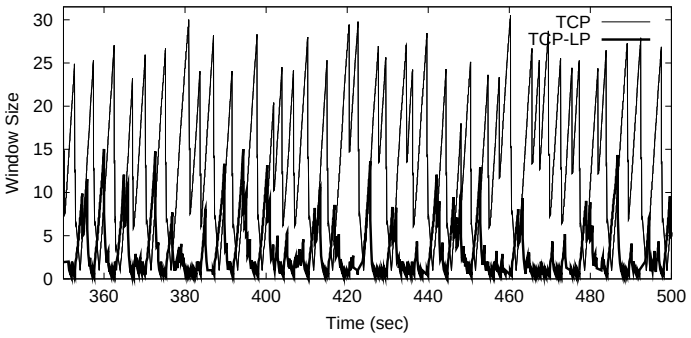


Figure 4.9 : TCP and TCP-LP's Congestion Window

TCP's congestion window widely oscillates in the range between zero and 30 packets. The window of the TCP-LP flow, also depicted, is able to track TCP's oscillation and increases its own window size when TCP's window decreases, and via early congestion inference, TCP-LP quickly backs off when the TCP flow ramps up its window size. By the time the TCP flow's window reaches its maximum of 30 packets, TCP-LP is in the inference phase, waiting for the next opportunity to utilize excess bandwidth.

4.4.2 Square-wave Background Traffic

This section next explores TCP-LP's performance in the presence of square-wave background traffic as described in Section 4.3.2.

Square Wave Period

The first experiments here investigate TCP-LP's ability to utilize excess bandwidth remaining from periodic on-off flows that transmit at constant rate when "on". Figure 4.10 depicts the bandwidth utilized by TCP, TCP-LP and TCP-LP/ECN, normalized to 6.6 Mb/s, the average excess bandwidth left unused by the square-wave background traffic. Each point in the figure represents the normalized bandwidth, utilized by the

respective protocol, for a given period of the square-wave's oscillation. For comparison, this work also depicts the normalized average available bandwidth curve.

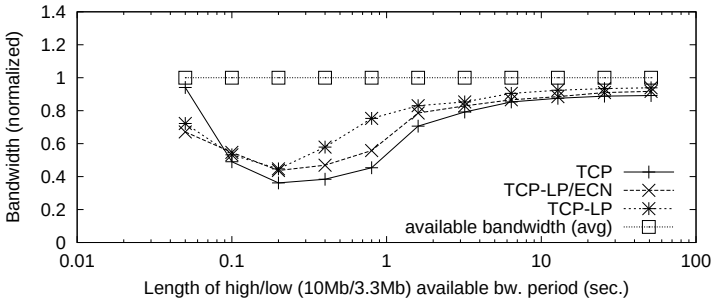


Figure 4.10 : Utilized Available Bandwidth vs. Square Wave Period

Observe that all three curves in Figure 4.10 have similar shape, and all three protocols utilize approximately only 50% of the available bandwidth when the square-wave period is too small (e.g., 0.2 seconds). Surprisingly, in this regime, both TCP-LP and TCP-LP/ECN utilize more available bandwidth than TCP. This is due to the early congestion indication and responsive congestion avoidance policy of the TCP-LP protocol, which is able to defer access to the cross-traffic bursts (from 0 to 2/3 C in this case) while avoiding entering the exponential-backoff phase.

Aggregation Level

Next, this section explore the impact of the number of flows under a fixed square wave period of 6.4 sec. Figure 4.11 illustrates that with higher levels of aggregation consisting of even 5 flows, TCP flows quickly overcome the performance problem of Figure 4.10. On the other hand, for TCP-LP utilization increases more slowly with aggregation level, as with a small number of flows, TCP-LP is not able to develop large congestion windows because it senses the existence of other competing TCP-LP flows and decreases its window accordingly. However, TCP-LP overcomes this

problem with a larger number of multiplexed flows.

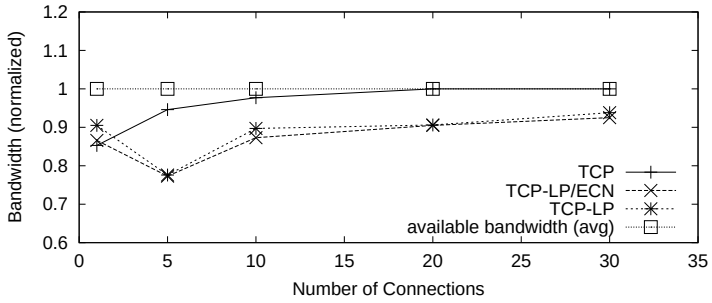


Figure 4.11 : Utilized Available Bandwidth vs. Number of Flows

Fairness

This section studies fairness among TCP-LP flows using Jain's fairness index [31]. The index, always between 0 and 1, is 1 if all flow throughputs are the same. If only k of the n users receive equal throughput and the remaining n - k users receive zero throughput, the fairness index is k/n. The experiments here include ten flows of the same type (TCP, TCP-LP or TCP-LP/ECN) that compete with the same non-responsive square wave background traffic.

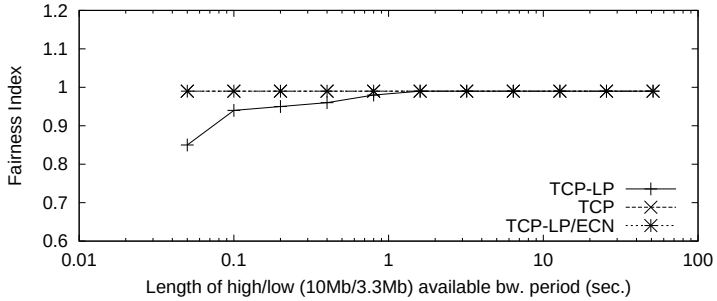


Figure 4.12 : Fairness Index vs. Square Wave Period

Figure 4.12 depicts the fairness indexes of three protocols for different periods

of square wave oscillations. First, observe that for both TCP and TCP-LP/ECN, the fairness index is approximately equal to 1 for all periods. However, TCP-LP's fairness index is slightly below one for time scales of up to 400 ms. By examining the traces, it is concluded that this originates from inaccurate estimates of the minimum and maximum delays. In most cases, one TCP-LP flow over-estimates the minimum delay value d_{min} due to wide and frequent oscillations of the background traffic. For this reason, it sends more than its fair share and the fairness index drops slightly. However, as the oscillation period increases, all flows use periods of low cross-traffic rate to accurately estimate the minimum one-way delay.

4.4.3 HTTP Background Traffic

Here, the thesis explores TCP-LP's behavior in an environment dominated by web-like transactions in the scenario described in Section 4.3.2. The performance measure of interest is the web-file retrieval (response) time, and thesis investigates TCP-LP's impact on this measure. As a standard of idealized performance, this research uses the measured retrieval times in a scenario with only web-traffic present in the system. Further, it performs multi-node experiments to study issues such as heterogeneous round-trip times and early congestion inference with multiple bottlenecks.

This thesis presents four experiments for the topology of Figure 4.8 with a link capacity of 1.5 Mb/s. In addition to web traffic between nodes zero and one, there is one FTP connection that operates in the same direction as the web-traffic. This connection is a long-lived bulk transfer and is a candidate for low-priority service. In the first three experiments, the FTP connection uses TCP-LP, TCP-LP/ECN, and TCP. Finally, to measure web-traffic response times without any cross-traffic, this thesis performs a fourth experiment in which no FTP traffic is generated. For the web transactions, it measures and averages the response times for different sized

objects.[6]

Impact on HTTP Response Times

To explore TCP-LP's impact on web traffic, this thesis compares HTTP file retrieval times with and without background TCP-LP bulk transfers.[7] Figure 4.13 depicts the averaged *difference* between the two transfer times. For example, when TCP-LP is used for a long-lived file transfer, the mean retrieval time for a 10 kB web-file is 0.49 sec. On the other hand, this retrieval time is 0.43 sec when there is no TCP-LP file transfer, hence the point (10, 0.06) in the figure. These experiments illustrate the non-intrusive aspect of TCP, as the long-lived TCP-LP bulk transfer flow only slightly increases the mean web-traffic response time, with increasing transparency achieved with larger HTTP file sizes.

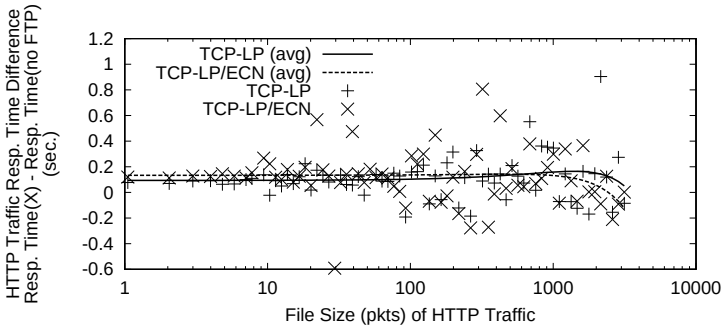


Figure 4.13 : Resp. Time Diff. (sec.) vs. File Size (kB) for HTTP Traffic

[6] As in [72], file sizes are grouped into 85 bins, each of which spans an interval [x,1.1x], and the average is taken over each bin.

[7] In the experiments, the HTTP response-time simulations use the same sample path of file sizes and think times.

Impact of High vs. Low Priority Bulk Transfer

This section next shows that if the bulk transfer flow uses TCP rather than TCP-LP, then the web response times are significantly degraded. Figure 4.14 depicts web-file response times normalized by the response times obtained when the background file transfer uses TCP. Because of this normalization, the curve labeled "TCP" in Figure 4.14 is a straight line with a value of one.

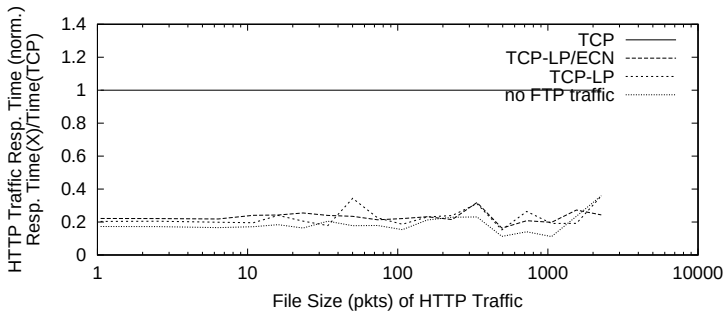


Figure 4.14 : Norm. Resp. Time vs. File Size (kB) for HTTP Traffic

Observe that use of TCP-LP for bulk data transfer reduces the web traffic response times by approximately 80% compared to TCP bulk transfer. For example, the average response time for the 10 kB file from the web-traffic stream is 2.46 sec when web traffic multiplexes with a TCP bulk-transfer background flow. This time is considerably larger than the 0.49 sec response time when TCP-LP is used for the bulk data transfer. TCP-LP's reduction in response time for web traffic occurs because without it, the TCP bulk-transfer demands its fair share of network bandwidth when competing with web-traffic. On the other hand, the bulk-transfer flow itself utilizes 61% of the bandwidth when TCP is used, only 10% more than when TCP-LP is used. This result emphasizes the benefits of low prioritization of bulk data transfers over web-traffic, which TCP-LP achieves in a distributed manner.

4.4.4 Multiple Bottlenecks

Further, this section considers a more realistic multiple bottleneck scenario using the topologies of Figures 4.15 and 4.18. In all experiments, links 0-1, 1-2 and 2-3 have capacity of 1.5 Mb/s, while all the others have capacity of 100 Mb/s.

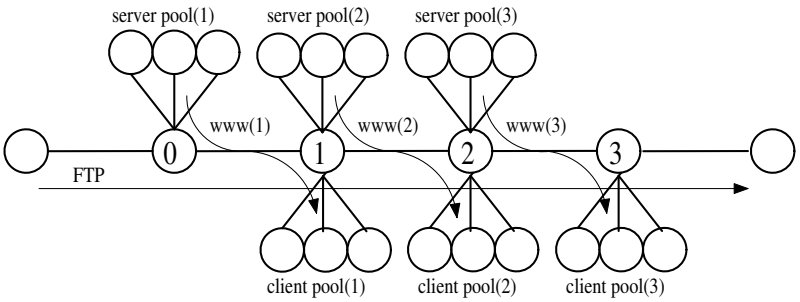


Figure 4.15 : First Topology for Multiple Bottlenecks

RTT Heterogeneity

To study TCP-LP when its round-trip time increases compared to round-trip times of competing HTTP flows, this research considers the scenario in which the bulk file-transfer flow traverses multiple bottlenecks as shown in Figure 4.15. There are three server and client pools, each of which generates cross-traffic on different bottleneck links.

Figure 4.16 depicts the averaged *difference* between HTTP file response times with and without the presence of a bulk-transfer TCP-LP flow. Observe that despite having the average round-trip time three times as large, TCP-LP retains its non-intrusiveness to the HTTP/TCP flows. This confirms the modeling result from Section 4.2.3, which states that TCP-LP flows are non-intrusive to TCP flows even if their round-trip times are much larger. Also, no substantial difference is observed be-

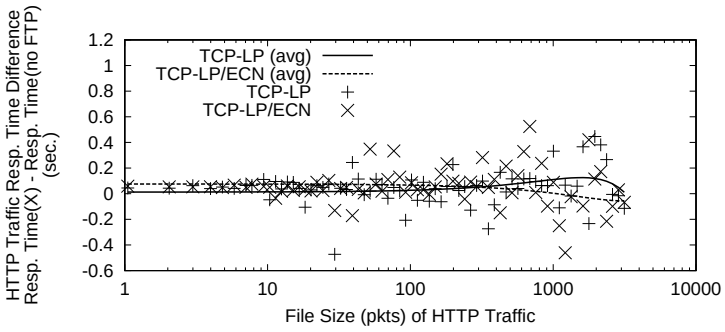


Figure 4.16 : Resp. Time Diff. (sec.) vs. File Size (kB) for HTTP Traffic

tween TCP-LP and TCP-LP/ECN, except that TCP-LP is slightly more responsive for large files.

Multi-hop Bulk Transfer

Figure 4.17 depicts the response times for different sized objects from all three pools normalized by the response times obtained when background FTP transfer uses TCP. Observe that the benefit of prioritization reported in the single bottleneck scenario still holds in this multiple-bottleneck scenario, although it is less pronounced. The difference is because the long-lived TCP flow is now less intrusive to web traffic due to its larger round-trip time.

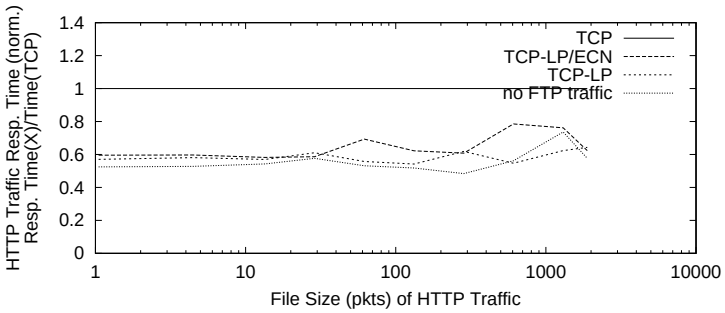


Figure 4.17 : Norm. Resp. Time vs. File Size (kB) for HTTP Traffic

Multi-hop Web Traffic

This section next considers the scenario in which web traffic traverses multiple hops and three FTP connections each traverse a single hop as depicted in Figure 4.18. Thus, the FTP flows in this scenario play the role of "fast elephants", a term for long-lived flows with short round-trip times [73].

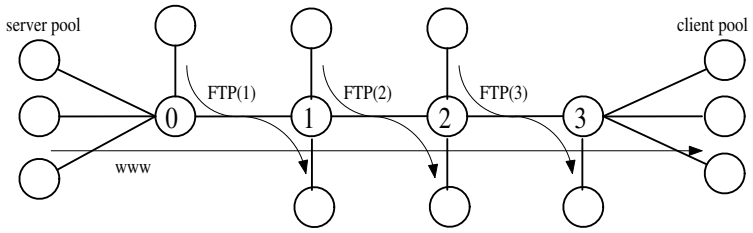


Figure 4.18 : Second Topology for Multiple Bottlenecks

Figure 4.19 depicts the averaged difference between web file response times with and without the three TCP-LP bulk transfers. In this scenario, the small TCP-LP round-trip time only improves its responsiveness and non-intrusiveness to competing web-traffic such that it becomes fully transparent to TCP. For example, the mean response time for the 10 kB file is 0.98 sec, while it is 0.74 sec in the idealized scenario when there are no FTP downloads in the system. This is revealed as the point (10, 0.24) for TCP-LP in Figure 4.19. Observe that the absolute difference in response times increases three times in this scenario when compared to the single-node scenario simply because the HTTP traffic now traverses three congested hops. However, the *per-node* impact of the bulk-transfer TCP-LP flows is approximately unchanged.

Finally, for comparison, this thesis again explores the system behavior when TCP is used for bulk data transfers. Figure 4.20 depicts the normalized response times for HTTP file retrievals. The figure indicates that "fast TCP elephants" severely impede the performance of web traffic that traverses multiple hops. For example, in this

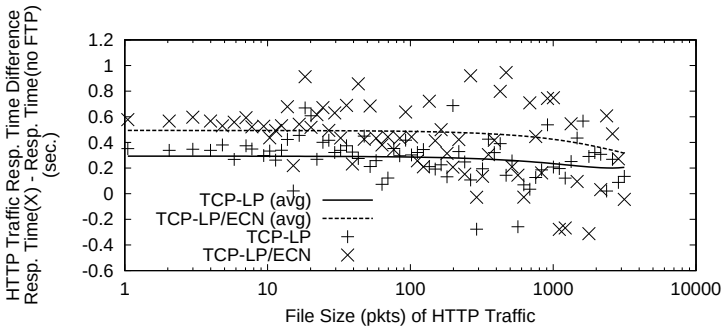


Figure 4.19 : Resp. Time Diff. (sec.) vs. File Size (kB) for HTTP Traffic

scenario, the average response time for a 10 kB file from the HTTP traffic stream is 14.27 sec.

This poor performance is because many web-traffic flows experience loss of their first packet which requires waiting for a default time-out interval of 3 sec before resending. According to the above results, each TCP flow from the web stream experiences four to five such timeout intervals on average. An interested reader can find more details on this problem in [72]. On the other hand, the results from Figure 4.20 indicate that simple two-class prioritization achieved by TCP-LP can successfully provide a desirable system behavior. While TCP-LP attains 52% of the bandwidth (10% less than TCP), it improves web-traffic response times by more than 90%.

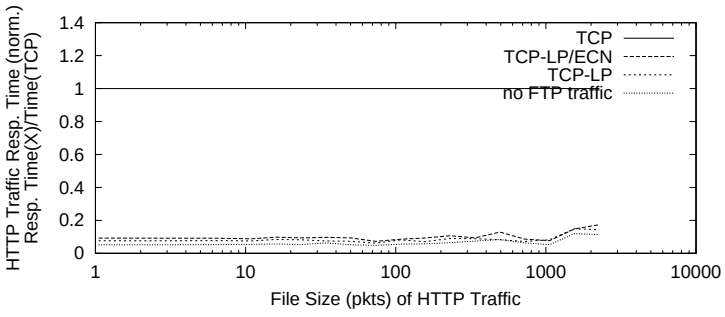


Figure 4.20 : Norm. Resp. Time vs. File Size (kB) for HTTP Traffic

Table 4.2 : Normalized TCP Throughput (%) vs. Number of Flows

Number of TCP and TCP-LP flows	1	2	5	10	15	20
Normalized TCP throughput	99.49	99.25	99.50	99.02	98.29	99.15

4.5 Protocol Implementation and Internet Experiments

This section first describes the implementation details of TCP-LP in Linux and explain the specific use of the TCP window-scaling and timestamping options from [66]. Next, it examines the protocol performance on a testbed and evaluate three important TCP-LP features. The first is non-intrusiveness to TCP traffic in large aggregation regimes; the second is the ability of both a single TCP-LP flow and aggregates to utilize the available bandwidth in the presence of highly dynamic cross traffic; the third is fairness among TCP-LP flows. Furthermore, it presents Internet measurements which are performed to evaluate the extent to which TCP-LP can utilize excess bandwidth in the presence of greedy TCP traffic. Finally, it presents TCP-LP's performance during a 24-hour period in a wide-area network and explores the impact of time-of-day effects on TCP-LP's throughput.

4.5.1 Implementation

The implementation of TCP-LP is derived by modifying the Linux-2.4.19-web100 kernel, which applies TCP Sack, and the TCP-LP source code is available at *http://www.ece.rice.edu/networks/TCP-LP*. Besides having monitoring and debugging features, the above kernel supports the window-scaling option [66], which allows the use of larger window sizes (about 1 GByte) and enables TCP-LP to fully utilize the available bandwidth. Note that many TCP stacks do not support this option by default,

such that the TCP header allocates only 16 bits for window advertisement, which limits maximum window size to 64 kBytes.

TCP-LP also uses the TCP timestamping option from [66] for the one-way delay measurements as explained in Section 4.2. In short, each TCP packet carries two four-byte timestamp fields. A TCP-LP sender timestamps one of these fields with its current clock value when it sends a data packet. On the other side, the receiver echoes back this timestamp value and in addition timestamps the ACK packet with its own current time. Thus, using these two values, the TCP-LP sender may measure one-way packet delays. However, each end-system measures time (and timestamps packet fields) in the "local unit" that corresponds to the number of clock ticks elapsed since a reference point in time. Since the clock granularity[8] may be different for the TCP-LP sender and receiver, the sender has to first estimate the receiver's clock granularity in order to use its timestamps for one-way delay measurements. The TCP-LP sender performs this task by monitoring the ACK packet's timestamp field and measuring the number of remote ticks elapsed during the one-second period after the connection establishment. Finally, the sender accurately estimates the receiver's clock granularity by choosing a value from the set of possible values (e.g., 100, 512 or 1024) that is closest to the measured number of remote clock ticks per second.

To the best of the writer's knowledge, TCP-LP is the first TCP stack that uses timestamping option for computing one-way delays. This option is originally developed to alleviate computation of round-trip times. Note that the use of one-way delays is an essential requirement for low-priority transport protocols in the Internet, as will be discussed in detail in Section 4.7. Finally, while we do not observe any problems with a drift between the sender and the receiver clocks, TCP-LP nevertheless

[8]Typically 100, 512 or 1024 ticks per second.

applies resets of d_{min} and d_{max} on three minute intervals by default.

4.5.2 Testbed Experiments

This section reports the results obtained on a testbed at Rice University. The testbed consists of two Linux clusters, as shown in Figure 4.8, with the difference that the bottleneck capacity is 100 Mb/s. Regular TCP (non-TCP-LP) flows apply the Linux-2.4.19-web100 kernel using TCP Sack.

TCP Transparency in Large Aggregation Regimes

To achieve TCP-transparent behavior in large aggregation regimes, TCP-LP reduces its window size to one packet per RTT in the presence of TCP flows and decreases packet size to 64 Bytes. Here. this thesis considers scenarios with many flows to evaluate whether TCP-LP remains non-intrusive in such regimes.

This section presents experiments with simultaneous TCP and TCP-LP file transfers where the number of flows in the system (both TCP and TCP-LP) increases from one to 20, as shown in Table 4.2. The second row of the table shows aggregate TCP throughput normalized to the throughput obtained when there are no TCP-LP flows in the network. Observe that the influence of TCP-LP flows is indeed marginal and that TCP throughput degrades by less than 1% on average. More importantly, observe that as the number of TCP-LP flows increases, the TCP throughput does not degrade. According to the theoretical computations from Section 4.2.2, one would require more than 390 TCP-LP flows in the above experiment to degrade TCP's throughput by more than 2%.

Available Bandwidth Utilization

Next, this section evaluates TCP-LP's ability to utilize the available bandwidth in the presence of extremely dynamic cross-traffic. To this end, it performs an experiment similar to the one from Section 4.4.2 with square-wave UDP cross-traffic oscillating between 0 and 2/3 of the link capacity, and where the period of oscillation changes from 50 ms up to 51.2 sec. The cross-traffic stream is generated using active probing software from [67]. Figure 4.21 depicts the bandwidth utilized by TCP-LP and TCP, normalized to the average excess bandwidth left unused by the square-wave's oscillation.

Observe that the curves in Figure 4.21 are somewhat different from the curves in Figure 4.10 since here no significant degradation of TCP and TCP-LP throughputs on shorter time-scales of the background traffic is noticed. This is due to shorter round-trip times in this scenario (the minimum RTT is 2 ms) that enable both TCP and TCP-LP flows to apply a more robust control and avoid entering the exponential-backoff phase. Furthermore, observe that again both protocols have similar behavior, with TCP-LP having slightly better performance on longer time-scales.

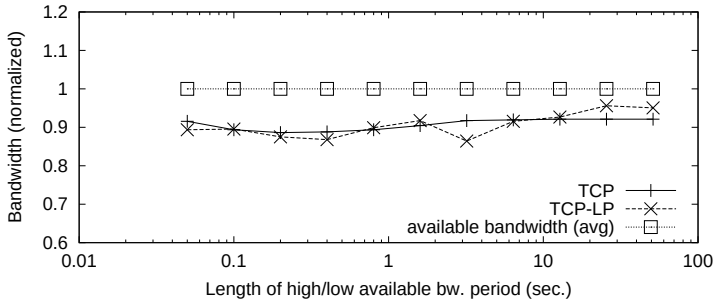


Figure 4.21 : Utilized Available Bandwidth vs. Square Wave Period

Aggregation Level and Fairness

Finally, this section repeats the simulation experiment from Section 4.4.2 in the testbed and explore the impact of the number of flows under a fixed square wave period of 6.4 sec. Figure 4.22 shows that TCP-LP utilization increases very quickly (quicker than TCP) with aggregation level in this scenario. Also, it measures the fairness index of ten TCP-LP flows and the results (not shown) confirm that TCP-LP flows achieve inter-TCP-LP fairness.

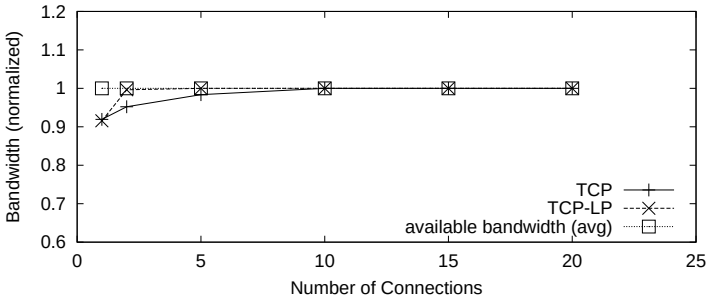


Figure 4.22 : Utilized Available Bandwidth vs. Number of Flows

4.5.3 Internet Experiments

Next, this thesis evaluates TCP-LP's performance in the Internet. All flows in the experiments (both TCP and TCP-LP) are originated from Rice University (Houston, TX) and the receivers are located 14 hops away, at SLAC (Stanford, CA). The flows traverse the Rice campus network, local and regional providers networks, and finally the Stanford campus network.

TCP-LP and a Greedy Long-Lived TCP Flow

This section considers simultaneous FTP downloads, where one flow uses TCP-LP and the other uses TCP. The objective is two-fold. First, to check TCP-LP's non-

intrusiveness property in a WAN environment. Second, to investigate to what extent can TCP-LP utilize excess bandwidth in the presence of a greedy long-lived TCP traffic. To obtain a time-dependent function of the utilized bandwidth, this section performs simultaneous downloads each five minutes, and thus obtains 12 throughput samples per hour for each of the flows.

Figure 4.23 depicts the TCP and TCP-LP throughput over a 12-hour (720 minutes) period.[9] First, observe that whenever TCP throughput is high (around 60 Mb/s), TCP-LP throughput remains low, thus confirming its TCP-transparent property. On the other hand, when the cross-traffic activity is strong enough (see Figure 4.23 when the x-axis is less than 200 minutes), TCP throughput drops down significantly, yet TCP-LP does not utilize substantially more bandwidth because it also gives priority to cross-traffic flows. However, observe that there are times (e.g., 200 min, 300 min, 440 min), when cross-traffic can simply hinder TCP from fully utilizing the available bandwidth (by forcing it to enter exponential backoff), yet leaving some bandwidth unused. TCP-LP detects such moments and successfully fills these gaps in TCP throughput. Thus, while these events are much less pronounced than in Figure 4.9 (due to the lack of reverse cross-traffic), TCP-LP demonstrates an important ability to detect and exploit such events.

Time-of-Day Effects

Finally, this section explores the impact of time-of-day effects on TCP-LP's throughput. Naturally, the hypothesis is that more bandwidth is available during nights when the network is less utilized. Furthermore, the goal is to quantify the amount of bandwidth available to a TCP-LP flow, and to compare it to TCP throughput.

[9] The experiments took place during a weekend, and that is why the figure lacks time-of-day effects.

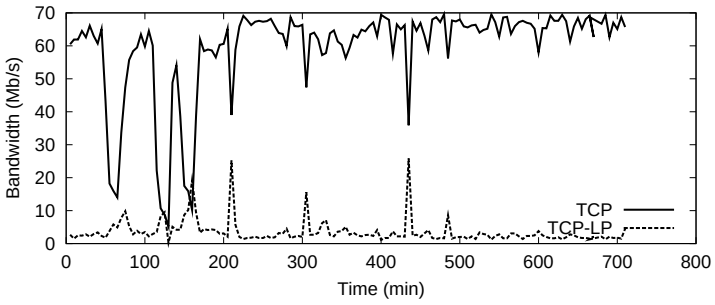


Figure 4.23 : TCP and TCP-LP Throughput vs. Time

To obtain the desired time-dependent functions of TCP and TCP-LP throughputs, while mitigating the above simultaneous file transfers effects (where TCP utilizes the entire available bandwidth), this work *interchangeably* measures TCP and TCP-LP throughputs in the 5-minute intervals over a 24-hour period. In this way, totally 12 samples per hour are obtained, six for each of the flows. While necessarily not comprehensive (due to traffic fluctuations over short time intervals), this methodology provides a reasonable way to independently measure TCP and TCP-LP throughputs on the same network path.

Figure 4.24 depicts the TCP and TCP-LP throughputs measured over a 24-hour period, starting at midnight. Note first that the time-of-day effects are clearly observable for both TCP and TCP-LP. As expected, the effects are more pronounced for the TCP-LP flow, which gives priority to *all* flows on the end-to-end path, and utilizes only the bandwidth that is left unused. On the other hand, TCP competes for resources with the cross traffic flows, and eventually utilizes its share of bandwidth. Figure 4.24 indicates that in the after-midnight hours (midnight to 8 a.m.) as well as in the after-working hours (5 p.m. to midnight), TCP-LP throughput fluctuates between 50% and 100% of TCP's throughput at the same time, utilizing approximately 75% of the TCP bandwidth on average. On the other hand, during working hours

(8 a.m. - 5 p.m.), TCP-LP throughput fluctuates between 0% and 100% of TCP's throughput, utilizing approximately 45% of the TCP bandwidth in this interval on average. Thus, the experiment illustrates that despite its low-priority nature, a TCP-LP flow is able to utilize significant amounts of available bandwidth in a wide-area network environment.

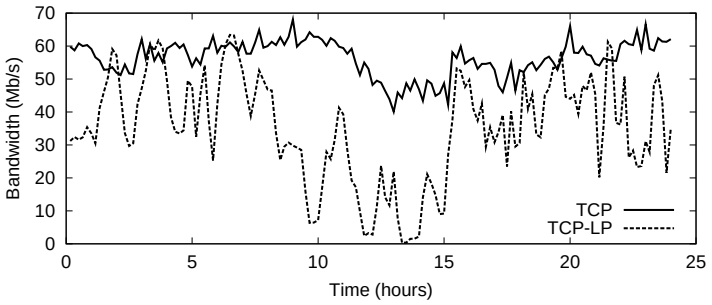


Figure 4.24 : TCP and TCP-LP Throughput vs. Time

4.6 High-speed TCP-LP

This section provides a brief overview of HSTCP-LP [17], a variant of TCP-LP developed for low-priority bulk data transfer in high-speed high-RTT networks (i.e., networks operating at 622 Mb/s, 2.5 Gb/s, or 10 Gb/s and spanning several countries or states). The key challenge in designing a low-priority protocol in such an environment is overcoming a magnified tradeoff (when compared to lower-rate links) between the ability to utilize the available bandwidth on one hand, and to quickly backoff in moments of congestion on the other.

HSTCP-LP is developed by merging two existing protocols: the first is High Speed TCP [74]; and the second is TCP-LP [15]. The goal is for HSTCP-LP to inherit the desired functionality of both, TCP-LP's ability to give strict priority to the cross-traffic, and HSTCP's efficiency in utilizing the excess network bandwidth. Moreover,

since HSTCP maintains strict fairness with current (non-high-speed) TCP implementations on low-speed links [74], it consequently enables HSTCP-LP to achieve a strict low-priority service in a broad span of networking environments: vs. current TCP implementations (e.g., TCP Sack) on low-speed links (in heavy or moderate packet drop ranges), and vs. high-speed TCP implementations (e.g., HSTCP [74], Scalable TCP [75], FAST TCP [76], BI-TCP [77], and H-TCP[78]) in high-rate networks.

On one hand, HSTCP-LP inherits two low-priority mechanisms from TCP-LP. First, in order to provide non-intrusive low-priority service, HSTCP-LP flows must detect oncoming congestion prior to cross-traffic flows. Consequently, HSTCP-LP uses inferences of one-way packet delays as early indications of network congestion rather than packet losses as used by the TCP-Sack-like cross-traffic flows. Second, HSTCP-LP inherits TCP-LP's congestion avoidance policy with two objectives: (1) quickly back off in the presence of congestion from the background flows and (2) achieve fairness among HSTCP-LP flows. On the other hand, HSTCP-LP inherits HSTCP's increase/decrease policy for large window sizes that enables it to quickly utilize and retain the available excess bandwidth in the absence of sufficient cross-traffic. In summary, HSTCP-LP is a TCP-LP version with HSTCP-like agile properties, or alternatively, a HSTCP stack with built-in TCP-LP-like low-priority mechanisms.

However, HSTCP-LP is far from being a trivial fusion of the two ancestor TCP stacks. The key challenge in designing the protocol is overcoming a magnified tradeoff (when compared to lower-rate links) between the ability to successfully utilize the excess bandwidth on one hand and to quickly backoff in moments of congestion on the other. For example, the original TCP-LP backoff policy that radically reduces window size when detecting persistent congestion (see reference [15] for details) is not entirely applicable to a high-speed environment since it can significantly degrade HSTCP-LP's performance. Consequently, HSTCP-LP applies a hybrid congestion avoidance

scheme that utilizes TCP-LP-like mechanisms only in low excess-bandwidth ranges, and then converges toward the less-backoff-responsive HSTCP policy as the window size increases.

The implementation of HSTCP-LP is derived by modifying the Linux-2.4-22-web100 kernel, which by default uses the HSTCP stack. The HSTCP-LP source code is available at *http://www.ece.rice.edu/networks/TCP-LP/*. This section briefly presents the results of an extensive set of Internet experiments on fast-production networks. In the majority of the experiments, the flows are launched from SLAC (Stanford, CA) to UFL (Gainesville, FL), as well as from SLAC to UMICH (Ann Arbor, MI), with the maximum achievable bandwidth on both paths being around 450 Mb/s. The experiments are performed with and without a light periodic UDP cross traffic (the average is 10% of the maximum bandwidth) to evaluate HSTCP-LP's ability to utilize the excess bandwidth. Also, a HSTCP-LP flow is multiplexed with the other TCP stacks to explore their mutual behavior.

The results show that HSTCP-LP is able to utilize significant amounts of the excess bandwidth when there is no cross-traffic in the network or when it multiplexes with a light periodic UDP traffic. On average, HSTCP-LP's performance is similar to the performance of other advanced TCP stacks, while the actual throughput varies in the 80% - 127% range (when compared to other high-speed TCP stacks) depending on various parameters such as the UDP cross-traffic period, the maximum window size or the sending-interface transmission queue length (*txqlen* in Linux). Next, the experiments show that HSTCP-LP is largely non-intrusive to other high-speed TCP stacks. HSTCP-LP consistently utilizes less bandwidth than the other stacks when it multiplexes with them, and the level of prioritization dominantly depends on the bottleneck-queue length: it ranges from *strict* low prioritization for larger bottle-

neck queue lengths (when the maximum queuing delay[10] is approximately $\geq 50\,ms$) to somewhat lighter levels of prioritization for smaller queue lengths. Finally, an HSTCP-LP flow is multiplexed with an aggregate of TCP Sack flows in a *high-speed* environment (on the SLAC-UMICH path). HSTCP-LP applies a more agile (than TCP Sack) window increase policy, yet uses one-way packet delays for early congestion indication. The goal is to evaluate which of the above mechanisms is more prevalent. The experiment shows that HSTCP-LP utilizes only 4.5% of the bandwidth in this scenario, thus confirming its low-priority nature.

4.7 TCP-LP: Related Work

The most related protocol to TCP-LP is TCP-Nice [79], which aims to provide a system support for background file replication. TCP-LP [15] and TCP-Nice were developed in parallel and independently from each other. TCP-Nice is designed as an extension to TCP-Vegas [33], with a more sensitive congestion detector. It uses an RTT-threshold-based congestion indication scheme where congestion is indicated if more than 50% of packets encounter the RTT-delay threshold. On the other hand, recall that TCP-LP reacts to one-way-delay threshold-based congestion indications and more aggressively decreases window size in times of persistent congestion. TCP-LP's use of one-way delays is critical because cross-traffic in the direction from the receiver to the sender may significantly prevent TCP-Nice from utilizing excess bandwidth. More precisely, if TCP-Nice's ACK packets are persistently delayed on the reverse path (such that the round-trip times are beyond the round-trip threshold), TCP-Nice may achieve near *zero* throughput, *independently* from the actual amount of excess bandwidth in the network. On the other hand, TCP-LP does not have this

[10]The maximum queuing delay is inferred by performing a parallel *ping* measurement.

problem as it suppresses the influence of reverse cross-traffic by using one-way delay measurements.

While no protocols other than TCP-LP and TCP-Nice provide an end-point realization of a low priority service, there are related efforts in several areas. First, one of the key TCP-LP mechanisms is the use of packet delay measurements for early congestion indications. Jain's delay-based congestion avoidance protocol [31], Wang *et al.'s* TCP/Dual [32], Brakmo *et al.'s* TCP/Vegas [33] all use delay-based congestion control in an effort to increase TCP throughput due to a reduced number of packet losses and timeouts, and a reduced level of congestion over the path. The key difference between TCP-LP and RTT-based congestion control protocols is in their primary objective. While the former aims to achieve *fair-share* rate allocations, TCP-LP aims to utilize only excess bandwidth. In this context, note that Martin *et al.* [80] suggest that RTT-based congestion avoidance is problematic to incrementally deploy in the Internet due to degraded throughput as compared to TCP/Reno flows. Observe that TCP-LP does not suffer from this problem again due to its different objective: TCP-LP targets the excess-capacity rate vs. the fair-share rate. Thus, TCP-LP is incrementally deployable and could be successfully used by *any* subset of Internet users. On the other hand, it may indeed be expected that TCP-LP shows a reduced performance in the networks (e.g., wireless) that induce high non-congestion-based delay variations.

Second, TCP-LP uses early congestion indication (earlier than TCP) as a basis for achieving class differentiation. Clark and Feng [81] proposed RIO (RED with In and Out) in which routers apply different marking/dropping functions for different classes of flows, thereby providing service differentiation. While similar in philosophy to TCP-LP, TCP-LP develops an *end-point* realization of early congestion indication for the purpose of low-priority transfer. Consequently, TCP-LP is applicable over routers

and switches that provide no active queue management or service differentiation.

Third, TCP-LP relates to adaptive bandwidth allocation schemes that aim to minimize file-transmission times using file-size-based service differentiation. Guo and Matta [72] use RIO in core routers and a packet classifier at the edge to distinguish between long- and short-lived TCP flows. Yang and de Veciana [82] develop TCP/SAReno in which the AIMD parameters dynamically depend on the remaining file size. While TCP-LP also substantially improves file-transmission times in the best-effort class, the key difference between TCP-LP and the above schemes is that it provides *strict* low-priority service, independent of the file size.

Next, as TCP-LP targets transmitting at the rate of available bandwidth, it is related to cross-traffic estimation algorithms which attempt to infer the available bandwidth via probing (see reference [54] for a thorough review of such algorithms). For example, Ribeiro *et al.* [62] and Alouf *et al.* [63] provide algorithms for estimation of parameters of competing cross-traffic under multifractal and Poisson models of cross traffic. In contrast, TCP-LP provides an adaptive estimation of available bandwidth by continually monitoring one-way delays and dynamically tracking the excess capacity. Similarly, Jain and Dovrolis [54] develop *pathload*, a delay-based rate-adaptive probing scheme for estimating available bandwidth. The key difference between *pathload* and TCP-LP is that the latter aims to *utilize* the available bandwidth, while the former only estimates it. Moreover, TCP-LP addresses the case of multiple flows *simultaneously* inferring the available bandwidth by providing each with a fair share (according to TCP fairness), an objective that is problematic to achieve with probes.

Finally, end-point admission control algorithms also use probes to detect if sufficient bandwidth is available for real-time flows [44]. Unfortunately, such techniques have a "thrashing" problem when many users probe simultaneously and none can

be admitted. While TCP-LP targets a low rather than high priority class, its basic ideas of adaptive and transparent bandwidth estimation could be applied to end-point admission control and alleviate the thrashing condition. In general, a probing flow should not assume that all measured available bandwidth is for itself alone, as this bandwidth will be shared among other probing flows. As TCP-LP partitions available bandwidth fairly among TCP-LP flows, this problem is eliminated.

4.8 Summary

This chapter presented TCP-LP, a protocol designed to achieve low-priority service (as compared to the existing best-effort class) from the network endpoints. TCP-LP allows low-priority applications such as bulk data transfer to utilize excess bandwidth without significantly perturbing non-TCP-LP flows. TCP-LP is realized as a sender-side modification of the TCP congestion control protocol and requires no functionality from the network routers nor any other protocol changes. Moreover, TCP-LP is incrementally deployable in the Internet. However, while the end-point congestion control is highly robust to diverse network conditions, its implicit assumption of end-system cooperation results in high vulnerability to malicious behavior. In the following chapters, this thesis analyzes the resiliency of end-point congestion control algorithms in environments with untrusted endpoints.

Chapter 5

Low-Rate TCP-Targeted Denial of Service Attacks

While TCP's congestion control algorithm is highly robust to diverse network conditions, its implicit assumption of end-system cooperation results in a well-known vulnerability to attack by high-rate non-responsive flows. This chapter investigates a class of *low-rate* denial of service attacks which, unlike high-rate attacks, are difficult for routers and counter-DoS mechanisms to detect. Using a combination of analytical modeling, simulations, and Internet experiments, this chapter shows that maliciously chosen low-rate DoS traffic patterns that exploit TCP's retransmission timeout mechanism can throttle TCP flows to a small fraction of their ideal rate while eluding detection. Moreover, as such attacks exploit protocol homogeneity, this chapter studies fundamental limits of the ability of a class of randomized timeout mechanisms to thwart such low-rate DoS attacks.

5.1 DoS Origins and Modeling

This section describes how an attacker can exploit TCP's timeout mechanism to perform a DoS attack. Next, it explains a scenario and a system model of such an attack. Finally, it develops a simple model for aggregate TCP throughput as a function of the DoS traffic parameters.

5.1.1 Origins

Recall that TCP's timeout mechanism (explained in detail in Section 2.2.2), while essential for robust congestion control, provides an opportunity for low-rate DoS attacks that exploit the slow-time-scale dynamics of retransmission timers. In particular, an attacker can provoke a TCP flow to repeatedly enter a retransmission timeout state by sending high-rate, but short-duration bursts having RTT-scale burst length, and repeating periodically at slower RTO time-scales. The victim will be throttled to near-zero throughput while the attacker will have low average rate making it difficult for counter-DoS mechanisms to detect.

This thesis refers to the short durations of the attacker's loss-inducing bursts as *outages*, and presents a simple but illustrative model relating the outage time-scale (and hence attacker's average rate) to the victim's throughput as follows.

First, consider a single TCP flow and a single DoS stream. Assume that an attacker creates an initial outage at time 0 via a short-duration high-rate burst. As shown in Figure 2.2, the TCP sender will wait for a retransmission timer of 1 sec to expire and will then double its RTO. If the attacker creates a second outage between time 1 and $1 + 2\,\mathrm{RTT}$, it will force TCP to wait another 2 sec. By creating similar outages at times 3, 7, 15, $\cdots$, an attacker could deny service to the TCP flow while transmitting at extremely low average rate.

While potentially effective for a single flow, a DoS attack on TCP aggregates in which flows continually arrive and depart requires periodic (vs. exponentially spaced) outages at the minRTO time-scale. Moreover, if all flows have an identical minRTO parameter as recommended in RFC 2988 [34], the TCP flows can be forced into continual timeouts if an attacker creates periodic outages.

Thus, this thesis considers "square wave" shrew attacks as shown in Figure 5.1

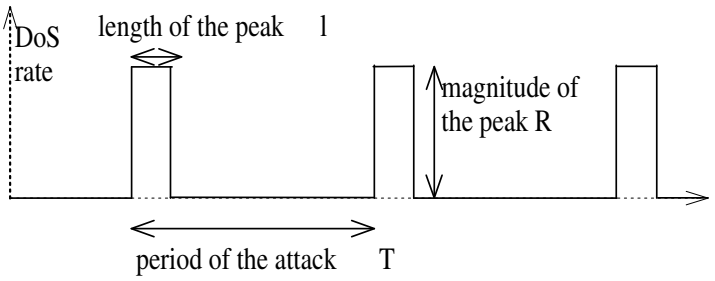


Figure 5.1 : Square-wave DoS stream

in which the attacker transmits bursts of duration l and rate R in a deterministic on-off pattern that has period T. As explored below, a successful shrew attack will have rate R large enough to induce loss (i.e., R aggregated with existing traffic must exceed the link capacity), duration l of scale RTT (long enough to induce timeout but short enough to avoid detection), and period T of scale RTO (chosen such that when flows attempt to exit timeout, they are faced with another loss).

5.1.2 Model

Consider a scenario of an attack shown in Figure 5.2(a). It consists of a single bottleneck queue driven by n long-lived TCP flows with heterogeneous RTTs and a single DoS flow. Denote RTT_i as the roundtrip time of the i-th TCP flow, $i = 1, \cdots, n$. The DoS flow is a periodic square-wave DoS stream shown in Figure 5.1. The following result relates the throughput of the TCP flows to the period of the attack.

DoS TCP Throughput Result. *Consider a periodic DoS attack with period T. If the outage duration satisfies*

(C1) $l' \geq RTT_i$

and the minimum RTO satisfies

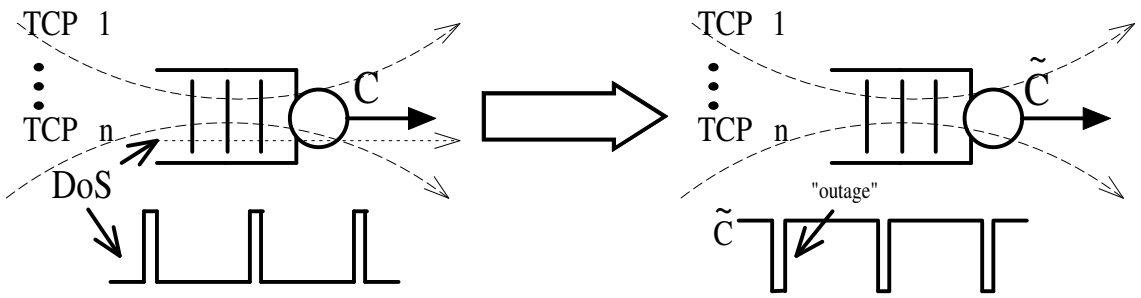


Figure 5.2 : DoS scenario and system model

(C2) $minRTO > SRTT_i + 4\,RTTVAR_i$

for all $i = 1, \cdots, n$*, then the normalized throughput of the aggregate TCP flows is approximately*

$$\rho(T) = \frac{\lceil \frac{minRTO}{T} \rceil T - minRTO}{\lceil \frac{minRTO}{T} \rceil T}. \tag{5.1}$$

This result is obtained as follows. As shown in Figure 5.2(b), the periodic l-length bursts create short l'-length outages having high packet loss.[1] If l' reaches the TCP flows' RTT time-scales, i.e., $l' \geq \text{RTT}_i$, for all $i = 1, \cdots, n$, then the congestion caused by the DoS burst lasts sufficiently long to force *all* TCP flows to simultaneously enter timeout. Moreover, if $\text{minRTO} > \text{SRTT}_i + 4\,\text{RTTVAR}_i$, for $i = 1, \cdots, n$, all TCP flows will have identical values of RTO and will thus timeout after minRTO seconds, which is the ideal moment for an attacker to create a new outage. Thus, in this case, despite their heterogeneous round-trip times, all TCP flows are forced to "synchronize" to the attacker and enter timeout at (nearly) the same time, and attempt to recover at (nearly) the same time. Thus, when exposed to outages with period T, Equation (5.1) follows.

Equation (5.1) expresses the *normalized* throughput of a TCP flow under a T-

[1] The relationship between l and l' is explored in Section 5.2.

periodic attack: a ratio of the bandwidth achievable by the TCP flow under the T-periodic attack, and the TCP bandwidth without any attack. For example, when $T = 1.5\,\text{sec}$, and $\text{minRTO} = 1\,\text{sec}$, the TCP flow utilizes the available bandwidth in the $[\text{minRTO}, T]$ period after each outage, such that the normalized TCP throughput becomes $(T - \text{minRTO})/T = 0.33$. On the other hand, when $T = 0.8\,\text{sec}$, only every second outage is effective, and the TCP flow utilizes bandwidth in the $[\text{minRTO}, 2T]$ period after each effective outage in this scenario. Consequently, the normalized throughput becomes $(2T - \text{minRTO})/2T = 0.375$ according to Equation (5.1).

Note that Equation (5.1) does not model throughput losses due to the slow-start phase, but simply assumes that TCP flows utilize all available bandwidth after exiting the timeout phase. In other words, it is assumed that the TCP flows utilize the full link bandwidth after the end of each retransmission timeout and the beginning of the following outage. Observe that if the period T is chosen such that $T \geq 1 + 2\,\text{RTT}_i$, all TCP flows will continually enter a retransmission timeout of 1 sec duration. Thus, because Equation (5.1) assumes that $\text{RTO} = \text{minRTO}$ for $T > \text{minRTO}$, while this is not the case in the period $(\text{minRTO}, \text{minRTO} + 2\,\text{RTT})$, Equation (5.1) behaves as an *upper bound* in practice. In other words, periodic DoS streams are not utilizing TCP's exponential backoff mechanism but rather exploit repeated timeouts.

Next, consider flows that do not satisfy conditions (C1) or (C2).

DoS TCP Flow-Filtering Result. *Consider a periodic DoS attack with period* T. *If the outage duration* $l' \geq RTT_i$ *and* $minRTO > SRTT_i + 4\,RTTVAR_i$ *for* $i = 1, \cdots, k$ *whereas* $l' < RTT_j$ *or* $minRTO \leq SRTT_j + 4\,RTTVAR_j$ *for* $j = k+1, \cdots, n$, *then Equation (5.1) holds for flows* $1, \cdots, k$.

This result, shown similarly to that above, states that Equation (5.1) holds for *any* TCP sub-aggregate for which conditions (C1) and (C2) hold. In other words, if a shrew attack is launched on a group of flows such that only a subset satisfies the two

conditions, that subset will obtain degraded throughput according to Equation (5.1), whereas the remaining flows will not. The thesis refers to this as "flow filtering", meaning that such an attack will deny service to a subset of flows while leaving the remainder unaffected, or even obtaining higher throughput. This issue is discussed in detail in Section 5.3.

5.1.3 Example

This section presents a baseline set of experiments to explore TCP's "frequency response" to shrew attacks. It first considers the analytical model and the scenario depicted in Figure 5.2 in which conditions (C1) and (C2) are satisfied and minRTO = 1 sec. The curve labeled "model" in Figure 5.3 depicts ρ vs. T as given by Equation (5.1). Throughput is normalized to the link capacity, which under high aggregation, is also the throughput that the TCP flows would obtain if no DoS attack were present.

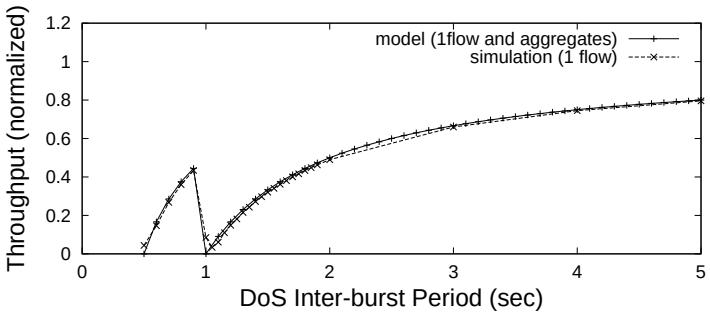


Figure 5.3 : DoS TCP throughput: model and simulation

Note that the average rate of the DoS attacker is decreasing with increasing T as its average rate is given by Rl/T. However, as indicated by Equation (5.1) and Figure 5.3, the effectiveness of the attack is clearly *not* increasing with the attacker's average rate. Most critically, observe that there are two "nulls" in the frequency response in

which TCP throughput becomes *zero*. In particular, $\rho(T) = 0$ when $T = \text{minRTO}$ and $T = \text{minRTO}/2$. The physical interpretation is as follows: if the attacker creates the minRTO-periodic outages, it will completely deny service to the TCP traffic. Once the brief outage occurs, all flows will simultaneously timeout. When their timeout expires after minRTO seconds and they again transmit packets, the attacker creates another outage such that the flows backoff again. Clearly, the most attractive period for a DoS attacker is minRTO (vs. minRTO/2), since it is the null frequency that minimizes the DoS flow's average rate. When $T > \text{minRTO}$, as the period of the attack increases, the TCP flows obtain increasingly higher throughput in periods between expiration of retransmission timers and the subsequent DoS outage.

Next, this section presents a set of *ns* simulations to compare against the model. The experiments again consider the scenario of Figure 5.2 but with a single TCP flow.[2] The TCP Sack flow has minRTO = 1 second and satisfies conditions (C1) and (C2). More precisely, the propagation delay is 6 ms while the buffer size is set such that the round-trip time may vary from 12 ms to 132 ms. The link capacity is 1.5 Mb/s, while the DoS traffic is a square-wave stream with the peak rate 1.5 Mb/s and burst length 150 ms.

The curve labeled "simulation" in Figure 5.3 depicts the measured normalized throughput of the TCP flow. Figure 5.3 reveals that Equation (5.1) captures the basic frequency response of TCP to the shrew DoS attack, characterizing the general trends and approximating the location of the two null frequencies.

[2]Recall that Equation (5.1) holds for any number of flows, and that TCP aggregates are simulated in Section 5.3.

5.2 Creating DoS Outages

This section explores the traffic patterns that attackers can use in order to create temporary outages that induce recurring TCP timeouts. First, it studies the instantaneous bottleneck-queue behavior in periods when an attacker bursts packets into the network. Next, it develops the DoS stream which minimizes the attacker's average rate while ensuring outages of a particular length. Finally, this section studies square-wave DoS streams and identifies the conditions in which they accurately approximate the optimal double-rate DoS streams.

5.2.1 Instantaneous Queue Behavior

Consider a bottleneck queue shared by a TCP flow and a DoS flow which every T seconds bursts at a constant rate R_{DoS} for duration l. Denote R_{TCP} as the instantaneous rate of the TCP flow, B as the queue size, and B_0 as the queue size at the onset of an attack, assumed to occur at $t = 0$.

Denote l_1 as the time that the queue becomes full such that

$$l_1 = \frac{(B - B_0)}{R_{DoS} + R_{TCP} - C}. \tag{5.2}$$

After l_1 seconds, the queue remains full for $l_2 = l - l_1$ seconds if $R_{DoS} + R_{TCP} \geq C$. Moreover, if $R_{DoS} \geq C$ during the same period, this will create an outage to the TCP flow whose loss probability will instantaneously increase significantly and force the TCP flow to enter a retransmission timeout with high probability (see also Figure 5.2).

5.2.2 Minimum Rate DoS Streams

Suppose the attacker is limited to a peak rate of $R_{\max}$ due to a secondary bottleneck or the attacker's access link rate. To avoid router-based mechanisms that detect

high rate flows, e.g., [12], DoS attackers are interested in ways to minimally expose their streams to detection mechanisms. To minimize the number of bytes transmitted while ensuring outages of a particular length, an attacker should transmit a double-rate DoS stream as depicted in Figure 5.4. To fill the buffer without help from background traffic or the attacked flow requires $l_1 = B/(R_{\max} - C)$ seconds. Observe that sending at the maximum possible rate $R_{\max}$ minimizes l_1 and consequently the number of required bytes. Once the buffer fills, the attacker should reduce its rate to the bottleneck rate C to ensure continued loss using the lowest possible rate.

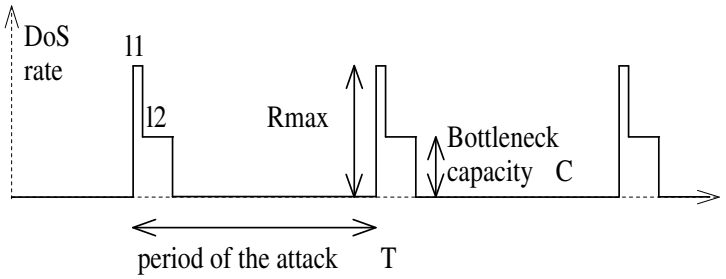


Figure 5.4 : Double-rate DoS stream

Thus, double-rate streams *minimize* the number of packets that need to be transmitted (for a given bottleneck queue size B, bottleneck capacity C, and range of sending rates from 0 to $R_{\max}$) among all possible sending streams that are able to ensure periodic outages with period T and length l_2.

To generate double-rate DoS streams in real networks, an attacker can use a number of existing techniques to estimate the bottleneck link capacity [51–53, 83, 84], bottleneck-bandwidth queue size [85] and secondary bottleneck rate [86].

Regardless of the optimality of double-rate DoS streams, this thesis considers the simpler square-wave DoS attack shown in Figure 5.1 as an approximation. First, these streams do not require prior knowledge about the network except the bottleneck rate.

Second, they isolate the effect of a single time-scale periodic attack.

To study the effectiveness of the square-wave, simulation experiments are performed in order to compare the two attacks' frequency responses. As an example, this section considers a square-wave DoS stream with peak rate 3.75 Mb/s and burst length $l = 50$ ms and a double-rate stream with $R_{\max} = 10$ Mb/s. For the double-rate stream, l_1 is computed as $B/(R_{\max} - C)$, while l_2 is determined such that the number of packets sent into the network is the same for both streams. The simulation parameters are the same as previously.

The resulting frequency responses in this example and others (not shown) are nearly identical. Consequently, since square-wave DoS streams accurately approximate the double-rate DoS stream and do not require knowledge of network parameters, the square-wave DoS streams are used henceforth in both simulations and Internet experiments.

5.3 Aggregation and Heterogeneity

This section explores the impact of TCP flow aggregation and heterogeneity on the effectiveness of the shrew attack. First, it experiments with long-lived homogeneous-RTT TCP traffic and explores the DoS stream's ability to synchronize flows. Second, it presents experiments in a heterogeneous RTT environment and explores the effect of RTT-based filtering. Third, this section studies the impact of DoS streams on links dominated by web traffic. Finally, it evaluates several TCP variants' vulnerability to shrews.

As a baseline topology (and unless otherwise indicated), this chapter considers many TCP Sack flows sharing a single congested link with capacity 1.5 Mb/s as in Figure 5.2. The one-way *propagation* delay is 6 ms and the buffer size is set such that the *round-trip* time varies from 12 ms to 132 ms. The DoS traffic is a square-wave

stream with peak rate 1.5 Mb/s, burst duration 100 ms, and packet size 50 bytes. In all experiments, there exists a FTP/TCP flow in the reverse direction, whose ACK packets multiplex with TCP and DoS packets in the forward direction. For each data point in the figures below, there are five simulation runs and averages are reported. Each simulation run lasts 1000 sec. The *ns* code and simulation scripts are available at *http://www.ece.rice.edu/networks/shrew*.

5.3.1 Aggregation and Flow Synchronization

The experiments of Section 5.1 illustrate that a DoS square wave can severely degrade the throughput of a *single* TCP flow. Here, the thesis investigates the effectiveness of low bit-rate DoS streams on TCP aggregates with homogeneous RTTs for five long-lived TCP flows sharing the bottleneck.

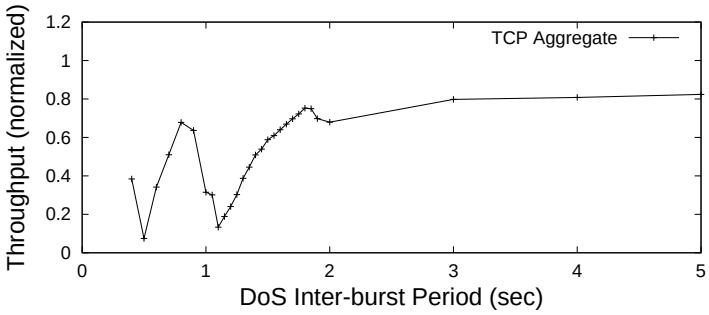


Figure 5.5 : DoS and aggregated TCP flows

Figure 5.5 depicts the normalized *aggregate* TCP throughput under the shrew attack for different values of the period T. Observe that similar to the one-flow case, the attack is highly successful so that Equation (5.1) can also model attacks on aggregates. However, note that when compared to the single-flow case, the throughput at the null 1/minRTO frequency is slightly larger in this case because the maximum RTT of 132 ms is greater than the DoS burst length of 100 ms such that a micro-flow

may survive an outage. Also observe that an attack at frequency 2/minRTO nearly completely eliminates the TCP traffic.

The key reasons for this behavior are twofold. First, *RTO homogeneity* (via minRTO) introduces a single vulnerable time-scale, even if flows have different RTTs (as explored below). Second, *DoS-induced synchronization* occurs when the DoS outage event causes all flows to enter timeout nearly simultaneously. Together with RTO homogeneity, flows will also attempt to exit timeout nearly simultaneously when they are re-attacked.

Synchronization of TCP flows was extensively explored in [87, 88] and was one of the main motivations for RED [69], whose goal is the avoidance of synchronization of many TCP flows decreasing their window at the same time. In contrast, the approach and scenario here are quite different, as an external malicious source (and not TCP itself) is the source of synchronization. Consequently, mechanisms like RED are unable to prevent DoS-initiated synchronization (see also Section 5.5).

5.3.2 RTT Heterogeneity

RTT-based Filtering

The above experiment shows that a DoS stream can significantly degrade throughput of a TCP aggregate, provided that the outage length is long enough to force all TCP flows to enter a retransmission timeout simultaneously. This section explores a heterogeneous-RTT environment with the objective of showing that a flow's vulnerability to low-rate DoS attacks fundamentally depends on its RTT, with shorter-RTT flows having increased vulnerability.

This section presents experiments with 20 long-lived TCP flows on a 10 Mb/s link. The range of round-trip times is 20 to 460 ms [89], obtained from representative Internet measurements [90]. These measurements are used to guide the setting of link

propagation delays for different TCP flows.[3]

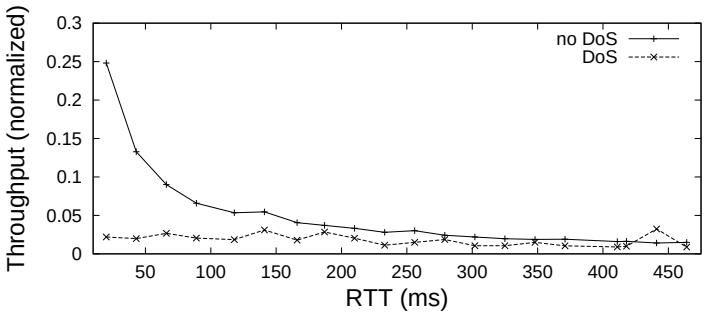


Figure 5.6 : RTT-based filtering

Figure 5.6 depicts the normalized TCP throughput for each of the 20 TCP flows. The curve labeled "no DoS" shows each flow's throughput in the absence of an attack. Observe that the flows re-distribute the bandwidth proportionally to 1/RTT such that shorter-RTT flows utilize more bandwidth than the longer ones. The curve labeled "DoS" shows each TCP flow's throughput when they are multiplexed with a DoS square-wave stream with peak rate 10 Mb/s, burst length 100 ms and period 1.1 sec. Observe that this DoS stream filters shorter-RTT flows up to a time-scale of approximately 180 ms, beyond which higher RTT flows are less adversely affected. Also, observe that despite the excess capacity available due to the shrew DoS attack, longer-RTT flows do not manage to improve their throughput.

However, in a regime with many TCP flows with heterogeneous RTTs, the *number* of non-filtered flows with high RTT will increase, and they will eventually be of sufficient number to utilize all available bandwidth left unused by the filtered smaller-RTT flows. Thus, the total TCP throughput will increase with the aggregation level for highly heterogeneous-RTT flows as illustrated in Figure 5.7. Unfortunately, the

[3]The experiments do not fit the actual CDF of this data, but uniformly distribute round-trip times in the above range.

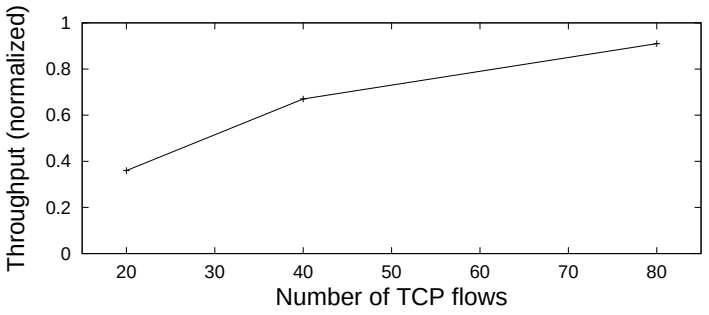


Figure 5.7 : High aggregation with heterogeneous RTT

high throughput and high link utilization with many flows (e.g., greater than 90% in the 80-flow scenario) is quite misleading, as the shorter-RTT flows have been dramatically rate-limited by the attack as in Figure 5.6. Hence, one can simultaneously have high utilization and an effective DoS attack against small- to moderate-RTT flows.

DoS Burst Length

The above experiments showed that DoS streams behave as a high-RTT-pass filter, in which the burst length is related to the filter cut-off time-scale. This section directly investigates the impact of burst length.

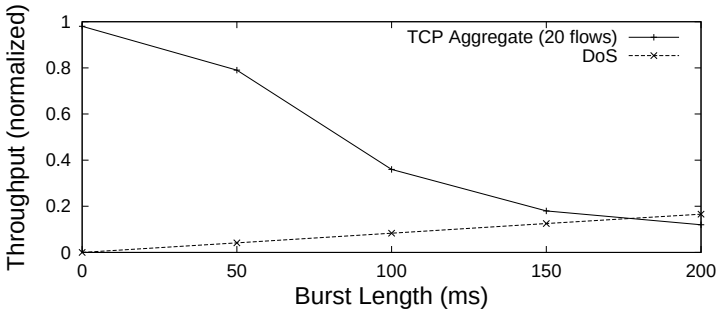


Figure 5.8 : Impact of DoS burst length

For the same parameters as above, Figure 5.8 depicts aggregate TCP through-

put as a function of the DoS burst length. The figure shows that as the burst length increases, the DoS mean rate increases, yet the aggregate TCP throughput decreases much more significantly. Indeed, as the burst length increases, the RTT-cut-off time-scale increases. In this way, flows with longer and longer RTTs are filtered. Consequently, the number of non-filtered flows decreases such that aggregate TCP throughput decreases. In other words, as the burst length increases, the sub-aggregate for which condition (C1) holds enlarges. With a fixed number of flows, the longer-RTT flows are unable to utilize the available bandwidth, and the aggregate TCP throughput decreases.

Peak Rate

Recall that the minimal-rate DoS streams studied in Section 5.2 induce outages without any help from background traffic and under the assumption that the initial buffer size B_0 is zero. However, in practice, the buffer will also be occupied by packets from reverse ACK traffic, UDP flows, etc. Consequently, in the presence of such background traffic, the DoS source can potentially lower its peak rate and yet maintain an effective attack.

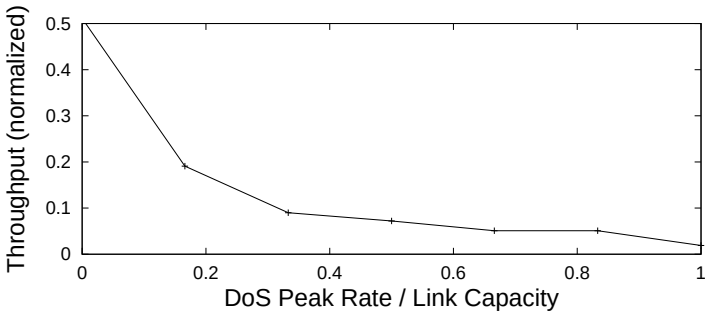


Figure 5.9 : Impact of DoS peak rate

Consider a scenario with five flows, a DoS flow and four long-lived TCP flows. The

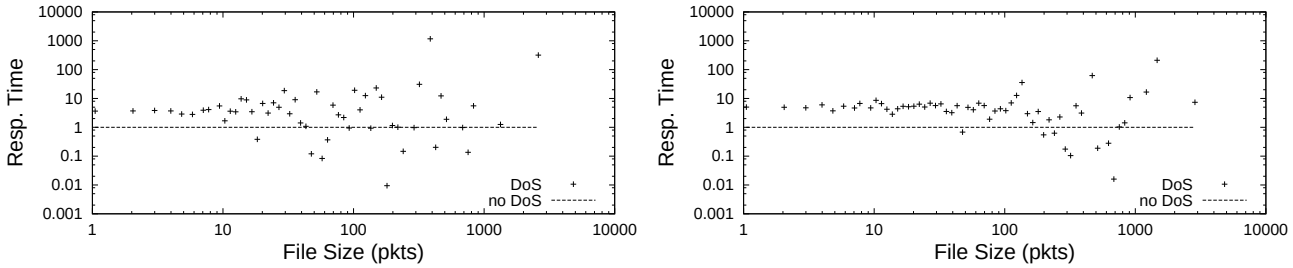
link propagation delays in the simulator are set such that one TCP flow experiences shorter RTT (fluctuates from 12 ms to 134 ms) while the other three have longer RTTs (from 108 ms to 230 ms). Figure 5.9 depicts the throughput of the short-RTT flow as a function of the normalized DoS peak rate varied from 0 to 1. Observe that relatively low peak rates are sufficient to filter the short-RTT flow. For example, a peak rate of one third of the link capacity and hence an average rate of 3.3% of the link capacity significantly degrades the short-RTT flows' throughput at the null time-scale. As hypothesized above, longer-RTT flows here play the role of background traffic and increase both B_0 and the burst rate in periods of outages which enables lower-than-bottleneck peak DoS rates to cause outages. This further implies that very low rate periodic flows that operate at one of the null TCP time-scales ($\frac{\text{minRTO}}{j}$, $j = 1, \cdots$) are highly problematic for TCP traffic. For example, some probing schemes periodically burst for short time intervals at high rates in an attempt to estimate the available bandwidth on an end-to-end path [54].

5.3.3 HTTP Traffic

Thus far, this chapter considered long-lived TCP flows. Here, it studies a scenario with flow arrival and departure dynamics and highly variable file sizes as incurred with HTTP traffic.

This thesis adopts the model of [71] in which clients initiate sessions from randomly chosen web sites with several web pages downloaded from each site. Each page contains several objects, each of which requires a TCP connection for delivery (i.e., HTTP 1.0). The inter-page and inter-object time distributions are exponential with respective means of 9 sec and 1 msec. Each page consists of ten objects and the object size is distributed according to a Pareto distribution with shape parameter 1.2. For the web transactions, the response times are measured and averaged for different sized

objects.



(a) 50% Offered Load (b) 100% Offered Load

Figure 5.10 : Impact on HTTP flows

Figure 5.10 depicts web-file response times normalized by the response times obtained when the DoS flow is not present in the system. Because of this normalization, the curve labeled "no DoS" in Figure 5.10 is a straight line with a value of one. The flows' mean HTTP request arrival rate is selected such that the offered HTTP load is 50% and near 100% for Figures 5.10(a) and 5.10(b), respectively.

On average, the file response times increased by a factor of 3.5 under 50% load and a factor of 5 under 100% load. Figures 5.10(a) and 5.10(b) both indicate that larger files (greater than 100 packets in this scenario) become increasingly and highly vulnerable to the shrew attacks with the response times of files increasing by orders of magnitude. Nevertheless, observe that some flows benefit from the shrew attack and significantly decrease their response times. This occurs when a flow arrives into the system between two outages and manages to transmit its entire file before the next outage occurs.

However, note that this effect is apparent in Figures 5.10(a) and 5.10(b) for the longer file sizes, whereas the effect is not observable for the shorter file sizes. This is due to the HTTP file-size distribution depicted in Figure 5.11, which shows that the number of short files is much larger than the number of longer files in a typical

web-browsing scenario. Consequently, while many of the short HTTP files actually manage to escape the attack and improve their response times, the response-times *average* is dominantly biased by the flows that are caught by the attack and whose response times are extremely degraded. Thus, while some flows actually benefit from the attack, the overall impact of the shrew attack on HTTP traffic remains quite effective.

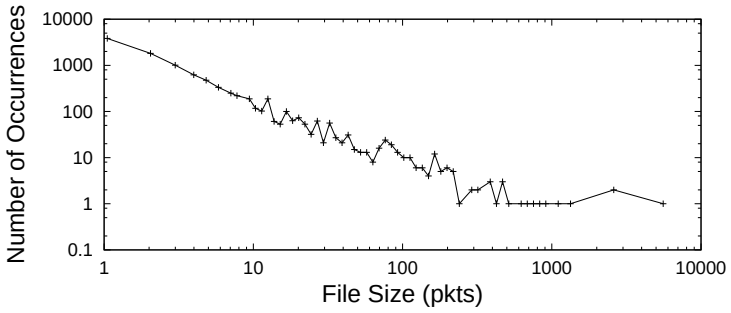


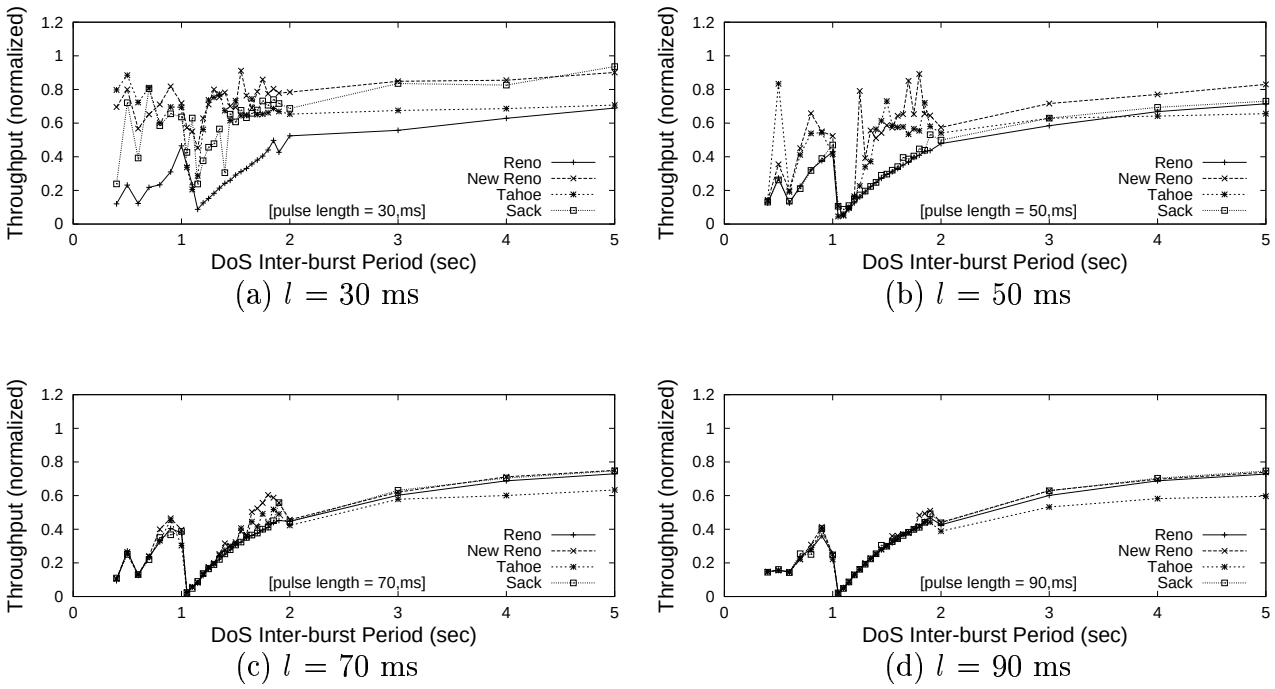
Figure 5.11 : HTTP file-size distribution

Next, observe that the deviation from the reference (no DoS) scenario is larger in Figure 5.10(a) than 5.10(b). This is because the response times are approximately 100 times lower for the no-DoS scenario when the offered load is 50% as compared to the no-DoS scenario when the system is fully utilized.

Finally, this research conducted experiments where DoS stream attack mixtures of long- (FTP) and short-lived (HTTP) TCP flows. The results (not shown) indicate that the conclusions obtained separately for FTP and HTTP traffic hold for FTP/HTTP aggregates.

5.3.4 TCP Variants

The effectiveness of low-rate DoS attacks depends critically on the attacker's ability to create correlated packet losses in the system and force TCP flows to enter re-



(a) $l = 30$ ms (b) $l = 50$ ms

(c) $l = 70$ ms (d) $l = 90$ ms

Figure 5.12 : TCP Reno, New Reno, Tahoe and Sack under shrew attacks

transmission timeout. While this chapter has studied the most robust TCP variant (TCP Sack) so far, many of the existing operating systems today still use some less advanced TCP versions. This section first provides a brief background on the work that has been done to build more robust TCP versions and help TCP flows to survive multiple packet losses within a single round-trip time without incurring a retransmission timeout. Then, it evaluates the performance of different TCP versions under the shrew attack.

It is well-known that TCP Reno is the most fragile TCP variant which enters the retransmission timeout whenever a loss happens and less than three duplicate ACKs are received. To overcome this problem, TCP New Reno [91] changes the sender's behavior during Fast Recovery upon receipt of a *partial ACK* that acknowledges some but not all packets that were outstanding at the start of the Fast Recovery

period. Further improvements are obtained by TCP Sack [36] when a large number of packets are dropped from a window of data [92] because when a Sack receiver holds non-contiguous data, it sends duplicate ACKs bearing the Sack option to inform the sender of the segments that have been correctly received. A thorough analysis of the packet drops required to force flows of a particular TCP version to enter timeout is given in [92].

Here, the thesis evaluates the performance of TCP Reno, New Reno, Tahoe and Sack under the shrew attack. Figures 5.12 (a)-(d) show TCP throughput for burst lengths of 30, 50, 70 and 90 ms, respectively. Figure 5.12(a) confirms that TCP Reno is indeed the most fragile TCP variant, while the other three versions have better robustness to DoS. However, when the peak length increases to 50 ms, *all* TCP variants obtain near zero throughput at the null frequency as shown in Figure 5.12(b). The Figure also indicates that TCP is the most vulnerable to DoS in the 1 - 1.2 sec time-scale region. During this period, TCP flows are in slow-start and have small window sizes such that a smaller number of packet losses are needed to force them to enter the retransmission timeout. Finally, Figures (c)-(d) indicate that all TCP variations obtain a throughput profile similar to Equation (5.1) when the outage duration increases, such that more packets are lost from the window of data. Indeed, if all packets from the window are lost, TCP has no alternative but to wait for a retransmission timer to expire.

5.4 Internet Experiments

This section describes several DoS experiments performed on the Internet. The scenario is depicted in Figure 5.13 and consists of a large file downloaded from a TCP Sack sender (TCP-S) to a TCP Sack receiver (TCP-R). While the RFC 2988 [34] recommendation for the minRTO parameter is already in the so-called *should*

phase,[4] to the best of the author's knowledge, it is not yet being widely deployed in the most popular operating systems. Hence, the TCP-S host is configured to have minRTO = 1 sec (by modifying the Linux-2.4.18 kernel) according to [34], and measure TCP throughput using *iperf*. The shrew attack is launched from three different hosts using a modified version of the UDP-based active probing software from [67] in order to send high-precision DoS streams. Three independent measurements are performed for each experiment and the average results are reported. Both the Linux TCP-kernel source code used in the experiments at the TCP-S side, and the modified UDP-based software used to generate the shrew attacks are available at *http://www.ece.rice.edu/networks/shrew*.

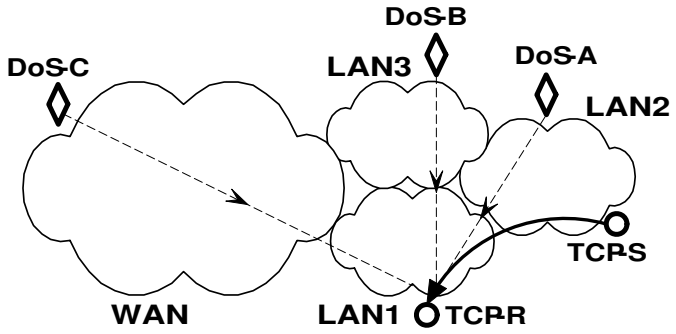


Figure 5.13 : DoS attack scenario

Intra-LAN Scenario. In this scenario, both the TCP sender (TCP-S) and DoS (DoS-A) hosts are on the same 10 Mb/s Ethernet LAN on Rice University, while the attacked host (TCP-R) is on a different 10 Mb/s Ethernet LAN, two hops away from both TCP-S and DoS-A. The peak rate of the square-wave DoS stream is 10 Mb/s while the burst length is 200 ms. The curve labeled "Intra-LAN" in Figure 5.14

[4]The IETF recommendations usually specify the parameter values that (1) *may*, (2) *should*, or (3) *must* be applied.

depicts the results of these experiments. The figure indicates that a null frequency exists at a time-scale of approximately 1.2 sec. When the attacker transmits at this period, it has an average rate of 1.67 Mb/s. Without the DoS stream, the TCP flow obtains 6.6 Mb/s throughput. With it, it obtains 780 kb/s throughput. Thus, the DoS attacker can severely throttle the victim's throughput by nearly an order of magnitude.

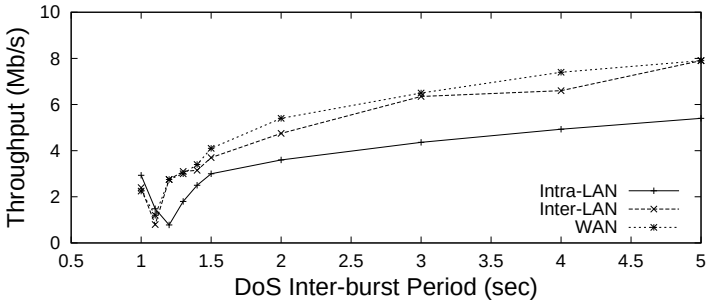


Figure 5.14 : Internet experiments

Inter-LAN Scenario. In this experiment, the TCP sender (TCP-S), DoS source (DoS-B) and attacked host (TCP-R) are on three different LANs of the ETH (Zurich, Switzerland) campus network. The route between the two traverses two routers and two Ethernet switches, with simple TCP measurements revealing that the TCP and DoS LANs are 100 Mb/s Ethernet LANs, while the attacked host is on a 10 Mb/s Ethernet LAN. The peak rate of the square-wave DoS stream is again 10 Mb/s while its duration is reduced as compared to the Intra-LAN Scenario to 100 ms. The curve labeled "Inter-LAN" in Figure 5.14 depicts the frequency response of this attack. In this case, a DoS time-scale of $T = 1.1$ sec is the most damaging to TCP, since here the TCP flow achieves 800 kb/s throughput, only 8.1% of the throughput it achieves without DoS flow (9.8 Mb/s). At this time-scale, the attacker has an average rate of 909 kb/s.

WAN Scenario. Finally, for the same TCP source/destination pair as in the Inter-LAN Scenario, source DoS-C initiates a shrew DoS attack from a LAN at EPFL (Lausanne, Switzerland), located eight hops away from the destination. The DoS stream has a peak rate of 10 Mb/s and a burst duration of 100 ms. The curve labeled "WAN" shows the frequency response of these experiments and indicates a nearly identical null located at $T = 1.1$ sec. For this attack, the TCP flow's throughput is degraded to 1.2 Mb/s from 9.8 Mb/s whereas the attacker has average rate of 909 kb/s. This experiment illustrates the feasibility of *remote* attacks. Namely, in the WAN scenario, the DoS attacker has traversed the local provider's network and multiple routers and Ethernet switches before reaching its victim's LAN. Thus, despite potential traffic distortion that deviates the attacker's traffic pattern from the square wave, the attack is highly effective.

Thus, while necessarily small scale due to their (intended) adverse effects, the experiments support the findings of the analytical model and simulation experiments. The results indicate that effective shrew attacks can come from remote sites as well as nearby LANs.

5.5 Counter-DoS Techniques

This section explores two classes of candidate counter-DoS mechanisms intended to mitigate the effects of shrew attacks: (a) router-assisted, and (b) end-point mechanisms. Out of many router-assisted schemes designed to detect and throttle malicious flows in the network, this section concentrates on the mechanisms that are based on preferential dropping of packets from malicious flows, and evaluate two representatives: RED-PD and CHOKe. From the end-point counter-DoS mechanisms, this thesis first evaluates the effect of the initial TCP congestion window size on the effectiveness of the attack, and then proposes and evaluates a counter-DoS mechanism

in which end-points randomize their minRTO parameter.

5.5.1 Router-Assisted Mechanisms

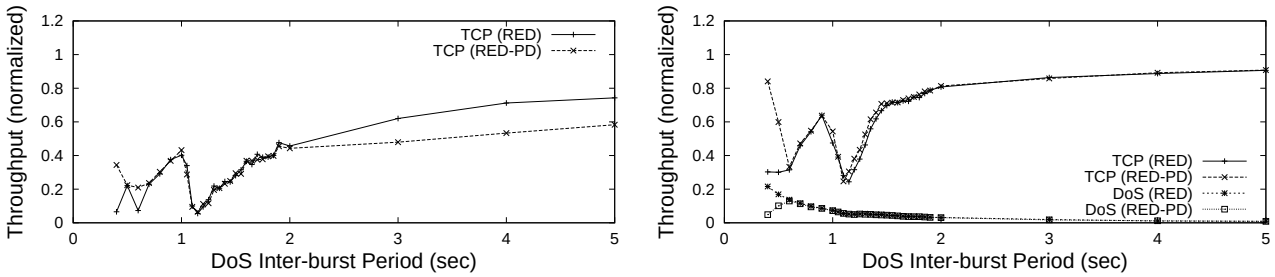
As described above, DoS flows have low average rate, yet do send relatively high-rate bursts for short time intervals. The key problem lies in the fact that relatively longer time-scales are needed to detect malicious flows with high confidence, while the shrew attack operates on relatively short time-scales. If these shorter time-scales are used to detect malicious flows in the Internet, many legitimate bursty flows would be incorrectly detected as malicious. This section investigates if the shrew traffic patterns can be identified as a DoS attack by router-based algorithms.

Mechanisms for per-flow treatment at the router can be classified as scheduling or preferential dropping. Due to implementation simplicity and other advantages of preferential dropping over scheduling (see reference [12]), this thesis concentrates on dropping algorithms for detection of DoS flows and/or achieving fairness among adaptive and non-adaptive flows. Candidate algorithms include Flow Random Early Detection (FRED) [21], CHOKe [20], Stochastic Fair Blue (SFB) [26], the scheme of reference [24], ERUF [23], Stabilized RED (SRED) [22], dynamic buffer-limiting scheme from [25] and RED with Preferential Dropping (RED-PD) [12]. Of these, this thesis studies the most popular representatives: RED-PD and CHOKe.

RED-PD

RED-PD uses the packet drop history at the router to detect high-bandwidth flows with high confidence. Flows above a configured target bandwidth are identified and monitored by RED-PD. Packets from the monitored flows are dropped with a probability dependent on the excess sending rate of the flow. RED-PD suspends preferential dropping when there is insufficient demand from other traffic in the output queue,

for example, when RED's average queue size is less than the minimum threshold.



(a) 1 TCP Sack flow (b) 9 TCP Sack flows

Figure 5.15 : Impact of RED and RED-PD routers

The simulation experiments with one and nine TCP Sack flows, RED-PD routers, and the topology of Figure 5.2 are performed. For one TCP flow, Figure 5.15(a) indicates that RED-PD is *not* able to detect nor throttle the DoS stream. For aggregated flows depicted in Figure 5.15(b), RED-PD only affects the system if the attack occurs at a time-scale of less than 0.5 sec, i.e., only unnecessarily high-rate attacks can be addressed. Most critically, at the null time-scale of 1.2 sec, RED-PD has no noticeable effect on throughput as compared to RED. Thus, while RED and RED-PD's randomization has lessened the severity of the null, the shrew attack remains effective overall.

Next, in the above scenario with nine TCP Sack flows, the DoS peak rate and burst length are varied to study the conditions under which the DoS flows will become detectable by RED-PD. The burst duration is first set to 200 ms and then the peak rate is changed from 0.5 Mb/s to 5 Mb/s. Figure 5.16(a) indicates that RED-PD starts detecting and throttling the square-wave stream at a peak rate of 4 Mb/s, which is more than twice than the bottleneck rate of 1.5 Mb/s. Recall that it has been shown in Section 5.3.2 that a peak rate of one third the bottleneck capacity and a burst length of 100 ms can be quite dangerous for short-RTT TCP flows.

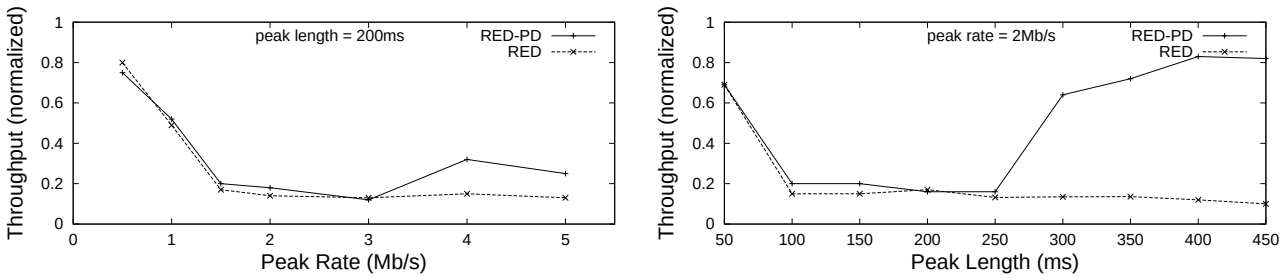


Figure 5.16 : Detecting DoS streams

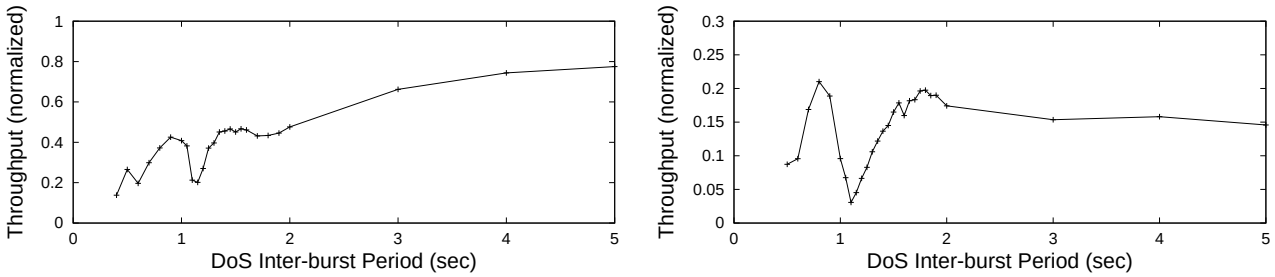
Further, the DoS peak rate is fixed to 2 Mb/s and the burst length is varied from 50 ms to 450 ms. Figure 5.16(b) shows that RED-PD begins detecting the DoS flow at 300 ms time-scales in this scenario. Recall again that much shorter burst time-scales are sufficient to throttle not only short-RTT flows, but the entire aggregates of heterogeneous-RTT TCP traffic.

Thus, Figure 5.16(b) captures the fundamental issue of time-scales: RED-PD detects high rate flows on longer time-scales, while DoS streams operate at very short time-scales. If these shorter time-scales are used to detect malicious flows in the Internet, many legitimate bursty TCP flows would be incorrectly detected as malicious. This issue is studied in depth in reference [12], which concludes that long time-scale detection mechanisms are needed to avoid excessively high false positives. However, there are schemes (e.g., [20, 21, 25, 26]) that use very short time-scales to detect high rate flows. While Mahajan *et al.* [12] indicate that the penalty for their use may be quite high, this thesis nevertheless evaluates below the ability of a representative of such schemes (CHOKe) to detect and throttle the shrew attack.

CHOKe

CHOKe is a dropping scheme designed to throttle unresponsive or misbehaving flows in a congested router. An incoming packet is matched against a random packet in

the queue. If they belong to the same flow, both packets are dropped, otherwise the incoming packet is admitted with a certain probability. The scheme tries to leverage the fact that high-bandwidth flows are likely to have more packets in the queue, and tries to approximate fair queuing in a scalable way. While Mahajan *et al.* [12] observe that CHOKe is not likely to perform well in high aggregation regimes (see reference [12] for details), and despite the indication that the penalty for the use of the scheme may be quite high (especially in low-aggregation regimes and HTTP scenarios), the main goal here is to evaluate its ability to detect the shrew attacks.



(a) 1 TCP Sack flow (b) Flow filtering

Figure 5.17 : Impact of CHOKe routers

This section initially presents a simulation experiment with one TCP Sack flow under the shrew attack, CHOKe router, and the topology of Figure 5.2. Figure 5.17(a) indicates that CHOKe outperforms RED-PD (compare Figures 5.15(a) and 5.17(a)) in thwarting the shrew attacks. This is exactly due to fact that CHOKe operates on much shorter time-scales: it observes instantaneous queue behavior (and not the drop history), and is thus able to mitigate the effectiveness of the attack more successfully. However, observe that CHOKe in this scenario does not completely eliminate the effectiveness of the attack, but only smooths the throughput "dip" on the minRTO time-scale. Finally, as the number of flows increases, CHOKe (like RED-PD) becomes more and more successful in smoothing the TCP aggregate null

frequencies (not shown).

However, recall that Section 5.3.2 demonstrated that in heterogeneous-RTT environments, the shrews are able to deny service to a subset of short-RTT TCP flows, yet *without* bursting at high instantaneous rates. Consequently, the attacker packets *do not* monopolize the buffer resources, and are thus hard to detect. This hypothesis is evaluated below.

This section repeats the experiment from Section 5.3.2 with a CHOKe router, to evaluate its ability to thwart the shrew attack in the flow-filtering scenario. The experiment consists of an aggregate of long-RTT TCP flows multiplexed with a short-RTT TCP flow. Figure 5.17(b) depicts the throughput of the short-RTT flow as a function of the shrew inter-burst period, where the peak of the shrew burst is kept to only 1/3 of the bottleneck link capacity. Observe that CHOKe fails to throttle the shrew flow, because this malicious flow is hidden in the aggregate of legitimate long-RTT TCP flows that are not significantly affected by the attack. Thus, while the shrew flow creates periodic outages and denies service to short-RTT flows, it actually never monopolizes the buffer resources, and remains undetected by the CHOKe router.

5.5.2 End-point Mechanisms

This section evaluates end-point-based counter-DoS mechanisms. The key idea is to make TCP more robust to shrew attacks by applying a more careful (DoS-resilient) protocol design. This section explores two modifications of the existing TCP parameters. The first is the increase of the initial window size parameter; and the second is the randomization of the minRTO parameter.

Increasing the Initial Window Size

The above experiments indicate that TCP flows are the most vulnerable to shrew attacks when they have small window sizes, simply due to fact that a smaller number of packet losses are needed to force them to enter the retransmission timeout. This section explores if increasing the window size after exiting the retransmission timeout (popularly known as "jump-starting" a TCP flow) may help in mitigating the effectiveness of the attack.

The parameter of interest here is the initial window size W. The default is two segments, whereas RFC 2414 [93] recommends increasing this parameter to a value between two and four segments (roughly 4 kbytes) to achieve a performance improvement. A number of experiments with TCP flows (with $W = 4$) under the shrew attack are performed, but do no noticeable improvement in such scenarios is observed. While increasing the initial window size parameter beyond four segments may lead to a congestion collapse [93], this section nevertheless performs experiments with $W = 8$ and $W = 16$ (not shown), for the sake of research curiosity. The only noticeable difference is that the TCP null time scale slightly moves closer to 1 sec. This happens because a "jump-started" TCP flow utilizes the available bandwidth much faster, but unfortunately the vulnerability to low-rate attacks remains. In summary, as long as the outage length is on the time scale of the flow's RTT, the increased number of packets in flight doesn't help in preventing the attack.

End-point minRTO Randomization

Since low-rate attacks exploit minRTO homogeneity, this section explores a counter-DoS mechanism in which end-points randomize their minRTO parameter in order to randomize their null frequencies. Here, the thesis develops a simple, yet illustrative model of TCP throughput under such a scenario. In particular, this section considers

a counter-DoS strategy in which TCP senders randomize their minRTO parameters according to a uniform distribution in the range $[a, b]$. The objective is to compute the TCP frequency response for a single flow with a uniformly distributed minRTO. Moreover, some operating systems use a simple periodic timer interrupt of 500 ms to check for timed-out connections. This implies that while the TCP flows enter timeout at the same time, they recover uniformly over the $[1, 1.5]$ sec range. Thus, the following analysis applies equally to such scenarios.

There are three cases according to the value of T as compared to a and b. First, if $T \geq b$. Then $\rho(T) = \frac{T - E(RTO)}{T}$, where $E(RTO) = (a+b)/2$ so that

$$\rho(T) = \frac{T - \frac{a+b}{2}}{T}, \text{for } T \geq b. \tag{5.3}$$

Second, for $T \in [a, b)$, denote k as $\lfloor \frac{b}{T} \rfloor$. Then,

$$\begin{aligned}\rho(T) = \frac{T-a}{b-a}\frac{T - \frac{T+a}{2}}{T} + \sum_{i=1}^{k-1} \frac{T}{b-a}\frac{\frac{T}{2}}{(i+1)T} \\ + \frac{b-kT}{b-a}\frac{(k+1)T - \frac{kT+b}{2}}{(k+1)T}.\end{aligned} \tag{5.4}$$

Equation (5.4) is derived as follows. Since only one outage at a time can cause a TCP flow to enter retransmission timeout, this thesis first determines the probability for each outage to cause a retransmission timeout and then multiplies it by the corresponding conditional expectation for the TCP throughput. In Equation (5.4), the first term denotes TCP throughput in the scenario when the retransmission timeout is caused by the next outage after the initial one. The term $\frac{T-a}{b-a}$ denotes the probability that the initial RTO period has expired, which further means that the first outage after time a will cause another RTO. The conditional expectation for TCP throughput in this scenario is $\frac{T - \frac{T+a}{2}}{T}$, where $\frac{T+a}{2}$ denotes the expected value of the end of the initial RTO, given that it happened between a and T. The second term of

Equation (5.4) denotes TCP throughput for outages $i = 2, \cdots, k-1$. The probability for them to occur is $\frac{T}{b-a}$, and the conditional expectation of TCP throughput is $\frac{T/2}{(i+1)T}$. Finally, the third term in Equation (5.4) denotes TCP throughput for the $(k+1)^{th}$ outage.

Finally, when $T < a$, it can be similarly shown that

$$\rho(T) = \frac{\lceil \frac{a}{T} \rceil T - \frac{a+b}{2}}{\lceil \frac{a}{T} \rceil T}, \text{for } k = 1, \tag{5.5}$$

and

$$\begin{aligned}\rho(T) = & \frac{\lceil \frac{a}{T} \rceil T - a}{b-a} \frac{\lceil \frac{a}{T} \rceil T - \frac{a+\lceil \frac{a}{T} \rceil T}{2}}{\lceil \frac{a}{T} \rceil T} \\ & + \sum_{i=\lceil \frac{a}{T} \rceil}^{k-1} \frac{T}{b-a} \frac{\frac{T}{2}}{(i+1)T} \\ & + \frac{b-kT}{b-a} \frac{(k+1)T - \frac{kT+b}{2}}{(k+1)T}, \text{for } k \geq 2. \end{aligned} \tag{5.6}$$

Figure 5.18 shows that the above model matches well with simulations for minRTO = uniform$(1, 1.2)$. Observe that randomizing the minRTO parameter shifts both null time scales and amplitudes of TCP throughput on these time-scales as a function of a and b. The longest most vulnerable time-scale now becomes $T = b$. Thus, in order to minimize the TCP throughput, an attacker should wait for the retransmission timer to expire, and then create an outage. Otherwise, if the outage is performed prior to b, there is a probability that some flows' retransmission timers have not yet expired. In this scenario, those flows survive the outage and utilize the available bandwidth until they are throttled by the next outage.

Because an attacker's ideal period is $T = b$ under minRTO randomization, this thesis presents the following relationship between aggregate TCP throughput and the DoS time-scale.

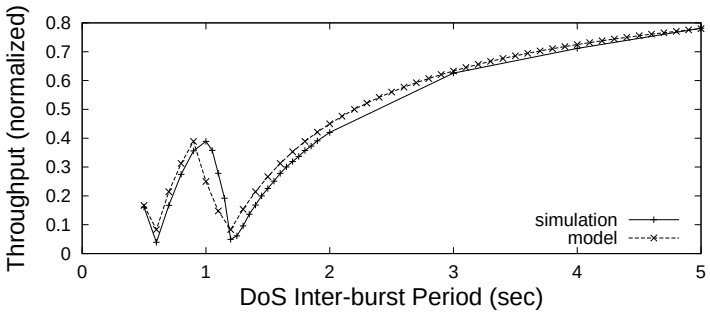


Figure 5.18 : DoS under randomized RTO

Counter-DoS Randomization Result. *Consider n long-lived TCP flows that experience b-periodic outages. The normalized* aggregate *throughput of the n flows is approximately*

$$\rho(T = b) = \frac{b - (a + \frac{b-a}{n+1})}{b} \tag{5.7}$$

The derivation is given in Appendix B.

Equation (5.7) indicates that as the number of flows n increases, the normalized aggregate TCP throughput in the presence of $T = b$ time-scale DoS attacks converges toward $\frac{b-a}{b}$. Indeed, consider the case that all flows experience an outage at the same reference time zero. When the number of flows in the system is high, a fraction of flows' retransmission timers will expire sufficiently near time a such that those flows can partially recover and utilize the available bandwidth in the period from time a to time b, when all flows will again experience an outage. For the scenario of operating systems that use a 500 ms periodic timeout interrupt, such that a flow "times out" uniformly in a [1,1.5] range, Equation (5.7) indicates that the TCP throughput degrades from 0.17 (a single TCP flow) to 0.34 (TCP aggregate with many flows) under the 1.5 sec periodic attack.

There are two apparent strategies for increasing throughput on $T = b$ time-scales. First, it appears attractive to decrease a which would significantly increase TCP

throughput. However, recall that conservative timeout mechanisms are fundamentally required to achieve high performance during periods of heavy congestion [2]. Second, while increasing b also increases TCP throughput, it does so only in higher aggregation regimes (when n is sufficiently large) and in scenarios with long-lived TCP flows. On the other hand, increasing b is not a good option for low aggregation regimes (when n is small) since the TCP throughput can become too low since $\rho(T = b) = \frac{n}{n+1}\frac{b-a}{b}$. Moreover, excessively large b could significantly degrade the throughput of short-lived HTTP flows which form the majority traffic in today's Internet. In summary, minRTO randomization indeed shifts and smooths TCP's null frequencies. However, as a consequence of RTT heterogeneity, the fundamental tradeoff between TCP performance and vulnerability to low-rate DoS attacks remains.

5.6 Summary

This chapter presented denial of service attacks that are able to throttle TCP flows to a small fraction of their ideal rate while transmitting at sufficiently low average rate to elude detection. It has been shown that by exploiting TCP's retransmission timeout mechanism, TCP exhibits null frequencies when multiplexed with a maliciously chosen periodic DoS stream. Moreover, it has been demonstrated that the existing solutions (particularly the core-based ones) are fundamentally limited in their ability to eliminate the effectiveness of the attack. The next chapter demonstrates further limitations of the core-based solutions in thwarting end-point misbehaviors.

Chapter 6

Receiver-Driven Transport Protocols: Vulnerabilities and Solutions

Recent advances in TCP congestion control design have demonstrated the ability to significantly improve TCP performance in a variety of scenarios, ranging from high-speed (e.g., [74, 94]) to mobile and wireless networks (e.g., [95, 96]). However, each such advance introduces the following dilemma: if a user can obtain a significant increase in throughput via an optimized congestion control algorithm, how can the network or the other end point distinguish among (i) users with optimized protocol stacks, (ii) "cheater's" that have modified protocol stacks that maximize their own throughput without regard to fairness or network stability, and (iii) attackers that seek only to transmit at a high rate in order to deny service to others. More precisely, the question becomes how can misbehavior be detected in the presence of widely variable protocol performance profiles? And most importantly, protocol innovations often introduce novel security challenges, which, if not considered *a priori*, may have devastating consequences once such innovations become deployed.

TCP variants that are widely deployed today are sender-centric protocols in which the sender performs important functions such as congestion control and reliability, whereas the receiver has minimum functionality via transmission of acknowledgements to the sender. Yet, as demonstrated in Chapter 2, it is becoming evident that increasing the functionality of *receivers* can significantly improve TCP performance [3–8]. Indeed, a key breakthrough in this design philosophy is represented by fully receiver-

centric protocols in which *all* control functions are delegated to receivers [9, 10]. The benefits that are being established for this innovative design include improved TCP throughput (see Section 2.2.3 for details) and an array of other performance enhancements: (i) improved loss recovery; (ii) more robust congestion control; (iii) improved power management for mobile devices; (iv) a solution to the handoff problem in wireless networks; (v) improved behavior of network-specific congestion control; (vi) easy migration to a replicated server during handoffs; (vii) improved bandwidth aggregation; and (viii) improved web response times.

However, both sender- and receiver-centric protocols implicitly rely on the assumption that both endpoints cooperate in determining the proper rate at which to send data, an assumption that is increasingly invalid today. With sender-centric TCP-like congestion control, the sending endpoint may misbehave by disobeying the appropriate congestion control algorithms and send data more quickly. Fortunately, the lack of a strong incentive for selfish Internet users to do so (uploading vs. downloading) appears to be the main guard against such misbehavior. Moreover, while it has been discovered that misbehaving *receivers* can perform DoS attacks or steal bandwidth even with sender-centric protocols [97], it has been shown that it is possible to modify TCP to entirely eliminate this undesirable behavior [97, 98].

On the other hand, receiver-centric congestion control presents a perfect match for a misbehaving user: the receiving endpoint performs *all* congestion control functions, and has both the incentive (faster web browsing and file downloads) and the opportunity (open source operating systems) to exploit protocol vulnerabilities. This chapter explores the tradeoffs and tensions between performance and trust for receiver-centric transport protocols. In particular, given the above benefits (i)-(viii), and clear vulnerabilities, *the goal is to evaluate whether it is possible for HTTP, file, and streaming servers in the Internet to deploy receiver centric transport protocols while striking a*

balance between performance enhancements and protection against misbehavior. This chapter focuses on the class of receiver-driven protocols because their deployment introduces a set of novel security challenges that can have devastating effects on the widely-deployed HTTP, file, and streaming servers in the Internet. Moreover, it shows that *none* of the existing solutions are able to efficiently protect the servers from such receiver misbehaviors.

6.1 Vulnerabilities

This section analyzes receiver misbehaviors which range from DoS attacks to more moderate (hence harder to detect) manipulations of congestion control parameters. It then develops an analytical model by generalizing [99] to predict the throughput that a misbehavior will obtain as a function of the modified AIMD parameters α and β, as well as the retransmission timeout RTO. Finally, for small files, it derives an expression for the response time for file download under modifications of the initial congestion window.

6.1.1 Receiver Misbehaviors

This section treats two classes of misbehaviors in the context of receiver-driven transport protocols: denial-of-service attacks and resource stealing. The key distinction between the two lies in the primary goal of the misbehaving client: DoS attackers aim to deny service to the background flows without necessarily achieving a particular benefit for themselves, whereas resource stealers aim to gain a performance benefit by stealing resources from the background flows (without necessarily starving them).

Denial of Service Attacks

This section begins with an extreme scenario and show that an RCP sender can become an easy target of a DoS attack.[1] Indeed, Figure 2.4 shows that the RCP sender listens to the request packets from the receiver, and replies by sending data packets without *any* control, as all control functions are delegated to the receiver for performance reasons. Hence, flooding the sender with short *req* packets (the same size as the *ack* packets, ∼40 Bytes) may force the RCP sender to flood the reverse path (from the server to the client) with much longer *data* packets (typically ∼1500 Bytes), and congest the network.[2]

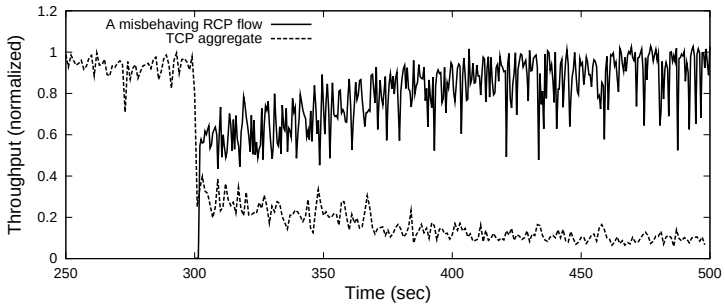


Figure 6.1 : RCP receiver performs a DoS attack by flooding the sender with requests

To demonstrate the vulnerability of fully receiver-driven transport protocols, the above request-flood attack is simulated and the result is shown in Figure 6.1. In the experiment, seven TCP Sack flows share a link, and at time 300 sec, an RCP flow joins the aggregate (the exact simulation parameters are provided in Section 6.3).

[1] Short overview of the RCP protocol is given in Section 2.2.3, while the entire protocol specification could be found in [10].

[2] While this thesis focuses on RCP, similar receiver incentives and protocol vulnerabilities hold whether protocols delegate some or all control functions to receivers, e.g., TFRC [4] and WebTP [9], respectively.

However, the congestion control functions are removed from the RCP flow (by re-tuning the appropriate RCP parameters at the receiver - details are given below), such that it floods the server with requests. Consequently, the RCP flow utilizes the entire bandwidth and denies service to the background traffic by exploiting TCP's well-known vulnerability to attacks by high-rate non-responsive flows.

Resource Stealing

In contrast, an unscrupulous receiver may moderately re-tune its parameters in an attempt to steal bandwidth from other flows in the network while eluding detection. Indeed, this thesis will quantify the extent to which it is harder to detect flows that moderately disobey some (but not all) congestion control rules (e.g., decrease the window size upon a packet loss, but do not halve it), than it is to detect flows that dramatically violate one or more congestion control rules. While this thesis do not underestimate the creativity of misbehaving receivers, this chapter treats only easy-to-implement misbehaviors that can be achieved by changing protocol parameters; namely, each parameter can be modified by changing a *single* line of code.

While the space of possible receiver misbehaviors is vast, this thesis focuses on parameter-based misbehaviors simply because they are easy to implement. While receivers could clearly use other mechanisms to achieve similar rates, Section 6.3 demonstrates that this does not affect the detection problem. Furthermore, this thesis does *not* treat the problem of application-level misbehaviors such as parallel download (where a malicious user opens multiple transport-layer connections to parallely download different partitions of a file from a server). Nevertheless, observe that the misbehaviors analyzed in this thesis are much more generic: (i) they can be simply and entirely implemented at the receivers; (ii) a malicious receiver can achieve a performance benefit even in scenarios where a single transport connection is used

for download (e.g., in the HTTP 1.1 web-server scenarios or in the non-partitioned FTP-download scenarios).

The first parameter of interest is the *additive-increase parameter* α, which has a default value of one packet per round-trip time. By increasing the window size more aggressively ($\alpha > 1$), a flow can achieve higher throughput.

The second parameter is the *multiplicative-decrease parameter* β which has a default value of 0.5 such that the congestion window is halved upon the receipt of congestion indication. Again, the receiver can potentially utilize more bandwidth by decreasing the window only moderately via $\beta > 0.5$.

The third parameter is the *retransmission timeout RTO*. Both TCP and RCP use a retransmission timer to ensure data delivery in the absence of any feedback from the remote peer.[3] In both cases, this value is computed using smoothed round-trip time and round-trip time variation. RFC 2988 [34] recommends to lower- and upper-bound this value to 1 and 60 sec, respectively. Thus, a malicious receiver may easily change these values. For example, by setting the *RTO* to a small value (e.g., 100 ms), one can expect to achieve throughput improvements in high packet loss ratio environments, because the misbehaving flow would back-off significantly less aggressively than behaving flows would.

Finally, the fourth parameter of interest is *the initial window size* W. The default is two segments, whereas RFC 2414 [93] recommends increasing this parameter to a value between two and four segments (roughly 4 Kbytes) to achieve a performance improvement. A misbehaving receiver might wish to further improve its performance (without caring much about problems such as congestion collapse), and increase this parameter even more. By doing so, the receiver can maliciously jump-start the RCP

[3] In the sender-driven TCP scenario, it is the absence of *ack* packets from the TCP receiver, while in the receiver-driven RCP scenario, it is the absence of *data* packets from the RCP sender.

flow (this is exactly what was done, among other things, in Figure 6.1 by setting $W = 10$) and improve its throughput. However, this parameter is expected to be crucial in improving the short file-size response times which are typical for web browsing.

6.1.2 Modeling Misbehaviors

Manipulations of parameters α, β, and RTO enable misbehaving receivers to steal bandwidth over longer time scales, whereas modifying the parameter W reduces latency for small files, hence over shorter time-scales. This section develops analytical models to predict the amount of stolen bandwidth and reduced latency over long- and short-time-scales, respectively.

Long Time Scales

This section begins with the well-known TCP throughput formula (Equation (30) in [99]) that expresses average TCP rate B as a function of the round-trip time RTT, steady-state loss event rate p, TCP retransmission timeout value RTO, and number of packets acknowledged by each ack b (typically $b = 1$ [100]):

$$B \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + RTO\min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)}. \tag{6.1}$$

Using the stochastic TCP model and methodology of [99], this thesis generalizes the above result to a scenario with arbitrary values of α and β.[4] Denoting d as $1/\beta$, it could be shown that

$$B \approx \frac{1}{RTT\sqrt{\frac{2bp(d-1)}{\alpha(d+1)}} + RTO\min(1, 3\sqrt{\frac{bp(1+d)(d-1)}{2\alpha d^2}})p(1+32p^2)}. \tag{6.2}$$

[4] A deterministic model for TCP-friendly AIMD congestion control with arbitrary α and β could be found in [101].

The derivation is provided in Appendix C. Note the two corner cases: for $\alpha = 1$ and $\beta = 0.5$, Equations (6.1) and (6.2) are equivalent; when $\beta = 1$ (when $d = 1$), then $B \rightarrow$ inf, i.e., if the congestion window is never decreased upon a packet loss, the throughput will theoretically converge to infinity. The intermediate cases are explored below as follows.

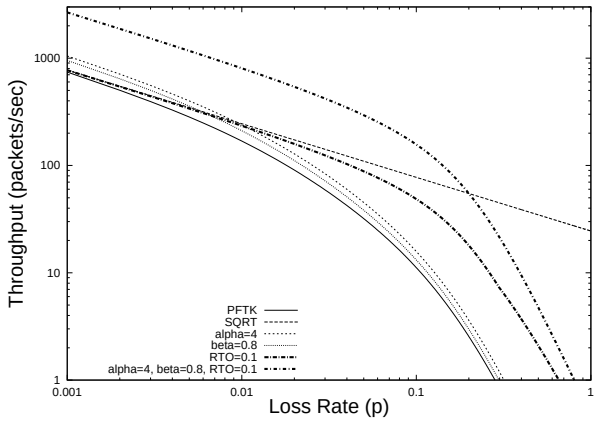


Figure 6.2 : Long-time-scale misbehaviors - numerical results

Figure 6.2 shows numerical results for TCP (and hence RCP) throughput as a function of the packet loss rate. *PFTK* denotes the formula from [99] (Equation (6.1), with $b = 1$ and $RTO = 1$), while *SQRT* is the "square-root" formula from [102] (the same as Equation (6.1), only without the *RTO* part). Next, the figure plots the throughput that a malicious receiver can achieve, according to the Equation (6.2), by manipulating α, β, and *RTO* (exact values are shown in the figure).

First, observe that by re-tuning α to four, one can double the throughput (y-axis is in logarithmic scale), while re-tuning β to 0.8 ($d = 1.25$) one can steal somewhat less bandwidth. More generally, according to Equation (6.2), setting α to a value larger than one, enables a flow to achieve approximately $\sqrt{\alpha}$ higher throughput as compared to a well-behaved TCP flow and for the same packet loss rate. Second,

observe that both curves ($\alpha = 4$ and $\beta = 0.8$) have a shape similar to the *PFTK* curve. This indicates that the amount of stolen bandwidth (the difference between the misbehaving and the *PFTK* curve) is approximately independent of the packet loss ratio. On the other hand, notice that this is not the case for the *RTO* parameter (e.g., $RTO = 100\,\text{ms}$), where the amount of stolen bandwidth increases as the packet loss ratio increases. This is because timeouts occur more frequently in higher packet-loss-ratio environments, and thus, disobeying the exponential backoff rules enables significant throughput gains in such environments. Furthermore, by re-tuning all parameters together ($\alpha = 4$, $\beta = 0.8$, $RTO = 0.1$), the model predicts significant stealing effects, where the misbehaving flow utilizes approximately ten (for $p = 0.02$) to twenty (for $p = 0.1$) times more bandwidth than behaving flows. Finally, observe that the *SQRT* formula significantly overestimates the TCP-friendly rate for higher packet loss ratios (where the exponential backoffs play a key role), hence this formula is not suitable for detection purposes (to be explained in detail below).

Short Time Scales

This section develops an expression for the response time for file download under modifications of the initial congestion window parameter *W*. It models only the exponential increase phase, which is the only phase that the majority of short-lived flows ever enter [72].[5] Section 6.4.2 shows that the expression accurately captures the response times of short files in a web browsing experiment.

The exponential increase phase for receiver-driven TCP is the same as in the sender-driven scenario, with the difference that the receiver has the leading role. It sends the first two *req* packets (the default initial window size is two segments) to

[5]See references [103–105] for more sophisticated models for the latency of *well-behaving* TCP flows.

the sender, which replies with the first two *data* packet. Next, the receiver doubles the congestion window and sends four *req* packets to the sender. Denote T_r as the response time of a regular (behaving) RCP flow, N as the file (flow) size in packets, and RTT as the round-trip time. In such a scenario, the response time for a flow of size N is

$$T_r = \max(RTT, \lceil \log_2 N \rceil RTT). \tag{6.3}$$

The equation indicates that files of length N packets will be downloaded in $\lceil \log_2 N \rceil RTT$ seconds. This is true for $N \geq 2$. On the other hand, the download time for a single-packet-long ($N = 1$) file cannot be smaller than RTT, hence Equation (6.3).

Next, denote T_m as the response time of a malicious flow who sets the initial window size W to a value larger than two. Further, denote s as the packet size in bits and C as the available bandwidth in bits/s. Then, when the file size $N \leq W$, it follows that

$$T_m = \max(RTT, Ns/C). \tag{6.4}$$

In other words, if the initial window size is set to a number larger than the file size, the file will be downloaded in a "single burst," and thus the actual response time equals the burst size, lower-bounded by RTT. Otherwise, if $N > W$, it follows that

$$T_m = \max(RTT, Ws/C) + \lceil \log_2 N - \log_2 W \rceil RTT. \tag{6.5}$$

The first part of Equation (6.5) is similar to Equation (6.4). It says that the first W packets are downloaded in a single burst, whereas the rest are transferred in a "jump-started" exponential increase phase. A simple calculation shows that a misbehaving

user can indeed significantly improve the file response time by manipulating the initial window size parameter. For example, this simple model indicates that for $C = 10\,\text{Mb/s}$, a 70 kByte file can be transferred within a single RTT when the initial window size W is set to 70 or more packets: *seven* times faster than what a behaving flow achieves.

6.2 Network Solutions

This section analyzes several state-of-the art network solutions (both core- and edge-based) designed to detect malicious flows. Common to all solutions is their fundamental limitation to accurately detect such flows due to their lack of the knowledge of the flows' round-trip times.

6.2.1 Core-Router-Based Solutions

This section first considers RED-PD (RED with Preferential Dropping) in detail, and then briefly discusses variants of Fair Queuing (FQ). In the absence of knowledge of flows' round-trip times, the above two schemes penalize flows based on the *absolute* throughput seen at a router, which in a heterogeneous-RTT environment typically means punishing short-RTT flows.

RED-PD

In [12], Mahajan *et al.* develop RED-PD, a scheme that uses the packet drop history at a router to detect high-bandwidth flows in times of congestion, and preferentially drop packets from these flows. In order to detect high-bandwidth flows, RED-PD sets a *target bandwidth*, above which a flow is identified as malicious. The target bandwidth is defined as the bandwidth obtained by a *reference* TCP flow with the *target RTT* (default is 40 ms), and the current drop rate measured at the output router queue.

The targeted bandwidth is computed using the square-root TCP-friendly formula. In other words, in the absence of per-flow RTT measurements, RED-PD sets the target RTT to 40 ms as a bound for distinguishing in- vs. out-of-profile flows.

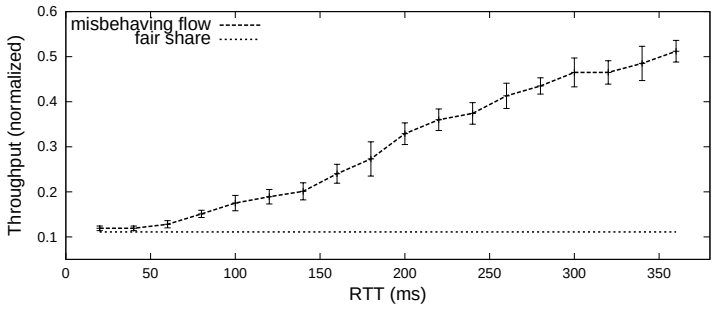


Figure 6.3 : RED-PD is unable to detect a malicious flow

While RED-PD can protect the system against certain misbehaviors, the lack of exact knowledge of the flow's RTT fundamentally limits its ability to detect severe end-point misbehaviors as demonstrated in Figure 6.3. The *ns* experiments with nine flows sharing a RED-PD router are performed. The round-trip times of the flows are varied from 20 to 350 ms (as shown on the x-axis), and the bandwidth of a single flow is plotted on the y-axis. When all flows are well-behaved, the bandwidth share is fair (the straight line in the figure). However, when one of the flows (whose normalized throughput is shown on y-axis) re-tunes α to 25, it can potentially steal up to five times more bandwidth than its fair share according to Equation (6.2). Observe that RED-PD successfully limits the malicious flow to its fair-share, but only when the RTT is less than or equal to 40 ms (recall that this is the RTT of the *reference* flow). However, as the flows' RTT increases, the malicious flow is able to steal more and more bandwidth, up to five times more than its fair share (the maximum for this scenario) when the RTT is 350 ms.

RED-PD's limitations in detecting misbehaving flows are more general than in-

dicated in the above example. First, it is important to notice that a misbehaving flow can steal bandwidth not only in homogeneous-RTT scenarios as in the above experiments, but also in heterogeneous-RTT environments, since the amount of stolen bandwidth depends on the RTT of a misbehaving flow. Second, while this chapter focuses on receiver-driven transport protocols, observe that the above RED-PD limitations apply equally to sender-based TCP stacks. Another problem arises from the fact that RED-PD uses a simple (and less accurate) square-root formula, which significantly overestimates the TCP-friendly rate for higher packet loss ratios because it doesn't account for retransmissions [99]. Hence, malicious TCP or RCP flows have the opportunity to steal dramatically more bandwidth as the packet loss ratio increases, e.g., 100 times more when $p = 0.3$, as indicated in Figure 6.2.

Finally, RED-PD's inability to determine with high confidence if a flow is malicious or not, limits its ability to punish a malicious flow (e.g., to completely starve it). Hence, "stealing pays off" for endpoints as they can freely re-tune their parameters without adverse effects: (i) they will not be completely starved; (ii) they will not utilize less bandwidth than a well-behaving TCP or RCP would; and yet (iii) they can quite often steal significant amounts of bandwidth.

Fair Queuing

While it may appear attractive to apply some version of fair queuing (including the preferential-dropping schemes developed to enforce fairness among adaptive and non-adaptive flows, e.g., Flow Random Early Detection (FRED) [21], CHOKe [20], or Stochastic Fair Blue (SFB) [26]) to solve the above problem, observe that such schemes are also unable to detect end-point misbehaviors and to enforce the proportional fairness targeted by TCP. Moreover, in a heterogeneous RTT environment, such schemes will significantly deviate from the proportional bandwidth share, and

even magnify the bandwidth-stealing effects. A simple, yet illustrative example, is given below. While not representative of an actual or realistic scenario, the main goal is to illustrate the difference between proportional (RTT-dependent) and max-min fairness as enforced by FQ.

Consider a link shared by three congestion-controlled flows, such that the proportional fair share is (0.9,0.05,0.05). Next, assume that flow 2 is malicious. It re-tunes its parameters and utilizes more bandwidth by stealing from flow number one, such that the bandwidth share is now (0.7, 0.25, 0.05). However, if FQ is used, all flows get their "fair-share", and the bandwidth share is now (0.33, 0.33, 0.33). Thus, FQ provides even more bandwidth to flow 2 than it could have stolen without it.

6.2.2 Edge-Router-Based Solutions

This section presents two solutions whose goal is to detect non-TCP-friendly behavior at the network edge. The key advantage of an edge-based vs. a network-based scheme is the opportunity to monitor packets in both directions (*data* in forward, and *ack* in reverse).[6]

D-WARD

In [11], Mirkovic *et al.* develop D-WARD, an edge-router based protection scheme for detecting DoS activity. For each traffic type, they establish a baseline traffic model. For a TCP session, they measure both outgoing (*data*) and incoming (*ack*) traffic and define the maximum allowable ratio of the two. When the ratio of the number of data vs. the number of ack packets goes over a certain threshold, they conclude that the flow is out of profile and rate-limit it.

[6]The *data* and *ack* paths of the same flow may not cross the same *network* router, but typically do cross the same *edge* router.

While the above scheme may indeed protect against TCP-based denial-of-service attacks (where the sender floods the network with data packets independent of the feedback from the receiver), this model clearly doesn't apply to the *receiver-driven* TCP scenario. Recall that in the receiver-based scenario, the number of requests and data packets is the same in both directions, even in the most severe denial-of-service scenarios. Moreover, the fact that the number of packets in the forward (*data*) and reverse (*req*) directions is the same is actually the core idea of the request-flood attack: the receiver floods the sender with requests, and the sender replies by transmitting the same number of data packets, yet with significantly larger size thereby congesting the network.

Tcpanaly

In [27], Paxson presents `tcpanaly`, a tool whose initial goal was to work in *one pass* over a packet trace by recognizing *generic* TCP actions. The goal of executing only one pass stemmed from the objective that `tcpanaly` might later evolve into a tool that could monitor an Internet link in real-time and detect misbehaving TCP sessions on the link. Unfortunately, the author was forced to abandon both of the goals. Among many obstacles, the key one is that one-pass analysis proved difficult due to vantage point issues (see reference [27] for details), in which it was often hard to tell whether a TCP flow's actions were due to the most recently received packet, or one received in the distant past.

6.3 An End-Point Solution

This section evaluates the potential of an end-point scheme to detect receiver misbehaviors. The key advantage of an end-point (vs. network-based) approach is the ability of the sender to estimate the round-trip time and loss rate on the path to

the receiver, and hence enforce a much "tighter" TCP-friendly throughput profile. However, a fundamental problem arises from the fact that in the absence of trust between the sender and receiver, it is problematic for the sender to infer whether the receiver is misbehaving as defined in Section 6.1 or legitimately trying to optimize its performance.

6.3.1 Sender-Side Verification

In order to detect receiver misbehavior, the sender requires increased functionality beyond its role as a slave to the receiver's request packets (see Figure 2.4). The objective is to add the minimum functionality to the sender that will enable it to robustly detect receiver misbehavior over long-time scales (the short-time-scale misbehavior detection problem is treated in Section 6.4.2), yet without *any* help from a potentially misbehaving receiver. While this new functionality inevitably increases the sender-side implementation complexity, it will be demonstrated that it represents a general solution to the bandwidth-stealing receiver-induced misbehaviors.

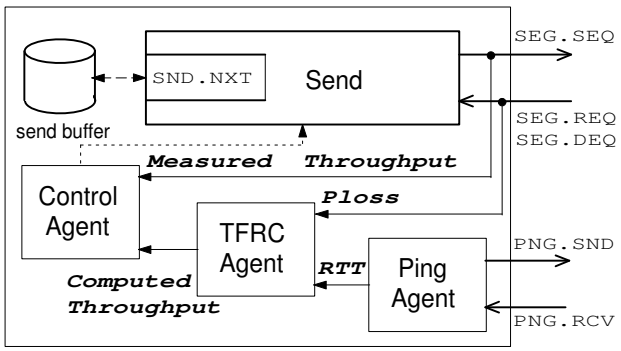


Figure 6.4 : Secure RCP sender

Figure 6.4 depicts the key components of such a solution. Equation (6.1) indicates that knowledge of RTT and packet loss ratio is enough to compute the TCP-fair

throughput, and consequently to detect out-of-profile flows. Unlike in network-based scenarios discussed in Section 6.2, an end-point scheme can measure RTT and the packet loss ratio, and hence enforce a more precise traffic profile than *any* network-based solution.

Because the sender must estimate RTT and packet loss ratio without any cooperation from the untrusted receiver, the sender transmits *ping* packets that the receiver has no incentive to delay, as a larger RTT implies a lower bandwidth profile.[7] Likewise, the sender must estimate the packet loss ratio and detect whether the receiver is actually re-requesting data packets that are dropped. Note that a node performing a DoS attack need not re-request dropped packets, whereas receivers that are stealing bandwidth will be forced to re-request packets for a reliable service. In any case, one possible solution to the above problem is for the sender to purposely drop a packet to test if the receiver will re-request it as the absence of a repeated request for the dropped packet would indicate a potential DoS attack. Note that this is a backward-compatible technique that could be used instead of the proposed *nonce* technique [98]. Nevertheless, this chapter focuses on bandwidth-stealing scenarios where receivers are forced to re-request dropped packets for a reliable service.

Once the RCP sender estimates RTT and the packet-loss-ratio, it can compute the TCP-friendly rate. However, because these parameters can vary significantly during a flow's lifetime, this thesis applies the methods developed for TCP-Friendly Rate Control (TFRC) [4] to estimate the TCP-friendly rate in real time. Namely, while existing use of TFRC focuses on setting the transmission rate based on RTT and loss

[7] If the receiver doesn't reply to the *ping* requests, the sender may either disconnect it, or rate-limit it to a moderate rate. Moreover, to prevent the receiver to simply send a response in anticipation of a request (thus thereby simulating a smaller RTT), the sender should randomize the period between the *ping* messages.

measurements, this thesis utilizes TFRC to *verify* TCP friendliness using the actual RTT (measured via the *ping agent*) and loss measurements incurred by the RCP flow itself.

In [106], Patel *et al.* designed an end-point scheme whose goal is to verify TCP friendliness in the context of untrusted mobile code. The key difference between the end-point scheme presented here and the one from [106] is that the scheme developed here aims to thwart possible *receiver* misbehaviors, and hence does not require any cooperation from a potentially malicious receiver. Moreover, in contrast to the scheme from [106], which compares the TCP sending rate to the TCP-friendly equation rate [99], the secure-RCP scheme applies the TFRC protocol to estimate the TCP-friendly rate in real time. This is particularly important in the presence of highly dynamic background traffic; while being an equation-based scheme, TFRC manages to adapt to relatively short time-scale available-bandwidth fluctuations [70].

Finally, by comparing the measured throughput (based on the number of packets sent) and the throughput computed by the *TFRC agent*, the *control agent* is able to detect, and eventually punish, a misbehaving receiver. This work does not implement the control module, as the primary goal here is to explore the ability of the above scheme to accurately *detect* receiver misbehaviors. Alternatives to punish include rate-limiting and preferentially dropping packets. However, given that the scheme can indeed accurately detect misbehaving receivers (to be shown below), the sender may simply disconnect the misbehaving client, and in that way discourage potentially malicious receivers from the temptation to steal bandwidth.

6.3.2 Detecting Misbehaviors

This section first evaluates the accuracy of the TFRC agent in measuring "TCP friendliness." Next, it re-tunes the RCP parameters at the receiver to mimic malicious

behavior, and then evaluates the sender's ability to detect such misbehaviors.

TFRC Agent

To robustly detect misbehaving receivers, it is essential to first evaluate the TFRC agent's accuracy in measuring TCP friendliness. Computed TFRC throughput may deviate from actual TCP throughput due to measurement errors (low RTT sampling resolution, *ping* packets sent once per second, etc.), system dynamics, and inaccuracies in the underlying TCP equation. Thus, to manage the detection scheme's false positives (incorrect declaration of a non-malicious flow as malicious), such inaccuracies must be incorporated into the detection process.

The *ns* simulation experiments are conducted and a link shared by a number of TCP Sack flows (varied from 1 to 600) is considered. The link implements RED queue management and has capacity 10 Mb/s; the buffer length, min_thresh, and max_thresh are set to 2.5, 0.25 and 1.25 times the bandwidth-delay product, respectively. The round trip time is 50 ms. Unless otherwise indicated, these parameters are used throughout the chapter. A number of simulations are performed, and the average results together with 95% confidence intervals are presented.

To establish a baseline of TFRC's behavior, the TFRC agent is first mounted on the sender side of a *sender-based* TCP Sack [36] flow and present the results in Figure 6.5. The figure depicts the ratio of measured (TCP Sack) vs. computed (by the TFRC agent) throughputs as a function of the packet loss ratio. When the measured vs. computed throughput ratio is one, this indicates that the TFRC agent exactly matches the TCP Sack throughput. Observe that this is indeed the case for low packet loss ratios (for the curve labeled as "TCP Sack"). As the packet loss ratio increases, the curve moderately increases, indicating a slight conservatism of the TFRC agent as the throughput computed by the TFRC agent is slightly lower than

the measured TCP Sack throughput. The problem of TFRC conservatism has been studied in depth in reference [107]. However, this problem is much less pronounced here than indicated in [107] as the TFRC agent measures the actual packet loss ratio incurred by the TCP Sack flow. This ratio is much lower than the loss ratio induced by a TFRC *flow* which backs-off less conservatively than TCP Sack (see reference [107] for further details). In summary, the throughput computed by the *TFRC agent* deviates from the TCP Sack throughput, yet the deviation is moderate, even for high packet loss ratios.

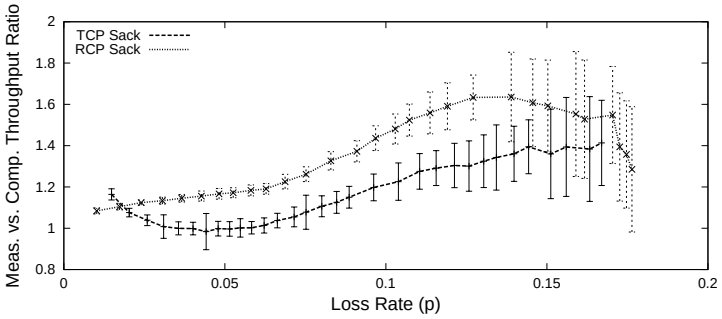


Figure 6.5 : TFRC agent mounted on the sender side of a well-behaved (a) TCP Sack and (b) RCP Sack

Finally, the above experiment is repeated, but now mount the TFRC agent on the RCP sender as in Figure 6.4. Observe that the ratio of the measured (RCP Sack) vs. computed throughput is somewhat higher than in the above sender-based TCP Sack scenario. Indeed, RCP Sack has an improved loss recovery mechanism (see reference [10] for details) and consequently improves throughput. The key problem is the sender side's difficulty in determining whether the receiver is trying to optimize its performance, or is simply stealing bandwidth. This problem is treated in detail in Section 6.3.3. Here, the reference measurement-based profile for a behaving RCP flow is obtained. This profile will next be used to demonstrate the capability of an end-point scheme to detect even moderate receiver misbehaviors.

Detecting Misbehaving Receivers

This section implements a misbehaving RCP node that re-tunes its congestion control parameters α, β, and RTO at the receiver. The goal is to evaluate the sender's ability to detect these misbehaviors and to evaluate the accuracy of the modeling result from Equation (6.2).

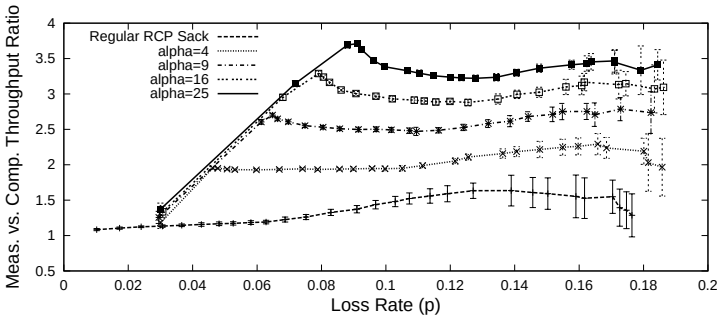


Figure 6.6 : Misbehaving receiver re-tunes the additive-increase parameter α

First, this section re-tunes the additive-increase parameter α and repeats the experiment above. Figure 6.6 depicts the measured vs. computed throughput ratio for misbehaving receivers (having α of 4, 9, 16 and 25), together with the same ratio for the behaving RCP flow having $\alpha = 1$. Recall that the left-most point on the curve corresponds to low loss and experiments in which the RCP flow competes with a single TCP Sack flow, whereas the right-most point on the curve corresponds to high loss and a single RCP flow competing with 600 TCP Sack flows. Observe first that the measured vs. computed throughput ratios for misbehaving flows clearly differ from the behaving flows' profile, indicating a strong potential for misbehavior detection (to be demonstrated below). Second, observe that the throughput ratio for misbehaving flows is approximately proportional to $\sqrt{\alpha}$ as predicted by the model except for extremely low aggregation regimes (e.g., $p = 0.03$ in which a single RCP flow competes with a single TCP Sack flow). In such low aggregation cases, while

the misbehaving flow indeed takes significantly more bandwidth than the competing TCP Sack flow (not shown), it is unable to fully utilize the bandwidth due to frequent backoffs.

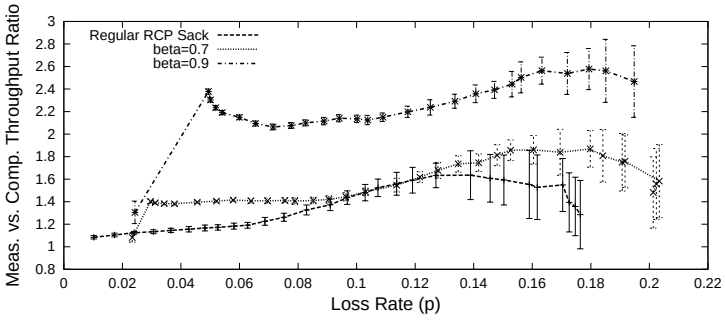


Figure 6.7 : Misbehaving receiver re-tunes the multiplicative-decrease parameter β

Second, this section considers misbehaviors via a re-tuned multiplicative-decrease parameter β and presents the results in Figure 6.7. Note that the curve for $\beta = 0.7$ is close to that of the behaving flow, indicating detection difficulties to be shown below. Also, observe that the curves for misbehavers are shifted to the right when compared to the behaving RCP flows (this also holds for Figure 6.6). This is due to fact that a misbehaving RCP flow increases the ambient packet loss rate.

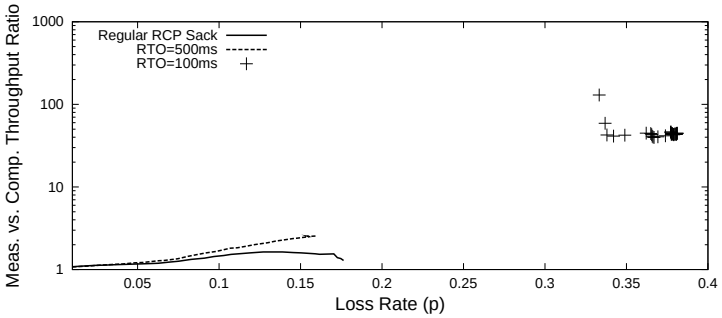


Figure 6.8 : Misbehaving receiver re-tunes the retransmission timeout parameter RTO

Next, this section explores misbehavers that re-tune the retransmission timeout

parameter by simultaneously re-tuning both *minRTO* and *maxRTO* parameters and present the results in Figure 6.8. Notice that when *RTO* is set to 500 ms, the receiver gradually steals more and more bandwidth as the packet loss ratio increases (as predicted by the model), since the number of time-outs increases with the packet loss ratio. However, 500 ms backoffs are sufficient to keep the system stable. On the other hand, observe that when re-tuning the *RTO* parameter to 100 ms (which in this scenario is smaller than the RTT), the system is pushed deeply into a loss regime ($p \approx 0.35$). In such a scenario, the amount of stolen bandwidth is so extreme that it may be characterized as a denial-of-service attack. Indeed, by re-tuning only a few parameters, it is possible to transform RCP (and TCP) into a powerful DoS tool (see Figure 6.1).

Detection Threshold

This section evaluates the sender's ability to detect receiver misbehaviors and study the false-alarm probability and correct misbehavior-detection probability. Denote *meas_thr* as the throughput measured by the RCP sender, and *comp_thr* as the throughput computed by the TFRC agent (as shown in Figure 6.4). Next, denote k as the threshold parameter, and define $P(k)$ as

$$P(k) = Prob(\frac{meas\_thr}{comp\_thr} > k). \tag{6.6}$$

For example, $P(1)$ denotes the probability that the measured vs. computed throughput ratio is larger than one, whereas $P(2)$ is the probability that the the measured throughput is more than twice the computed one. If the receiver is behaving, then $P(k)$ is the *false-alarm* probability (i.e., it is falsely concluded that the receiver is misbehaving with probability $P(k)$). On the other hand, if the receiver is misbehaving, then $P(k)$ is the *correct misbehavior-detection* probability (i.e., it is

correctly concluded that the receiver is misbehaving with probability $P(k)$).

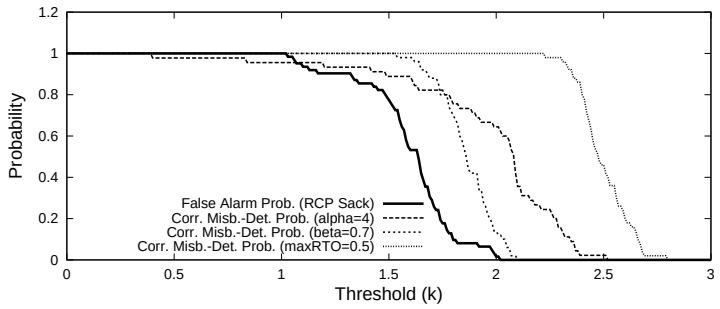


Figure 6.9 : Detecting out-of-profile flows

Figure 6.9 plots the false alarm probability (for the behaving RCP flow), together with the correct misbehavior-detection probabilities for three moderately misbehaving receivers (exact parameters are shown in the figure). The packet loss ratio is set to 0.15 representing a scenario in which the throughput ratio deviates (approximately) the most as indicated in Figure 6.5. Consequently, the false-alarm probability for the behaving RCP flow is largest, indicating that this scenario is the most challenging from the detection point of view.

The key observations from Figure 6.9 are as follows. First, note the tradeoff in setting the threshold parameter k. If it is too small (e.g., $k = 1$), it is possible to detect the misbehaving receivers with high probability, but the false alarm probability is also one. On the other hand, if it is set too high (e.g., $k = 3$), the false alarm probability becomes zero, but the correct misbehavior-detection probability also becomes zero. However, observe that the fact that the false-alarm probability decreases faster (for smaller k), makes it possible to set the threshold (e.g., $k = 1.8$ in this scenario), such that the false positives are acceptably small, yet it is possible to detect *all* of the above cheaters with high probability. Thus, this *worst-case* scenario confirms the high precision of the end-point scheme in detecting a wide range of receiver

misbehaviors.[8] However, it will be shown next that setting the parameter k incurs an additional challenge when confronted with versions of TCP employing performance enhancements.

6.3.3 Advanced Congestion Control Mechanisms

This section studies RCP stacks that implement advanced congestion control mechanisms, e.g., Explicit Loss Notification (ELN) [95, 108] or Westwood [96]. While such mechanisms can significantly improve throughput [10], they introduce a fundamental receiver-misbehavior detection problem. In essence, it is problematic from the network endpoint (sender side) to distinguish between a malicious receiver that legitimately optimizes its performance by applying an advanced congestion control mechanism and a malicious receiver, who unscrupulously steals bandwidth from other flows in the network. This section first provides a brief background on advanced congestion control mechanisms and their performance in the receiver-driven protocol scenarios, and then discusses in detail the misbehavior detection problem.

There is a significant body of work proposed to improve the TCP performance in wireless environments, where high channel losses may disproportionately degrade TCP Sack performance. Here, this section explains two well-known protocols, TCP-ELN and TCP Westwood. TCP-ELN has been proposed to distinguish wireless random losses from congestion losses. It relies on an external trigger to classify the losses, and fast retransmits lost segments due to wireless errors without decreasing down the congestion window. It has been shown in [10] that when this mechanism is applied in the *receiver-driven* protocol scenario, the throughput improvements are

[8] While misbehaviors other than parameter modification may indeed occur, this research focuses on parameter-based misbehaviors because they are easy to implement; other misbehaviors (e.g., algorithmic) will be out-of-profile anyway.

quite significant (this thesis repeats this experiment and confirms the result below). This is mostly due to the fact that RCP-ELN benefits from having accurate loss classifications about all missing segments in the receive buffer.

Another protocol that significantly improves the throughput over wireless links is TCP Westwood. It does so by using a less conservative decrease parameter β that depends on the online estimate of the available bandwidth. In this way, TCP Westwood avoids significant throughput losses due to link errors. It is expected that the same mechanism could provide further throughput improvements in receiver-driven protocols. Below, this thesis focuses on RCP-ELN and does not further consider sender- or receiver-based TCP Westwood.

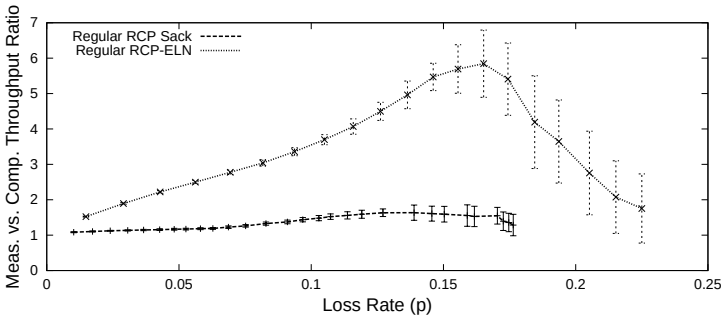


Figure 6.10 : RCP-ELN significantly improves throughput

This section first presents the simulation of an RCP-ELN flow in a lossy wireless-like environment. Figure 6.10 depicts the measured vs. computed throughput ratio as a function of loss. Observe that the RCP-ELN throughput ratio increases significantly as compared to the RCP Sack profile, indicating that RCP-ELN indeed significantly improves throughput, e.g., achieving a six-fold increase for a loss ratio of 0.17. However, the key problem is that from the sender perspective, the RCP-ELN flow is difficult to distinguish from a misbehaving flow.

Figure 6.11 depicts the false-alarm probability for the behaving RCP-ELN flow for

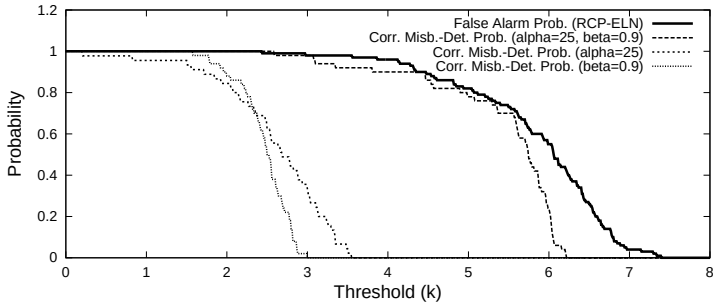


Figure 6.11 : From the sender's perspective, RCP-ELN looks like a misbehaving flow

a packet loss ratio of 0.15. To emphasize the detection problem, the figure also plots the correct misbehavior detection probabilities (without any advanced congestion control mechanisms), with maliciously re-tuned parameters (i) $\alpha = 25$, (ii) $\beta = 0.9$, and (iii) $\alpha = 25$ and $\beta = 0.9$. Observe that using a small threshold (e.g., $k = 1$) ensures a high detection probability for any of the above misbehaviors, but the RCP-ELN is also falsely detected as malicious. However, simply increasing the threshold k does *not* eliminate the problem. For example, for $k = 4$, the false alarm probability for ELN-RCP is still one, while the probability to detect misbehaviors (i) and (ii) has already dropped to *zero*. Finally, by using a very large k (e.g., $k = 7$ in this scenario), the false alarm probability for RCP-ELN becomes acceptably small, but it the scheme is unable to detect any of the (quite severe) receiver misbehaviors.

Thus, these experiments illustrate a fundamental tradeoff between system performance and security (the ability to detect bandwidth stealers), as both cannot be maximized simultaneously. Ironically, while advanced congestion control mechanisms at the receiver significantly improve throughput, the resulting false-alarm probability further increases, further emphasizing the tradeoff. This thesis argues that setting the parameter k to a larger value strikes the best balance for the file- or streaming-servers in the Internet. A large value protects servers from severe denial-of-service

attacks, while enabling innovation in protocol design by preserving the performance benefits of receiver-centric transport protocols. The downside is the fact that the scheme is unable to detect some bandwidth stealers. In contrast, strictly enforcing today's TCP-Sack throughput profile via a lower k would indeed make it possible to catch even modest bandwidth stealers. However, a small k would remove most of the RCP benefits, and indeed remove the incentive for designing and deploying enhanced TCP stacks.

6.4 Short Time Scale Misbehavior

The secure RCP sender is designed to detect receiver manipulations of congestion control parameters (e.g., α, β, RTO) that would enable the receiver to steal bandwidth over longer time periods. Hence, these misbehaviors can be detected on longer time-scales. This section explores the minimum time scale for which the sender can accurately identify receiver misbehavior. Moreover it studies a receiver misbehavior targeted towards short-lived flows in which receivers begin with a large initial congestion window.

6.4.1 Minimum Detection Timescales

To explore the minimum detection time-scale, this work first performs ten experiments, and the results are shown in Figure 6.12. In all experiments, a single RCP flow competes with 20 TCP Sack cross-traffic flows. Figure 6.12 depicts the measured vs. computed throughput ratio (measured at the RCP sender) as a function of time, where the reference time zero identifies the start time of the RCP flow. In five of the ten experiments, the RCP receiver behaves well (only the random seed is changed for each simulation run), while in the remaining experiments a malicious receiver with

$\alpha = 9$ is created.

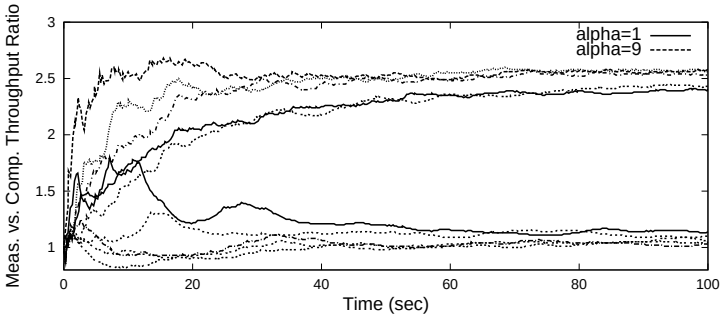


Figure 6.12 : Throughput ratio vs. time for (a) a well-behaving flow and (b) a misbehaving flow ($\alpha = 9$) (for five different random seeds)

Observe that the ratios for both of the stacks (behaving and malicious) converge relatively quickly: toward one for the behaving flows, and approximately to $\sqrt{\alpha}$ for the misbehaving flows. However, note that the curves for the two stacks can be quite similar, and may overlap, over short time scales. The overlaps are due to the fact that a behaving RCP flow (just like a TCP flow) can be quite bursty over shorter time-scales (e.g., due to the exponential increase phase), and thus may deviate from the TCP-friendly rate computed by the TFRC agent.

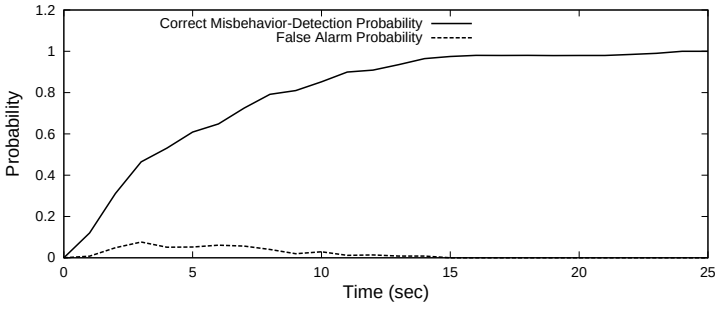


Figure 6.13 : Probability to detect a misbehaving flow increases as the time evolves

Next, this section presents an extensive set of simulations to statistically quantify the above observations. Figure 6.13 depicts the correct misbehavior detection proba-

bility (for the misbehaving flow with $\alpha = 9$), together with the false alarm probability (for the behaving flows) as a function of time and for $k = 2$. Observe first that the correct misbehavior-detection probability converges to one as time evolves, indicating that it becomes more and more certain that the receiver is misbehaving. On the other hand, observe that the false-alarm probability for the same scenario is quite low (only several percent up to 10 sec) and approaches zero beyond 10 sec. Thus, beyond this time scale, it is possible to detect the receiver misbehavior with high confidence, and the sender can freely punish the receiver given that the probability to falsely detect a behaving flow drops to near zero beyond 10 sec.

However, very short-lived flows transmitting up to tens or hundreds of packets are common in today's Internet due to web traffic. The file transmission times typically last for only several *ms* to several hundreds of *ms*, and the above scheme (targeted to detect bandwidth stealers in file- or streaming-server scenarios) is not designed to detect very short time-scale misbehaviors. Below, this work first explores additional short time-scale receiver misbehaviors targeted for web-browsing and short files, and then analyzes appropriate protection mechanisms.

6.4.2 Initial Congestion Window

This section considers web RCP flows that increase their initial congestion window in order to obtain decreased response time. It will be shown that it is possible for a malicious receiver to not only significantly improve its own response time, but to also severely degrade the response times for the background traffic.

Figure 6.14 shows the simulation scenario. This thesis adopts the model developed in [71], which is described in detail in Chapter 5. Here, the inter-page and inter-object time distributions are exponential, while the means are changed in different experiments such that the utilization on the bottleneck link varies from 10% to 90%.

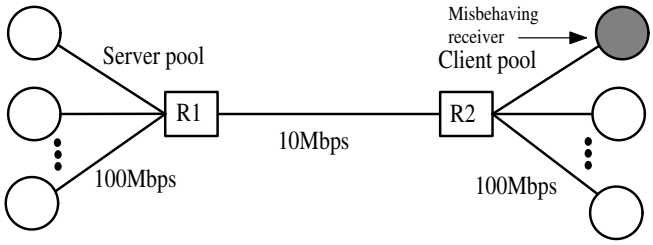


Figure 6.14 : Web simulation scenario

There is a single misbehaving client in the client pool, which uses a mis-configured RCP (details are given below), while the other clients from the pool behave and use unmodified TCP Sack.

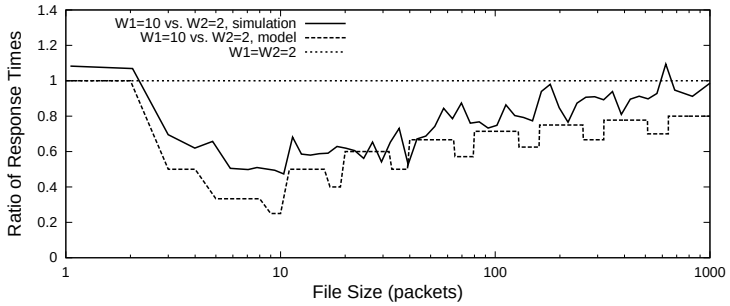


Figure 6.15 : Misbehaving receiver re-tunes the initial window size parameter W (link utilization 10%)

Figure 6.15 depicts the average file response time for the RCP flow (normalized by the response times for the same flow when the RCP client is well behaving) as a function of file size. Because of the normalization, the curve labeled as "W1=W2=2" is a straight line with a value of one. On the other hand, notice that the misbehaving RCP client is able to significantly improve its response times by increasing the initial window size parameter W to 10. Observe next that the malicious receiver achieves the maximum improvement exactly for the files that are 10-packets long, and this is because such files are downloaded in a single burst (files with size less than 10 packets

are also downloaded in a single burst, but the improvement is most prominent for the longest files in this single-burst-category). On the other hand, files longer than 10 packets also improve their response times, simply due to the fact that their congestion windows are jump-started with $W = 10$. Next, observe that the modeling result from Section 6.1.2 accurately tracks the simulation results.[9] The non-monotonic and alternating quasi-periodic shape of the modeling curve is due to the use of the ceiling function ($\lceil . \rceil$) in Equations (6.3) and (6.5).

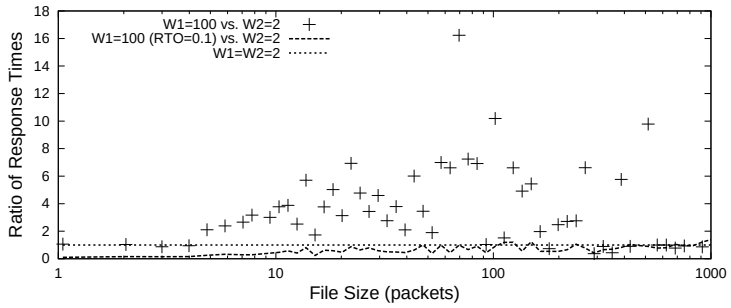


Figure 6.16 : A greedy receiver ($W = 100$) may degrade its own response times; but "turning off" the backoff timers ($W = 100$, $RTO = 0.1$) "improves" the response times (link utilization 90%)

While it may appear attractive for a malicious client to maximally increase the initial window size parameter W in order to steal more and more bandwidth, this is not necessarily a good option, especially in more congested environments. This is illustrated in Figure 6.16, where the link utilization is increased to 90%, and the malicious clients sets the initial window size parameter W to 100 packets. Here, this greedy user significantly degrades not only the background traffic (not shown), but also degrades its *own* response times (shown in the figure) by an order of magnitude. This degradation is due to the fact that when the malicious user sends large bursts

[9]In Equations (6.4) and (6.5), C is set to 10 MB/s, which due to the low average utilization of 10%, is close to the flow's available bandwidth.

of requests, it forces the web server to reply with large bursts of data packets, many of which are themselves lost in the congestion. These packet losses force even the RCP user to enter the exponential backoff phase and degrades its response time. To overcome the above problem, the malicious user needs to "turn off" the exponential backoff timers. This is done by re-tuning the *RTO* parameter to 100 ms. In this way, the malicious user is able both to "push-out" and significantly degrade the background traffic, and at the same time improve its own response times, as also shown in the figure.

6.4.3 Solutions

This section explores two possible solutions to the above short-time-scale misbehaviors. One is to rate-limit flows, which while effective in thwarting cheaters, is a non-work conserving solution in which it is problematic to determine the appropriate rate. The second solution is to have a "smart" RCP client at the sender side that would enforce a "TCP-friendly" exponential window increase. It would estimate the RTT to the client, and release the *data* packets accordingly. While also effective in thwarting cheaters, this approach unfortunately mitigates some of the benefits of RCP.

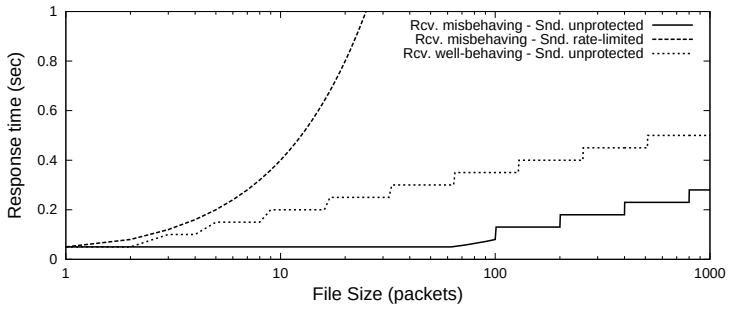


Figure 6.17 : Protecting against short-time-scale misbehaviors

To study the performance of the above solutions, Figure 6.17 plots the file-response times in three different scenarios for the RCP flow with the available bandwidth of 10 Mb/s and RTT of 50 ms: (i) when a malicious user sets the initial window W to 100 packets and the sender does not rate limit (labeled as "Rcv. misbehaving - Snd. unprotected"); (ii) the receiver sets $W = 100$, but the sender rate limits to 200 kb/s (labeled as "Rcv. misbehaving - Snd. rate-limited") and (iii) the receiver is well behaving and is not rate-limited (labeled as "Rcv. well-behaving - Snd. unprotected"). Figure 6.17 illustrates problems in setting the rate-limit value. Setting it to 200 Kb/s degrades the file response times significantly, as shown in Figure 6.17.

But the key insight from the above experiment is that using a large initial window sizes can significantly (up to *ten* times in the above scenario - and much more in larger-bandwidth networks) improve file response times. Such methodologies have been studied in depth in [91, 109–111], but in the context of sender-based TCP, where the web-server increases the initial window size in an attempt to improve system performance. However, in the receiver-driven RCP scenario, it is hard to distinguish whether the receiver is jump-starting the TCP flow or is simply malicious. Thus, applying rate limiting or the "smart" RCP client methodology may indeed protect the system against receiver misbehavior, but at the same time prevents attempts as in [91, 109–111] to improve performance. This illustrates the tradeoff between system security and performance in that strict enforcement of protocol rules would not only reduce performance, but would also inhibit protocol innovation.

However, unlike in the RCP-ELN scenario (in which it is concluded that using larger values for the detection threshold parameter k can protect against DoS attacks, but not from relatively moderate bandwidth stealing), here, the conclusion is that either rate-limiting or a "smart" RCP client has to be *strictly* applied, because a receiver with an excessively large W in combination with manipulated exponential

backoff timers can significantly degrade the legitimate background traffic (Figure 6.16). Yet, applying any of the short-time-scale protection methodologies inevitably reduces the incentive for receivers to use RCP for short-lived flows, as sender-based TCP enhanced with jump-starting methodologies is able to achieve the best response-time curve from Figure 6.17 without any security considerations.

Chapter 7

Conclusions and Future Work

This chapter summarizes the major results of this thesis and then discusses some of the directions for the future work.

7.1 Conclusions

This thesis designed, implemented, and evaluated a series of edge-based algorithms and protocols for efficient inference and control of the Internet from its endpoints. The proposed solutions together form a new foundation for a robust quality-of-service communication via a scalable edge-based architecture where the novel functionality is added strictly at either edge routers or end hosts. The major findings of this thesis are summarized below.

Chapter 3 developed a framework for clients of multi-class services to assess a system's core QoS mechanisms. The thesis developed a scheme for clients to perform a series of hypothesis tests across multiple time scales in order to infer the request service discipline among class-based weighted fair queuing, earliest deadline first, and strict priority. The scheme can be applied to any other scheduler for which a statistical service envelope is derived. For a particular scheduler, this work devised techniques for clients to obtain maximum likelihood estimations of the system's class differentiation parameters, such as WFQ weights and EDF delay bounds. Finally, this research showed how parameters of non-work-conserving elements such as rate limiters can be estimated.

The thesis further evaluated the methodology in a two-class setting in both networking and QoS web-server scenarios. The main findings of this part of the thesis are as follows. **(i)** For networks, the results show high accuracy in both scheduler inference and unknown parameter estimation. **(ii)** For web servers, the inference scheme achieves high accuracy provided that the variability of service times due to factors such as different CPU processing times, disk service times and variable file sizes is not significantly larger than the service variability due to the other class' workload. In both cases, the scheme utilized a general multiple-time-scale traffic and service model to characterize a broad set of behaviors within a unified framework. **(iii)** The inference techniques developed in Chapter 3 are generally applicable and computationally feasible up to a moderate number of service classes.

Chapter 4 presented TCP-LP, a protocol designed to achieve low-priority service (as compared to the existing best-effort class) from the network endpoints. TCP-LP allows low-priority applications such as bulk data transfer to utilize excess bandwidth without significantly perturbing non-TCP-LP flows. TCP-LP is realized as a sender-side modification of the TCP congestion control protocol and requires no functionality from the network routers nor any other protocol changes. Moreover, TCP-LP is incrementally deployable in the Internet. The thesis presented an extensive set of *ns-2* simulations and Internet experiments and showed that **(iv)** TCP-LP is largely non-intrusive to TCP traffic (including very large aggregation regimes) while at the same time, TCP-LP flows can successfully utilize a large portion of the excess network bandwidth. **(v)** In practice, significant excess capacity is available even in the presence of "greedy" long-lived FTP/TCP flows due to factors such as ACK delays from reverse traffic. **(vi)** Competing TCP-LP flows share excess bandwidth fairly. **(vii)** File transfer times of best-effort web traffic are significantly reduced when long-lived bulk data transfers use TCP-LP rather than TCP. **(viii)** Despite their low-priority

nature, even longer-RTT TCP-LP flows are able to utilize substantial amounts of spare available bandwidth in a wide-area network environment.

Chapter 5 presented low-rate denial of service attacks that are able to throttle TCP flows to a small fraction of their ideal rate while transmitting at sufficiently low average rate to elude detection. This work showed that by exploiting TCP's retransmission timeout mechanism, TCP exhibits null frequencies when multiplexed with a maliciously chosen periodic DoS stream. The thesis developed several DoS traffic patterns (including the minimum rate one) and through a combination of analytical modeling, an extensive set of simulations, and Internet experiments it showed that **(ix)** low-rate DoS attacks are successful against both short- and long-lived TCP aggregates and thus represent a realistic threat to today's Internet; **(x)** in a heterogeneous-RTT environment, the success of the attack is weighted towards shorter-RTT flows; **(xi)** low-rate periodic open-loop streams, even if not maliciously generated, can be very harmful to short-RTT TCP traffic if their period matches one of the null TCP frequencies; and **(xii)** both network-router and end-point-based mechanisms can only mitigate, but not eliminate the effectiveness of the attack.

The thesis further concluded that the underlying vulnerability is not due to poor design of DoS detection or TCP timeout mechanisms, but rather to an inherent tradeoff induced by a mismatch of defense and attack time-scales. Consequently, to completely defend the system in the presence of such attacks, one would necessarily have to significantly sacrifice system performance in their absence.

Finally, Chapter 6 analyzed the class of receiver-driven transport protocols that delegate key control functions to receivers. While this radically new protocol design achieves significant performance and functionality gains in a variety of wireless and wireline scenarios, this work showed that a high concentration of control functions available at the receiver leads to an extreme vulnerability. Namely, receivers would

have both the means and incentive to tamper with the congestion control algorithm for their own benefits. This thesis analyzed a set of easy-to-implement receiver misbehaviors and analytically quantified the substantial benefits that a malicious client can achieve in terms of stolen bandwidth over long time-scales (e.g., in file- or streaming-server scenarios) and response time improvements for short-files in HTTP scenarios.

This thesis further evaluated a set of state-of-the-art network-based solutions, and proposed and analyzed a set of end-point solutions. The main findings of this part of the thesis are as follows. **(xiii)** Network-based solutions are fundamentally limited in their ability to detect and punish even severe endpoint misbehaviors. **(xiv)** End-point solution can accurately detect long-time-scale receiver misbehaviors and strictly enforce the TCP-friendly rate, but such enforcement entirely *removes* the performance benefits of receiver-driven protocols. **(xv)** In the file- and streaming-server scenarios, it is possible to strike an acceptable balance between protocol performance on one hand, and vulnerability to misbehavers on the other, due to the fact that moderate bandwidth stealers do not represent a critical threat to the system security. **(xvi)** On the contrary, short time-scale receiver misbehaviors can extremely degrade the response times of well-behaving clients in the HTTP-server scenarios; hence, such servers have to *strictly* apply sender-based short-time-scale protection mechanisms; unfortunately, such mechanisms can often limit the receiver-driven TCP performance to a level which is *below* the level achievable by sender-based TCP.

7.2 Future Work

In the end, this thesis identifies several research directions for future work. The first direction merges more closely the concepts of end-point inference and control; the second direction promotes the design of robust and dependable network services.

7.2.1 Adaptive Networking

While a significant amount of work has been done in designing better and better end-point algorithms (e.g., TCP congestion control), the Internet keeps bringing a wide range of stresses, from new applications such as streaming or multicast to diverse environments with vastly different link capacities (ranging from several kb/s up to several Gb/s per flow), congestion levels, etc. As a new network technology or environment (wireless, high-speed, asymmetric, etc.) evolves, new protocols are developed to improve the system performance by taking advantage of some unique network-environment property. However, a single algorithm or mechanism is unlikely to be uniformly applicable to all network environments. On the other hand, it is impossible for network clients to "tune" their protocol parameters whenever they change the network or the bottleneck of their end-to-end path changes location.

The above scenario makes a case for a new generation of protocols that would be able not just to detect a specific network environment, but to determine the *dominant network state* (e.g., based on the measured round-trip time, available bandwidth, packet loss ratio, a specific packet-loss pattern, etc.), and to adapt to it by applying the appropriate mechanism for that state. Necessarily, such control protocols would need sophisticated inference algorithms, able not just to determine the specific network state, but to detect and track possible state changes (e.g., due to bottleneck hoping), and locally optimize performance objectives such as throughput or fairness. Such globally-applicable hybrid protocols would not only more firmly merge the concepts of inference and control, but would push the concept of protocol adaptation to a completely new level.

7.2.2 Building DoS-Resilient Network Services

While this thesis focused on the vulnerabilities of end-point protocols - such vulnerabilities, unfortunately, exist in almost each and every existing or emerging network protocol, architecture, or technology. Examples of potential vulnerabilities are the ones recently explored in peer-to-peer systems (e.g., [112]), ad-hoc networks (e.g., [113]), or routing protocols (e.g., [114]). The main cause of such vulnerabilities comes from the fact that network services were built targeting functionality, scalability, and efficiency, but much less security and DoS-resiliency. The key reason for such a design philosophy is an assumption of global cooperation that is increasingly invalid today. However, the fundamental problem arises from the fact that cooperation among different entities in a network or a distributed system is not an "unnecessary design assumption", but indeed an essential requirement for a proper system functioning. Hence, building robust network services remains to be an important goal of the future Internet.

This thesis promoted a *pro-active* approach in solving the above end-point misbehavior problems. Such an approach preferences active discovery of possible system vulnerabilities versus only reacting to the problems once they actually happen. The pro-active approach is generally applicable to non-end-point-based vulnerabilities, and is certainly valuable as it enables us to discover and prevent new classes of denial-of-service attacks before they become widely exploited. The second promissory approach in solving the above problems (that has not been applied in this thesis) seems to be the one that employs the concepts of *cooperation* and *coordination* among the well-behaving entities in detecting the misbehaving ones. Indeed, if the majority of the communication parties are well-behaving, then their coordinated effort in detecting misbehaviors should be more successful then if they act independently. But on

the other hand, nothing stops the misbehaving parties to also cooperate and coordinate their efforts in launching DoS attacks or achieving certain performance benefits. Such activities, their potentials, and possible consequences are still left very much unexplored. Hence, the "war" between DoS attackers and defenders continues, and the final outcome is still uncertain. In any case, this remains to be an active and interesting research area.

Appendix A

Summary of Notation from Chapter 3

a_j^i	arrival time of request j in class i
d_j^i	departure time of request j in class i
ϕ_i	relative weight of class i in WFQ scheduler
r_i	rate limit bound of class i
C	deterministic service capacity
$S^i(t)$	class i's theoretical service envelope over intervals of length t
$B^i(t)$	class i's theoretical arrival envelope over intervals of length t
$A^i[s, s+t]$	total class i's arrivals in the interval [s,s+t]
δ_i	class i's delay bound for EDF scheduler
T	measurement window
I_k	interval length ($I_k = kI_1$)
$R_{k,j}^{i,A}$	class i's empirical arrival rate in the $[s + (j-1)I_k, s + jI_k]$ interval
N_k	number of successive intervals of length I_k in the measurement window T
$\bar{R}_k^{i,A}$	mean of the empirical arrival *rate* envelope of class i for intervals of length I_k
$RV_k^{i,A}$	variance of the empirical arrival *rate* envelope of class i for intervals of length I_k
$U[s, s+t]$	service received during *backlogging* interval $[s, s+t]$
$M^S(t)$	service *rate* received in the backlogging interval $[s, s+t]$
$\vec{M}_k^S$	vector of empirical service rates over backlogging intervals of length I_k

Table A.1 : Notation from Chapter 3 - Part I

$\vec{C_k}$	vector of empirical *aggregate* service rates over backlogging intervals of length I_k
C_k	aggregate rate service envelope over intervals of length I_k
$\bar{C}_k$	mean of C_k
CV_k	variance of C_k
$\vec{R}_k^{i,S}$	vector of empirical class i's service rates over backlogging intervals of length I_k
$p_{S_k^i}^{SCH}(\cdot)$	pdf of $S^i(I_k)$ for scheduler SCH and for constant aggregate service capacity
$\bar{D}_i$	mean delay of class i requests
$p_{C_k I_k}(\cdot)$	pdf of the aggregate service envelope in time scale I_k
$\tilde{p}_{S_k^i}^{SCH}(\cdot)$	pdf of $S^i(I_k)$ for scheduler SCH and for variable aggregate service capacity
$\hat{\phi}_{i,k}$	MLE of ϕ_i in time scale I_k
$\hat{\phi}_i$	final estimate of ϕ_i
$\hat{r}_k^i$	MLE of class i's rate limit bound r_i in time scale I_k
$\gamma_{k,i}$	rate variance ratio for class i and time scale I_k
γ^*	threshold for rate variance ratio

Table A.2 : Notation from Chapter 3 - Part II

Appendix B

Computing the throughput of a TCP aggregate under the *shrew* attack

Assume that an initial outage causes all TCP flows to enter the retransmission timeout and assume that $T = b$. Then, the throughput of the TCP aggregate can be computed as

$$\rho(T = b) = \frac{b - E(x)}{b}, \tag{B.1}$$

where $E(X)$ denotes expected value of a random variable X which corresponds to an event that at least one TCP flow's timeout expired at time x, $x \in [a, b]$. Assuming that each TCP flow's minRTO is uniformly distributed between a and b, the CDF of X becomes

$$P(X \leq x) = 1 - (\frac{b - x}{b - a})^n. \tag{B.2}$$

Denoting the corresponding pdf of random variable X as $p(x)$, it follows that

$$p(x) = \frac{\partial P(X \leq x)}{\partial x} = n\frac{(b - x)^{n-1}}{(b - a)^n}. \tag{B.3}$$

The expected value of X, $E(X)$ can be computed as

$$E(X) = \int_a^b xn\frac{(b - x)^{n-1}}{(b - a)^n}\, dx. \tag{B.4}$$

The integral from Equation (B.4) can be solved by using integration by parts with the substitutes $n\frac{(b-x)^{n-1}}{(b-a)^n} = dv$ and $x = u$. The solution is $E(X) = a + \frac{b-a}{n+1}$. Thus,

based on Equation (B.1), it follows that Equation (5.7) holds.

Appendix C

Computing the throughput of a misconfigured RCP/TCP flow

This section applies exactly the same assumptions, methodology, and notation as in [99]. A reader interested in following the derivation below needs to use reference [99] in parallel.

Loss indications are exclusively "triple-duplicate" ACKs:

Assume that a user increases window size by α packets, and that it decreases it by a factor of β. Denote d as $1/\beta$. Then, Equation (7) from [99] becomes

$$W_i = \frac{W_{i-1}}{d} + \frac{X_i}{b/\alpha}, i = 1, 2, ... \tag{C.1}$$

where X_i is the number of increase rounds in the i-th tripple-duplicate period (TPD_i). Equation indicates that during TDP_i, the window size increases between W_{i-1}/d and W_i, and the increase is linear with slope α/b. Consequently, the number of packets transmitted in TPD_i is expressed by

$$Y_i = \frac{X_i}{2}(\frac{W_{i-1}}{d} + W_i - \alpha) + \beta_i, \tag{C.2}$$

where β_i is the number of packets sent in the last round. Next, assuming that X_i and W_i are mutually independent sequences of i.i.d. random variables, it follows from the above two equations and Equation (5) from [99] that

$$E[W] = \frac{\alpha}{b}\frac{d}{d-1}E[x], \tag{C.3}$$

and,

$$\frac{1-p}{p} + E[W] = \frac{E[X]}{2}(\frac{E[W]}{d} + E[W] - \alpha) + \frac{E[W]}{2}. \tag{C.4}$$

From Equations (C.3) and (C.4), it follows that

$$E[W] = \frac{d\alpha}{1+d}\frac{b(d-1)+d}{2b(d-1)} + \\ + \sqrt{(\frac{b(d-1)+d}{2b(d-1)})^2(\frac{d\alpha}{1+d})^2 + 2\frac{\alpha\, d^2}{b(1+d)(d-1)}\frac{1-p}{p}}. \tag{C.5}$$

Observe that,

$$E[W] = \sqrt{\frac{d^2}{(1+d)(d-1)}\frac{2\alpha}{bp}} + o(1/\sqrt{p}). \tag{C.6}$$

i.e., $E[W]$ converges to the first term of Equation (C.6) for small values of p. From Equations (C.3), (C.8) as well as Equation (6) from [99], the expressions for the expected number of rounds ($E[X]$) in the TD period, as well as the expected duration $E[A]$ of the same period are derived. A simplified expression for $E[X]$ is

$$E[X] = \sqrt{\frac{2b(d-1)}{(1+d)p\alpha}} + o(1/\sqrt{p}). \tag{C.7}$$

and,

$$E[A] = RTT(\frac{b(d+1)+d}{2(1+d)} + \\ + \sqrt{(\frac{b(d+1)+d}{2(1+d)})^2 + \frac{1-p}{p}\frac{2b}{1+d}\frac{d-1}{\alpha}} + 1). \tag{C.8}$$

Finally, based on Equation (18) from [99], as well as $E[X]$ and $E[A]$ derived here, it could be shown that the RCP (TCP) throughput $B(p)$ is

$$B(p) = \frac{1}{RTT}\frac{1}{\sqrt{\frac{2bp(d-1)}{\alpha(d+1)}}} + o(1/\sqrt{p}). \tag{C.9}$$

Loss indications are "triple-duplicate" ACKs and timeouts:

Using Equations (25) and (28) from [99], as well as Equations (C.6) and (C.7) from above, it could be shown that Equation (6.2) follows.

Bibliography

[1] J. Saltzer, D. Reed, and D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 195–206, Nov. 1984.

[2] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *Proceedings of ACM SIGCOMM '99*, Vancouver, British Columbia, Sept. 1999.

[3] D. Clark, M. Lambert, and L. Zhang, "NETBLT: A high throughput transport protocol," in *Proceedings of ACM SIGCOMM '87*, Stowe, VM, Aug. 1987.

[4] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proceedings of ACM SIGCOMM '00*, Stockholm, Sweden, Aug. 2000.

[5] P. Mehra, A. Zakhor, and C. De Vleeschouwer, "Receiver-driven bandwidth sharing for TCP," in *Proceedings of IEEE INFOCOM '03*, San Francisco, CA, Apr. 2003.

[6] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "WTCP: A reliable transport protocol for wireless wide-area networks," in *Proceedings of ACM MOBICOM '99*, Seattle, WA, Aug. 1999.

[7] N. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. Bershad, "Receiver based management of low bandwidth access links," in *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.

[8] V. Tsaoussidis and C. Zhang, "TCP-Real: Receiver-oriented congestion control," *The Journal of Computer Networks*, vol. 40, no. 4, pp. 477–497, Apr. 2002.

[9] R. Gupta, M. Chen, S. McCanne, and J. Walrand, "A receiver-driven transport protocol for the web," in *Proceedings of INFORMS '00*, San Antonio, TX, Nov. 2000.

[10] H.-Y. Hsieh, K.-H. Kim, Y. Zhu, and R. Sivakumar, "A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces," in *Proceedings of ACM MOBICOM '03*, San Diego, CA, Sept. 2003.

[11] J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the source," in *Proceedings of IEEE ICNP '02*, Paris, France, Nov. 2002.

[12] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[13] A. Kuzmanovic and E. Knightly, "Measuring service in multi-class networks," in *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, Apr. 2001.

[14] A. Kuzmanovic and E. Knightly, "Measurement-based characterization and classification of QoS-enhanced systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 671–685, July 2003.

[15] A. Kuzmanovic and E. Knightly, "TCP-LP: A distributed algorithm for low priority data transfer," in *Proceedings of IEEE INFOCOM '03*, San Francisco, CA, Apr. 2003.

[16] A. Kuzmanovic and E. Knightly, "TCP-LP: Low-priority service via end-point congestion control," Submitted to *IEEE/ACM Transactions on Networking*.

[17] A. Kuzmanovic, E. Knightly, and R. L. Cottrell, "HSTCP-LP: A protocol for low-priority data transfer in high-speed high-RTT networks," in *Proceedings of PFLDnet '04*, Argonne, IL, Feb. 2004.

[18] A. Kuzmanovic and E. Knightly, "Low-rate TCP-targeted denial of service attacks (the shrew vs. the mice and elephants)," in *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, Aug. 2003.

[19] A. Kuzmanovic and E. Knightly, "A performance vs. trust perspective in the design of end-point congestion control protocols," in *Proceedings of IEEE ICNP '04*, Berlin, Germany, Oct. 2004.

[20] R. Pain, B. Prabhakar, and K. Psounis, "CHOKe, a stateless active queue management scheme for approximating fair bandwidth allocation," in *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.

[21] D. Lin and R. Morris, "Dynamics of Random Early Detection," in *Proceedings of ACM SIGCOMM '97*, Cannes, France, Sept. 1997.

[22] T. J. Ott, T. V. Lakshman, and L. Wong, "SRED: Stabilized RED," in *Proceedings of IEEE INFOCOM '99*, New York, NY, Mar. 1999.

[23] A. Rangarajan and A. Acharya, "ERUF: Early regulation of unresponsive best-effort traffic," in *Proceedings of IEEE ICNP '99*, Toronto, CA, Oct. 1999.

[24] F. Anjum and L. Tassiulas, "Fair bandwidth sharing among adaptive and non-adaptive flows in the Internet," in *Proceedings of IEEE INFOCOM '99*, New York, NY, Mar. 1999.

[25] F. Ertemalp, D. Chiriton, and A. Bechtolsheim, "Using dynamic buffer limiting to protect against belligerent flows in high-speed networks," in *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[26] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Stochastic fair BLUE: A queue management algorithm for enforcing fairness," in *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, June 2001.

[27] V. Paxson, "Automated packet trace analysis of TCP implementations," in *Proceedings of ACM SIGCOMM '97*, Cannes, France, Sept. 1997.

[28] A. Kuzmanovic and E. Knightly, "Low-rate TCP-targeted denial of service attacks and counter strategies," Submitted to *IEEE/ACM Transactions on Networking*.

[29] J. Qiu and E. Knightly, "Inter-class resource sharing using statistical service envelopes," in *Proceedings of IEEE INFOCOM '99*, New York, NY, Mar. 1999.

[30] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, Addison-Wesley, 2000.

[31] R. Jain, "A delay based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM Computer Comm. Review*, vol. 19, no. 5, pp. 56–71, Oct. 1989.

[32] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm," *ACM Computer Comm. Review*, vol. 22, no. 2, pp. 9–16, Apr. 1992.

[33] L. Brakmo and L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol.

13, no. 8, pp. 1465–1480, Oct. 1995.

[34] V. Paxson and M. Allman, "Computing TCP's retransmission timer," Nov. 2000, Internet RFC 2988.

[35] V. Jacobson, "Congestion avoidance and control," *ACM Computer Comm. Review*, vol. 18, no. 4, pp. 314–329, Aug. 1988.

[36] S. Floyd, J. Madhavi, M. Mathis, and M. Podolsky, "An extension to the selective acknowledgement (SACK) option for TCP," July 2000, Internet RFC 2883.

[37] B. Teitelbaum et al., "Internet2 QBone: Building a testbed for differentiated services," *IEEE Network*, vol. 13, no. 5, pp. 8–17, Sept. 1999.

[38] C. Dovrolis and P. Ramanathan, "A case for relative differentiated services and the proportional differentiation model," *IEEE Network*, vol. 13, no. 5, pp. 26–35, Sept. 1999.

[39] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: A scalable architecture to approximate fair bandwidth allocations in high speed networks," in *Proceedings of ACM SIGCOMM '98*, Vancouver, British Columbia, Sept. 1998.

[40] S. Blake et al., "An architecture for differentiated services," 1998, Internet RFC 2475.

[41] C. Chuah, L. Subramanian, R. Katz, and A. Joseph, "QoS provisioning using a clearing house architecture," in *Proceedings of IWQoS '00*, Pittsburgh, PA, June 2000.

[42] K. Nichols, V. Jacobson, and L. Zhang, "Two-bit differentiated services architecture for the Internet," 1999, Internet RFC 2638.

[43] A. Terzis, L. Wang, J. Ogawa, and L. Zhang, "A two-tier resource management model for the Internet," in *Proceedings of Global Internet Symposium '99*, Rio de Janeiro, Brazil, Dec. 1999.

[44] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang, "Endpoint admission control: Architectural issues and performance," in *Proceedings of ACM SIGCOMM '00*, Stockholm, Sweden, Aug. 2000.

[45] C. Cetinkaya and E. Knightly, "Scalable services via egress admission control," in *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.

[46] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proceedings of ACM SIGCOMM '99*, Cambridge, MA, Aug. 1999.

[47] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster reserves: A mechanism for resource management in cluster-based network servers," in *Proceedings of ACM SIGMETRICS '00*, June 2000.

[48] N. Bhatti and R. Friedrich, "Web server support for tiered services," *IEEE Network*, vol. 13, no. 5, pp. 64–71, Sept. 1999.

[49] V. Kanodia and E. Knightly, "Multi-class latency-bounded web services," in *IEEE/IFIP IWQoS '00*, Pittsburgh, PA, June 2000.

[50] K. Li and S. Jamin, "A measurement-based admission controlled web server," in *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.

[51] R. L. Carter and M. E. Crovella, "Measuring bottleneck link speed in packet-switched networks," *Performence Evaluation*, vol. 27, no. 28, pp. 297–318, 1996.

[52] V. Paxson, "End-to-end Internet packet dynamics," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277–292, June 1999.

[53] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," in *Proceedings of ACM SIGCOMM '00*, Stockholm, Sweden, Aug. 2000.

[54] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," in *Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, Aug. 2002.

[55] J. Schlembach, A. Skoe, P. Yuan, and E. Knightly, "Design and implementation of scalable admission control," in *Proceedings of the International Workshop on QoS in Multiservice IP Networks*, Rome, Italy, Jan. 2001.

[56] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[57] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, pp. 1374–1399, Oct. 1995.

[58] D. Wrege, E. Knightly, H. Zhang, and J. Liebeherr, "Deterministic delay bounds for VBR video in packet-switching networks: Fundamental limits and practical

tradeoffs," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 352–362, June 1996.

[59] R. Boorstyn, A. Burchard, J. Liebeherr, and C. Oottamakorn, "Effective envelopes: Statistical bounds on multiplexed traffic in packet networks," in *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.

[60] D. Wrege and J. Liebeherr, "Video trafic characterization for multimedia networks with a deterministic service," in *Proceedings of IEEE INFOCOM '96*, San Francisco, CA, Mar. 1996, pp. 537–544.

[61] J. Bennett and H. Zhang, "WF^2Q: Worst-case Fair Weighted Fair Queueing," in *Proceedings of IEEE INFOCOM '96*, San Francisco, CA, Mar. 1996.

[62] V. Ribeiro, M. Coates, R. Riedi, S. Sarvotham, B. Hendricks, and R. Baraniuk, "Multifractal cross-traffic estimation," in *Proceedings of ITC '00*, Monterey, CA, Sept. 2000.

[63] S. Alouf, P. Nain, and D. Towsley, "Inferring network characteristics via moment-based estimators," in *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, Apr. 2001.

[64] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, Aug. 1995.

[65] K. K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," Jan. 1999, Internet RFC 2481.

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

[67] A. Pasztor and D. Veitch, "High precision active probing for Internet measurement," in *Proceedings of INET '01*, Stockholm, Sweden, 2001.

[68] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Comm. Review*, vol. 24, no. 5, pp. 10–23, 1994.

[69] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[70] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, "Dynamic behavior of slowly-responsive congestion control algorithms," in *Proceedings of ACM SIGCOMM '01*, San Diego, CA, Aug. 2001.

[71] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proceedings of ACM SIGCOMM '99*, Vancouver, British Columbia, Sept. 1999.

[72] L. Guo and I. Matta, "The war between mice and elephants," in *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[73] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of network traffic," in *Proceedings of IEEE/ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, Nov. 2001.

[74] S. Floyd, "Highspeed TCP for large congestion windows," Dec. 2003, Internet RFC 3649.

[75] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," Dec. 2002, submitted for publication.

[76] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST TCP: From theory to experiments," Apr. 2003, submitted for publication.

[77] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," July 2003, submitted for publication.

[78] R. N. Shorten, D. J. Leith, J. Foy, and R. Kilduff, "Analysis and design of congestion control in synchronized communication networks," June 2003, submitted for publication.

[79] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: A mechanism for background transfers," in *Proceedings of OSDI '02*, Boston, MA, Dec. 2002.

[80] J. Martin, A. Nilsson, and I. Rhee, "The incremental deployability of RTT-based congestion avoidance for high speed TCP internet connections," in *Proceedings of ACM SIGMETRICS '00*, Santa Clara, CA, June 2000.

[81] D. D. Clark and W. Fang, "Explicit allocation of best-effort packet delivery service," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 362–373, Aug. 1998.

[82] S. Yang and G. de Veciana, "Size-based adaptive bandwidth allocation: Optimizing the average QoS for elastic flows," in *Proceedings of IEEE INFOCOM '02*, New York, NY, June 2002.

[83] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?," in *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, Apr. 2001.

[84] V. Jacobson, "Pathchar: A tool to infer characteristics of Internet paths," *ftp://ftp.ee.lbl.gov/pathchar/*, Apr. 1997.

[85] J. Liu and M. Crovella, "Using loss pairs to discover network properties," in *Proceedings of IEEE/ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, Nov. 2001.

[86] A. Pasztor and D. Veitch, "The packet size dependence of packet pair like methods," in *Proceedings of IWQoS '02*, Miami, FL, May 2002.

[87] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, pp. 115–156, Sept. 1992.

[88] L. Zhang, S. Shenker, and D. Clark, "Observation on the dynamics of a congestion control algorithm: The effects of two-way traffic," in *Proceedings of ACM SIGCOMM'91*, Zurich, Switzerland, Sept. 1991.

[89] S. Floyd and E. Kohler, "Internet research needs better models," in *Proceedings of HOTNETS '02*, Princeton, New Jersey, Oct. 2002.

[90] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," *ACM Computer Comm. Review*, vol. 32, no. 3, pp. 5–21, July 2002.

[91] J. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," in *Proceedings of ACM SIGCOMM '96*, Stanford University, CA, Aug. 1996.

[92] K. Fall and S. Floyd, "Simulation-based comparison of Tahoe, Reno and SACK TCP," *ACM Computer Comm. Review*, vol. 5, no. 3, pp. 5–21, July 1996.

[93] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's initial window," 1998, Internet RFC 2414.

[94] C. Jin, D. Wei, and S. Low, "FAST TCP: Motivation, architecture, algorithms, performance," in *Proceedings of IEEE INFOCOM '04*, Hong Kong, China, Mar. 2004.

[95] H. Balakrishnan and R. Katz, "Explicit loss notification and wireless web performance," in *Proceedings of IEEE GLOBECOM '98*, Sydney, Australia, Nov. 1998.

[96] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of ACM MOBICOM '01*, Rome, Italy, July 2001.

[97] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "TCP congestion control with a misbehaving receiver," *ACM Computer Comm. Review*, vol. 29, no. 5, pp. 71–78, Oct. 1999.

[98] D. Ely, N. Spring, D. Wetherall, S. Savage, and T. Anderson, "Robust congestion signaling," in *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[99] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, Apr. 2000.

[100] M. Handley, J. Padhye, S. Floyd, and J. Widmer, "TCP friendly rate control," July 2001, IETF Internet draft.

[101] S. Floyd, M. Handley, and J. Padhye, "A comparison of equation-based and AIMD congestion control," *Technical Report*, May 2000.

[102] M. Mathis, J. Semke, J. Madhavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance," *ACM Computer Comm. Review*, vol. 27, no. 3, pp. 67–82, July 1997.

[103] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.

[104] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the performance of HTTP over several transport protocols," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 616–630, Oct. 1997.

[105] C. Patridge and T. Shepard, "TCP/IP performance over satelite links," *IEEE Network*, vol. 11, no. 5, pp. 44–49, Sept./Oct. 1997.

[106] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack, "Upgrading transport protocols with untrusted mobile code," in *Proceedings of ACM SOSP '03*, New York, NY, Oct. 2003.

[107] M. Vojnovic and J.-Y. Le Boudec, "On the long-run behavior of equation-based rate control," in *Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, Aug. 2002.

[108] H. Balakrishnan, V. Padmanabhan, S. Seshana, and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, Dec. 1997.

[109] V. Padmanabhan and R. Katz, "TCP Fast Start: A technique for speeding up web transfers," in *Proceedings of IEEE GLOBECOM '98*, Sydney, Australia, Nov. 1998.

[110] R. Wang, G. Pau, K. Yamada, M.Y. Sanadidi, and M. Gerla, "TCP start up performance in large bandwidth delay networks," in *Proceedings of IEEE INFOCOM '04*, Hong Kong, China, Mar. 2004.

[111] Y. Zhang, L. Qiu, and S. Keshav, "Speeding up short data transfers: Theory, architectural support, and simulations," in *Proceedings of NOSSDAV '00*, Chapel Hill, N. Carolina, June 2000.

[112] D. Dumitriu, E. Knightly, I. Stoica, and W. Zwaenepoel, "Denial of service resilience in peer-to-peer systems," Submitted to *OSDI '04*.

[113] I. Aad, J.P. Hubaux, and E. Knightly, "Denial of service resilience in ad hoc networks," in *Proceedings of ACM MOBICOM '04*, Philadelphia, PA, Sept. 2004.

[114] O. Nordstrom and C. Dovrolis, "Beware of BGP attacks," *ACM Computer Comm. Review*, vol. 34, no. 2, pp. 1–8, Apr. 2004.