

The Design and Evaluation of Advanced TCP-based Services over an Evolving Internet

A Thesis
Presented to
The Academic Faculty

by

Qi He

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
August 2005

The Design and Evaluation of Advanced TCP-based Services over an Evolving Internet

Approved by:

Professor Mostafa Ammar, Advisor
College of Computing
Georgia Institute of Technology

Professor George Riley
College of Engineering
Georgia Institute of Technology

Professor Constantinos Dovrolis
College of Computing
Georgia Institute of Technology

Professor Ellen Zegura
College of Computing
Georgia Institute of Technology

Professor Richard Fujimoto
College of Computing
Georgia Institute of Technology

Date Approved: July 13th, 2005

*To my parents,
for their endless and unreserved support.*

ACKNOWLEDGEMENTS

I am grateful to many people inside and outside of the College of Computing at Georgia Tech, without whose support I would not have finished this thesis the way it is now.

First of all, I feel privileged to have the opportunities to work with a number of experienced professors. The advices they were able offer were often complementary and I feel extremely lucky to have the best in several aspects.

My advisor Mostafa Ammar has guided me through every piece of the thesis work. Not only has he imparted to me a wealth of knowledge in networking, telecommunications and performance evaluation, but also he has taught me the skills essential to a PhD student and a researcher, including the general problem-solving skills, system-level thinking, and the ability to identify and define research problems.

Much of the thesis work is collaboration with Constantinos Dovrolis. The quality of this thesis has been greatly improved with his help. Every discussion with him has been constructive to strengthening my knowledge and skills, especially in the areas of measurement and network performance evaluation. It is his thought-provoking and detailed questions, and his encouragement for persistence, that have made some of the thesis work as solid as it is now.

Richard Fujimoto has given me a lot of suggestions that greatly enhanced the various pieces of simulation work in my thesis. He has also been very supportive of the other pieces of work that I decided to pursue.

I learned from George Riley the basics as well as the advanced topics in network simulation. The simulation methodologies work in this thesis embodies a lot of input from him. He also helped much in the experimentations involving network simulations, which I used for every piece of my thesis work.

Ellen Zegura has given me valuable comments and suggestions on my work at different stages of my graduate study. She has also shared with me her opinions and advices on

graduate study in general, on writing outstanding papers and giving good presentations.

I owe a lot to Karsten Schwan for the system skills and insights I acquired through the work with him, which turn out to be so essential to my research. He also gave me great advices on research-related communications and writing.

I am thankful to Jim Xu, who has helped me on some statistics related problems and offered valuable suggestions on my work. I am thankful to Kalyan Perumalla for his help with the use of his federated simulation libraries.

The brilliant fellow students in the Networking and Telecommunications Group, and those in the Parallel and Distribution Simulations, have given me a lot of input, helped me tackle problems encountered in experiments, and made my graduate study so enjoyable. I would like to specifically thank Ruomei Gao, Wenrui Zhao, Shashi Merugu, Hao Jiang, Ravi Prasad and Manish Jain for the numerous discussions we had. Ravi Prasad and Manish Jain helped me on using several measurement tools and testbeds.

I am also grateful for the support from the following people who I worked with during my internships at IBM Austin and at HP Labs: Farhad Fatahalian, Ray Bryant, Rajiv Arora, Weiming Gu, Robert Dimpsey, Dan Muntz, Dejan Milojicic.

All the professors and colleagues have also been extremely supportive during the process that I prepared for a career. Their help made it a smooth ride from proposal to thesis to graduation.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	x
LIST OF FIGURES	xi
SUMMARY	xiv
I INTRODUCTION	1
1.1 Overview of Research Challenges	1
1.2 Summary of Research Objectives and Contributions	2
1.2.1 Part I: Performance Evaluation-Assisted Design of Advanced TCP-based Services	2
1.2.2 Part II: Efficient Large Scale Simulation of TCP/IP Networks	6
1.3 Thesis Organization	8
II RELATED WORK	10
2.1 TCP Modeling and Throughput Prediction	10
2.1.1 TCP Modeling	10
2.1.2 TCP Throughput Prediction	11
2.2 Peer-to-Peer Systems and Overlay Networks	14
2.2.1 Overview	14
2.2.2 Resource Management in Overlay	15
2.2.3 Modeling and Evaluation of Peer-to-Peer Systems	16
2.3 Simulation Performance Optimization	17
2.4 Network Traffic Analysis, Modeling and Generation	19
III PREDICTION OF LARGE TRANSFER TCP THROUGHPUT	21
3.1 Introduction	21
3.2 Formula-based prediction	23
3.2.1 A formula-based TCP throughput predictor	24
3.2.2 Effect of the extra load due to the target flow	26
3.2.3 Errors due to the TCP sampling behavior	27

3.2.4	Errors due to the difference between available bandwidth and TCP throughput	28
3.3	FB prediction accuracy	29
3.3.1	Overview of measurement methodology	29
3.3.2	Results	31
3.3.3	Summary	37
3.4	History-based prediction	37
3.4.1	Linear Predictors	38
3.4.2	Detection of Level Shifts and Outliers	39
3.5	HB prediction accuracy	41
3.5.1	Results	41
3.5.2	Summary	46
3.6	Two predictability factors	47
3.6.1	Effect of Load	48
3.6.2	Effect of Degree of Multiplexing	51
3.6.3	Summary	51
3.7	Summary	52
IV	CONGESTION CONTROL IN GNUTELLA-LIKE OVERLAY NETWORKS	54
4.1	Introduction	54
4.2	Congestion Control in Gnutella Networks	55
4.2.1	Overview of the Gnutella Operations	56
4.2.2	Congestion Control Mechanisms	56
4.2.3	Message Type Specific Limit	59
4.2.4	Comparison of Different Gnutella Servents	60
4.3	Congestion Control Behavior	61
4.3.1	Message Drop Patterns	62
4.3.2	Distribution	66
4.3.3	Impact of Access Bandwidth and Peer Connectivity	69
4.4	Understanding the Congestion Control Mechanisms in Gnutella Networks .	69
4.4.1	Priority Queuing	73

4.4.2	TCP Backpressure	78
4.4.3	Rate Limiting	81
4.5	Summary	83
V	PACKET-LEVEL SIMULATION OF PEER-TO-PEER NETWORKS	85
5.1	Introduction	85
5.2	Simulator Architecture	86
5.2.1	Overview	87
5.2.2	PeerApp	88
5.2.3	PeerAgent	89
5.2.4	Socket Adaptation Layer	90
5.2.5	Describing the Simulation	91
5.3	Specializing the Framework to Simulate Gnutella Systems	92
5.4	Experimental Results on Gnutella Performance	94
5.4.1	System Throughput	95
5.4.2	Query Response Time	96
5.4.3	Considering the Rest of the Network	99
5.5	Large Scale Simulation with <i>PDNS</i>	100
5.6	Extensible for Different Research Purposes	102
5.6.1	GWebCache Simulation	102
5.6.2	Testbed for Alternative Gnutella Flow Control Mechanisms	102
5.7	Summary	103
VI	EXPLOITING THE PREDICTABILITY OF TCP STEADY STATE BEHAVIOR TO SPEED UP NETWORK SIMULATION	105
6.1	Introduction	105
6.2	Prediction-based Simulation Design	107
6.2.1	Simulation Context	107
6.2.2	Simulation Framework Details	107
6.3	Predicting Steady-state Throughput	110
6.3.1	Summary of TCP behavior	110
6.3.2	Measurement Period	110
6.3.3	Direct Throughput Measurement	111

6.3.4	Formula Based Prediction	111
6.4	Mixing Packet-based and Predicted Flows	112
6.5	Experimental Results	114
6.5.1	Implementation	114
6.5.2	Speedup	115
6.5.3	Approximation	116
6.5.4	Experience with Larger IP Networks	125
6.6	Suggestions for Usage	129
6.7	Comparison with Fluid Flow Models	133
6.8	Summary	135
VII	OPTIMAL CONFIGURATION OF TCP TRAFFIC IN NETWORK SIMULATIONS UNDER LINK LOAD CONSTRAINTS	137
7.1	Introduction	137
7.2	Problem Statement	139
7.2.1	Overview of the Traffic Generation Context	139
7.2.2	Traffic Generation Context Specification	140
7.3	Traffic Generation Algorithm	143
7.3.1	Single Link Example	143
7.3.2	Problem Formulation	145
7.3.3	Populating the Linear Program	147
7.4	Evaluation	149
7.4.1	Single Link Example	150
7.4.2	Network of Links	151
7.4.3	Explaining the Impact of Lossy Links	160
7.4.4	Summary of the Evaluation Results	162
7.5	Summary	162
VIII	CONCLUSION AND FUTURE WORK	163
8.1	Research Summary	163
8.2	Future Directions	166
	REFERENCES	169

LIST OF TABLES

1	Main Symbols	24
2	(A) Increased RTT (B) Increased Loss Rate.	26
3	Different Loss Rate Estimates during a TCP Flow.	28
4	Servent Flow Control Features	60
5	Priority Queueing Mechanisms	61
6	Hops Distribution of Drops (%)	69
7	TTL Distribution of Drops (%)	70
8	Two Classes of Peers	73
9	Heterogeneous Priority Policy	78
10	Socket Upcalls	90
11	classinfo.txt	92
12	Approximation-RED vs. DropTail (% error)	118
13	Run Time Comparison	127
14	Terms and Notations	143

LIST OF FIGURES

1	Thesis Organization Chart	8
2	A measurement epoch. 150 such epochs were recorded during each trace, with 7 traces collected per path.	30
3	CDF of E for all predictions, for predictions in lossy paths, and for predictions in lossless paths.	31
4	CDF of RTT and loss rate increase due to target flow.	32
5	Prediction errors using $\tilde{T}$ and $\tilde{p}$ (RTT and loss-rate during the target flow) and using $\hat{T}$ and $\hat{p}$ (RTT and loss-rate prior to the target flow).	33
6	Variation of the prediction error across different paths.	34
7	CDF of per-trace RMSRE.	35
8	Actual throughput vs. prediction error.	36
9	Prediction accuracy for window-limited vs. congestion-limited flows.	36
10	Examples of TCP throughput traces and the prediction errors (RMSRE) with various predictors.	38
11	Moving Average prediction error.	41
12	Holt-Winters prediction error.	42
13	Prediction error versus CoV.	43
14	A: Predictable paths (low RMSRE), B: Paths with small and predictable errors (stable RMSRE), C: Paths with small but unpredictable errors (varying RMSRE), D: Unpredictable paths (high RMSRE, notice the different Y-axis ranges).	44
15	Prediction error for window-limited vs. congestion-limited flows.	45
16	Prediction error with different TCP transfer periods.	46
17	Gaussian process.	49
18	Poisson process.	49
19	Processor Sharing model.	50
20	Gnutella servant forwarding	57
21	Drop pattern (Gtk-Gnutella)	63
22	Drop pattern (Mutella)	64
23	Drop pattern (LimeWire)	64
24	CDF of drops/sec	65

25	Expanded view (Gtk-Gnutella)	66
26	Expanded view (Mutella)	67
27	Expanded view (LimeWire)	67
28	Type distribution of drops	68
29	Impact of connectivity	70
30	Performance of priority queuing	74
31	Performance with utility function	75
32	Performance vs. buffer size	76
33	Heterogeneity test setup	78
34	Throughput with TCP backpressure	79
35	Response time with TCP backpressure	80
36	Heterogeneous backpressure capability	82
37	Rate limiting performance	83
38	Heterogeneous rate limiting capability	84
39	Peer simulator architecture	87
40	System throughput	96
41	Connectivity	97
42	Hit time vs. access B/W	98
43	Hit time vs. link delay(1)	98
44	Hit time vs. link delay (2)	99
45	Performance with background traffic	100
46	Scaling with number of processors	101
47	Simulation epochs	107
48	Topology I	114
49	Topology II	115
50	Speedup vs. transfer size	116
51	Approximation vs. loss (Direct Measurement)	117
52	Approximation vs. loss (Formula)	118
53	Approximation vs. number of flows	120
54	Throughput approximation (1)	121
55	Throughput approximation (2)	121

56	Queue occupancy–packet-based	122
57	Queue occupancy–formula-based	123
58	Packet sequence	124
59	Queue occupancy with mixed simulations	124
60	Prediction error vs. network configuration	125
61	Prediction error vs. traffic mix	126
62	Approximation vs. transfer size	127
63	Queue occupancy vs. number of flows	128
64	Queue occupancy (heterogeneity)	128
65	Queue occupancy–RED vs. DropTail	130
66	Formula vs. direct measurement	131
67	Approximation on LANs	134
68	Capturing the transience	135
69	Traffic generation context	139
70	Utilization approximation	151
71	Loss rate approximation	152
72	CDF of the magnitude of relative error (backbone topology)	154
73	Relative error vs. target load	155
74	E vs. E_r (5 lossy links, 70 target links)	155
75	E vs. E_r (14 lossy links, 70 target links)	156
76	Impact of the number of target links	157
77	Impact of the number of lossy links	158
78	Impact of average hop count (7 lossy links)	158
79	Impact of average hop count (14 lossy links)	159
80	Loss rate approximation (scatter plot)	160
81	CDF of packet queuing delay	161

SUMMARY

Performance evaluation continues to play an important role in network research. Two types of research efforts related to network performance evaluation are particularly noteworthy: (1) using systematic performance evaluation to understand specific problems and to design better solutions, and (2) designing efficient performance evaluation methodologies.

This thesis addresses several performance evaluation challenges, encompassing both categories of effort listed above, related to a seemingly familiar topic, TCP-based services. For many years throughout the Internet evolution, end-to-end congestion control, and TCP in particular, has remained the prevailing mechanism through which network usage is regulated among end users and network performance is controlled. Much research has been invested in understanding TCP and in improving or customizing TCP. It is yet to be fully answered, however, as to what and how to evaluate TCP-based services. Such questions are especially important to address when new network services and technologies emerge. This thesis is motivated by building high-performance TCP-based network services in the context of overlay routing and peer-to-peer systems. While working towards solutions to concrete network service design and efficient network performance evaluation methodologies, this thesis demonstrates just how much the detailed knowledge of TCP and of network traffic could help in achieving these solutions.

With respect to the first type of research effort listed above, this thesis addresses two issues related to the design of TCP-based network services:

- *Prediction of large transfer TCP throughput:* Throughput is by far the most important performance metric of large TCP transfers. Predicting the TCP throughput attainable on different paths is frequently practiced for server or route selections in peer-to-peer systems and overlay/multihomed routing networks. Based on a thorough measurement study, we evaluate the accuracy of two categories of TCP throughput

prediction techniques. We then analyze the factors that affect the accuracy of each and suggest the directions for improvement.

- *Congestion control and message loss in Gnutella peer-to-peer networks:* Neighboring nodes in the Gnutella peer-to-peer overlay network connect to each other using TCP tunnels. Congestion could take place at the overlay layer and cause message losses when the throughput of a TCP tunnel cannot keep up with the rate of arriving messages. We evaluate the congestion control mechanisms and message loss behavior in the existing Gnutella overlay networks. The challenges for congestion control in such a message-oriented network are analyzed, as are the design tradeoffs of alternative mechanisms.
- *Packet-level simulation of peer-to-peer systems:* In order to study systems such as the above with details of the underlying network, we design and develop a scalable and extensible peer-to-peer system simulator that is run on a native network simulator with *packet-level details*. The simulator has an interface that is intended to maximize its portability to different packet-level network simulators, making it possible to utilize existing parallelization techniques for scalability. It can be extended to study different aspects of a peer-to-peer system.

The second part of the thesis, representing the second type of effort outlined above, proposes two techniques to improve network simulations by exploiting the detailed knowledge of TCP:

- *Speed up network simulations by exploiting TCP steady-state predictability:* Processing of packet events is by far the most time consuming aspect of large scale network simulations. We develop a technique that saves on the processing of packet events for TCP flows using the well established results showing that the average behavior of a TCP flow is predictable given a steady-state path condition. We exploit this to predict the average behavior of a TCP flow over a future period of time where steady-state conditions hold, thus allowing for a reduction (or elimination) of the processing required for packet events. Our technique can integrate well with existing packet-level

simulations. In terms of the approximation to packet-level simulations, it is also more faithful in several respects than previous optimization techniques.

- *TCP workload generation under link load constraints:* Realistic load level, congestion locality and load diversity are reflections of real network conditions. In order to have such load conditions in a large scale network simulation setup, we need an approach to generating an appropriate traffic matrix. We develop an algorithm that generates traffic for a specific network configuration such that realistic and user-designated load conditions are obtained on certain links. At the same time, the algorithm minimizes the simulation memory requirement.

CHAPTER I

INTRODUCTION

1.1 Overview of Research Challenges

The evolution of network technologies constantly inspires new applications, which in turn, change the network traffic characteristics significantly. For instance, the dominance of Web traffic on the Internet in the late 90's has gradually been replaced by a combined dominance of Web and peer-to-peer (P2P) traffic in recent years [130]¹.

Emerging applications tend to have high demands for network resources and employ sophisticated and diversified protocols, which stress network resource management. One constancy over the evolution is that end-to-end congestion control, and TCP in particular, remain the prevailing mechanisms through which network usage is regulated among end users and network performance is controlled. As such, TCP and end-to-end congestion control keep influencing the design and evaluation of applications [78], systems [40] and some network mechanisms [81]. On the other hand, emerging technologies and Internet applications present new challenges for congestion control. In response to those challenges, proposals to modify TCP [29, 71, 99] or to adjust the usage of TCP at the application layer [60, 107, 144], as well as new congestion control schemes [149], have appeared frequently in different contexts.

This thesis is concerned with a seemingly familiar topic, TCP-based services, in the emerging context of overlay and P2P networks. Our focus is on the TCP-related performance evaluation issues in this context. As demonstrated by much previous work, performance evaluation continues to play an important role in most areas of network research. Recently, Floyd announced a new IRTF group that is chartered to promote the research efforts in the evaluation of transport protocols and TCP in particular [16, 17]. This is yet

¹See <http://netflow.internet2.edu/weekly/> for weekly Abilene network usage report.

another example that performance evaluation remains an important issue, even for a topic as familiar as TCP. This thesis will be another testimony of how performance evaluation can help with the application and protocol design.

With respect to performance evaluation, two types of effort are particularly noteworthy. First, performance evaluation has been an important tool to understand the nature of specific problems and to design the solutions to them. Second, much research has been invested in improving performance evaluation. Network performance evaluation is often a daunting task, given all the idiosyncrasies of network protocols, and especially for large scale networks that involve millions of nodes. This thesis tackles problems in both these fronts, and we organize the thesis according to this categorization.

1.2 Summary of Research Objectives and Contributions

1.2.1 Part I: Performance Evaluation-Assisted Design of Advanced TCP-based Services

The first part of the thesis is concerned with the use of performance evaluation in understanding the design issues related to specific applications in the context of overlay networks [26, 77, 121, 136]. Especially with the recent endorsement from the Virtual Networks proposal [105], overlay networks have attracted much attention and are considered a promising computing paradigm to overcome many crucial performance problems of the current Internet [121], and to maximize and extend its usage [26, 136]. Peer-to-peer systems [12, 44, 3] are a type of application that often embodies the use of overlay networks to deliver or share information and resources in a decentralized manner among a large number of users. One of the most distinctive characteristics of overlay networks is that collaborating nodes form an application layer network on top of the native Internet. Packets or messages are then routed and forwarded on the overlay, typically based on certain user-perceived performance criteria or application layer message semantics.

This thesis looks into two issues in designing TCP-based services in these systems, namely, overlay route or peer selection, and overlay congestion control, as described below.

- To benefit most from the overlay architecture, applications need to construct the optimal overlay networks, and to make intelligent routing decisions. For TCP-based services, these demand much knowledge of the complex behavior of TCP in different paths and under different usages [25, 78, 125, 127]. For example, when a node in a throughput sensitive application selects neighboring overlay nodes, one key criterion is the TCP throughput between a candidate and the node itself [43, 127]. We also need to be aware that overlay and peer-to-peer systems also offer unique ways to improve end-to-end performance [61, 78, 132, 138] of TCP-based services. For example, concatenated TCP tunnels have been used to improve multicast performance [78, 138].
- Forwarding and routing at the application layer engenders new congestion control problems. For example, congestion can occur at the overlay layer, either within an overlay or among competing overlays. Intra-overlay congestion and message losses have been observed, e.g., in the Gnutella application [3]. To deal with overlay layer congestion such as this, one has to consider not only the operation of TCP in this new environment, e.g., forwarding on paths with concatenated TCP tunnels, but also the application layer performance criteria and the semantics of application messages.

This thesis addresses the following specific challenges related to the two issues above. While motivated to tackle these two issues, some of the methodologies we use and the conclusions we draw are applicable to a broader range of studies.

Prediction of Large Transfer TCP Throughput Performance prediction of network paths is an essential task in overlay and P2P networks [26, 43], as it is in Grid computing [51] and CDNs [77]. To name just a few of their applications, such predictions are used in route selection schemes for overlay and multihomed networks [20, 21, 25, 94], dynamic server selection [109, 134, 151], and peer-to-peer parallel downloads [44].

Arguably, the most important performance metric of a path is the average throughput of TCP transfers. The reason is that most data-transfer applications, and about 90% of the Internet traffic, use the TCP protocol. When it comes to performance prediction, the focus is typically on bulk TCP transfers, lasting more than a few seconds. Short TCP

flows are often limited by slow-start, and their performance is much more sensitive to the randomness in the background traffic [52]. This thesis looks at the problem of predicting the throughput of a bulk TCP transfer in a particular network path, *prior* to actually starting the flow. For many applications, such as server selection and overlay route selection, a throughput prediction is needed before the flow starts. The reason is that rerouting an established TCP connection to a different network path or server can cause problems such as migration delays, packet reordering, re-initialization of the congestion window. Note that TCP throughput prediction is different than TCP throughput *estimation*. The latter is performed while the flow *is in progress* with the objective to estimate the TCP throughput or the TCP-Friendly rate of the flow. An example of a TCP throughput estimation scheme is TCP-Friendly Rate Control (TFRC) [47].

In Chapter 3 of this thesis, we classify existing TCP throughput prediction techniques into two categories: *Formula-Based* (FB) and *History-Based* (HB). FB prediction relies on analytic models [28, 38, 54, 85, 89, 100] that express the TCP throughput as a function of the characteristics, such as loss rate and RTT, of the underlying network path. Despite the accuracy of the models themselves, this work shows that FB schemes can lead to large *prediction* errors and the major causes are revealed. HB approaches predict the throughput of TCP flows from a time series of previous TCP throughput measurements on the same path. HB predictions are demonstrated to be quite attractive in the sense that even simple HB predictors, such as Moving Average and Holt-Winters, using a history of limited and sporadic set of prior samples, can be fairly accurate. With simple queuing models, it is shown that the performance of HB predictors is highly dependent on the load and the degree of statistical multiplexing on a particular path [66, 67].

Congestion Control and Message Loss in Gnutella Networks In peer-to-peer overlay networks, peers act as routers that forward messages on *overlay links*, which are often transport connections (or tunnels) between peers. Realizing that P2P system performance is highly dependent on the routing efficiency of the underlying overlay network, previous research has focused on building better connected overlay networks [94].

Overlay congestion, which may occur during the forwarding process on peers, can have an adverse impact on the performance of an overlay network. However, little is known about its impact and the current practice of overlay congestion control. In Chapter 4 of this thesis, we investigate and characterize the message loss behavior and the congestion control implementations in a particular peer-to-peer overlay network, the *Gnutella* network. Our work starts with an analysis of different Gnutella server (referring to a Gnutella peer) implementations. This reveals the heterogeneity among their congestion control mechanisms. We next undertake a measurement study to characterize the message loss behavior on typical Gnutella peers and how it relates to the congestion control mechanisms implemented by the servers. Packet-level simulations are then used to understand the impact different factors have on the effectiveness of the current Gnutella congestion control mechanisms. We also show that the message-oriented forwarding, prioritized message types and the heterogeneous congestion control mechanisms present unique challenges for congestion control in Gnutella-like networks [63].

Packet-level Simulation of Gnutella Networks As peer-to-peer traffic starts to claim an increasingly larger proportion of the total Internet traffic, it is crucial that new peer-to-peer research includes considerations for the interaction with the underlying network.

However, much of the evaluation of peer-to-peer proposals has used simplified models that fail to include a detailed model of the underlying network. As stated in some publications [55], the failure to include a detailed packet-level network model can be largely attributed to the complexity of experimenting with a scalable peer-to-peer system simulator built on top of a scalable network simulator with packet-level details. To address the issue, in Chapter 5 of this thesis, we design and develop a framework for an extensible and scalable peer-to-peer simulation environment that can be built on top of existing packet-level network simulators. The simulation environment is portable to different network simulators, which enables us to simulate a realistic large scale peer-to-peer system using existing parallelization techniques [64]. We implement an instantiation of this framework for the Gnutella network, *GnutellaSim* [5]. This tool is used to study the Gnutella congestion

control described above.

1.2.2 Part II: Efficient Large Scale Simulation of TCP/IP Networks

The second part of the thesis addresses the challenges in large-scale network simulations. Network research studies concerning congestion control and detailed protocol design, exemplified by the work in the first part of the thesis, very often use packet-level experimentations involving simulation or measurement. Research in the overlay and peer-to-peer networks, in particular, reinforces a need for experimentation methodologies that are *scalable*, *faithful* to real Internet environment and the detailed traffic behavior, and can be easily *tailored* by users. The two pieces of work in this part address the scalability and realism requirements, respectively, of network simulation. While the first piece of work is based on the predictability of TCP steady-state behavior, the second utilizes the knowledge of TCP bandwidth sharing, and the relation between TCP throughput and link load conditions. Although motivated by the overlay network contexts, the techniques and solutions we offer are generally applicable to most efforts that involve large-scale network simulations.

Exploiting the Predictability of TCP Steady-State Behavior to Speed Up Network Simulations In discrete-event network simulations, a significant portion of resources and computation are dedicated to the creation and processing of packet transmission events. For large-scale network simulations with a large number of high-speed data flows, the processing of packet events is the most time consuming aspect of the simulation. As a result any technique that would save on the processing of packet events can result in significant speedup of the simulation, i.e., reduction of the time required to run a simulation. In this thesis we develop a technique that saves on the processing of packet events for TCP flows. Needless to say, TCP traffic represents a large portion of the Internet traffic today and, therefore, large-scale network simulations typically involve the simulation of a large number of TCP packet flows.

The methodology we develop in Chapter 6 exploits the predictability of TCP behavior under a steady path condition so as to reduce (or eliminate) the processing required for packet events. This method integrates well with existing packet-level simulations, and

preserves the statistical behavior of every entity in the system so as to maintain the accuracy of the network simulation as a whole. Two approaches to estimating TCP throughput are compared for their accuracy and efficiency. Note, however, that these are throughput *estimations* based on observation *during* the steady-state, as compared to the *prediction* that we go into great length in Chapter 3. We also compare our techniques with previous approaches, such as fluid-flow simulation [22, 84, 91, 92, 118, 146], to improving TCP simulation performance [65].

Optimal Configuration of TCP Traffic in Network Simulations under Link Load

Constraints To gain meaningful insights from network simulations, it is important to have realistic and controlled network conditions [74], in terms of link utilization, queuing delay and loss rate. The ability to explicitly control link conditions is useful for studying a subject under a spectrum of controlled path conditions. For example, it is demonstrated that link utilization has a significant impact on the performance of AQM schemes [81]. Explicit control of link conditions also enables the simulation of important network characteristics such as diversity, locality and correlation of path conditions, which can affect applications such as overlay routing [25], Content Distribution Networks [77] and reliable multicast [49].

Given that over 90% of the Internet load is carried by TCP, most network simulation studies use TCP flows to generate the background traffic. A basic, but unresolved, question however is: how can one decide how many TCP flows to simulate from one network node to another? Simulating too many flows on a link can cause an unrealistically high loss rate on that link, while simulating too few flows can result in undesirably light load conditions. Similarly, to simulate realistic network conditions, one has to carefully control the load distribution on various network links (e.g., edge vs. core links), as well as the hop count (path length) of the simulated TCP flows. Previous simulation studies have dealt with these issues in a trial-and-error manner, experimenting with several traffic configurations until a realistic distribution of link load and loss rate is achieved. In Chapter 7 of this thesis, we present a methodology that determines the number of TCP flows that should be simulated between each pair of nodes in a network, based on the network topology, a specification of

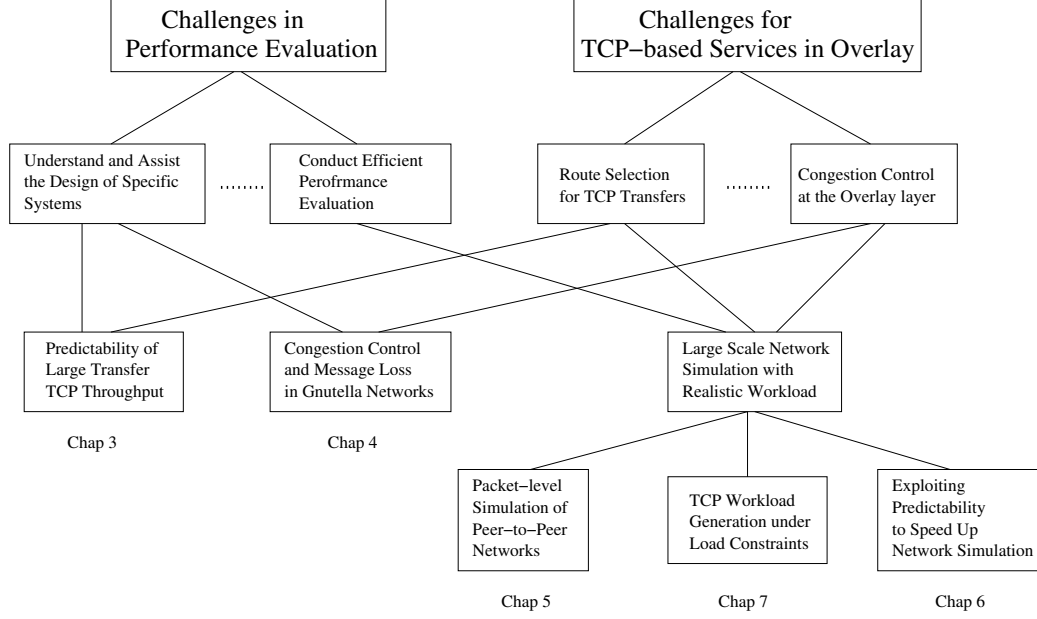


Figure 1: Thesis Organization Chart

the utilization and loss rate for certain links, and an average number of hops for the TCP flows. Our methodology is based on a linear program formulation that, while meeting the utilization and loss rate specifications, minimizes the number of required TCP flows. This optimization criterion minimizes the memory requirement of the simulation, which is often a key limiting factor in large network simulations involving TCP flows. Our evaluations show that the proposed methodology can closely approximate the specified link conditions in terms of utilization and loss rate. We also analyze the largest approximation errors and reveal their causes.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 discusses existing work related to the topics in this proposal. Chapter 3 discusses the evaluation and analysis of TCP throughput prediction methods. Chapter 4 investigates the congestion control and message loss on *Gnutella* networks. Chapter 5 describes the packet-level simulation framework of peer-to-peer networks and the specific instance of Gnutella network simulator (GnutellaSim). Chapter 6 presents the methodology to speed up TCP simulation. Chapter 7 discusses the

algorithm to generate traffic under link load constraints. In Chapter 8, we summarize the research contributions and discuss the future directions suggested by this thesis.

Figure 1 organizes the different pieces of thesis work based on their motivations or goals, both in the area of performance evaluation, and in the area of overlay and peer-to-peer networks. For instance, Chapter 3 addresses a problem related to the route selection for TCP transfers in overlay networks, and it also demonstrates how performance evaluation is used to understand and to assist the design of specific systems.

CHAPTER II

RELATED WORK

This chapter provides an overview of the related work on the topics of TCP modeling and throughput prediction, peer-to-peer systems and overlay networks, simulation performance optimization, traffic modeling and generation.

2.1 TCP Modeling and Throughput Prediction

The basic understandings of TCP behavior and models of TCP throughput are among the most important bases of this thesis work.

2.1.1 TCP Modeling

2.1.1.1 Steady-State Behavior

A TCP flow can be characterized as an *initial slow start* phase followed by a *steady-state* phase [38, 100]. The steady-state of a TCP flow comprises of a series of *timeout* periods, each of which, in turn, consists of a *slow start* phase and multiple *triple duplicate* periods. The transitions among these different phases are based on certain congestion avoidance and congestion control algorithms, of which the most important input is often the losses detected during the transfer. In other words, TCP behavior is fairly predictable given steady-state path conditions, such as a stationary average loss rate. This is the key characteristic that we take advantage of to speed up network simulations (see Chapter 6).

Given the well-defined behavior of TCP, numerous analytical models have been proposed to express the average TCP throughput in steady-state as a function of the underlying path characteristics or the details of competing flows [28, 38, 54, 85, 89, 100]. They differ in the type of mathematical model involved, e.g., probability models, queueing models, control theories, game theories, optimization problems, and fluid models. They also differ in terms of complexity and accuracy, modeling assumptions, and TCP flavor. The model by Padhye

et al. [100], known as the *PFTK* model, is one of the most popular because it is of close-form and the parameters required are relatively easy to obtain. That model expresses TCP throughput R as a function of the loss event probability p and Round-Trip Time (RTT) of the flow T , as well as some properties of the flow itself:

$$R = \min \left(\frac{M}{T\sqrt{\frac{2bp}{3}} + T_o \min(1, \sqrt{\frac{3bp}{8}})p(1 + 32p^2)}, \frac{W}{T} \right) \quad (1)$$

In (1), W is the TCP maximum congestion window size, M is the TCP data segment size, T_o is the TCP retransmission timeout period and b is the number of new segments released per new ACK.

2.1.1.2 Transient Behavior

TCP transient behavior typically refers to its convergence properties and responsiveness to changes in network conditions [147]. A number of models have been proposed to study those properties [72, 97]. When it is important to know the full dynamics of network traffic, e.g., when designing new router queuing schemes or TCP-friendly transport protocols, such models can be used to complement the steady-state TCP models. When developing the throughput prediction-based simulation method in Chapter 6, we leverage prior results about TCP transient behavior to isolate its impact on the estimate of the steady-state TCP throughput.

2.1.2 TCP Throughput Prediction

The throughput prediction we discuss here and use in Chapter 3 of this thesis, refers to estimating the throughput of a TCP flow *prior* to actually starting the flow. Notice that this is a different notion than throughput *estimation*. The latter is performed while the flow *is in progress*, which is much easier to accomplish and is the key technique we use in Chapter 6.

Unlike the prediction of RTT and loss rate, which can be based on direct and low-overhead measurements, predicting TCP throughput is significantly harder. First, TCP throughput depends on a large number of factors, including the transfer size, maximum sender/receiver windows, various path characteristics (RTT, loss rate, available bandwidth,

the nature of cross traffic, reordering, router/switch buffering, and others) and the exact implementation of TCP at the end-hosts. Second, direct measurement of TCP throughput using large “probing” transfers can be highly intrusive because the latter can saturate the underlying paths for significant time periods. What is really desired is *a low-overhead TCP throughput prediction technique that either avoids probing transfers altogether, or requires only a limited amount of probing traffic.*

Formula-based Throughput Prediction Recently, several low-overhead TCP throughput prediction schemes have been proposed in the literature [26, 57, 133, 139, 142]. One category of these schemes, which is called Formula-based (FB) prediction, rely on analytical throughput models as described in the previous subsection. Throughput models were initially proposed to *estimate* TCP-equivalent throughput in TCP-Friendly Rate Control [47], given estimates of the relevant path characteristics [38, 100] measured during the flow. Another motivation for some of these models has been to *predict* network characteristics [47, 89, 100]. FB predictions apply to throughput models the path characteristics measured prior to the flow to predict the TCP flow throughput. For instance, the throughput-optimizing routing component of RON follows the FB approach [26], predicting TCP throughput based on the “square-root” formula [89] and the RTT/loss rate measurements obtained *prior* to the flow.

While throughput models are used for prediction when a path is lossy, it is not clear what would be a good model for prediction when the path is lossless. One customary approach to deal with lossless paths is to predict the TCP throughput as the *available bandwidth* of the path prior to the TCP flow, when the flow is limited by its maximum window size [68, 70, 107]. Certain overlay routing schemes have proposed using available bandwidth for TCP throughput prediction in lossless paths [26]. Available bandwidth is the non-utilized part of the tight link’s capacity, and it can be measured non-intrusively [70].

Although FB predictions are used frequently, there has been little empirical evaluation of their accuracy, or a critical analysis of the factors that affect their accuracy. Chapter 3

of this thesis is the first to use systematic measurement and analytical studies to evaluate the real-world accuracy of FB predictions, and to reveal the major causes of error for FB predictions: (1) differences between the a priori measurements of RTT and loss rate, (2) difference between the available bandwidth and the TCP throughput that is attainable [107].

The problem of using throughput formulas for prediction has been noted before by Goyal et al. [57]. They show that the end-to-end packet loss rate p on a path can be quite different from the “congestion event probability” p' required by the Padhye model (1), and they have proposed a way to derive p' from p . Notice though, their work does not solve the more general problem of estimating the path characteristics during a flow from those observed prior to the flow.

History-based Throughput Prediction Another category of approaches to throughput prediction, which is called history-base (HB) predictions, use standard time series forecasting techniques to predict TCP throughput based on a history of throughput measurements from previous TCP transfers on the same path. These approaches are primarily based on the stationarity and predictability of network traffic and path conditions [108]. HB prediction is applicable only when large TCP transfers are performed repeatedly on the same path. Expediently, this is the case with some applications of TCP throughput prediction [26, 44, 140].

HB TCP throughput prediction has been previously studied, mostly in the context of Grid computing [133, 139, 140, 142]. One operational system is the Network Weather Service (NWS) project [143]. In NWS, throughput prediction is based on small (64KB) TCP transfer probes with a limited socket buffer size (32KB). Vazhkudai et al. use bulk TCP transfers (1MB-1GB) and a large socket buffer (1MB), performed sporadically (1 minute-1 hour) [139]. They show that various linear predictors (including ARIMA models) perform similarly, and that the average prediction error on two paths ranges from 10% to 25%. Zhang et al. examine TCP throughput predictability based on a large set of paths and transfers [155]. Their TCP throughput measurements use 1MB transfers performed every minute, with 200KB socket buffers. Their main results are that 1) with several simple

linear predictors, about 95% of the prediction errors are below 40%, and 2) predictions using a very long history (e.g., Moving Average with 128 samples) perform rather poorly. A study by Qiao et al. has shown that the predictability of network traffic is highly path dependent [108]. Mathematical models (such as MMPP) have been previously used to analyze the predictability of aggregate network traffic [123], however, they are not directly applicable to the predictability of TCP throughput.

Even though the previous work on HB prediction is substantial, it left three important open issues that we attempt to address with this thesis. First, it did not distinguish between congestion-limited and window-limited flows; typically, the latter have a small socket buffer at the receiver or sender compared to the path’s bandwidth-delay product, and so they do not impose a heavy load on the network. Second, the previous work did not examine the effect of the TCP transfer frequency on the HB prediction accuracy; this frequency is a crucial parameter for HB prediction. Third, previous work has not investigated the underlying path characteristics that determine the predictability of TCP throughput. Instead, the underlying network path has been viewed as a “black box”, and so it was not possible to relate its characteristics (such as load and degree of statistical multiplexing) to the resulting TCP throughput predictability.

2.2 Peer-to-Peer Systems and Overlay Networks

2.2.1 Overview

An overlay network consists of collaborating nodes that form an application layer forwarding network, and packets or messages are routed in the overlay network according to certain user-perceived performance criteria or application layer message semantics. The use of overlay dates back to MBone [46], where a group of nodes form a forwarding network to the purpose of gradual deployment of Internet multicast. Recently, there is a surge of interest in overlay network research, where numerous proposals exist that use overlay networks for application layer multicast [121], Content Distribution [77], peer-to-peer sharing [3, 4], Quality of Service [132], optimized routing [26, 136], among others.

The peer-to-peer system represents one relatively early application of overlay networks,

and it is being viewed as a novel approach for people to share resources such as files [3, 4] and computing cycles [12], or to support collaborative tasks, among a large number of users distributed across the Internet. The growing popularity of peer-to-peer applications has inspired numerous research activities, many of which involve modeling [55, 75, 157], and architecture or protocol designs to improve their scalability and performance [82, 41].

2.2.2 Resource Management in Overlay

Resource management is key to the scalability of any system. The growing popularity of peer-to-peer systems and overlay networks makes overlay network resource management crucial to maintain the health of the Internet.

Numerous demonstrations have been performed that suggest that the performance of peer-to-peer systems is highly dependent on the underlying network characteristics [115, 116, 125]. On the other hand, it has also been shown that peer-to-peer systems have significant impact on the rest of the network [127]. For example, it is argued in [116] that the Gnutella networks will not scale eventually because the network cannot support the large bandwidth they demand. In [115], it is demonstrated that the Gnutella virtual network topology does not match the physical network topology, thus putting additional stress on the infrastructure. However, much of the evaluation of peer-to-peer proposals use simplified models that fail to include a detailed model of the underlying network.

Compared to the work on peer-to-peer systems, research in overlay networks has shown a much stronger connection to the underlying network. Most of the existing work on overlay, however, models only the network topology and static link properties (such as capacity and delay) [134]. Network resource dynamics have been addressed with simplified traffic models at best, e.g., by assuming infinitesimal traffic and an open-loop traffic model [145].

There are congestion control related problems arising in the context of overlay that defy analysis based on such simplified models. Some existing works investigated the use of TCP-tunnel based overlay networks to improve user-perceived performance. ROMA uses loosely coupled concatenated TCP tunnels to reduce the length of congestion control loops and improve the performance of reliable multicast [78]. OverQoS uses overlay networks built

on top of TCP tunnels to provide QoS services to overlay users while taking into account the global network resource usage [132]. Harrison et al. propose a router-level overlay framework to deploy QoS provisioning in the Internet [61].

One congestion-related problem that is quite different from the above is the *overlay layer congestion*, which has received little attention. Overlay congestion occurs when the throughput of an overlay link cannot keep up with the rate of traffic that is forwarded onto it. Among the causes of overlay congestion is the heterogeneity in peering node resources [125] and the insufficient considerations for the heterogeneity in overlay construction [134]. Overlay congestion can result in message losses and adversely impact application layer system performance. Yet, little is known about the impact and how well it is handled currently. In the work documented in Chapter of this thesis, we initiate a study of the overlay layer congestion control problem in the Gnutella peer-to-peer network. Chawathe et al. have since proposed a congestion control mechanism as part of an effort to improve the scalability of Gnutella-like systems [42].

2.2.3 Modeling and Evaluation of Peer-to-Peer Systems

Many analytic models of peer-to-peer systems are derived from the classic client/server model [95, 75, 55], with extensions in which the available peers become a composite server. These models are very useful in analyzing the impact of application layer protocols and user behavior. By ignoring the basic network service component, however, they fall short of revealing the impact of the interaction between P2P systems and the network. In Chapter 5 of this thesis, we discuss a packet-level peer-to-peer simulator framework that is intended to fill this gap.

The complexity of a packet-level network model may be avoided by using detailed flow-level models in which the average TCP throughput behavior can be obtained. We observe, however, that it is difficult to incorporate a complex flow-level model into the analytic model of a peer-to-peer system in the first place. More importantly, the end-to-end flow-level TCP throughput is not always directly related to the throughput of most peer-to-peer systems. In other words, better TCP throughput on individual connections between peers does not

always imply better peer-to-peer system throughput.

2.3 Simulation Performance Optimization

To improve the performance of network simulations, a number of researchers have reported good results by using the *fluid flow models* for TCP flow simulation [22, 84, 91, 92, 118, 146]. These techniques approximate network flows as fluids and describe the flows and their interaction using flow equations. A few characteristics distinguish these techniques from ours, documented in Chapter 6:

- They are typically not derived from event-based simulation models (as our technique is) and as such become hard to integrate with existing event-based simulation approaches.
- For a fluid model to approximate the real TCP throughput closely, a detailed characterization of the time between congestion events is necessary [22]. However, loss characterization is a daunting task by itself, given the extreme heterogeneity of the Internet, and therefore most fluid models sacrifice accuracy by making simple assumptions about loss patterns. Even with a detailed loss model, fluid models may overestimate the real throughput on certain paths due to the sub-linearity in TCP transmission rate modeling [22].
- A fluid model often results in reduced accuracy also because it assumes average behavior over certain time horizon (step size), and does not capture any behavior related to second order characteristics [22, 84].
- Different fluid models have to be developed for different types of traffic (e.g., variants of TCP) and/or queuing discipline. This is not a concern to our technique since we utilize the predictable steady-state TCP behavior, which is independent of these factors.

Dead reckoning is a related technique to speed up event-based simulation. With this technique, the state of a simulated entity is estimated at some future time based on past

history and some knowledge of the current state. This has been mostly used in the context of Distributed Interactive Simulation (DIS) (see for example [35, 31]). In Chapter 6, we explore related ideas in the specific context of network simulation and the simulation of TCP flows. Whereas dead reckoning is used to reduce the overhead of state updating across distributed systems, our goal is to reduce the packet events of individual TCP flows. In dead reckoning, the actual state of each entity is owned by one participating system, which can always correct predictions with large errors. Since our technique does not maintain the actual state of TCP flows, it requires more accurate prediction methods.

Another technique that has been proposed to speed up large-scale network simulation is the *Optimistic Parallel Simulation* of TCP flows [148]. This technique divides and distributes the simulation workload to different processors, and uses optimistic approaches for the synchronization among them. Optimistic Parallel Simulation does not save on packet event processing, instead, it scales up simulation by utilizing redundant physical resources. Performance improvement is limited by the availability of physical resources and the need for inter-process synchronization[148]. Inter-process synchronization prevents the full utilization of computing resources, and the speedup of Optimistic Parallel Simulation is often constrained by the propagation delay in the simulated network topology. This approach is orthogonal to Optimistic Parallel Simulation.

Commercial simulator tools rely on more consolidated approaches to improving speed and scalability. These include techniques such as small core simulation engine, critical path code optimization [18].

Real-time network simulation is the holy grail of large scale network simulations and it has attracted much attention in recent years [83, 15]. The challenges for real-time network simulation are well-documented by Riley et al. [112, 53]. To achieve real-time simulation, some level of fidelity is often sacrificed. For example, Gu et al. propose an approach to use system identification [96] techniques to simulate a large network on slower speed processors with lower fidelity [59].

Whereas the technique we describe in Chapter 6 can potentially deliver real-time simulations even for very large networks, this technique was not designed with the goal of

real-time simulation and currently does not guarantee that. As we carefully document, the speedup attainable by our technique is dependent on the traffic pattern.

2.4 *Network Traffic Analysis, Modeling and Generation*

Background traffic is a key component of network simulations, as it shapes the network environment under which network mechanisms or protocols are evaluated. As such, network traffic modeling and generation have attracted much attention in the literature [14, 30, 80, 87, 129, 150]. Much research has been invested in understanding the basic characteristics of network traffic, such as the traffic composition and the traffic process as a time series. Perhaps the most notable finding attained is that network traffic tend to have long-range dependence (LRD) and self-similarity [103, 158]. These properties have important implications for network research [58, 101]. For example, they affect the design of router queue management schemes [81]; on the other hand, they can be exploited for prediction [62] and congestion control [23, 137].

There are traditionally two categories of traffic generation: *open-loop* and *closed-loop*. The open-loop approach, exemplified by trace-driven traffic generation, generates the packet arrival sequence on a link [86]. Since it does not simulate the end-points of traffic, it cannot capture the flow and congestion control provided by TCP, which carries over 90% of the Internet traffic. As it is increasingly accepted that it is important to preserve the congestion responsiveness of network traffic [50, 74], closed-loop traffic generation [14, 30, 37, 150] has become the prevailing approach in recent years. This traffic generation method simulates the source and destination of each flow, and noticeably, the flow and congestion control of TCP. Existing traffic generation methods have typically been focusing on the accurate modeling either of the packet arrival process or of the traffic source behavior, so as to capture network traffic characteristics such as long range dependence and self-similarity [58].

None of them sufficiently addresses first-order link characteristics such as average utilization and loss rate, which are quite relevant to network simulations. The need to explicitly control link conditions has been only occasionally acknowledged by previous work. For example, in order to control the utilization of a simulated link, it has been suggested that

one can tune the traffic iteratively until the desired utilization is reached [150]. Such an approach is expensive because one has to run the simulation multiple times to check the utilization. In addition, this approach works only for simple topologies. It is not clear how it will work when the flows on multiple links interact with each other and every link has a target link condition to be satisfied. As to queuing delay and loss rate, some network simulation and emulation tools provide a stock of stochastic models for users to control these two parameters [34, 39]. Since losses and queuing delays generated by these models do not result from a TCP workload, however, this approach defeats the goal of closed-loop simulation.

In Chapter 7, we propose an algorithm to configure traffic for a specific network configuration in order to obtain user-specified link load conditions. While LRD and load conditions are two important characteristics of network traffic that we need to consider in our network performance studies. There are other factors that we believe warrant more attention. We identify those factors in Chapter 8 as directions for future work.

CHAPTER III

PREDICTION OF LARGE TRANSFER TCP THROUGHPUT

3.1 Introduction

In this chapter, we focus on the design, empirical evaluation, and analysis of TCP throughput predictors for a broad class of applications. The common requirement of such applications is that they rely on an accurate throughput prediction prior to the start of the TCP transfer. We first give more detailed introductions of the two categories of TCP throughput prediction techniques: Formula-Based and History-Based. Within each class, we develop representative prediction algorithms, which we then evaluate empirically over the RON testbed [26]. Note that our objective is not to compare FB and HB predictors. In fact, the two schemes are complementary, as they require different types of measurements and previous information about the underlying path. Instead, our objective is to examine the key issues in each prediction scheme, evaluate their accuracy under different conditions, explain the major causes of prediction errors, and provide insight regarding the factors that affect the predictability of large transfer TCP throughput in a given path.

As described in Chapter 1, FB prediction relies on mathematical models that express the TCP throughput as a function of the characteristics of the underlying network path (e.g., RTT, loss rate). In this chapter, we prefer to use the PFTK model of [100], because it is both simple and quite accurate. The main advantage of FB prediction is that it does not require any history of previous TCP transfers. In addition, FB prediction can be performed with relatively lightweight, non-intrusive network measurements of parameters such as RTT and loss rate. Unfortunately, however, our measurements show that FB schemes can lead to large prediction errors. The main reason is that throughput models require knowledge of the path characteristics *during* the TCP flow, whereas FB predictions measure the corresponding a

priori characteristics *before* the flow starts. If the flow itself causes significant changes in those characteristics, the resulting prediction errors can be unacceptably large. Another reason is that the delays or losses that a TCP flow experiences are not necessarily the same as those observed by a periodic probing stream, such as *ping* [57]. On the positive side, we do observe that the prediction errors are much lower, and probably acceptable for most applications, if the TCP transfer is limited by the receiver’s advertised window to the point that the transfer does not saturate its path.

On the other hand, HB approaches use standard time series forecasting techniques to predict TCP throughput based on a history of throughput measurements from previous TCP transfers on the same path. Our measurements over the RON testbed show that even simple linear HB predictors, such as Moving Average and non-seasonal Holt-Winters, are quite accurate. Furthermore, in agreement with previous work on HB prediction [139, 155], we found no major differences among a few candidate HB predictors. We do find, however, that two simple heuristics can noticeably improve the accuracy of HB predictors. The first is to detect and ignore outliers, and the second is to detect level shifts and restart the HB predictors. We next show, perhaps surprisingly, that even with a short history of a few previous transfers performed sporadically in intervals up to 30-40 minutes, prediction errors are still fairly low. On the negative side, our measurements show that HB predictors are highly path-dependent, which begs for answers to the following two questions. What makes TCP throughput much more predictable on some paths than on others, and which are the fundamental factors that affect the throughput predictability on a path? Using simple queueing models, we focus on two factors that we believe are the most important: the load on the path, and the degree of statistical multiplexing. Specifically, we show that the prediction error increases with the load on the bottleneck link, and decreases with the number of competing flows under constant load. Consequently, paths that are heavily loaded with just a few big flows are expected to be most difficult to predict.

The structure of the chapter is as follows. In Section 2, we develop a representative FB predictor and highlight some important issues in that type of prediction. Section 3 presents

measurement results for the accuracy of FB prediction. Section 4 introduces several existing HB predictors, and describes two simple techniques that can improve such predictors significantly. Section 5 presents measurement results for the accuracy of HB prediction. Section 6 focuses on two major factors that affect the throughput predictability: the load of the network path, and the degree of statistical multiplexing. We conclude in Section 7.

3.2 *Formula-based prediction*

The central component of an FB predictor is a mathematical formula that expresses the average TCP throughput as a function of the underlying path characteristics. Probably the most well-known such model is the “square-root” formula of [89]:

$$E[R] = \frac{M}{T\sqrt{\frac{2bp}{3}}} \quad (2)$$

where $E[R]$ is the *expected TCP throughput* (as opposed to R which denotes the *actual or measured throughput* and $\hat{R}$ which denotes the *predicted throughput*). In the previous formula, M is the flow’s Maximum Segment Size, b is the number of TCP segments per new ACK, while T and p are the RTT and loss rate, respectively, as experienced by the TCP flow. The main mathematical symbols we use are summarized in Table 1. This model is fairly accurate for bulk TCP transfers in which packet losses are recovered with Fast-Retransmit. Analytical results such as (2) have been very useful in understanding the relation between TCP throughput and certain key path characteristics, such as loss rate and RTT.

Another motivation for the research that led to these TCP models was the ability to *predict* the throughput of a TCP flow given estimates of the relevant path characteristics [38, 57, 100]. In this section, we first present a more complete TCP throughput formula, as well as the corresponding FB predictor. Although several similar models exist in the literature (see [128] and the references therein), we emphasize that our remarks regarding the accuracy and limitations of FB prediction are not specific to the particular formula we use.

Table 1: Main Symbols

T	RTT experienced by target flow
$\hat{T}$	RTT measured with periodic probing prior to target flow
$\tilde{T}$	RTT measured with periodic probing during target flow
p	loss rate experienced by target flow
$\hat{p}$	loss rate measured with periodic probing prior to target flow
$\tilde{p}$	loss rate measured with periodic probing during target flow
p'	congestion event probability experienced by target flow
R	actual throughput of target flow
$\hat{R}$	predicted throughput of target flow
$\tilde{R}$	expected throughput of target flow based on $\tilde{T}$ and $\tilde{p}$
$\hat{A}$	available bandwidth measured prior to target flow
W	maximum window of target flow

3.2.1 A formula-based TCP throughput predictor

The TCP throughput formula that we use is the PFTK result of [100], which improves on the square-root formula especially in the presence of retransmission timeouts and/or a limited maximum window:

$$E[R] = \min \left(\frac{M}{T\sqrt{\frac{2bp}{3}} + T_o \min(1, \sqrt{\frac{3bp}{8}})p(1 + 32p^2)}, \frac{W}{T} \right) \quad (3)$$

where T_o is the TCP retransmission timeout period, and W is the maximum window size (limited by the socket buffer size at the sender or receiver). We emphasize that p and T , in the previous equation, are the average loss rate and RTT that the *target flow* (i.e., the TCP flow whose throughput we try to predict) experiences. Notice that the loss rate p may be zero, in which case the flow is *lossless* and $E[R]$ is given by the term W/T .

Suppose now that we want to apply (3) to TCP throughput prediction. The main problem is that we do not know, when predicting, the loss rate and RTT that the flow will experience during its lifetime. The obvious approach, which has been previously followed in practice (e.g., in overlay routing [24]), is to measure the loss rate and RTT *before* the transfer with a utility such as *ping*, and then apply those estimates of p and T in (3). Suppose that $\hat{p}$ and $\hat{T}$ are the loss rate and RTT estimates based on measurements prior to the flow. Then, if $\hat{p} \approx p$ and $\hat{T} \approx T$, the prediction accuracy will be only limited by the

accuracy of these approximations and the accuracy of the mathematical model that was used to derive (3). We can expect that $\hat{p} \approx p$ and $\hat{T} \approx T$ when the TCP flow imposes a minor load on the path's bottleneck, and so it does not affect significantly the RTT and loss rate of the path.

A limitation of the previous approach is that it does not apply to *lossless paths*, i.e., when $\hat{p}=0$. In that case, $W/\hat{T}$ can be totally unrelated to the realized throughput, especially if W is much larger than the bandwidth-delay product of the underlying path. One approach to deal with lossless paths is to predict the TCP throughput based on the *available bandwidth* $\hat{A}$ of the path prior to the TCP flow, when $\hat{A} < W/\hat{T}$. The available bandwidth is the non-utilized part of the bottleneck link's capacity, and it can be measured non-intrusively with end-to-end probing techniques [68, 70, 110, 131]. Although the available bandwidth and TCP throughput are not expected to be exactly equal, $\hat{A}$ can be used as a first-order approximation of R when the flow is not limited by its maximum window size W [68]. On the other hand, if $W/\hat{T} < \hat{A}$, the flow cannot obtain all the available bandwidth due to its limited maximum window, so $W/\hat{T}$ is a more reasonable predictor; we refer to such flows as *window-limited*.

To summarize, the FB predictor that we consider in the rest of this chapter is given by the following equation:

$$\hat{R} = \begin{cases} \min \left(\frac{M}{\hat{T} \sqrt{\frac{2b\hat{p}}{3}} + \hat{T}_o \min(1, \sqrt{\frac{3b\hat{p}}{8}}) \hat{p} (1+32\hat{p}^2)}, \frac{W}{\hat{T}} \right) & \text{if } \hat{p} > 0 \\ \min \left(\frac{W}{\hat{T}}, \hat{A} \right) & \text{if } \hat{p} = 0 \end{cases} \quad (4)$$

where $\hat{R}$ is the predicted throughput, while $\hat{T}$, $\hat{p}$, and $\hat{A}$, are the measured RTT, loss rate, and available bandwidth prior to the TCP flow. We estimate the retransmission timeout period as

$$\hat{T}_o = \max(1\text{sec}, 2\text{SRTT}) \quad (5)$$

where SRTT is set to the measured RTT $\hat{T}$ prior to the target flow. Note the differences between (3) and (4): the latter relies on the estimates $\hat{T}$, $\hat{p}$, $\hat{T}_o$, rather than on the actual values T , p , T_o , and it also has a component that depends on the available bandwidth estimate $\hat{A}$.

In the following, we explain three important limitations of the above predictor using basic insight and simple *ns2* simulation scenarios.

3.2.2 Effect of the extra load due to the target flow

Basic queueing theory tells us that an increase in the utilization of a queue (with non-periodic arrivals) increases the average queueing delay. Similarly, in a queue with a limited buffer, an increase in the utilization can cause a higher loss probability. The increase in the queueing delays and/or the loss probability tends to be more significant when the utilization becomes significantly higher, or when the utilization was already high even before the additional load.

These basic facts can cause major errors in FB prediction. The reason is that the RTT $\hat{T}$ measured prior to the target flow will not reflect the queueing delay during that transfer. So, $\hat{T}$ can be lower than the RTT T that the target flow experiences. Similarly for the loss rate, it can be that $\hat{p} < p$. The net result of either effect is that the FB predictor can overestimate the TCP throughput, especially when the target flow increases the utilization of the bottleneck significantly or when the latter is already very high. Note that the experimental validation of the PFTK result, reported in [100], was based on the “posthumous” estimation of p and T , i.e., from *tcpdump* packet traces collected at the sender/receiver while the target flow was in progress. Of course the same approach is not possible in the prediction context.

Table 2: (A) Increased RTT (B) Increased Loss Rate.

Simulation	C (Mbps)	B (pkts)	$\hat{T}$ (ms)	$\tilde{T}$ (ms)	$\hat{R}$	R
1	20	200	16.7	23.4	13.3	9.2
2	50	200	20.1	20.1	10.6	10.6

(A)

Simulation	C (Mbps)	B (pkts)	$\hat{p}$ (%)	$\tilde{p}$ (%)	$\hat{R}$	R
1	10	50	0.14	0.46	9.2	4.6
2	100	300	0.2	0.22	6.2	6.1

(B)

We use simple *ns2* simulations to demonstrate how such prediction errors could take place. The simulations use a simple dumbbell topology with the bottleneck link (capacity C ,

buffer space B) in the center, and the TCP flows (both the cross traffic and the target flow) traversing that link. Table 2-(A) gives an example of the discrepancy between the path's RTT $\hat{T}$ prior to the target flow and the RTT $\tilde{T}$ during the target flow. The target flow as well as a single cross traffic TCP flow are window-limited ($W=30\text{KB}$) so that the bottleneck does not experience any packet losses ($p=0$). Note that the capacity C in Simulation-1 is lower than in Simulation-2, causing a higher utilization in the former. Specifically, the utilization in Simulation-1 increases from 72% to 98.5% after the target flow starts; the corresponding utilization increase in Simulation-2 is from 24% to 48%. As a result, $\tilde{T}$ is significantly different than $\hat{T}$ in Simulation-1, causing a substantial prediction error. The prediction is very accurate, on the other hand, in Simulation-2 because the utilization remained relatively low even after the target flow started.

Similarly, Table 2-(B) exemplifies the discrepancy between the loss rate during the target flow $\tilde{p}$ and the loss rate prior to the target flow $\hat{p}$. The cross traffic is a single TCP flow in Simulation-1, and 15 TCP flows in Simulation-2. None of the flows, including the target flow, are window-limited. Here, we see a large loss rate increase in Simulation-1 after the target flow starts. The loss rate increase is much smaller in Simulation-2 because the buffer size B is significantly larger in that case. As a result, the prediction error is substantial in Simulation-1 but minor in Simulation-2.

3.2.3 Errors due to the TCP sampling behavior

Even when the target flow does not affect significantly the path's RTT and loss rate, it is still hard to estimate the RTT and loss rate that the TCP target flow experiences. TCP reduces its packet transmission rate when it experiences losses, which means that it tends to "sample" the RTT and packet loss processes less frequently when the path is congested. This is a very different sampling behavior than that of a utility such as ping, which typically sends periodic probing packets. Also, TCP tends to send bursts of data packets when self-clocking fails (e.g., due to ACK compression), which also leads to a different sampling behavior than periodic probing.

To make things more complex, a mathematical model for the TCP throughput may

be based on certain assumptions that affect the interpretation of parameters such as T or p . For instance, the PFTK model assumes that T is constant, and that when a packet is dropped all the remaining packets in that “flight” are also dropped (referred to as a “congestion event”). As a result, the parameter p in (3) should not be the unconditional loss probability among all packets of the target flow, but the congestion event probability. The discrepancy between these two parameters was one of the main focus points in [57].

Table 3 shows three different “loss rates”, all obtained from the same simulation as in Table 2-(B). In this Table, $\tilde{p}$ is a ping-based estimate of the loss rate measured with periodic probing packets (40 bytes every 100ms) during the target flow, p is the (unconditional) loss rate that the target flow experienced, and p' is the congestion event probability estimated from a detailed analysis of the *ns2* packet trace. Notice the striking difference, more than an order of magnitude, between $\tilde{p}$ and the other two metrics. Ping estimates a larger loss rate, due to its non-adaptive sampling behavior that we mentioned earlier. The difference between p and p' is also noticeable, although not major. Unfortunately, it is not known how to measure p' or p *prior* to the start of the target flow. For this reason, existing FB prediction schemes use ping-based loss rate estimates, which are also much simpler to obtain.

Table 3: Different Loss Rate Estimates during a TCP Flow.

Simulation	$\tilde{p}$	p	p'
1	0.04	0.0046	0.0028
2	0.03	0.0022	0.0015

3.2.4 Errors due to the difference between available bandwidth and TCP throughput

As previously mentioned, when $\hat{p}=0$ and $\hat{A} < W/\hat{T}$, we predict the throughput of the target flow based on the path’s available bandwidth $\hat{A}$ prior to that flow. These two metrics, however, can be significantly different in certain cases [70].

First, whether a TCP flow can saturate the available bandwidth of a path depends on the buffer space B at the bottleneck. If B is not sufficiently large, packet losses can

cause significant underutilization and the resulting TCP throughput can be lower than $\hat{A}$. Second, if the competing cross traffic at the bottleneck is made of elastic flows (e.g., persistent TCP flows), the target flow can capture more than $\hat{A}$, by receiving some of the bandwidth previously occupied by cross traffic flows. The actual difference between available bandwidth and TCP throughput in that case depends on the number and the RTTs of the competing TCP flows.

Consequently, the available bandwidth $\hat{A}$ prior to the target flow can cause either overestimation or underestimation of the flow’s throughput, depending on the amount of buffering and the ”elasticity” of the cross traffic in the path. Given that it is hard to infer network buffering and cross traffic elasticity in practice, it is unclear whether we can design a better FB predictor than $\hat{A}$ in the case of lossless paths.

3.3 FB prediction accuracy

The previous section argued that FB prediction can be inaccurate under certain network conditions. In this section, we show measurement results from several Internet paths that quantify the inaccuracy of FB prediction, and further analyze these prediction errors. First, we describe the measurement methodology and the dataset we use throughout this chapter.

3.3.1 Overview of measurement methodology

Our measurements were collected on 35 Internet paths that interconnect nodes of the RON testbed [26]¹. The RON nodes that we used are located mostly in US universities, but there are also two nodes in Europe and one in Korea. Out of the 35 paths, five are transatlantic paths, one between Korea and New York-NY, and the rest within the US. Most paths can deliver at least 10Mbps, but seven of the paths had a DSL or T-1 bottleneck.

We collected seven measurement ”traces” on each path, with a total of 245 traces across all paths. Each trace consists of 150 back-to-back measurement ”epochs”. An epoch starts with an available bandwidth measurement using pathload, followed by a 60-sec measurement of $\hat{p}$ and $\hat{T}$ using a homespun ping utility that generates a 41-byte probing packet every

¹ We prefer RON instead of PlanetLab, because the latter is often too heavily loaded for accurate network measurement.

100ms, followed by a 50-sec TCP transfer (target flow) generated by IPerf [69] (see Figure 2). RTT and loss rate estimates are also measured during the TCP transfer. A 50-second transfer on these paths is long enough to ensure that the flow spends a negligible fraction of its lifetime in the initial slow-start. A total of 36750 TCP transfers (in the same number of epochs) were performed. The duration of each epoch (and also the time interval between successive TCP transfers) was about 2-3 minutes, while the duration of each trace was about 6 hours. The measurements were collected during a week in May 2004.

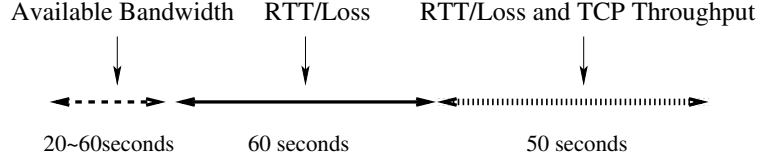


Figure 2: A measurement epoch. 150 such epochs were recorded during each trace, with 7 traces collected per path.

IPerf allows us to directly control the maximum TCP window size W by limiting the receiver socket buffer size. Unless otherwise noted, we used $W=1\text{MB}$, which is large enough to saturate all the paths we experimented with and cause congestion. To examine the effect of W , we also performed the same measurements with $W=20\text{KB}$, which, as will be shown later, limits the transfer to only a fraction of the available bandwidth on most paths.

Each epoch provides the following measurements: the pre-transfer estimates $\hat{p}$, $\hat{T}$, $\hat{A}$, the actual TCP throughput R , and the estimates of the loss rate $\tilde{p}$ and RTT $\tilde{T}$ during the transfer. The first three estimates are used in (4) to predict the TCP throughput $\hat{R}$, which is then compared with the actual throughput R . We collected $\tilde{p}$ and $\tilde{T}$ in order to evaluate how the corresponding metrics change due to the target flow, and also to quantify the prediction error if it was possible to estimate $\tilde{p}$ and $\tilde{T}$ before the target flow.

We define the *relative prediction error* E of an individual measurement epoch as

$$E = \frac{\hat{R} - R}{\min(\hat{R}, R)} \quad (6)$$

Notice that the denominator $\min(\hat{R}, R)$ gives E the property that overestimation or underestimation by the same factor $w > 1$, i.e., $\hat{R}=wR$ for the former and $\hat{R}=R/w$ for the latter, yields the same relative error $w - 1$ (in absolute value).

To report a single figure for n measurements in a time series (specifically, for all the 150 epochs of a trace), we use the *Root Mean Square Relative Error (RMSRE)* statistic, defined as

$$\text{RMSRE} = \sqrt{\frac{1}{n} \sum_{i=1}^n E_i^2} \quad (7)$$

where E_i is the relative error of measurement i .

3.3.2 Results

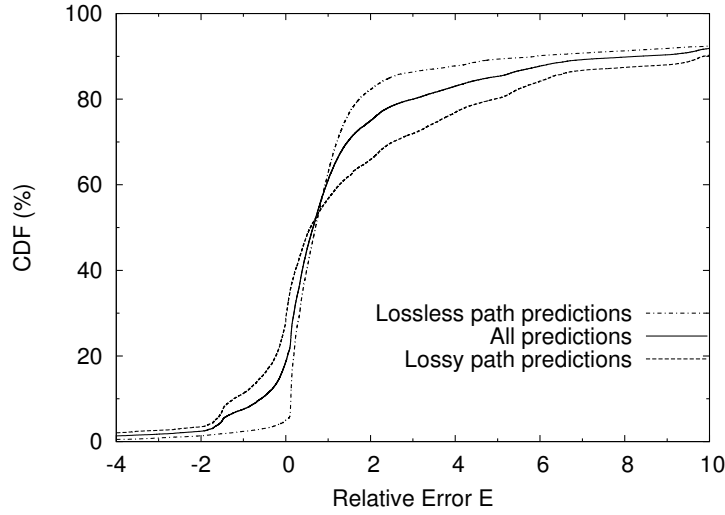


Figure 3: CDF of E for all predictions, for predictions in lossy paths, and for predictions in lossless paths.

Prediction error in lossy and lossless paths: Figure 3 shows the CDF of E for all measurements, across all traces and paths. It also shows separately the CDFs of E for the subset of lossy path predictions (based on the PFTK model) and for the subset of lossless path predictions (based on the available bandwidth estimate $\hat{A}$)². Let us first focus on the “all predictions” curve. Notice that *for roughly 40% of all measurements, the prediction is an overestimation by more than a factor of two ($E \geq 1$)*. In fact, the overestimation errors are larger than an order of magnitude ($E \geq 9$) for almost 10% of the measurements. The underestimation errors are much less dramatic and common, with only 10% of the measurements suffering from an underestimation by more than a factor of two ($E < -1$).

²For $W=1\text{MB}$, we have $\hat{A} < W/T$ in all paths.

In the case of lossless paths, underestimation errors occur very rarely, while the overestimation errors are considerably lower and less common than in lossy paths. The reason is that in lossless paths, our FB predictor does not rely on the erroneous RTT and loss rate estimates prior to the target flow. The remaining errors can be attributed to the differences between TCP throughput and available bandwidth, discussed in § 3.2.4. The fact that, in lossless paths, overestimation is the only major type of prediction error implies that either pathload overestimates the path’s available bandwidth, or that TCP cannot saturate the available bandwidth in its path due to random losses or insufficient buffering at the bottleneck link.

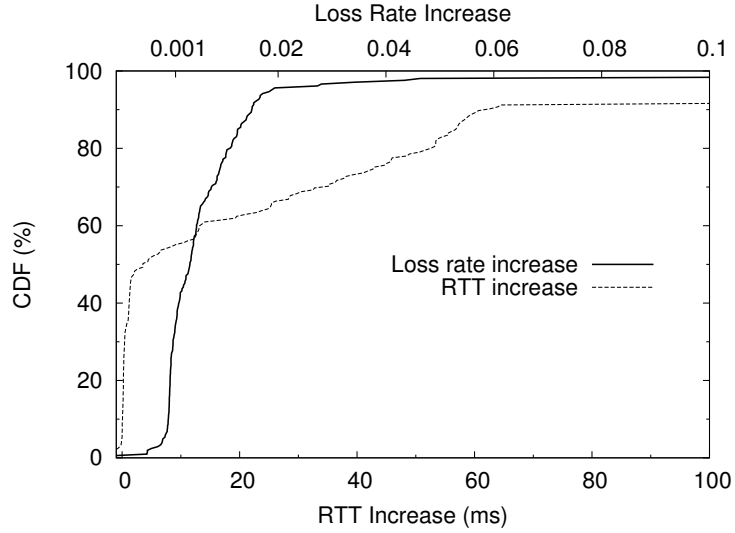


Figure 4: CDF of RTT and loss rate increase due to target flow.

RTT and loss rate increases due to target flow: Returning to the case of lossy paths, the fact that overestimation is much more dramatic than underestimation illustrates the dominance of the issue discussed in § 3.2.2, namely $\hat{T} < T$ and $\hat{p} < p$. Figure 4 shows the distributions of the increases in RTT and in loss rate after the start of the target flow. The increases were measured as $\tilde{T} - \hat{T}$ and $\tilde{p} - \hat{p}$ respectively (recall that $\tilde{T}$ and $\tilde{p}$ are estimates of T and p during the target flow). Notice that in about 50% of the measurements, the RTT did not increase significantly. In 40% of the measurements, however, the target flow caused an RTT increase between 5ms and 60ms. In 10% of the measurements the RTT increase was higher than 100ms, probably due to congested low-capacity links (DSL or T-1). The

loss rate, on the other hand, increased by 0.1% to 2% in almost all measurements. Even though this loss rate increase may appear small in magnitude, recall that TCP throughput is inversely proportional to the square-root of the loss rate (see (2)). For example, an increase of the loss rate from 0.1% to 1% can cause a throughput overestimation by a factor of about 3.2.

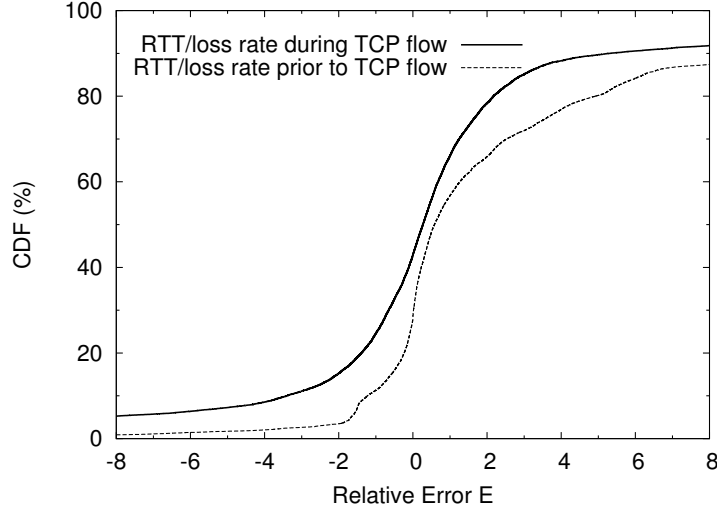


Figure 5: Prediction errors using $\tilde{T}$ and $\tilde{p}$ (RTT and loss-rate during the target flow) and using $\hat{T}$ and $\hat{p}$ (RTT and loss-rate prior to the target flow).

Errors due to periodic RTT and loss rate sampling: An interesting hypothetical question is the following: *how accurate would FB prediction be, if we had the estimates of the path's RTT $\tilde{T}$ and of the loss rate $\tilde{p}$ during the target flow?* In theory, it may be possible to estimate $\tilde{T}$ and $\tilde{p}$, given $\hat{T}$ and $\hat{p}$, based on a model that captures the impact of the target flow on the queue of the bottleneck link. Figure 5 shows the CDF of the FB prediction error when we feed in (4) the ping-based RTT $\tilde{T}$ and loss-rate $\tilde{p}$ during the target flow. The CDF refers only to lossy paths. Note that using $\tilde{T}$ and $\tilde{p}$ makes the relative error significantly lower than using $\hat{T}$ and $\hat{p}$ ($-3 < E < 3$ for about 80% of the predictions). Also, overestimation and underestimation become equally likely (the CDF of E is practically symmetric). Despite the benefits of knowing $\tilde{T}$ and $\tilde{p}$, the prediction errors are still significant: more than half of the prediction errors are still larger than a factor of two. These prediction errors can be attributed to the fact that TCP samples the RTT and

loss rate processes in an adaptive way, rather than periodically. In terms of our notation, the remaining prediction errors are due to the differences between $\tilde{T}$ and T , and between $\tilde{p}$ and p .

Variation of prediction error across different paths and traces:

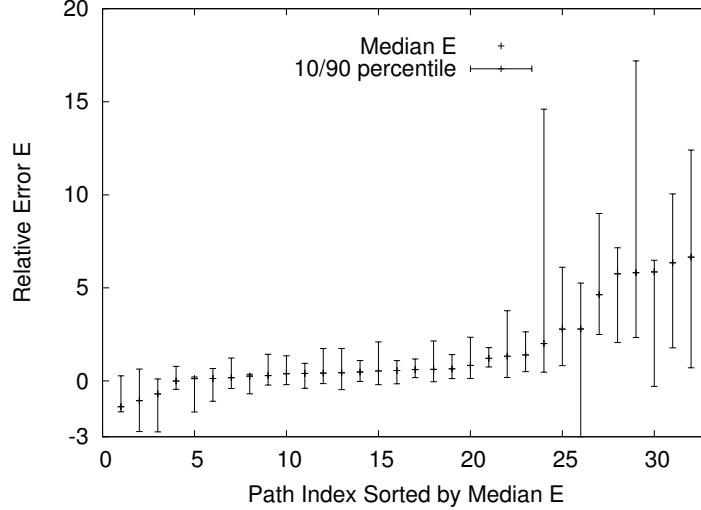


Figure 6: Variation of the prediction error across different paths.

Figure 6 shows the median, as well as the 10/90-th percentiles, of the relative prediction error on a per path basis (recall that we have 7×150 measurements from each path). There are three paths that we did not include in this graph because they have excessive prediction errors. With the exception of 4-5 paths that mostly give small underestimation errors, most paths give only overestimation errors. Another interesting point is that *different paths exhibit widely different predictability*. About 10 out of the 35 paths have much larger ranges of prediction error than the rest, extending up to $E=10$ or larger. This implies that, not only is it hard to predict TCP throughput with an FB method, but also it is hard to bound the prediction error that should be anticipated. We will return to the reasons behind this large variation of the prediction error across different paths in § 3.6.

Figure 7 shows the distribution of the per-trace RMSRE. Recall that we calculate a single RMSRE value for each trace (with 150 successive epochs per trace). The important observation here is that about 70% of the traces have an RMSRE that is larger than 1.0. This is a rather disappointing accuracy for a predictor, as it implies that in most cases, the

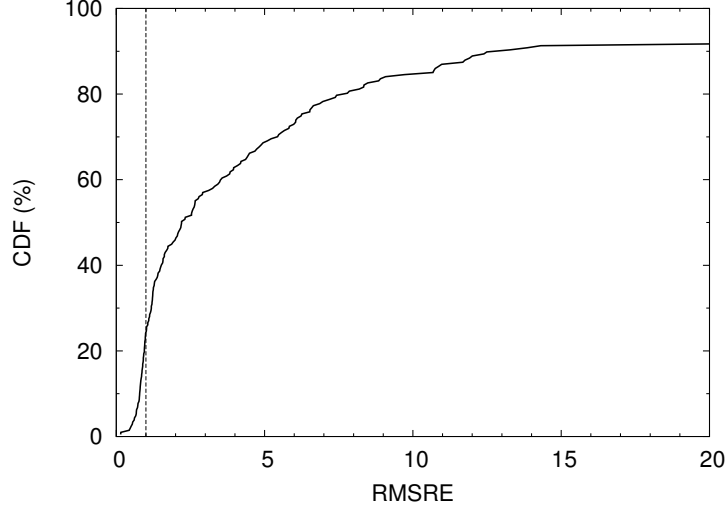


Figure 7: CDF of per-trace RMSRE.

prediction error will be more than a factor of two.

Figures 6 and 7 naturally lead to the question of what kind of paths or transfers tend to have the largest prediction errors. Figure 8 is a scatter plot that shows the relationship between actual throughput of each transfer R and the prediction error E for its throughput. Clearly, the largest prediction errors occur to transfers that have very small throughput ($R \leq 0.1\text{Mbps}$). We cannot, however, make general connections between FB predictability and basic path characteristics, given the limited knowledge we have about the underlying paths. Notice that the well-shaped curves present in the figure are caused by a series of very similar predictions (based on the almost constant RTT and loss rate measurements on the same path) and the corresponding series of realized throughput that vary slightly within a small range.

Predictability of window-limited flows: Another interesting question is whether the FB predictor would be more accurate for window-limited flows (i.e., $W/\hat{T} < \hat{A}$), given that those flows do not attempt to saturate the network path. To answer this question, we extended each epoch with another IPerf TCP transfer with $W=20\text{KB}$, and performed one experiment on each path. We verified that this transfer was window-limited on 18 of the 35 paths, and the ratio $W/(\hat{T}\hat{A})$ varied between 0.02 to 0.81. Figure 9 compares the RMSRE between the transfers with a large maximum window ($W=1\text{MB}$) and a small maximum

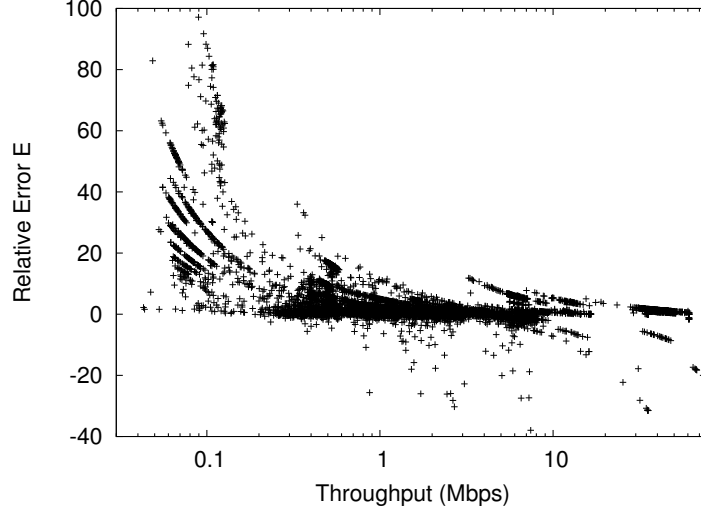


Figure 8: Actual throughput vs. prediction error.

window ($W=20\text{KB}$). Note the log-scale of the Y-axis. In all paths, the prediction error of window-limited flows was lower, often by a large factor. In particular, 14 out of the 18 paths have an RMSRE that is less than 1.0 for window-limited flows. We anticipate that for many applications a TCP throughput prediction that is accurate within a factor of two would be adequate. For such applications, an FB predictor may be appropriate as long as the transfer does not attempt to saturate the underlying available bandwidth.

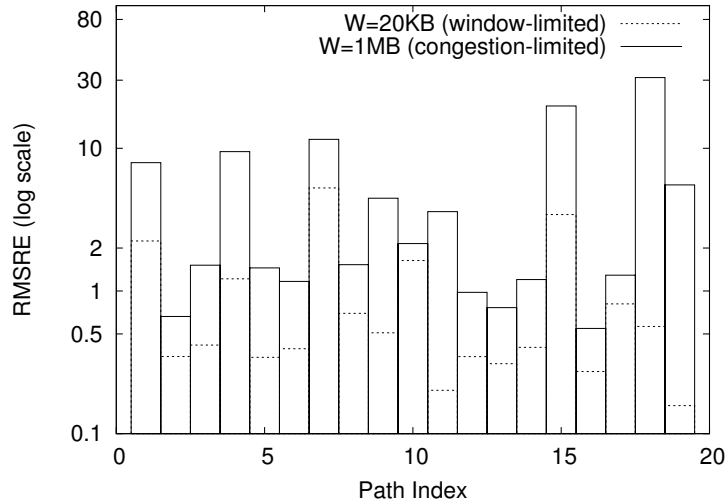


Figure 9: Prediction accuracy for window-limited vs. congestion-limited flows.

3.3.3 Summary

The results of this section showed that FB prediction can be inaccurate, mostly in lossy paths and when the target flow saturates the underlying path. The major cause of prediction errors is that the RTT and/or loss rate before the transfer are significantly different than while the transfer is in progress. We note again that this cause of prediction errors is not specific to the PFTK formula. This implies that it is unlikely that other TCP throughput models would have produced more accurate FB predictions. Other important causes of prediction errors are the difference between periodic and TCP sampling of the RTT and loss rate processes, and the difference between TCP throughput and available bandwidth.

Our results also suggest that more sophisticated techniques that estimate the RTT and loss rate to be experienced by the target flow can significantly improve FB prediction. Such techniques, however, should take into account the load that will be exerted by the target flow, and the impact of that load on the queueing delays and losses in the path. More information about the underlying path, such as the capacity, available bandwidth, buffer size, number of competing flows, etc, may help achieving this goal.

3.4 *History-based prediction*

A fundamentally different approach to predicting the throughput of a large TCP transfer is to use throughput measurements of previous transfers in the same path. This *History-Based* (HB) prediction method is similar to traditional time series forecasting, where past samples of an unknown random process are used to predict the value of the process in the future. The HB approach is possible in applications where large TCP transfers are performed repeatedly over the same path.

In this section, we first introduce three families of simple linear predictors (Moving Average, Exponential Weighted Moving Average, and non-seasonal Holt-Winters). We do not examine more complex linear predictors such as ARMA or ARIMA because the selections of both their order and of their linear coefficients require a large number of past measurements [33]; instead, we expect that applications will have to perform TCP throughput HB prediction based on a limited number of past transfers (say 10-20). We

then show that two distinct time series “pathologies”, namely *outliers* and *level shifts*, can have a major impact on the prediction error, and propose simple heuristics that can deal with these pathologies effectively.

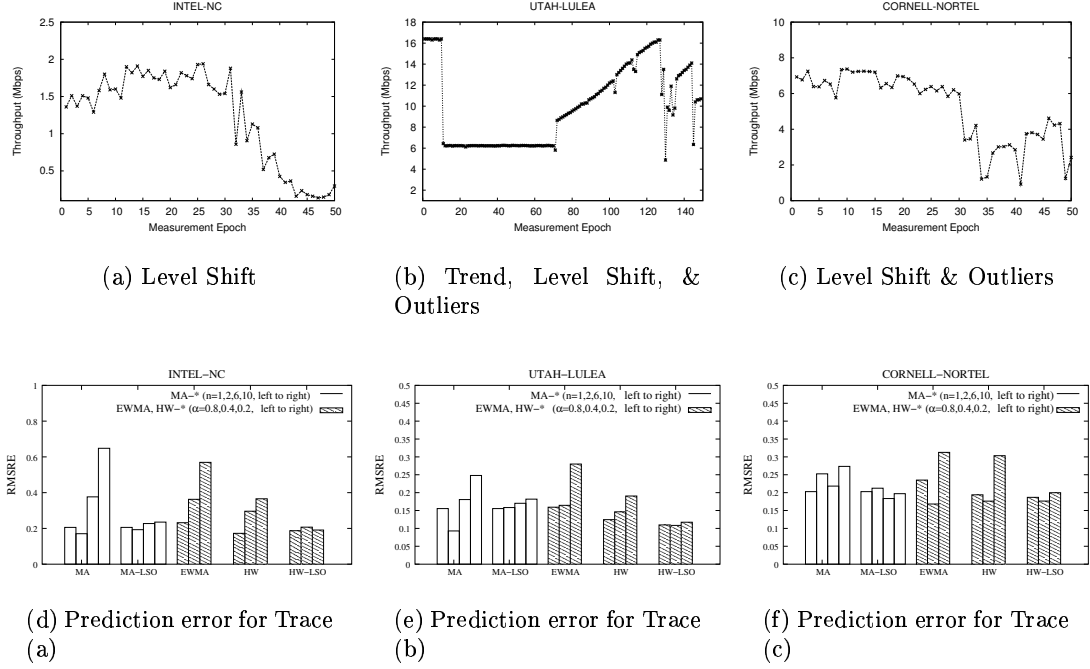


Figure 10: Examples of TCP throughput traces and the prediction errors (RMSRE) with various predictors.

3.4.1 Linear Predictors

- *Moving Average (MA)*. Given a time series X , the one-step n -order MA (n -MA) predictor is

$$\hat{X}_{i+1} = \frac{1}{n} \sum_{k=i-n+1}^i X_k$$

where $\hat{X}_i$ is the predicted value and X_i is the actual (observed) value at time i . If n is too small, the predictor cannot smooth out the noise in the underlying measurements. On the other hand, if n is too large the predictor cannot aptly adapt to non-stationarities (e.g., level shifts due to load variations or routing changes).

- *Exponentially Weighted Moving Average (EWMA)*. The one-step EWMA predictor is

$$\hat{X}_{i+1} = \alpha X_i + (1 - \alpha)\hat{X}_i$$

where α is the weight of the last measurement ($0 < \alpha < 1$). Similar to the MA predictor, a higher α cannot smooth out the measurement noise, while a lower α is slow in adapting to changes in the underlying time series.

- *Holt-Winters (HW)*. The non-seasonal Holt-Winters predictor is a variation of EWMA that attempts to capture the *trend* in the underlying time series, if such a trend exists³. This predictor is more appropriate than EWMA for non-stationary processes, especially if the latter exhibit a linear trend. A non-seasonal HW predictor maintains a separate smoothing component $\hat{X}_i^s$ and a trend component $\hat{X}_i^t$, and it depends on two parameters α and β , both in $(0, 1)$. Specifically, the predicted value at time i is

$$\hat{X}_i^f = \hat{X}_i^s + \hat{X}_i^t,$$

where

$$\hat{X}_{i+1}^s = \alpha X_i + (1 - \alpha)\hat{X}_i^f,$$

$$\hat{X}_{i+1}^t = \beta(\hat{X}_i^s - \hat{X}_{i-1}^s) + (1 - \beta)\hat{X}_{i-1}^t.$$

The initial values of $\hat{X}^s$ and $\hat{X}^t$ are X_0 and $X_1 - X_0$, respectively, assuming that the time series starts at $i=0$.

3.4.2 Detection of Level Shifts and Outliers

While experimenting with various predictors, we found out that the largest prediction errors are often caused by level shifts and outliers in the observed time series. Furthermore, if we manage to somehow avoid these two characteristics in the throughput time series, then the exact choice of the predictor, or of its parameters, does not make a significant difference.

A level shift is a type of non-stationarity, and it causes a significant and typically sudden change in the mean of the observed time series. An outlier is a measurement that

³The forecasting literature sometimes refers to this non-seasonal version of the HW predictor as Holt's predictor.

is significantly different, beyond the typical level of statistical variations, relative to nearby measurements. Both outliers and level shifts have been studied extensively in the theory of forecasting [120]. In Figures 10(a), 10(b) and 10(c) we show examples of traces that exhibit both outliers and level shifts, observed in our TCP throughput measurements. One way to deal with level shifts, after they are detected, is to restart the predictor, ignoring all previous history. Outliers, on the other hand, can be just ignored.

We next describe simple heuristics to detect level shifts and outliers. Suppose that $\{X_1, \dots, X_n\}$ is the sequence of past measurements, ignoring outliers, where X_1 is the first measurement after the last detected level shift. We determine that the measurement X_k is an increasing (decreasing) level shift if it satisfies the following three conditions:

1. The measurements $\{X_1, \dots, X_{k-1}\}$ are all lower (higher) than the measurements $\{X_k, \dots, X_n\}$,
2. The median of $\{X_1, \dots, X_{k-1}\}$ is lower (higher) than the median of $\{X_k, \dots, X_n\}$ by more than a relative difference χ , and
3. $k + 2 \leq n$.

The last condition aims to avoid misinterpreting an outlier as a level shift. Upon the detection of a level shift, we ignore all measurements prior to X_k and restart the predictor from X_k . On the other hand, a measurement X_k (with $k < n$) is considered an outlier if it differs from the median of the measurements in $\{X_1, \dots, X_n\}$ by more than a relative difference of ψ . Outliers are discarded from the history of previous measurements.

Figures 10(d), 10(e) and 10(f) show the RMSRE for the three sample traces with five different predictors: MA, MA-LSO, EWMA, HW, and HW-LSO. The *LSO* acronym is used when we use the previous heuristics for the detection of Level Shifts and Outliers. For the MA and MA-LSO predictors, we show results for four different values of n (see the figure's legend). For the EWMA and HW predictors, we show results for three values of α . We observed that, at least for our datasets, the RMSRE does not strongly depend on β , χ and ψ . We found empirically that the following values perform reasonably well, in terms of minimizing the RMSRE, at least in our datasets: $\beta=0.2$, $\chi=0.3$, and $\psi=0.4$. On the other hand, the parameters n and α play a major role in the prediction accuracy when the

LSO heuristic is *not* used. The LSO heuristic decreases the prediction error significantly, and makes the predictors more robust to the choice of n or α . The difference between the accuracy of MA-LSO and HW-LSO is not major, although the latter tends to perform slightly better. More results for the accuracy of these two predictors will be given in the next section.

3.5 *HB prediction accuracy*

In this section, we apply the HB predictors of the previous section to the measurements described in § 3.3. Our objective is to compare the most promising HB predictors that we experimented with, and to examine how the HB prediction accuracy varies in different paths, with window-limited flows, and with different transfer frequencies.

3.5.1 Results

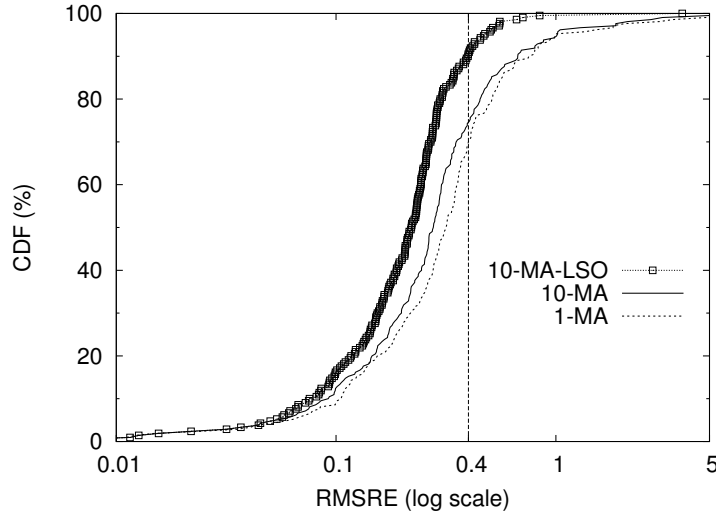


Figure 11: Moving Average prediction error.

Comparison of HB predictors: Figures 11 and 12 summarize the prediction error (in terms of RMSRE) of several MA and HW predictors, respectively. The EWMA predictor performs similarly to HW. Without LSO, the n -MA predictors perform very similarly when $n < 20$ (we do not show all of them), except the trivial case of $n=1$ that performs slightly worse. With LSO, there is a significant reduction in the RMSRE of MA predictors. For HW

predictors, $\alpha=0.8$ (0.8-HW) performs visibly better than $\alpha=0.4$. Further experimentation showed that $\alpha=0.8$ is close to the optimal for our dataset, and we use this value for HW predictor hereafter unless otherwise noted. HW predictor is also significantly improved with LSO. A comparison of MA-LSO (with $n=10$) and HW-LSO shows that the accuracy of the latter is only slightly better. This is an indication that not many of our traces exhibit linear trends.

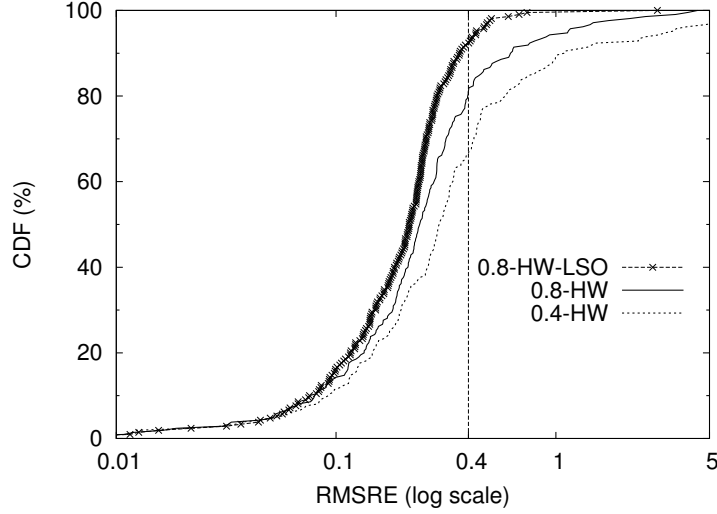


Figure 12: Holt-Winters prediction error.

Comparison of FB and HB predictors: Even though these two classes of predictors are complementary, in some cases it may be possible to use either FB or HB predictor. Comparing the RMSRE of the FB predictor (see Figure 7) with that of the HB predictors, we can see that the accuracy of the latter is dramatically better. Specifically, HB predictors give RMSRE less than 0.4 for about 90% of the traces. The same RMSRE percentile for the FB predictor is 20, while the median RMSRE is about 2. One may argue that this comparison is not fair for FB, since FB is applicable without any knowledge of previous TCP transfer throughput measurements. If it is possible to collect and use such historical data, however, this comparison shows that HB prediction should be preferred to FB prediction.

RMSRE vs. CoV of throughput measurements: We are interested in the relation between the prediction RMSRE for a given trace and the Coefficient of Variation (CoV) of

the corresponding TCP throughput time series⁴. The reason for this comparison will become clear in the following section, where we use the CoV as an indication of the TCP throughput *predictability* in a path. To calculate the CoV of a trace, we isolate stationary periods based on the detected level shifts and exclude outliers. We then calculate the weighted average of the CoVs for different periods (with the weight of each period being the number of corresponding measurements). In the RMSRE calculations, we also exclude measurements that were identified as outliers. Figure 13 shows the correlation of the CoV and RMSRE for each collected trace, using the HW-LSO predictor. Note the strong correlation between the two metrics. The correlation coefficient between them is 0.91. We can thus assume, at least as a first-order approximation, that the RMSRE prediction error with HW-LSO is equal to the CoV of the corresponding time series, at least in the datasets we experimented with.

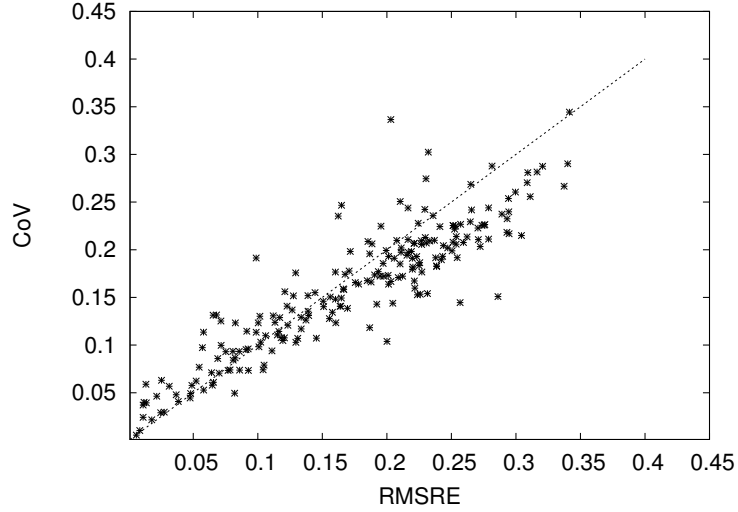


Figure 13: Prediction error versus CoV.

Variations in path predictability: Figure 14 provides close-up views of the accuracy of several predictors in 12 sample paths. We classify these paths into four representative classes (described in the figure’s caption), based on the average prediction error as well as the variation of the error across different traces in the same path. Each subfigure represents a

⁴Recall that the CoV is the ratio of the standard deviation to the mean.

specific path, with the X-axis numbers indicating different traces. For each trace, successive bars show the RMSRE with 1-MA, 10-MA, HW, and HW-LSO, from left to right. As previously noted, *the HW-LSO predictor is almost always the best in terms of RMSRE*. A more important observation from these graphs, however, is that *there are major differences in the prediction error between different paths*. Some paths have quite low RMSRE and they are fairly predictable, others have larger RMSRE but the RMSRE is quite stable (predictable errors), while others have either large RMSRE variations (unpredictable errors), or high RMSRE (unpredictable throughput). What causes different paths to behave so differently in terms of their TCP throughput predictability? We focus on this important question in the next section.

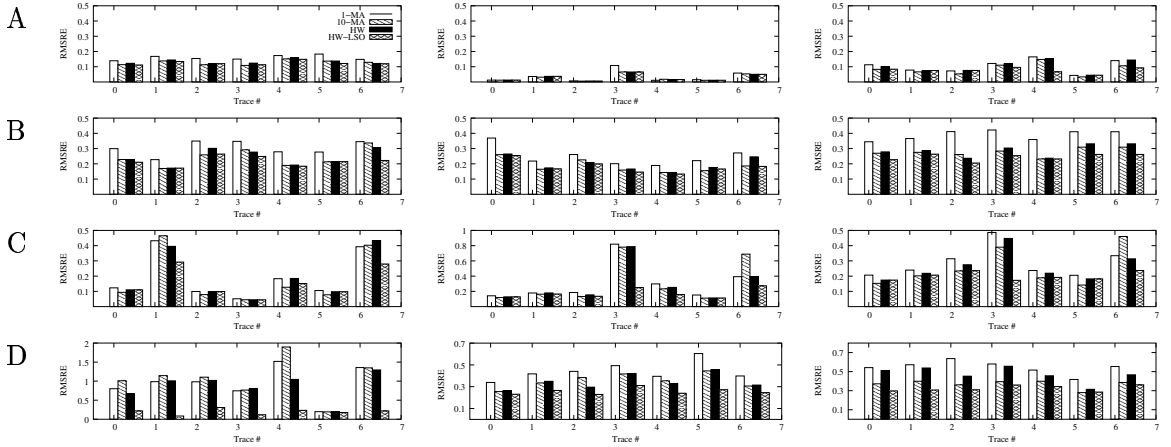


Figure 14: A: Predictable paths (low RMSRE), B: Paths with small and predictable errors (stable RMSRE), C: Paths with small but unpredictable errors (varying RMSRE), D: Unpredictable paths (high RMSRE, notice the different Y-axis ranges).

Predictability of window-limited flows: When the target flow is window-limited, it would probably not be subject to the dynamic variations of the available bandwidth, and so we can expect higher predictability. Figure 15 compares the prediction error for window-limited flows ($W=20\text{KB}$) and for congestion-limited flows ($W=1\text{MB}$), using the same traces as in Figure 9. Notice that *window-limited flows have a lower RMSRE, confirming the insight that the throughput is more predictable when the target flow does not attempt to saturate the path*. The RMSRE difference is not always major, however, especially when the RMSRE for congestion-limited flows is already low (around 0.1). These remaining errors

are probably due to short-term load variations in the underlying path, or random packet losses that the target flow experiences, causing unavoidable variations in the resulting TCP throughput, independent of W .

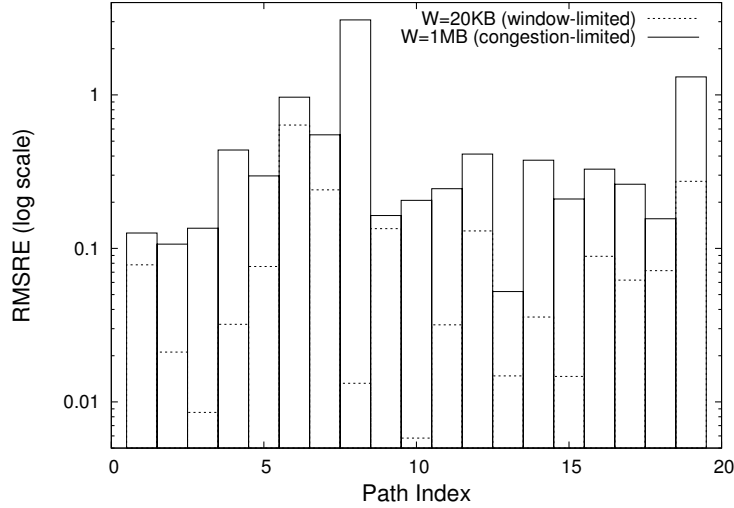


Figure 15: Prediction error for window-limited vs. congestion-limited flows.

The effect of the target flow frequency: All previous results are based on periodic TCP transfers, performed approximately every 3 minutes. We expect the prediction accuracy to depend on this “TCP transfer period”. A time series with a larger period spans a wider history horizon, and so route changes or major load variations become more likely.

To see how the measurement period affects the prediction error, we down-sample the original traces at different frequencies. We then apply the HW-LSO predictor to the down-sampled traces, producing RMSRE of predictions for transfer periods of 6, 24, and 45 minutes. Figure 16 shows the results. As we would expect, *the prediction accuracy degrades as we increase the measurement period*. Fortunately, though, the prediction errors remain reasonable even with the largest measurement period. Specifically, with the 45-min period, 65% of the traces have an RMSRE below 0.4. At the 90-th percentile of the traces the RMSRE is less than 0.4 with the 3-min period, and less than 1.0 with the 45-min period. This is an encouraging result, as it implies that *HB prediction is fairly accurate even if it relies only on sporadic previous TCP transfers, every few minutes, on the given paths*. Of course we emphasize once more that this conclusion is based only on our dataset, and it is

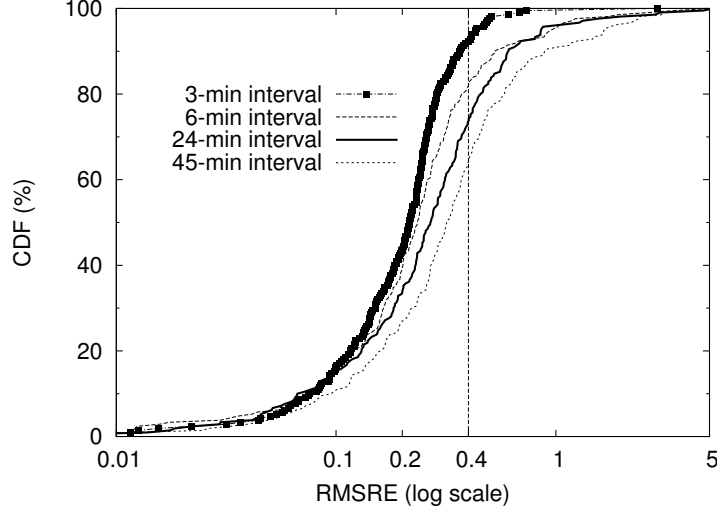


Figure 16: Prediction error with different TCP transfer periods.

possible that other Internet paths behave differently.

3.5.2 Summary

This section has evaluated the accuracy of HB prediction with respect to several factors, some of which have not been examined before. Specifically, we have shown that:

1. Even a limited history of sporadic previous TCP transfers is often sufficient to achieve a fairly good prediction accuracy.
2. Simple heuristics to detect outliers and level shifts can significantly reduce the number of large prediction errors.
3. If HB prediction is feasible, i.e., if there is a history of previous TCP transfers in the same path, then HB prediction is more accurate than FB prediction.
4. Different paths can exhibit distinct patterns of prediction accuracy. Consequently, even with the same prediction algorithm and available history, the resulting accuracy can be significantly different from path to path.
5. The predictability of an HB predictor is higher when the transfer is window-limited. Consequently, if predictability is more important than throughput maximization, then

the TCP flow should have a limited advertised window such that it does not saturate the underlying path.

3.6 Two predictability factors

The empirical results in the previous section raise the following question: *what makes TCP throughput much less predictable in some network paths than in others?* In this section, we focus on this question and identify two major factors that affect the accuracy of HB prediction in a path: load and the degree of multiplexing. Specifically, we rely on simple queueing models that provide a framework for reasoning about the relationship between TCP throughput predictability and the previous two factors.

First, we focus on the connection between the relative prediction error and the Coefficient of Variation (CoV) of a given time series. Consider a second-order stationary time series X with mean μ_X , variance σ_X^2 , and covariance $\gamma_X(k)$. According to the Yule-Walker forecasting model [106], an autoregressive one-step predictor based on the n most recent samples of X has the following prediction error variance:

$$\text{Var}[e_n] = \text{Var}[X_{n+1} - \hat{X}_{n+1}] = \sigma_X^2 - \sum_{k=1}^n a_{X,n}(k) \gamma_X(k)$$

where X_i and $\hat{X}_i$ are the actual and predicted values of X , respectively, at time i , and $\{a_{X,n}(i), i = 1, \dots, n\}$ are the autoregressive coefficients of X that minimize the mean square prediction error. The corresponding relative prediction error, in terms of the Normalized Root Mean Square Error (NRMSE)⁵ is given by:

$$\frac{\sqrt{\text{Var}[e_n]}}{\mu_X} = \frac{\sqrt{E[e_n^2]}}{\mu_X} = \sqrt{\text{CoV}_X^2 - \frac{\sum_{k=1}^n a_{X,n}(k) \gamma_X(k)}{\mu_X^2}}, \quad (8)$$

where $\text{CoV}_X = \sigma_X / \mu_X$. The key point here is that *the relative prediction error increases with the CoV of the underlying time series*. Also, if the time series has a weak correlation structure then the relative prediction error is approximately equal to the time series CoV,

$$\text{if } \gamma_X(k) \approx 0, \text{ then NRMSE} = \frac{\sqrt{E[e_n^2]}}{\mu_X} \approx \text{CoV}_X \quad (9)$$

⁵Notice that although NRMSE is not exactly the same as RMSRE, they are reasonably close as long as μ_x does not vary significantly, say spanning an order of magnitude, in a time series. This is true for most of the paths we measured.

Also recall the observation from Figure 13: the RMSRE with the HW-LSO predictor and the CoV of the corresponding time series are approximately equal. Consequently, in the following we are interested in the effects of load and degree of multiplexing on the CoV of the TCP throughput time series, rather than examining directly the effect of these factors on the RMSRE or on the NRMSE.

3.6.1 Effect of Load

Consider a link of capacity C , modeling the bottleneck of a path. We next examine the effect of that link's load conditions through two different models: first, an Independent and Identically Distributed (IID) process for the aggregate traffic at a bufferless server, and second, a Poisson process of IID session arrivals at a Processor Sharing server.

3.6.1.1 IID arrival process at bufferless server

Suppose that the *arriving* traffic rate at a given time scale T can be modeled as an IID process Y . Without loss of generality, $T=1$ time unit. Let Z be the *observed* traffic rate at the output of the link at the same time scale. For a bufferless link, the observed rate process is given by

$$Z = \begin{cases} Y & \text{if } Y < C \\ C & \text{if } Y \geq C \end{cases} \quad (10)$$

and so the probability distribution function of Z can be obtained from that of Y . The available bandwidth is given by $A=C - Z$, and its CoV is

$$\text{CoV}(A) = \frac{\sqrt{\text{Var}[Z]}}{C - E[Z]}$$

If we assume that the TCP throughput is, as a first-order approximation, equal to the available bandwidth, then the previous expression also gives the TCP throughput CoV.

We used Mathematica to derive $\text{CoV}(A)$ for two offered load processes Y : a Gaussian process and a Poisson process. The resulting $\text{CoV}(A)$, as well as the std-deviation of A , are shown in Figures 17 and 18, respectively, as a function of the link utilization $\rho=(C - A)/C$. The key observation is that *the CoV of the available bandwidth increases with the link utilization*. If the TCP throughput follows the variations of the available bandwidth,

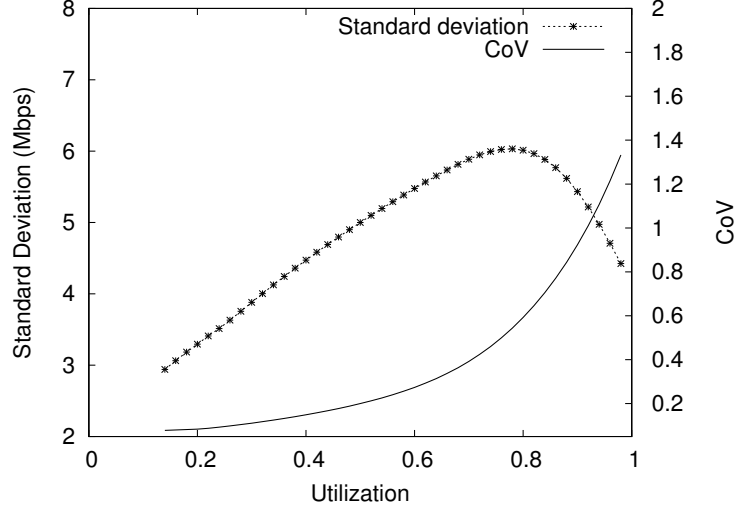


Figure 17: Gaussian process.

then based on (9) we should expect *a higher relative prediction error under heavier load conditions.*

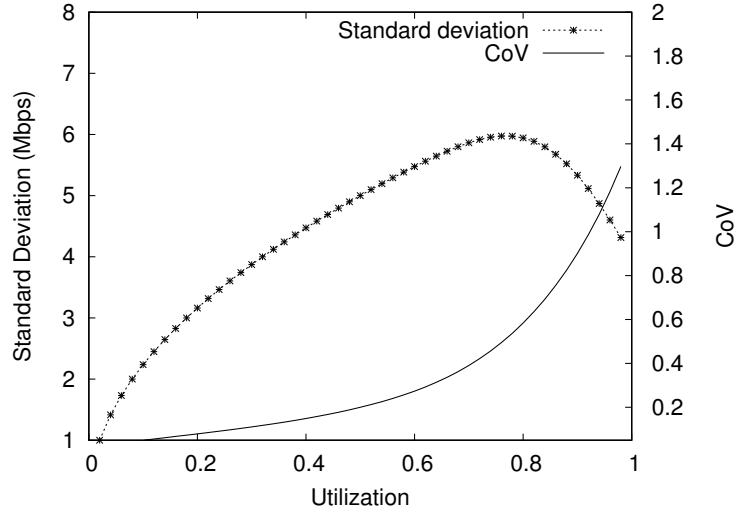


Figure 18: Poisson process.

As an interesting side-note, note that the standard deviation reaches a maximum as ρ increases, and then it drops. The reason for that drop is that, in heavy-load conditions, the link is almost always utilized and so there is little *absolute* variation in the available bandwidth. This point has been studied in more depth by Tian et al. in [135]. In relative

terms, however, the variability of A increases monotonically with ρ , as shown by the CoV curve.

3.6.1.2 Processor Sharing model with Poisson session arrivals

The previous model does not capture what happens at a congested link, in which the available bandwidth is zero. In this paragraph, we model the traffic as a stream of IID sessions arriving at a link, based on a Poisson process with average rate λ . The mean size of the sessions is θ . The normalized offered load is $\rho = (\lambda\theta)/C$. Furthermore, we model the link as a Processor Sharing server, meaning that if there are N sessions in the link then their instantaneous service rate is $r(N)=C/N$. Since the available bandwidth is zero when the link is not idle, this a more appropriate model for a congested link [52]. An arriving session, modeling the target flow, will obtain the same throughput $r(N)$ as any other active flow. So, in this model, we are not interested in the CoV of the available bandwidth, but in the CoV of the per-flow throughput $r(N)$.

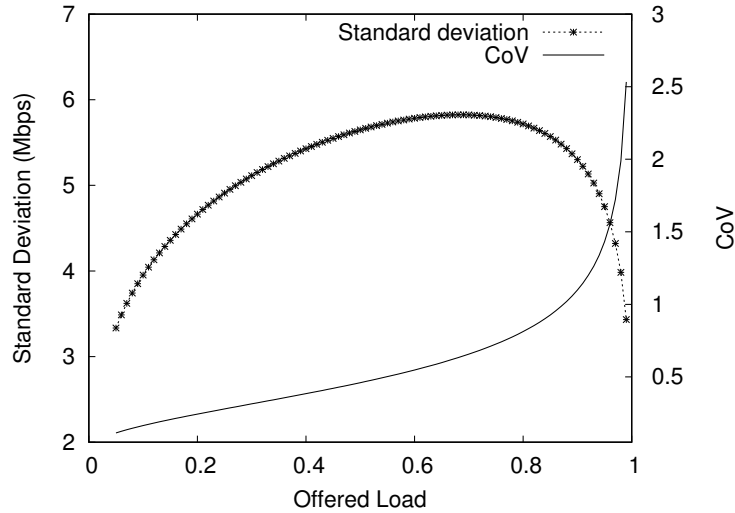


Figure 19: Processor Sharing model.

The probability distribution for the number of active flows N in the above Processor Sharing model is given by

$$\pi(N) = \rho^N(1 - \rho)$$

We again use Mathematica to derive the CoV of the target flow's throughput $r(N)$:

$$\text{CoV}[r(N)] = \frac{(1 - \rho)\log(1 - \rho)^2 + \rho \cdot L(2, \rho)}{(\rho - 1)\log(1 - \rho)^2}$$

where $L(n, x) = \sum_{k=1}^{\infty} \frac{x^k}{k^n}$. Figure 19 shows the standard deviation and CoV of $r(N)$ as a function of the offered load ρ . The main observation is the same as in the IID traffic model: *the CoV of a flow's throughput increases with the offered load ρ , implying that we should expect a higher relative prediction error under heavier load conditions.*

3.6.2 Effect of Degree of Multiplexing

The conventional wisdom is that network traffic is “smoother” in links with a higher degree of multiplexing, i.e., with a larger number of simultaneously active flows. Using a simple queuing model, we aim to better understand this insight, and the conditions under which it is valid.

Consider again a model of Poisson session arrivals. Instead of the Processor Sharing model (which leaves no available bandwidth), suppose that sessions are rate limited, and for simplicity, the rate for each session is constant and equal to r . The number of sessions N on the link follows a Poisson distribution with mean and variance $E[N] = \text{Var}[N] = (\lambda\theta)/r$ [52].

The utilized link capacity at any point in time is $Y = Nr$, with mean $E[Y] = rE[N] = \lambda\theta = \rho C$, and variance $\text{Var}[Y] = r^2\text{Var}[N]$. So, the CoV of the available bandwidth is

$$\text{CoV}[A] = \text{CoV}[C - Y] = \frac{1}{\sqrt{E[N]}} \frac{\rho C}{C(1 - \rho)} \quad (11)$$

Suppose that we keep the utilization ρ constant, but decrease the session service rate r so that the average number of sessions $E[N]$ increases. Equation (11) shows that the CoV of A decreases with the square root of $E[N]$. This confirms that *we should expect a lower relative prediction error as the number of competing flows on the link increases, but only when the utilization remains constant.*

3.6.3 Summary

This section used some simple queuing models to confirm the following insights:

- the relative prediction error increases with the CoV of the underlying time series,
- the CoV of the available bandwidth process in a non-congested link, or the CoV of a flow's throughput in a congested link, increases with the offered load on that link,
- the CoV of the available bandwidth process decreases with the number of competing flows on the link, if the utilization remains constant.

Obviously, our models are based on quite restrictive assumptions and they do not consider the many idiosyncrasies of TCP. In particular, the previous analysis assumed that the TCP throughput follows the variability of the available bandwidth at the bottleneck link of its path. This assumption is obviously not true in short time scales (less than a few RTTs), and so the previous insights may not be true for short TCP flows. We note that we have also validated the previous conclusions in the case of long TCP transfers with simulations of both TCP and non-TCP cross traffic.

3.7 Summary

This chapter investigated two classes of throughput predictors for large TCP transfers. FB prediction is an attractive option, given that it does not require intrusive measurements or any history of prior TCP transfers. We demonstrated however that it can be inaccurate, especially when the transfer attempts to saturate the path, and we explained the reasons for these errors. HB prediction, on the other hand, is quite accurate but it is feasible only when there is a history of previous TCP transfers in the same path. Although the accuracy of HB prediction does not depend so much on the actual predictor, it does depend on the transfer's maximum congestion window size and on the underlying path. We explained the path dependency based on two factors: the load and the degree of multiplexing on the bottleneck link of the path.

In future work, it would be interesting to examine hybrid predictors, which rely on TCP models as well as on recent history. Another direction would be to develop TCP throughput models that are specifically designed for prediction, and that take as inputs various estimates of the path load, buffering, and cross traffic nature. In terms of HB

prediction, more complex predictors (such as ARIMA models) can be also evaluated, even though our measurements indicate that the prediction error is already quite low, probably for any practical purposes, in most paths.

CHAPTER IV

CONGESTION CONTROL IN GNUTELLA-LIKE OVERLAY NETWORKS

4.1 *Introduction*

Gnutella represents a category of peer-to-peer systems that use TCP tunnels and scoped broadcast routing, with no end-to-end application layer reliability or flow control¹. Each servent maintains a TCP tunnel² with each neighboring peer. When a servent receives a message, it may need to forward it, either to a particular neighbor (unicast routing), or to all the neighbors except the one from which the message is received (broadcast routing). Notice that only control messages are involved in the Gnutella forwarding. File downloading uses a direct TCP connection between the file requester and the file provider.

Congestion occurs in Gnutella when a TCP tunnel is blocked for sending, and the symptom is the loss of Gnutella messages. The existence of congestion and message loss in Gnutella has been known for quite a while [13, 116] and previous work has taken measures against it [3, 98, 122]. These mechanisms, which we call *Gnutella congestion control*, try to reduce overlay layer congestion and improve the system performance under congestion. Due to the lack of a common understanding of the nature of Gnutella congestion and how it should be best handled, the (at least) dozens of Gnutella servent implementations each designs its own congestion control schemes, hoping to be compatible with other servents so as to achieve a global performance improvement.

In this chapter, we undertake a systematic study of the congestion control and message loss on Gnutella networks. Our work aims to provide a better understanding of congestion

¹Congestion control and message loss in an overlay network are dependent on the characteristics of its architecture and the application traffic. For example, it makes differences whether TCP tunnels or UDP tunnels are used, whether end-to-end application layer reliability or flow control is exercised, and whether broadcast is used extensively.

²Although UDP packets are also accepted by most implementations, they are rarely seen in practice.

control in such networks and it consists of the following parts:

1. An investigation of the congestion control mechanisms used by several popular Gnutella server implementations.
2. A measurement-based characterization of message losses on Ultrapeers³.
3. An analysis, based on large scale packet-level simulations [64], of the performance of certain Gnutella congestion control mechanisms.

The first two parts of our work reveal substantial variations among servers in their congestion control mechanisms and message loss behavior. It is also shown that most of the Gnutella congestion control mechanisms are derived from those designed for operation on network routers. The last part of our work is to understand how these *native congestion control* mechanisms perform in the Gnutella environment. While congestion control is a well-researched topic in networking, the specific overlay environment exemplified by the Gnutella system is distinguished by **1)** the lack of a strong notion of “flow”, and **2)** the significant heterogeneity in the congestion control mechanisms implemented and the need for them to coexist.

The rest of the chapter is organized as follows. In Section 2, we describe and compare the congestion control mechanisms adopted in different Gnutella server implementations. Section 3 demonstrates the measurement results of message dropping behavior on different Ultrapeer implementations. In Section 4, we present our simulation-based investigation of the performance of several congestion control mechanisms in the Gnutella network. The chapter is concluded in Section 5.

4.2 Congestion Control in Gnutella Networks

Congestion control in Gnutella networks tries to reduce overlay layer congestion, and to improve system performance when congestion does take place. Although congestion control

³In the original Gnutella architecture, peers are homogeneous in terms of their forwarding responsibilities. The new architecture distinguishes Ultrapeers and Leaf nodes to address the scalability issue. Only the Ultrapeers, which are assumed to have higher bandwidth and CPU processing power, and a longer lifetime, are responsible for routing in this new architecture.

is not specified for Gnutella, most servent implementations have incorporated certain mechanisms. In this section, we will examine how congestion is handled in the following servents: LimeWire[8]/Phex[10], Gtk-Gnutella [6], Mutella[9]/Gnucleus[56] and Qtella [11]. Notice that Phex implements its congestion control based on the *SACHRIFC* [122, 3] proposal by LimeWire, and Mutella uses the source code base from Gnucleus. Therefore, each pair of servents have similar congestion control implementations.

4.2.1 Overview of the Gnutella Operations

We summarize below the Gnutella operations for the purpose of discussions in this chapter. Details of the Gnutella protocol can be found in the specification [3].

Gnutella servents form an overlay that forwards the Gnutella protocol messages. To maintain the overlay as peers join and leave the system dynamically, *Ping* messages are generated by servents to discover new peers. Gnutella uses scoped-broadcast (controlled by TTL) for Pings. A peer that receives a Ping responds by sending a *Pong*, which is unicast routed back to the Ping originator via the same path. To search for a file, a servent generates a *Query*, which is also broadcast routed. A servent that receives a Query and has matching files responds with *QueryHit* messages, which are unicast routed back. There are other types of messages which appear less frequently and are unicast forwarded.

4.2.2 Congestion Control Mechanisms

Before we introduce the Gnutella congestion control mechanisms, we shall first describe the default Gnutella forwarding process. We use Figure 20 to facilitate the descriptions of this process and the following congestion control operations. A servent maintains a number of TCP tunnels, each of which has its own socket receive buffer and send buffer. In the default forwarding process, when a servent receives a message⁴ that needs to be forwarded, it will try to enqueue it to the back of the socket send buffer of every outgoing tunnel for the message⁵. If a socket send buffer happens to be blocked, the message will be dropped for

⁴Although TCP is a byte-stream transport protocol and does not preserve message boundaries, most current implementations use the PUSH flag to indicate the last segment of each application message, which TCP is supposed to send/deliver without buffering. Therefore, the Gnutella applications actually receives *messages*, instead of *packets*.

⁵There is only one outgoing tunnel if it is a unicast routed message.

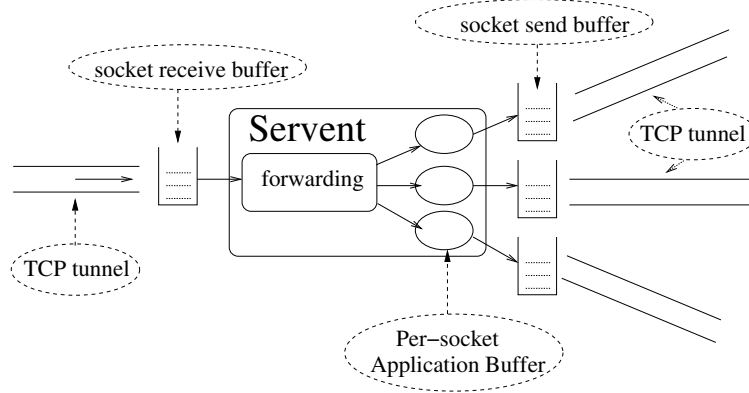


Figure 20: Gnutella servent forwarding

that output tunnel. This is what is traditionally called *drop tail queuing*, and the congestion control mechanisms described in the rest of this section are applied either additionally or as a modification to this process. Notice that the per-socket application layer output buffer shown in the figure is not part of the default forwarding process. It is, however, used by some congestion control mechanisms deployed in Gnutella, as will be described later.

Although their implementations could differ substantially from one Gnutella servent to another, congestion control approaches that are commonly adopted by Gnutella can be categorized as one of the following: *priority message queuing*, *rate limiting*, *TCP backpressure*, or *active queue management*.

Priority Message Queuing Priority message queuing [152] is a mechanism that prioritizes messages for sending, and it is often used on network routers for QoS or Differentiated Service [32] support. On congestion, lower priority messages are more likely to be dropped.

Priority queuing is used in Gnutella to improve the overall Gnutella system performance. To implement this scheme, an application layer output buffer (referred to as output buffer/queue in the rest of the chapter), which may be divided and allocated to different priority levels, is created for each tunnel, as shown in Figure 20. When a message is to be forwarded to a tunnel, it is first entered into the output buffer (or dropped) based on its relative priority. Messages are prioritized based on some Gnutella specific semantics, which are often considered in the following order:

- **Message Type.** Different types of Gnutella messages have different contributions to system performance. QueryHit throughput is usually considered the most important performance metric. The loss of a QueryHit directly contributes to throughput degradation. A lost Query compromises the chance of getting QueryHits from the group of peers that are reachable by the Query. Ping and Pong messages contribute indirectly to system throughput by establishing connections or improving network connectivity. Those other types of messages that appear much less frequently are often considered of the highest priority.
- **Hops and TTL.** A Gnutella message has a servent implementation dependent TTL and a hop count (Hops) of 0 when it is initialized. TTL is decreased by one and Hops is increased by one by each servent that forwards the message. When it comes to priority, Hops is considered a measure of the number of potential receivers yet to be reached, the potential value of the message to the receiver, or the network resource already used to deliver the message, depending on the message type. Some servents consider TTL, in addition to Hops, for Pong and QueryHit messages. For Pongs and QueryHits, TTL is a measure of the proximity to the end receiver, therefore, smaller TTLs are favored.
- **Age.** Newer messages are favored over older ones by some priority queues to help minimize delay [122].

Rate Limiting and TCP Backpressure To prevent the onset of congestion, receiving and/or sending rate limits can be enforced, either on a per TCP connection basis, or on a per servent basis.

Incoming rate limiting in Gnutella is often enforced by suspending the read operations on a TCP socket when data received during a time slot exceeds a predefined or user configured limit. This technique, when used alone, merely slows down the sender and shifts the message dropping to the sender. Therefore, it is often complemented by another technique dubbed the *hops-flow* control on the sender. When a servent is rate limiting the receiving on a socket, it sends a control message to the sender to instruct it to stop sending Query

messages exceeding a certain hop count limit, which is often 0.

Another technique that throttles the sender is TCP backpressure [104]. Although both are implemented by stopping the consumption of incoming data, the condition that triggers the throttle in TCP backpressure is the blocking on the output socket on which the incoming data is supposed to be sent. For broadcast routed messages, TCP backpressure takes place when a certain fraction of output sockets are blocked. With the TCP advertised receiver window flow control scheme, TCP backpressure creates a chain event which pushes the data back to the servents that have buffers to accommodate them. When the congestion is transient, TCP backpressure avoids message losses and slowing down many sources, and can achieve a higher system throughput.

There are also Gnutella implementations that balance the bandwidth usage among concurrent tunnels and/or between HTTP traffic (for downloading/uploading) and the Gnutella protocol traffic. LimeWire is an example of sophisticated scheduling of bandwidth usage.

Active Queue Management Active Queue Management (AQM) [81] is a scheme that proactively controls the queue occupancy to prevent the occurrences of congestion. In Gtk-Gnutella, when the prioritized output queue associated with a socket has an occupancy beyond a predefined high watermark, the socket enters a congestion control mode, in which the servent will avoid sending any message on the socket. To prevent the explosion of QueryHit messages flowing onto the connection, Query messages from a congestion-controlled socket will be dropped and the corresponding peer will be notified to stop sending queries. A socket leaves the congestion control mode when the queue occupancy drops below a predefined low watermark.

4.2.3 Message Type Specific Limit

It has been observed that the scalability of Gnutella is handicapped by the overabundance of messages flowing through the Gnutella network including Pings and Queries. It is often the case that Queries are sent to a large number of hosts in an attempt to secure a hit from a few of them [13]. Some servents impose limits on Pings and Queries when they experience or anticipate congestion. For example, Gtk-Gnutella servents use Ping throttle to accept

(and forward) at most one Ping every three seconds from a particular Gnutella peer.

When there are already a large number of Pong/QueryHit messages responding to a particular Ping/Query message, more Pongs/QueryHits are often considered to have lesser values. Most Gnutella servents impose certain limits on the number of Pong/QueryHit messages returned/forwarded for the same Ping/QueryHit. In the case of QueryHit, this approach could improve the average success ratio of Queries (percentage of Queries that receive at least one QueryHit), due to the long-tail distribution of file popularities [124].

4.2.4 Comparison of Different Gnutella Servents

Table 4 compares the Gnutella servent implementations with regard to the major congestion control features⁶ they adopt. While the table does not show the detailed differences among the implementations of each feature, it is sufficient to demonstrate the heterogeneity among servents.

Table 4: Servent Flow Control Features

Features	LimeWire/Phex	Mutella/Gnucleus	Gtk-Gnutella	Qtella
Priority Message Queue	Yes	Yes	Yes	No
TCP Backpressure	No/Yes*	No	No	No
Rate Limit	Yes	No**	Yes	No
AQM	No	No	Yes	No

*: Phex implements TCP backpressure, but LimeWire does not.

**: The latest source code of Mutella has incorporated incoming rate limiting, but it is not fully functioning.

Priority queuing schemes in Gnutella servents differ in their prioritization policies, as well as implementations. Table 5 lists some of the distinguishing characteristics among the servents⁷. We highlight below some differences shown in the table:

- While most servents give priority to Pong messages over Query messages, LimeWire does the opposite.
- In Gtk-Gnutella, low hops are favored for Ping and Query, while high hops are favored

⁶Since the message type specific limits used by different servents have many detailed differences, they are not individually listed for comparison.

⁷Notice that Qtella has been excluded from the table since it does not have priority queuing.

for Pong and QueryHit. TTL is compared for QueryHit messages that have the same TTL.

In LimeWire, QueryHits are prioritized based on the number of replies already seen for the corresponding Query. For other messages, hop count is used to determine the relative priority of messages of the same type, with the same policy as that of Gtk-Gnutella. TTL is not used for prioritization.

Mutella does not prioritize messages based on Hops or TTL.

- Gtk-Gnutella and Mutella use a form of preemptive scheduling where higher priority messages are always scheduled before lower priority messages. LimeWire uses weighted round-robin scheduling to prevent the starvation of lower priority message types. In each scheduling period, time slots of different lengths are allocated to different types of message.

Table 5: Priority Queueing Mechanisms

Characteristics	LimeWire/Phex	Mutella/Gnucleus	Gtk-Gnutella
Pong favored over Query	No	Yes	Yes
Prioritized by TTL and/or Hops	Yes	No	Yes
Newer messages favored	Yes	Yes	No
QueryHits prioritized by popularity	Yes	No	No
Output buffer scheduling	Weighted RR*	Highest**	Highest**

*: Weighted Round-Robin

**: Highest priority first

4.3 Congestion Control Behavior

In this section, we present our measurement and analysis of the message dropping (loss) behavior of several servent implementations. Our experiments are intended to show, for each servent implementation, the message drop rate and pattern, and what kind of messages are more likely to be dropped. We also relate these phenomena to their congestion control strategies described in Section 4.2.

Measurement Methodology Since the transition to the new Ultrapeer-Leaf architecture shifts most of the forwarding task to Ultrapeers, our measurement is performed on Ultrapeers of three servent implementations: LimeWire, Gtk-Gnutella and Mutella⁸. We instrument the source code of each servent to log the messages dropped. For each message dropped, we log the following information whenever available: time, type of message, Hops, TTL, peer address and vendor information. We also keep track of the total number of messages received during a measurement. Unless otherwise specified, the results presented in this section are obtained in the experiment described below.

We run the three Ultrapeers on three machines that have similar, if not exactly the same, connectivity to the Internet from the same department at Georgia Tech. We use similar configurations for all the three, so as to provide a similar exposure to the Gnutella network traffic. The Ultrapeers are all configured to have 4-8 connections to other Ultrapeers and a maximum of 15 leaf nodes. Our monitoring shows that they have 20-23 connections most of the time. We do not set customized bandwidth limits even if they exist as an option. The experiment was started around 3PM EST on Friday, May 30 and lasted 24 hours. We started to log after the servents had relatively stable initial connections.

Although we cannot verify if the data collected in this way represent the average message dropping behavior, we believe it is fair to say that they are typical of many servents on the network. This is based on the observation that many users do not bother to change the default settings, and the configuration we choose for the servents is very close to the default configurations recommended by different implementations for our Ultrapeers' Internet connectivity.

4.3.1 Message Drop Patterns

The results presented in this subsection show the evolution of message drop rate (number of drops per second), which we call message drop pattern, of the three servents.

⁸Qtella does not support Ultrapeer.

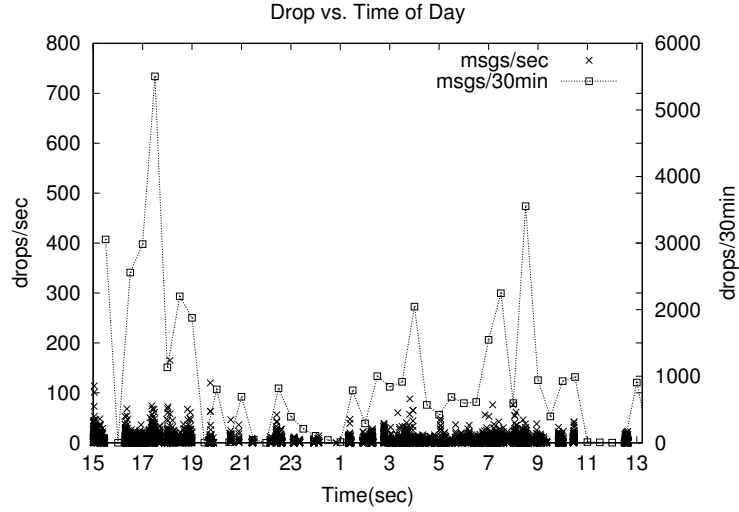


Figure 21: Drop pattern (Gtk-Gnutella)

The 24-hour View Figures 21, 22 and 23 demonstrate the message drop patterns of the three Ultrapeers over a 24-hour time period. Notice that the average rate is so different that we have to use different Y-axes (left and right) ranges for the three servants.

Assuming that most of the traffic seen by our servants originated from U.S. users, the figure for Gtk-Gnutella seems to exhibit a time-of-day pattern that is closest to the pattern of user activity during a day [127], with peaks around late afternoon and evening. This is explained by the fact that most of the messages dropped by Gtk-Gnutella are Queries (see Figure 28), which are user-initiated. Although this is also true for Mutella, the losses on a Mutella Ultrapeer can be significantly affected by the bandwidth quality of its peers during a time period, since it does not drop bad peers as proactively as LimeWire or Gtk-Gnutella, neither does it enforce incoming rate limiting, as we will explain shortly. The larger number of drops around midnight and early morning in the LimeWire figure could be caused by a large number of disconnections resulting from users going offline. Changes of peer connections result in a large number of Pings and Pongs, causing more drops by LimeWire. Figure 24 compares the CDFs of message drop rate for the three servants. This figure, as well as the previous three figures, shows that with the same degree of connectivity to the rest of the Gnutella network, Mutella drops much more messages than the other two. Additional information we gather from the logs suggests that the difference may be mainly

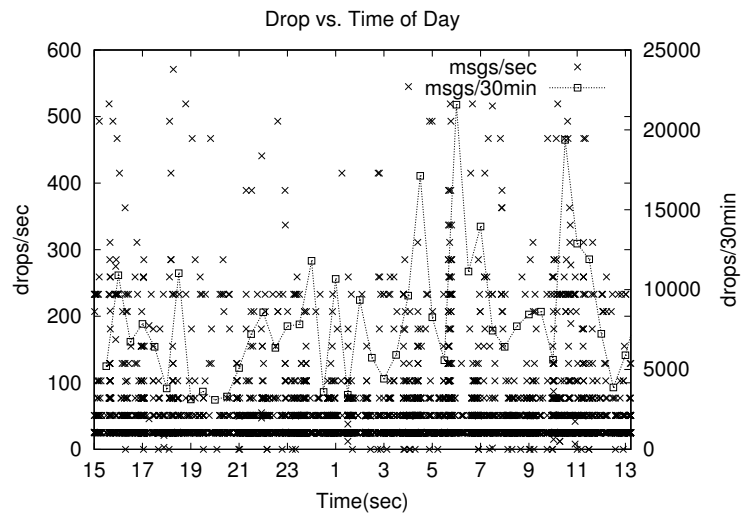


Figure 22: Drop pattern (Mutella)

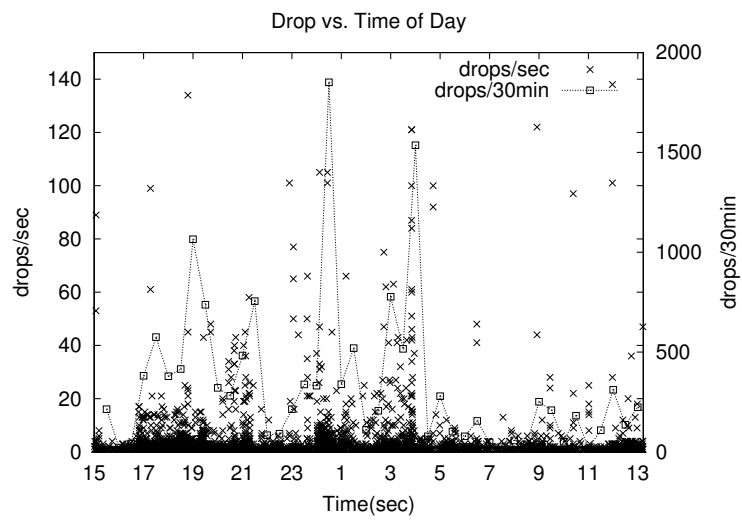


Figure 23: Drop pattern (LimeWire)

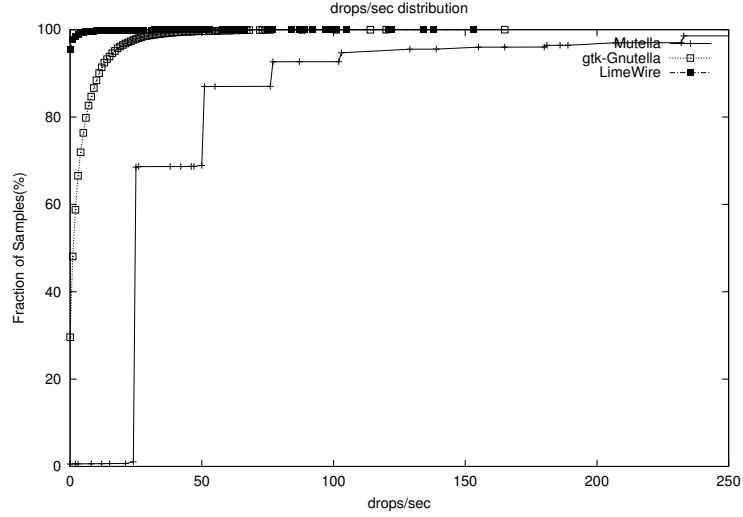


Figure 24: CDF of drops/sec

attributed to the fact that Mutella is the only one that does not impose an incoming rate limit:

- Mutella also receives more messages, although its average *drop ratio*, i.e., the number of messages dropped versus the number of messages received, is still much higher than the other two. The average drop ratios are 7%, 0.4% and 0.5% for Mutella, Gtk-Gnutella and LimeWire, respectively.
- The Ultrapeers connected to the Mutella Ultrapeer are more heterogeneous than those connected to the other two Ultrapeers, and there are less of them we know (based on the vendor information) that have implemented rate limiting. We believe that this is an artifact of the different bootstrapping strategies they use and the different popularities of these servants [76].

The Expanded View Figures 25, 26 and 27 are the expanded views of part of the 24-hour traces between 3PM and 5PM. The drop patterns shown by these figures highlight the following facts:

- Gtk-Gnutella servant has distinct congestion-controlled periods and non-congestion-controlled periods, due to its Active Queue Management scheme.

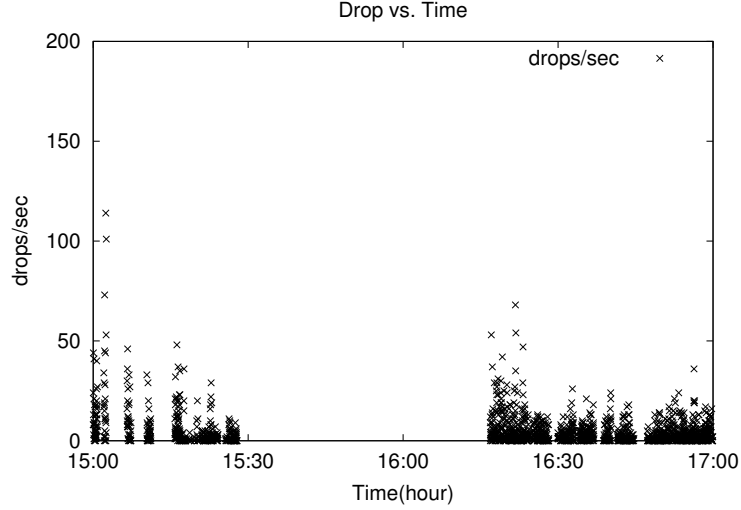


Figure 25: Expanded view (Gtk-Gnutella)

- In Mutella, the number of messages dropped per second seems to have a step difference of about 25 drops. Mutella checks the length of an priority message queue only on every socket output event, and it will cut the queue length to 25 if it is longer than 50. Therefore, at least 25 messages are dropped together at a time.
- Message drop rate in LimeWire is very low most of the time, with some occasional peaks. Drops are evenly spread over time.

4.3.2 Distribution

In this subsection, we will show the distribution of the messages dropped by different servants in terms of message type, hops and TTL. The results expose noticeable differences among the three servants, corresponding to the differences in their prioritization policies.

Figure 28 shows the message type distribution of the messages dropped. The results have already been cited by the previous subsections, but the causes may merit a further explanation. In the LimeWire priority queues, the four lowest priority message types are: Ping, Pong, Query and QueryHit, in order from the lowest to the highest. In Mutella and Gtk-Gnutella, they are Ping, Query, Pong and QueryHit. Therefore, most of the drops are

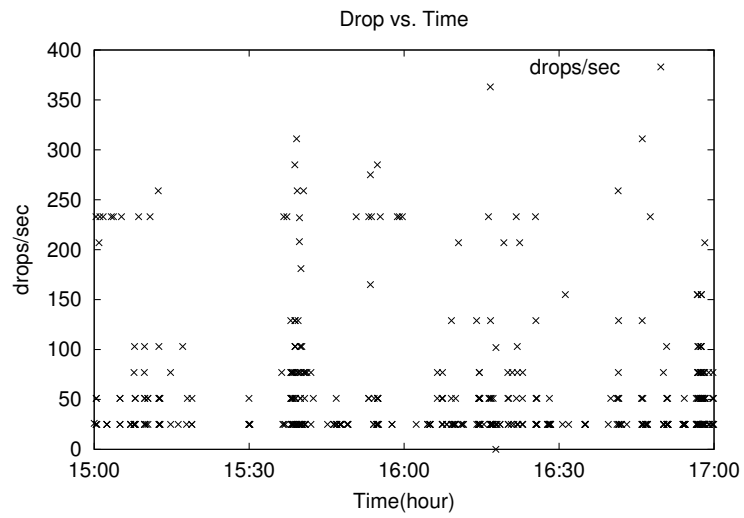


Figure 26: Expanded view (Mutella)

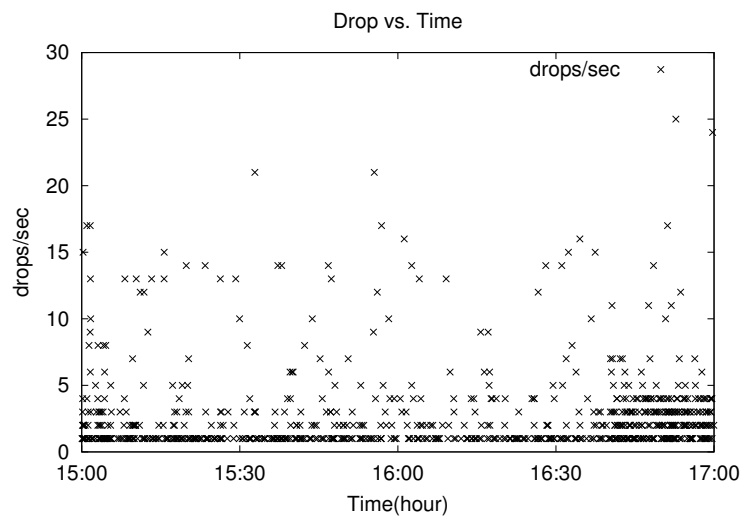


Figure 27: Expanded view (LimeWire)

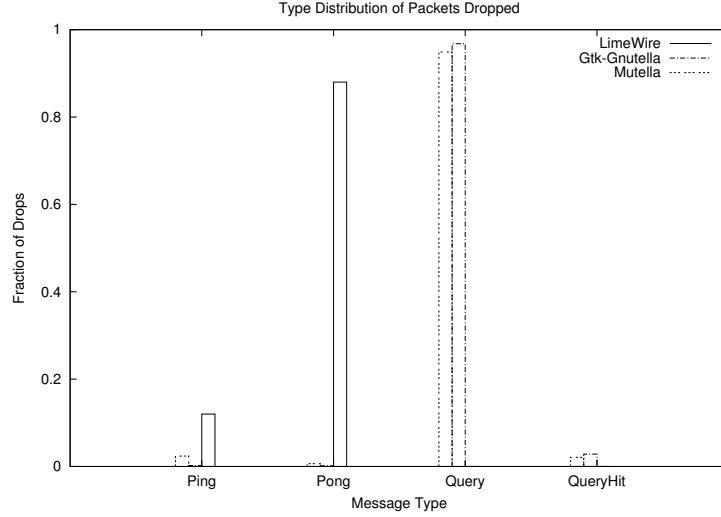


Figure 28: Type distribution of drops

Query messages in Mutella and Gtk-Gnutella⁹, whereas Pong and Ping messages dominate those dropped by LimeWire.

Table 6 shows the hop count distribution of the drops. Since most of the messages dropped by LimeWire are Pongs, for which high hops are preferred, almost all the messages dropped have a hop count below 3, with a majority of them having a hop count of 1. Contrary to LimeWire, Gtk-Gnutella mostly drops Queries, for which low hops are preferred. Therefore, most drops have a hop count of 6 and the rest mostly have a hop count of 5. Mutella does not prioritize based on Hops, therefore, it has a more even distribution across different hop counts. The slight unbalance among different hop count values could be attributed to the hop count distribution of messages received, as our logs reveal. For all the servants, hardly any drops (or messages to be forwarded) have a hop count of 7 or higher, presumably because the Gnutella default maximum TTL is 7.

Table 7 shows the TTL distribution of the drops. Assuming a fixed initial TTL of 7, the distributions for Mutella and Gtk-Gnutella drops correspond well to their hop count distributions ($TTL + hop\ count = initial\ TTL$). LimeWire, however, does not have that correspondence. In LimeWire, the TTL for a Pong is initialized to be the hop count of the

⁹There are much less Pings than Queries and Pongs in the Gnutella network.

Table 6: Hops Distribution of Drops (%)

Hops	LimeWire		Mutella	Gtk-Gnutella
<=3	0	4	20	2
	1	59		
	2	12		
	3	25		
4	0		23	2
5	0		20	12.3
6	0		36	83.5
>=7	0		0	0.2

corresponding Ping incremented by 1. Considering that most of the Pongs dropped has an initial TTL within the range 3-5, the TTL distribution of LimeWire corresponds well to its hop count distribution too.

4.3.3 Impact of Access Bandwidth and Peer Connectivity

The message loss on an Ultrapeer is also affected by its access bandwidth and connectivity. For example, a cable modem servent is more likely to experience congestion than a campus network servent when it tries to forward a message to a certain number of peers, and it is also more likely to experience congestion if it is connected to more peers. We perform two additional experiments on LimeWire Ultrapeers and compare the results to the previous measurement on LimeWire Ultrapeer. One of them is performed on an Ultrapeer that has a cable modem connection to the Internet, and the other one is performed on an Ultrapeer that is on the same Georgia Tech campus as before, but uses a different peer connectivity setting (a maximum of 32 Ultrapeer connections and 30 leaf connections). Figure 29 shows that the message drop rate increases dramatically with the number of neighboring peers, or with a lower access bandwidth (notice that the Y-axis uses a log scale).

4.4 *Understanding the Congestion Control Mechanisms in Gnutella Networks*

The congestion control mechanisms used in the current Gnutella network are mostly derived from existing proposals designed to operate on native network routers. However, the

Table 7: TTL Distribution of Drops (%)

TTL	LimeWire	Mutella	Gtk-Gnutella
1	36	38	86
2	27	21	12
3	18	22	1.4
4	15	10	0.3
≥ 5	4	9	0.3

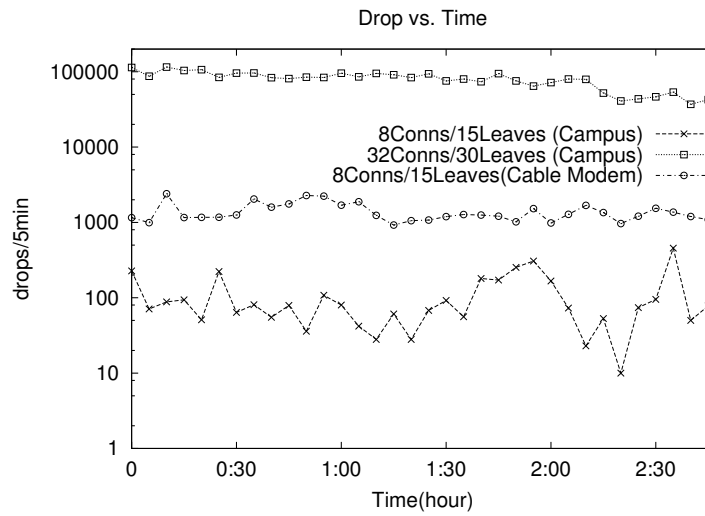


Figure 29: Impact of connectivity

Gnutella network is a very different environment from the Internet and other traditional networks, in the sense that:

- there is no strong notion of “flow” in Gnutella. The congestion control functions are enforced either on a per-message basis as with priority queuing, or on a per tunnel basis as with rate limiting, whereas the native congestion control mechanisms are usually exercised with certain notions of “flow”.
- heterogeneous congestion control mechanisms exist in Gnutella and we cannot control their organization in the connected overlay. Native congestion control mechanisms, however, usually assume that the same or compatible mechanisms are deployed on other routers.

Given these unique characteristics, the performance of the Gnutella congestion control mechanisms are yet to be studied. The measurement results of message losses on Ultrapeers, as presented in the previous section, are not sufficient for that purpose. A system-wide measurement study is not a feasible solution for several reasons: *i)* the scale of the Gnutella network makes it a prohibitive task; *ii)* it is impossible to have a real Gnutella network that consists solely of servents using a same congestion control mechanism; *iii)* it is hard to decouple the system performance impact of different aspects, e.g., congestion control and bootstrapping, of a servent implementation.

In this section, we use packet-level simulations as the main tool to study the following congestion control schemes individually: *i)* priority queuing, *ii)* rate limiting and *iii)* TCP backpressure. There is one case where we use experiments with real Gnutella implementations. Our focus in this section is to understand: *i)* whether and how the performance of each is affected by the Gnutella overlay characteristics and workload, and, *ii)* the impact of heterogeneity on the performance of each.

The main findings from this part of our work can be summarized as follows:

- *priority queuing* could improve system performance substantially, but the design of an optimal priority policy is particularly challenging because of the dynamic workload composition and multiple dimensions of performance requirements;

- *TCP backpressure* can be very effective in reducing congestion while improving system throughput, but its performance is sensitive to the traffic load;
- *rate limiting* reduces congestion at the cost of slowing down sources to a degree that it may not maintain the same throughput as without it;
- heterogeneity is a concern for almost all the mechanisms, since it precludes the full realization of their advantages.

Simulation Overview We build the three congestion control mechanisms into an *ns*-based packet-level Gnutella simulator we had in our previous work [64] to analyze the system performance. Each mechanism can be individually enabled or disabled in an experiment so that it can be studied in isolation. Our experimental setups are based on existing models. Unless otherwise specified, the following models and parameters are used.

We use a 100-node transit-stub network topology generated with GT-ITM [36] as the backbone. We attach a random number (between 10 and 20) of leaf nodes to each stub node to get a network of 1642 nodes. A total of 816 peers are attached to some of those leaf nodes. The access bandwidth of a peer, which is represented by the bandwidth of the link between the leaf node to which it is attached and the stub node it is linked to, follows a distribution based on the statistics presented in [125].

The peer behavioral model is based on paper [55] with the parameters shown in Table 8. Peers switch between the online state and the offline state. While online, they could be idle when they do not search for files. Columns “avg offline” and “avg idle” denote the average lengths for a peer to stay offline and stay idle, respectively. The column “prob. going offline” is the probability for a peer to go offline after a successful query, and “freeloader” indicates whether a peer uses the system without sharing any of its own resources. There are two classes of peers, “freeloader” and “non-freeloader”.

We use two different models of file popularity distribution. When an average hit probability (the probability that a peer receiving a Query request will return a QueryHit) is specified for an experiment, we use a uniform file popularity distribution. Otherwise, a Pareto distribution [55] is assumed, with the parameter α set to 2. Notice that the hit

Table 8: Two Classes of Peers

	avg offline(sec)	avg idle(sec)	prob. going offline	freeloader?
class 0	4320	300	0.2	0
class 1	4320	30	0.1	1

probability of a Query is assumed to be proportional to its popularity as in [55]. We use 100 popular files in the system.

4.4.1 Priority Queuing

The priority of a message in a queuing system is decided based on its relative weight in the current queue. When used for native network congestion control, a priority policy can often be predefined among competing flows based on certain fairness criteria, and can be enforced strictly by monitoring the local resource usage by each flow.

However, there is no strong notion of “flow” in Gnutella and the priority of a message is dependent on its semantics. A priority policy is defined based on a perception of how different messages contribute to the overall system performance. This perception, however, is often complicated by the fact that the relative weight of a message may vary with the workload composition, and that when multiple performance metrics are to be considered, the relative weight of a message may be conflicting for different metrics. The difficulty of having a definitive priority policy is evident from the heterogeneous policies among server implementations.

The experiments in this subsection are to demonstrate the following: **1)** priority queuing can improve performance substantially when a single performance metric is considered; **2)** when multiple performance metrics are considered and the system workload composition varies dynamically, a static priority policy may not always perform the best; **3)** setting a proper buffer size on Gnutella server has quite different considerations from those for network routers; **4)** heterogeneous priority policy can potentially hurt system performance.

Performance with A Single Performance Metric We use QueryHit throughput as the sole performance metric in this experiment, and show that performance improvement by priority queuing increases with the system scale. Gnutella messages are prioritized based on

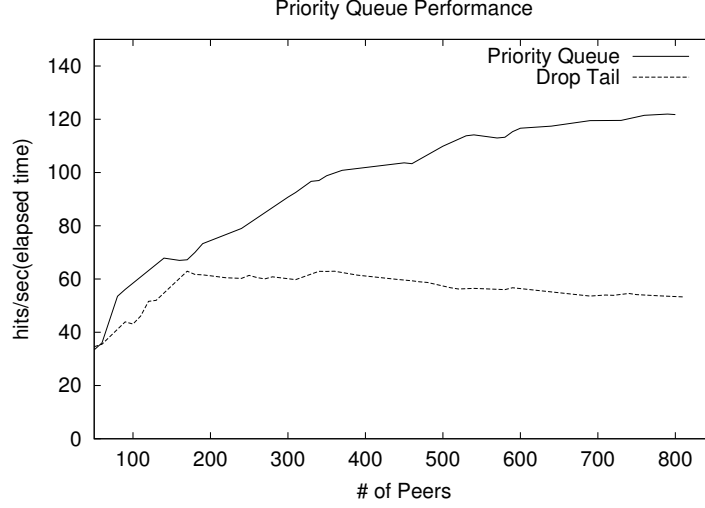


Figure 30: Performance of priority queuing

their types and hops. Message types are prioritized as: Ping, Pong, Query and QueryHit, from the lowest to the highest. High hops are favored over low hops for QueryHit and Pong, and the opposite for Query and Ping. Each TCP tunnel has an output queue of size 100, shared by all the message types. Peers join the Gnutella system following a synthetic trace. We measure the periodic system throughput as more servants join the system.

Figure 30 shows that with an increasing number of active peers, the performance improvement over drop tail queuing is more visible. It also shows that when drop tail queuing is used in this specific system, the scale of the system has a negative impact on the throughput, evident from the downward trend of throughput after the system size exceeds 180.

Static Policy vs. Dynamic Workload Composition When multiple performance metrics have to be considered, a static priority policy may not always perform the best for different traffic compositions.

For example, consider a system where QueryHit throughput and average response time (the time it takes the first QueryHit to arrive after a Query is sent) are both considered in a utility function as follows:

$$\frac{\text{throughput}}{\text{average hits per query}} + \frac{10}{\text{average response time}} \quad (12)$$

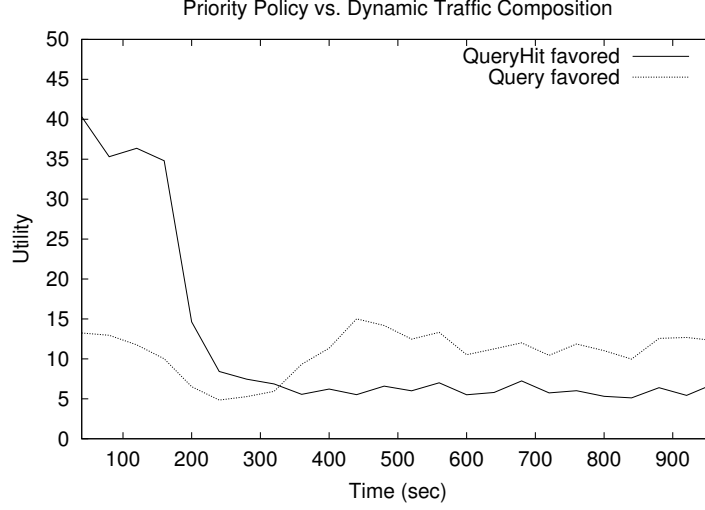


Figure 31: Performance with utility function

The utility function adds a dynamic weight ($1/\text{average hits per query}$) to the QueryHit throughput which essentially discounts the value of a QueryHit when the average number of hits per query increases. With the policy that favors QueryHit over Query, as the average number of hits for a query increases, the response time may increase as a result of Queries being queued longer due to a larger number of higher priority QueryHit messages, and the value of a QueryHit decreases as a result of the increased hits per query. Eventually, the increased throughput may be offset by the increased average hits per query and average response time.

In the experiment shown in Figure 31, we increase the average hit probability from 0.2 to 0.8 at time $t = 150\text{sec}$ and we compare the utilities of two priority policies, one favoring QueryHit and another favoring Query. We can see that after the average hit probability jumps up to 0.8, the policy that favors Query gradually outperforms the policy that favors QueryHit.

Buffer Size Setting a proper router buffer size has always been an intriguing problem. Guidelines for network routers are generally based on some flow-related system characteristics. For example, one principle is to use a size that is at least twice the longest end-to-end TCP flow round-trip time multiplied by the bottleneck link bandwidth [141]. Another rule

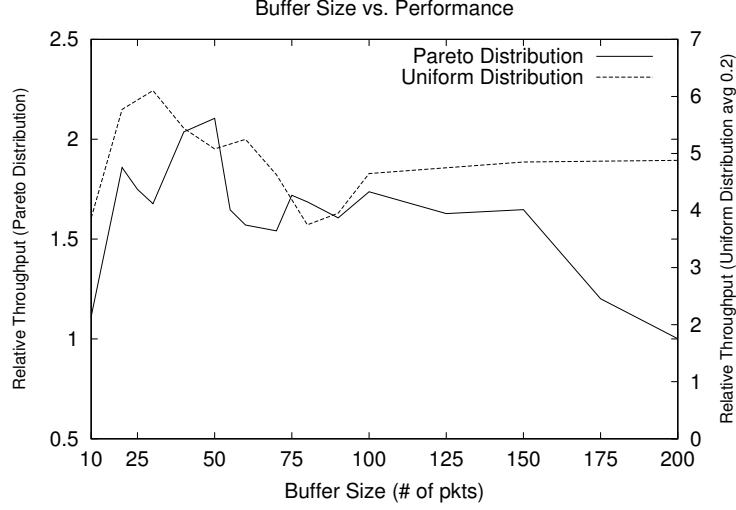


Figure 32: Performance vs. buffer size

is to preserve at least 1 slot in the queue for each TCP flow [93]. Clearly, these techniques cannot be applied to the Gnutella network.

Gnutella network has its own characteristics that influence the setting of a proper buffer size. Contrasting to the high-speed links associated with typical Internet routers, the bandwidth of an overlay link (TCP tunnel) is the throughput of the tunnel, which is often very low compared to the capacity of physical links. Therefore, queuing delay will be much more pronounced, especially for the lower priority messages, which suggests that a relatively small buffer size is more appropriate. In addition, the degree of multiplexing is much lower on the overlay, and incoming traffic on individual overlay links is already smoothed out by the TCP congestion window and receiver window. Therefore, a small buffer may be enough to accommodate burstiness. Most of the Gnutella servents use small buffer sizes ranging from 25 to 100 messages. Our experiments suggest that it is a reasonable choice to have a small buffer size.

We perform experiments under a varied buffer size with two different file popularity distributions: one that uses the Pareto distribution and another that uses a uniform distribution with an average hit rate of 0.2. Figure 32 shows that the highest throughput is obtained when the buffer sizes are 50 and 30 for the two settings, respectively. QueryHit throughput can be significantly affected by the delivery rate of Queries. As the buffer size

increases, Queries are buffered longer, while the number of Queries delivered may not increase a lot (QueryHits occupy most of the buffers). In the case of Pareto distribution, when the buffer size increases to 200, the impact of longer delay eventually dominates the effect of increased deliveries of Queries, and the system throughput is reduced to virtually the same as that of drop tail queuing.

Heterogeneity Consider two neighboring servents that have different priority policies. It is conceivable that this heterogeneity may cause problems if the second servent drops those messages that are favored by the first router, and the messages it favors are dropped by the first router and rarely even reach it. The system may end up without any performance improvement intended by either of the two policies. Unlike the other experiments in this section, the following experiments are performed on real Gnutella servents. We intend to show that the impact of heterogeneous priority policy is visible in the real Gnutella network.

We perform two experiments in a controlled environment with the setup shown in Figure 33. Ultrapeers $L1$ and C are run on the same campus network and connected to each other via Fast Ethernet. $L1$ runs LimeWire and is connected to the external Gnutella network. C runs Gtk-Gnutella and LimeWire, respectively, in the two experiments. Another LimeWire Ultrapeer $L2$ is connected *only* to C . The TCP connection between C and $L2$ has a relatively low throughput, which we obtain with a cable modem connection and some constant bit rate UDP cross traffic. For each experiment, multiple tests are performed when C is connected to a different number of LimeWire Ultrapeers like $L1$. In each test, we keep track of the numbers of Query messages received by C and by $L2$, respectively.

Table 9 compares the Query messages received by $L2$ in the two experiments, shown in the second and third columns respectively, as a percentage of those received by C from the Gnutella network through $L1$. We can see that the Gtk-Gnutella servent drops Query messages that would have been forwarded to the stub LimeWire ($L2$) servent by the LimeWire C , and that the difference is more pronounced when the congestion on C is heavier (more Ultrapeer connections). Notice that LimeWire does not forward all Query messages either, but Queries are dropped mostly for reasons other than congestion control, such as Duplicate

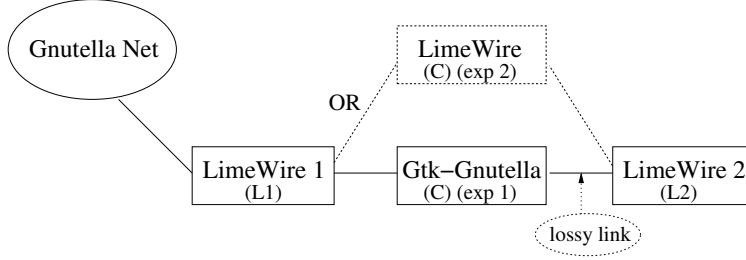


Figure 33: Heterogeneity test setup

Table 9: Heterogeneous Priority Policy

Connectivity of C	received by $L2(1)$	received by $L2(2)$
2	46%	86%
3	21%	77%
4	7%	64%

or Invalid Queries.

The experiments above show the potential problem when two subsets of LimeWire servers are connected through a Gtk-Gnutella Ultrapeer. Query messages, which are considered high priority in LimeWire may be dropped by the Gtk-Gnutella Ultrapeer. However, it should be noted that Gtk-Gnutella does not necessarily perform worse than LimeWire just because it drops more Queries. Gtk-Gnutella favors Pong over Query in the priority queue, but it controls the volume of Pong traffic through Ping throttle. LimeWire, on the other hand, favors Query over Pong, but it has a more stringent rate limit on the generation of Queries.

4.4.2 TCP Backpressure

TCP backpressure, when implemented on backbone and edge routers on the Internet, pushes the backbone congestion back to the lighter loaded edges. The reduction (or elimination) of message losses avoids slowing down TCP sources and therefore improves end-to-end throughput. The performance improvement is contingent on the transience of congestion, since otherwise the queue on the edge router will build up and eventually the source will be slowed down. Notice that TCP backpressure is implemented on a per-flow basis on those

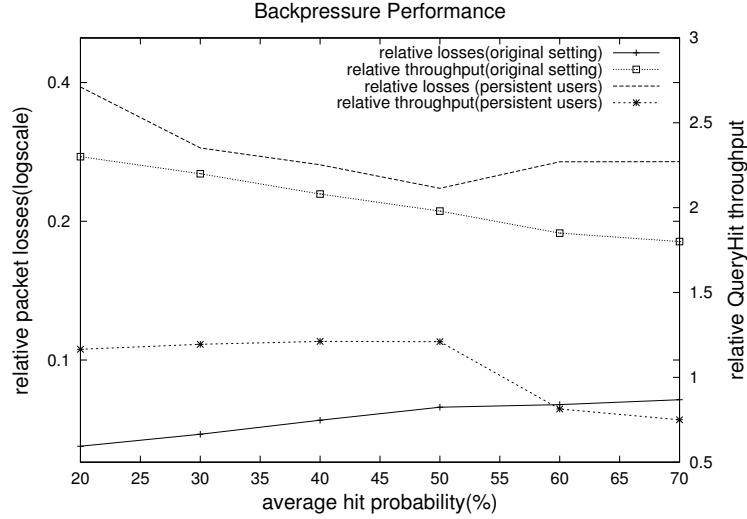


Figure 34: Throughput with TCP backpressure

routers.

On a TCP backpressure enabled Gnutella servent, when a certain fraction of the next hop tunnel(s) for an incoming message are blocked, the servent stops consuming data from the incoming TCP socket. When this functionality is implemented by all the servents, the congestion will be pushed back to those servents that have available buffers.

However, it is less clear how TCP backpressure affects the Gnutella throughput and average response time. In the Gnutella network, the chain of TCP backpressure events builds up a tree of servents that are slowed down, some of which may not need to be because they do not forward traffic that will pass through the congested servent. The larger the average load in the system, the larger the tree will be, and the more likely there are servents being unnecessarily slowed down. In this section, we will look at different aspects of the performance of TCP backpressure, as well as the performance implication of having heterogeneous TCP backpressure capabilities among servents.

Congestion Reduction and Throughput Improvement Figure 34 shows the relative performance of TCP backpressure as compared to drop tail queuing when the traffic load varies with the average hit probability. In these experiments, we use a threshold of 50% to trigger backpressure, i.e., when more than 50% of the output tunnels to forward an

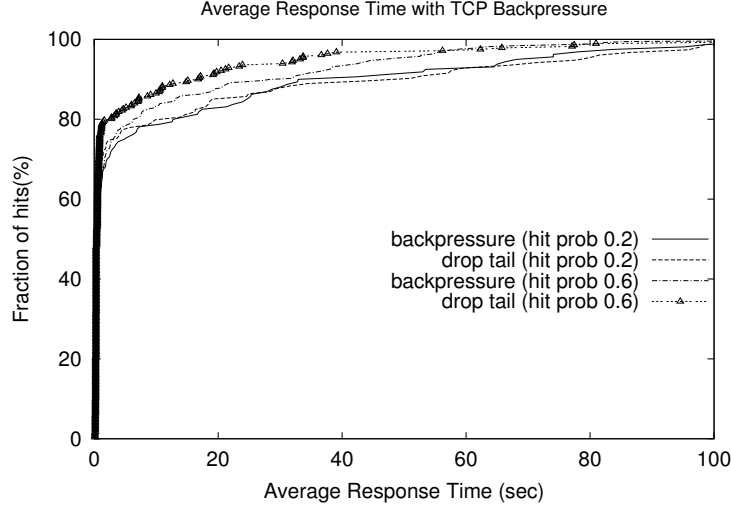


Figure 35: Response time with TCP backpressure

incoming message onto are blocked, the incoming socket will be blocked for reading. In the first experiment, we use the peer behavior model described originally. We can see that although the performance improvement becomes smaller when the system load increases with the average hit probability, it is consistently substantial with this setup. Losses are reduced to 6%-9% as compared to the drop tail queuing system, and QueryHit throughput is improved by 80%-130%.

In the second experiment, we change the peer behavior model such that both classes of peers are persistent, i.e., they do not go offline. With this change, the success of a previous query is more likely to introduce a new query. The feedback process greatly increases system load and the probability of sources being slowed down unnecessarily by TCP backpressure. Although TCP backpressure still reduces message losses to 20%-40% in the second experiment, it does not improve throughput significantly. As the average hit probability increases to 60%, it is actually outperformed by drop tail queuing.

To summarize the two experiments, we highlight that TCP backpressure is fairly effective in reducing congestion and improving throughput under light load. It may not improve throughput under heavy load and it is sensitive to the positive feedback in user behavior on the success of the previous queries.

Response Time For most native network congestion control mechanisms, throughput at the flow level is much more a focus than the packet latency. In Gnutella, however, Query response time is an important performance metric and it can be affected by the message latency. One of the implications of using TCP backpressure is that the messages, if delivered, may have been buffered longer in the overlay network. Our experiments, however, show that the longer message delay does not necessarily translate to a longer Query response time. Figure 35 compares the CDFs of Query response time in some of the experiments (with the original peer behavior settings) we described above. In the experiment with an average hit rate of 0.2, there is hardly any difference between TCP backpressure and drop tail queuing. At a hit rate of 0.6, TCP backpressure actually has a slightly shorter average Query response time than drop tail queuing. Although the average message latency may have increased by TCP backpressure, its negative effect on Query response time is offset by the fact that the reduction of losses increases the chance of securing a hit that is close to the Query originator.

Heterogeneity If a servent without TCP backpressure capability is encountered before the traffic is pushed back to where there are available buffers, the congestion will be shifted to this servent and no performance improvement will be achievable. The probability for this to happen will increase when there are more servents in the system that are not TCP backpressure capable, or when the system load is heavier.

Figure 36 shows the QueryHit throughput (shown as the progress of the number of QueryHits) with different mixtures of TCP backpressure enabled peers and disabled peers. The improvement is reduced when 50% of the servents do not support TCP backpressure, to about half of the improvement when all the servents support TCP backpressure. When only 25% servents are capable of TCP backpressure, there is hardly any throughput improvement versus the drop tail scheme.

4.4.3 Rate Limiting

Rate limiting is a scheme that tries to avoid message losses by limiting the input and/or outgoing rate, and it is very commonly implemented by Gnutella servents. As the previous

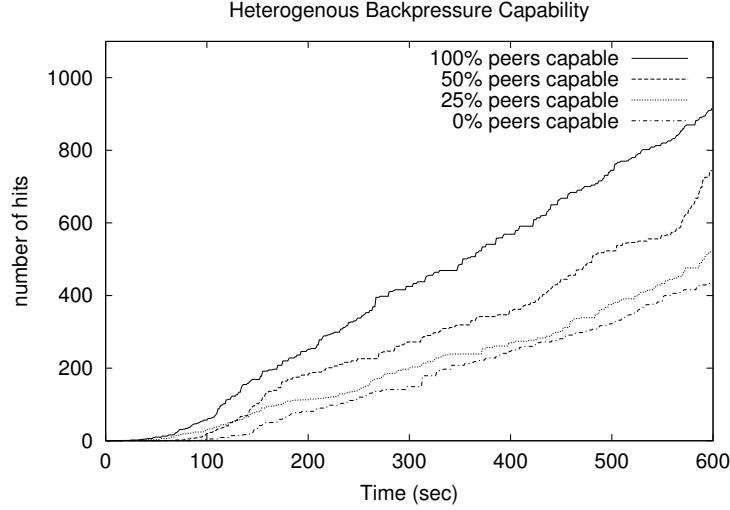


Figure 36: Heterogeneous backpressure capability

two schemes, rate limiting is often used on network routers with some notion of flow. In Gnutella, when a predefined incoming rate limit is reached, a servent will stop reading from a socket (*per-connection limit*) or any incoming socket (*per-servent limit*). In this subsection, we will look at the performance of incoming rate limiting, in terms of congestion reduction and relative throughput. Both per-servent rate limiting and per-connection rate limiting are experimented with.

On the X-axis of Figure 37, xc refers to per-connection rate limiting with a limit of x (messages per second), and xs refers to per-servent rate limiting with a limit of x . b/p refers to TCP backpressure. All the numbers shown are relative to those of drop tail queuing. We can see that rate limiting does not achieve significant loss reduction and throughput improvement at the same time, as is achieved by TCP backpressure. The highest throughput is achieved when a 500 msgs/sec per-connection limit is used, but it comes with almost the same level of message losses as drop tail queuing. Message losses can be reduced substantially only when the sources are slowed down by a rate limit that is much lower than the TCP tunnel throughput, such as the 10 msgs/sec per-connection limit in Figure 37. In practice, the default rate limit set by different servent implementations are all very low. For example, Gtk-Gnutella recommends 1Kbps for each inter-Ultrapeer connection, which is about 20-30 Gnutella messages.

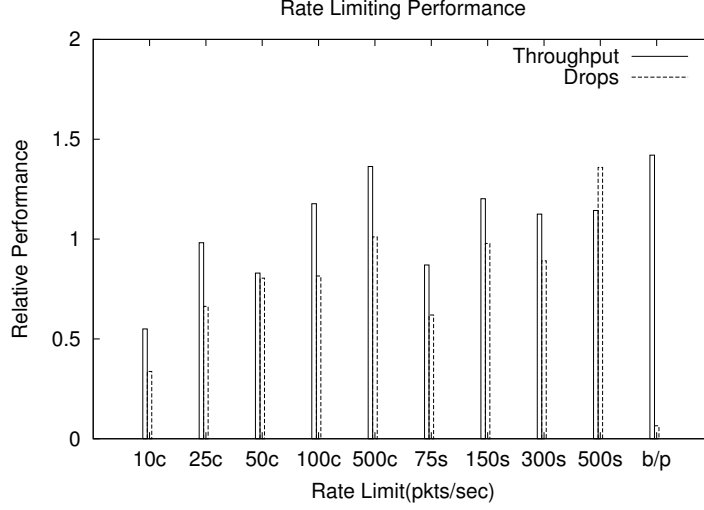


Figure 37: Rate limiting performance

Heterogeneity Based on the above discussions, we know that rate limiting is used primarily as an approach to reducing congestion. Clearly, a larger loss reduction can be achieved when a larger portion of peers implement rate limiting, as demonstrated by Figure 38. Nonetheless, Figure 38 also shows that when there are 80% peers supporting rate limiting, increasing the percentage of rate-limited peers does not further enhance the loss reduction significantly. Unlike TCP backpressure, for which a peer lacking the capability can reduce the benefits brought by many TCP backpressure capable peers, with rate limiting, a peer that has the capability can have limiting effect on peers that do not have the capability. Therefore, we do not need 100% peers supporting rate limiting to achieve that effect. In each of these experiments, a different percentage of peers enforce a 10 msgs/sec per-connection rate limit. The numbers shown are relative to those of the drop tail queuing.

4.5 Summary

In this chapter, we examine the congestion control mechanisms and message loss in the current Gnutella network. Our investigation reveals the heterogeneous congestion control strategies existing among different server implementations. Through measurement, we show the different Ultrapeer message loss behavior resulting from this heterogeneity. To understand the global performance of individual congestion control mechanisms, most of

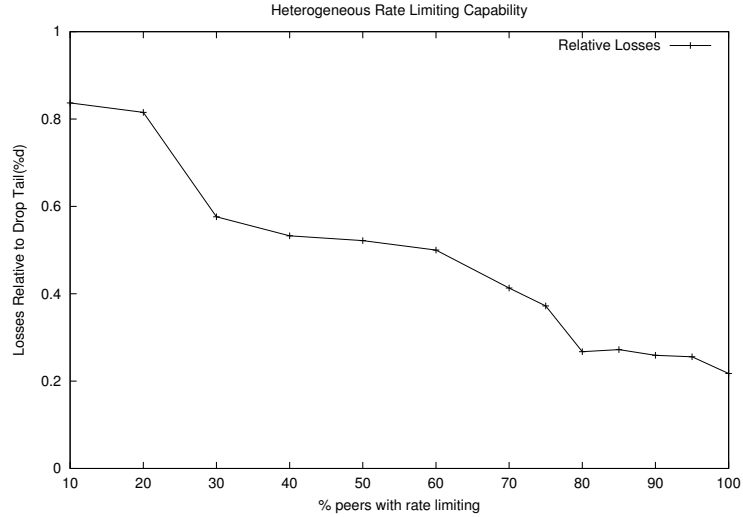


Figure 38: Heterogeneous rate limiting capability

which are derived from those designed for native network routers, we perform packet-level simulations of the Gnutella network. We show that the usage and performance of these mechanisms are strongly affected by the characteristics of the Gnutella network, among which are the lack of a strong notion of “flow”, and the heterogeneity of congestion control mechanisms. In particular, we show that: 1) the design of an optimal priority policy for the priority queuing scheme is challenged by multiple dimensions of performance requirement and the dynamic workload composition; 2) the performance of TCP backpressure is sensitive to the traffic load; 3) rate limiting reduces congestion effectively only at the cost of throughput degradation; 4) for each of the above schemes, heterogeneity precludes the full realization of its full advantage. The study in this work brings us closer to our ultimate goal of understanding overlay congestion in general and designing better resource management schemes for overlays.

CHAPTER V

PACKET-LEVEL SIMULATION OF PEER-TO-PEER NETWORKS

5.1 *Introduction*

In this chapter, we develop a *scalable and extensible packet-level peer-to-peer simulator*. The primary contributions of this work are:

- the development of a framework of an extensible peer-to-peer simulator that will allow a complete evaluation of peer-to-peer systems that include the details of the underlying network, and
- the demonstration of the importance of incorporating a network model and the use of packet-level simulations in peer-to-peer research, through the use of Gnutella simulation developed according to our framework

To address the needs of a wide range of studies, our framework simulator has the following characteristics:

- **Generic.** Peer-to-peer systems can differ in many aspects, e.g., initialization (a procedure often called bootstrapping) and maintenance of peering relationships, protocol message routing, and user behavior. The simulator should capture all the common aspects, while accommodating a large degree of diversity in each aspect. To address this requirement, a high-level characterization of peer-to-peer systems is required.
- **Platform independent.** Depending on the needs of their simulations, users may prefer one network simulator over another. For users that are not interested in the packet-level details of a simulation, the simulator should also be able to run without a network simulator. To accomplish this, we define a common socket-like API between the peer-to-peer simulator and the lower layer network simulator. Our simulator can run on

top of network simulators such as *ns2*[34], *GTNets* [111] and *pdns*[114].

- Scalable. A peer-to-peer system can consist of thousands of nodes or more, which requires a highly scalable simulator when packet-level details are pursued. Platform-independence makes it possible for our simulator to utilize existing techniques, such as *pdns* and *NixVector* routing [113], to scale network simulations.
- Easy to extend and use. Most users of the simulator either need to experiment with an existing peer-to-peer system, e.g., for performance analysis and workload characterization [75], or they are interested in redesigning/adding a few components in an existing peer-to-peer system. To avoid duplicate efforts by users and to make it easy to extend, the simulator is designed to be modular and composable. It is also easy for users to set up and describe their simulations.

We demonstrate the application of the framework by building a Gnutella system simulator based on our framework. Using the Gnutella simulator, we show that peer-to-peer system performance can be affected by network characteristics to a large extent, and that only packet-level details can reveal some aspects of the impact. Several usages of the Gnutella simulator are to be described to illustrate how extensibility and composability facilitate its use for different research purposes.

The rest of the chapter is organized as follows. Section 2 describes the simulator framework in details. Section 3 introduces a Gnutella system simulator built based on the simulator framework. Section 4 demonstrates how the Gnutella system performance is impacted by network characteristics through some simple experiments. Section 5 describes how we scale up peer-to-peer simulations using existing parallelization techniques. Section 6 concludes the chapter.

5.2 Simulator Architecture

In this section, we describe the overall architecture of the peer-to-peer framework simulator and its implementation. We start with a general description of peer-to-peer systems and then explain the simulator in details.

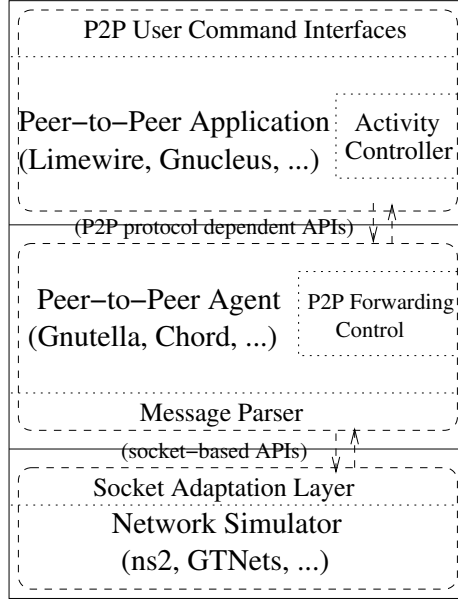


Figure 39: Peer simulator architecture

5.2.1 Overview

Peer-to-Peer Systems are decentralized applications with which people share resources or collaborate on certain tasks. The peers in one system usually have the same basic functionalities, but can be dissimilar in certain characteristics and may behave very differently. The basic functionalities of a peer can be generally categorized as follows [95]:

- share and search for resources
- use a certain peer-to-peer protocol to implement the resource sharing: establish and maintain connections with peers, accept user input and initiate protocol messages, process and route protocol messages

In our simulator, a peer-to-peer system consists of a group of peer nodes, each of which has the three-layer architecture shown in Figure 39. At the bottom layer is the packet-level network simulator which provides packet transfer up to the transport layer. The middle layer is the peer-to-peer agent (PeerAgent) which deals with a specific peer-to-peer protocol. Among its responsibilities are the transmission, reception, processing and forwarding of protocol messages, as well as the maintenance of transport connections with

peers. The peer-to-peer application entity (PeerApp) at the top of the architecture performs application related actions such as resource sharing, query processing and peer relation maintenance. This separation of functionalities between PeerAgent and PeerApp are based on two considerations:

- There is often a many-to-one mapping between the application and the peer-to-peer protocol or vice versa. For example, LimeWire, Gnucleus and BearShare all use the Gnutella protocol and they can be built on top of the same Gnutella protocol agent. In some other cases, people may want to compare different protocols by running a same application on top of them.
- People conducting research on the two layers often have different concerns or areas of interest. Protocol layer research may focus on the system's communication behavior. Application layer research may be more concerned with issues such as user behavior. It is desirable if they can independently consider the components of interest.

We implement the framework simulator as a set of C++ base classes. The rest of the section describes each of these components.

5.2.2 PeerApp

PeerApp is the direct interface to peer-to-peer application users. It embodies the behavior of a peer-to-peer application. PeerApp is therefore a component where diversity is most likely to take place. However, we are able to identify the activities common to most applications, which can be categorized as follows.

- User interface. User interface includes APIs that are called by application users and may trigger certain protocol messages and/or changes to the peer state. Functions in this category include: *join()*, *leave()*, *idle()*, *share()*, *search()*. There are also upcalls from PeerApp to users, the most important one being *queryhit()*.
- Maintenance of peer relationship. Peers in a system connect to each other in certain ways. Two operations commonly involved are: 1) *bootstrap*: which learns of, selects

and connects to a number of peers as a peer initially joins the system, and 2) *maintenance*: which adapts to the system changes by possibly dropping old peers and adding new peers for improved connectivity.

ActivityController One of the most important inputs to a peer-to-peer system simulation is the user behavior, which is often derived from a real or synthetic trace. Another possibility is to use the results obtained by some peer characterization studies and associate a certain model with the peer behavior. ActivityController is an entity associated with a PeerApp that generates user actions dynamically based on a behavioral model. Therefore, the interface between ActivityController and PeerApp is the same as the User/PeerApp interface. This natural correspondence facilitates the development of different classes of ActivityController.

We implement the model proposed in [55] with two classes: PeerSys and ClassSpec. PeerSys models system-wide parameters such as the number and popularity distribution of popular files, and the number of different classes of peers. ClassSpec is the model of a specific class of users, such as on/off probability, the average length of idle period and offline period. An ActivityController is created with two parameters: a PeerSys and a ClassSpec. ActivityController is implemented basically as a timer of which the expiration triggers a change of peer state and schedules the next event of state change.

5.2.3 PeerAgent

PeerAgent performs protocol dependent message transmission and processing. The APIs between PeerAgent and PeerApp usually correspond to the message types defined in a specific peer-to-peer protocol. A packet delivered to a PeerAgent by the network simulator is first parsed by a message parser. Based on the details of the message, the PeerAgent either drops it, delivers it to the PeerApp, and/or forwards it to adjacent peer(s), after some protocol-dependent changes to the message, if necessary.

P2P Forwarding Control In most peer-to-peer systems, peers perform some sort of protocol message forwarding. During the forwarding process, it can easily happen that the

Table 10: Socket Upcalls

Name	Function
int upcall_recv(Socket*, PacketData*, Handler*)	Data Received
void upcall_connected(Socket *)	Active Connection Established
void upcall_passconn(Socket *)	Passive Connection Established
void upcall_closing(Socket *)	Socket Being Closed
void upcall_send(Socket *)	Socket Ready to Send
void upcall_accept(Socket *)	Incoming Connection Request

outgoing connection(s) is blocked. PeerAgent can invoke certain forwarding control component to perform peer-to-peer layer flow control or congestion control functions, instead of simply discarding the message. A forwarding control mechanism can be protocol dependent or independent and it is especially relevant to systems that use broadcast-style routing.

5.2.4 Socket Adaptation Layer

Part of the frustration in using existing network simulators for peer-to-peer research is the awkwardness of transferring real application layer data. The transport layer messaging services provided by different network simulators often have different APIs. To make the peer-to-peer simulator portable, we choose to use a socket-based interface between PeerAgent and the network simulator. The socket interface provides the necessary system socket operations: *bind*, *listen*, *connect*, *send*, *recv* and *poll*. Due to the sequential nature of discrete-event network simulations, we cannot have blocking socket operations. All changes of the socket status should be conveyed to the upper layer through upcalls. Upcalls defined in our system are listed in Table 10.

To use network simulators that do not provide the standard socket interface, we need to build a socket adaptation layer that wraps its transport layer services with the socket interface. For example, for *ns2*, we need to first build a new subclass of FullTcpAgent that can listen on a specific port and accept an incoming connection request by spawning a new TCP connection. The new subclass also remedies the fact that FullTcpAgent does not transfer the actual TCP payload across nodes.

Another issue related to the portability among network simulators is the different address formats used. The socket adaptation layer should perform the translation between the

address format used by the network simulator and that used by the peer-to-peer simulator.

Not Interested in Packet-level Details? To support users that are not interested in packet-level details, we implement a `NullSocket` interface. `NullSocket` conforms to the `Socket` interface in our system, but bypasses the connections to the network simulator and has a null operation in most of the routines. When a `NullSocket` connects to another socket, it merely records the pointer to that socket as its peer. When data is sent from the socket, the *upcall_recv()* procedure of its peer will be called directly.

5.2.5 Describing the Simulation

One of the desirable features of a simulator is the ease of description and setup of simulations. A common peer-to-peer simulation scenario involves a group of peers, each running on a physical node, going online and offline dynamically and probably sending some search queries when it is online.

Assume that we have a physical network already set up, to attach a peer to a physical node, we only need to create a `PeerApp` with the node address as the parameter. As mentioned previously, to control the behavior of a peer, either a trace or an `ActivityController` can be used. With a trace file, events related to peer behavior are pre-scheduled based on it. With an `ActivityController`, it is only necessary to initialize the `ActivityController` with a `PeerSys` and a `ClassSpec`. The following table is an excerpt from a sample *ns2* script, along with a specification of the system parameters (`PeerSys`) and the two classes of peers (`ClassSpec`, shown in Table 11). In Table 11, “prob. going offline” is the probability for a peer to go offline after a successful query, and “freeloader” indicates whether a peer uses the system without sharing any of its own.

Table 11: classinfo.txt

class	avg offline(sec)	avg idle(sec)	prob. going offline	freloder?
0	4320	300	0.2	0
1	4320	30	0.1	1

```

01: set ns [new Simulator]
02: set sys [new PeerSys 100 2 2]
03: $sys init-class "classinfo.txt"
04: set rng_ [new RNG]
05: for {set i 0} {$i<2000} {incr i} {
06: set node($i) [$ns node]
07: if { $i%4 == 0 } {
08: set p_($i) [new PeerApp $i]
09: # attach as a class 0 peer
10: $p_($i) attach-peersys $sys 0
11: set stime [$rng_ uniform 0.0 100.0]
12: $ns at $stime "$p_($i) start"
13: }
14: }

```

P2P System Parameters: 100 popular files pareto ($\alpha = 2$) popularity distribution 2 classes of peers 2000 nodes and 500 <i>class</i> 0 peers
Classes Specification: classinfo.txt (Table 11)

5.3 *Specializing the Framework to Simulate Gnutella Systems*

Gnutella[3] is one of the most popular protocols used by peer-to-peer file sharing applications. Following the framework described in the previous section, the implementation of a Gnutella simulator is straightforward.

The major components, GnutellaAgent and GnutellaApp, are subclasses of PeerAgent

and PeerApp, respectively. The APIs between them are downcalls and upcalls that correspond to the Gnutella protocol messages: *Connect*, *Ping*, *Pong*, *Query*, *QueryHit* and *Push*. A GnutellaAgent uses upcalls to deliver protocol messages intended for local GnutellaApp. In response to a protocol message, or as a result of some user requests, a GnutellaApp uses downcalls to ask its GnutellaAgent to send protocol messages.

Internal design of the GnutellaAgent also follows directly from the Gnutella protocol. For example, the Gnutella message parser recognizes the Gnutella message format, and the routing engine maintains a cache of Gnutella Descriptor IDs to make forwarding decisions.

At the application layer, we shall see that many subcomponents, e.g., the ActivityController and the user interfaces, of the PeerApp can be reused by the Gnutella simulator. On the other hand, GnutellaAgent and GnutellaApp can be extended to allow for optimizations at both the protocol layer and the application layer, as we will demonstrate in the following subsection.

Extensibility Through the descriptions of the framework and the Gnutella simulator, we can pinpoint two design rationales that make them extensible: *i) functional isolation*, and *ii) protocol-centric structuring*. Functional isolation is a standard principle to build modular and extensible systems. We use the term protocol-centric structuring to refer to the organization of the system components based on their relation to the peer-to-peer protocol, which simplifies the extension of the system. For example, PeerApp is the end consumer/producer of the protocol messages, PeerAgent usually only routes/delivers protocol messages, and message parser/forwarding control facilitates/controls the routing. Following are two examples that demonstrate how we can easily extend our Gnutella simulator to evaluate some proposed schemes intended to improve the Gnutella system performance.

In [117], the author proposes a revised Ping/Pong scheme. The scheme caches Pongs to avoid broadcasting Pings too rapidly, multiplexes Pings to avoid additional broadcasts while the cache is filling, and limits the number of Pongs sent in response to a Ping. These techniques do not involve any changes to the Gnutella protocol. To simulate the revised scheme with our simulator, it is only necessary to overload the upcalls corresponding to the

Ping request and Pong response, although additional data structures need to be added to GnutellaApp for use in these techniques.

In [82], the authors propose a keyword posting protocol for decentralized peer-to-peer systems to improve search success rate. The proposal involves adding new protocol messages for Gnutella peers (servents) to post keywords and to retrieve file names matching certain posted keywords. To evaluate the proposal using our simulator, new upcalls and downcalls corresponding to the posting protocol messages need to be added to the interface between GnutellaAgent and GnutellaApp. In addition, the upcall corresponding to QueryHit should be overloaded to cache the search results. Notice that the keyword posting protocol is related to the resource sharing function of a peer, and therefore part of the user interface of GnutellaApp, the *search()* method in this case, should also be overloaded, which is different from the simulation of the revised Ping/Pong scheme.

5.4 *Experimental Results on Gnutella Performance*

By building different socket adaptation layers, we are able to port the peer-to-peer simulator framework to *ns2*, *GTNets* and *pdns*, and run the Gnutella simulator on them. In this section, we will use some simple experiments to show that packet-level detail simulations capture factors that influence the Gnutella system performance but that are not covered by analytic peer-to-peer models such as [55], or even flow-level models such as [52].

In [55], the authors proposed a model (we call it UMASS model in the following discussions) of peer-to-peer file sharing systems. They model the file search requests as customers and the available peers as servers. With the application of queuing network theory, the authors obtained interesting results on such system performance metrics as system throughput, query failure probability, and mean query response time.

In our experiments, we have a certain number (742) of peers with a certain initial join sequence (Poisson arrival of new peers). Peer behavior is derived from the UMASS model and except for the experiments in the first subsection below, we use the parameters shown in Table 11, which are also similar to the ones used in paper [55] and are based on existing

studies. *Ns2* is used as the underlying network simulator. We use a 100-node transit-stub network topology generated with GT-ITM [36] as the backbone and attach a random number (between 10 and 20) of leaf nodes to each stub node. Other parameters of the network setup are specified for different experiments as needed. Peers are attached to some of those leaf nodes. The access bandwidth of a peer, which is represented by the bandwidth of the link between the leaf node to which it is attached and the stub node it is linked to, follows a certain (but varied in some experiments) distribution, based on statistics such as [2]. It should be noted, though, that it is not our intention to validate or disprove the results in [55], and that we do not model the downloading activity. Without the download component, system throughput is defined as the number of query hits per second.

5.4.1 System Throughput

In the first set of experiments, we will show, through packet-level simulations, that the Gnutella system performance can differ substantially with different network throughput settings. All the settings and parameters are the same across the three experiments in this set except the distribution of the access bandwidth or the link delay on the backbone (which can be used to represent different degrees of proximity among peers), both of which affect the average throughput of the TCP tunnels used for communication in the Gnutella network. Results in Figure 40 show substantially different Gnutella system throughput values with different TCP throughput settings.

Whereas the different results in Figure 40 may be intuitive, analytic models that fail to include a detailed network model will not be able to reveal such differences. In the UMASS model, for example, system throughput is a function of the average number of available peers and the connectivity among them. Notice that the two classes of peers we simulate in this set of experiments both have a θ probability of going offline after a successful query, therefore, the number of available peers at any instance of time is not affected by system throughput and stays the same for the three experiments. In addition, as we can see from Figure 41, the connectivity among peers is hardly changed across the three settings. Therefore, all the parameters used to compute the system throughput, and hence the result,

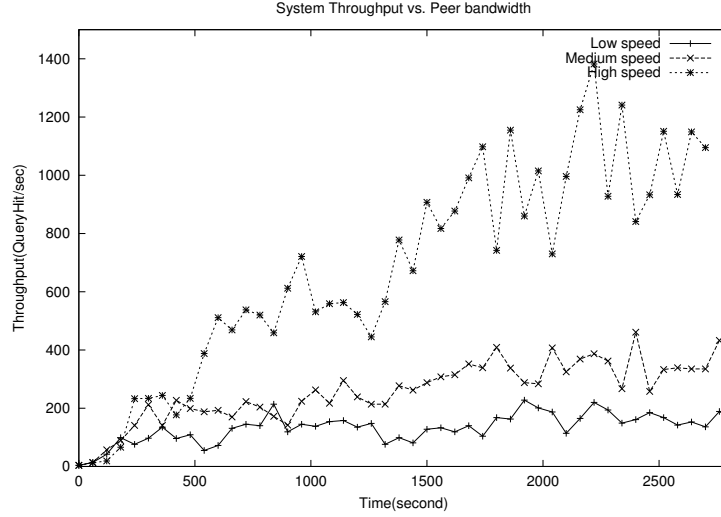


Figure 40: System throughput

should be very close for the three experiments, which is not consistent with the packet-level results in Figure 40. We observe that the discrepancy results not only from the different TCP throughput values, but also from the different loss ratios of QueryHit packets. Losses can happen when there are bottleneck TCP tunnels in the Gnutella network.

5.4.2 Query Response Time

Query response time is one of the most user-perceivable performance metrics of a peer-to-peer system. In Gnutella systems, the Query response time consists of two parts: the time it takes a user to get the first QueryHit corresponding to a Query (*hit time*), and the time it takes to download a file (*downloading time*). People often assume that the hit time is negligible as compared to the downloading time. We believe that the Gnutella hit time is an important performance metric as it affects user behavior. In [55], hit time has been modeled as a function of non-network-related parameters such as the available online peers and the average hit probability corresponding to the resource being searched for.

In this set of experiments, we will first show that hit time is substantially impacted by the access bandwidth and the backbone link delay. For the experimental results shown in Figure 42, we vary the distribution of the peer access bandwidth to obtain two values of average access bandwidth that differ by 50%. The hit time presented is averaged over a

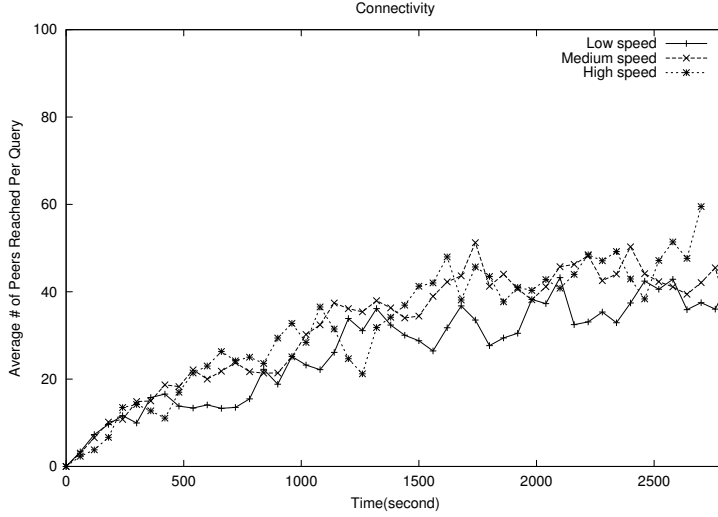


Figure 41: Connectivity

60-second period. It is evident that a higher access bandwidth achieves a shorter average hit time. Similarly in Figure 43, we show that link delays (a ratio of 1:2:3 among the three curves) affect the hit time. Notice that in this figure and the figures in the rest of the section, the hit time is averaged from the start of the trace.

In the experiments shown in Figures 42 and 43, different values of average access bandwidth or average link delay also result in different TCP throughput. It might be argued that the different TCP throughput can explain all the discrepancies and therefore a sophisticated flow-level model that can associate a certain throughput with each of the peer connections can model the hit time. To understand the problem further, we perform another subset of experiments. We make the throughput of all peer connections limited by peer access bandwidth, such that the TCP throughput is all limited to the same degree. Nonetheless, Figures 44 shows that the average hit time still varies substantially with different back bone link delay configurations.

Due to the heterogeneity among peers in terms of characteristics such as rate limit, re-attempt probability, it is hard to incorporate a detailed flow-level model such as [52]. More importantly, their model applies to the throughput of end-to-end flows and we can not map it directly to the goodput of most peer-to-peer systems, which use application-layer broadcasting and may have packet losses on forwarding peer nodes.

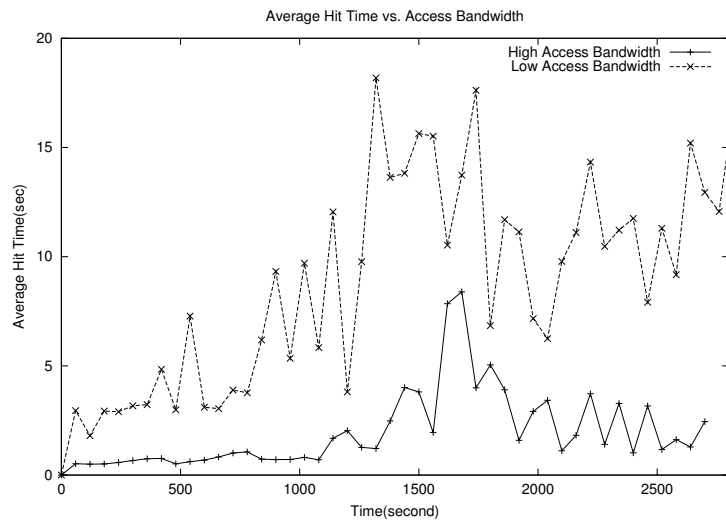


Figure 42: Hit time vs. access B/W

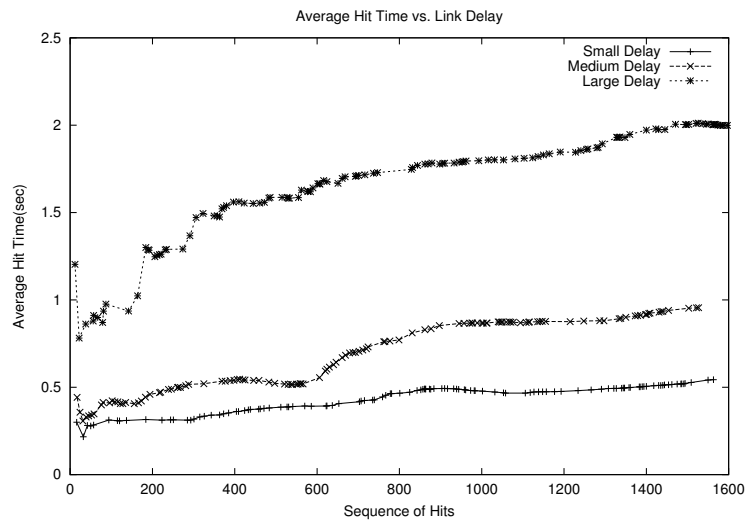


Figure 43: Hit time vs. link delay(1)

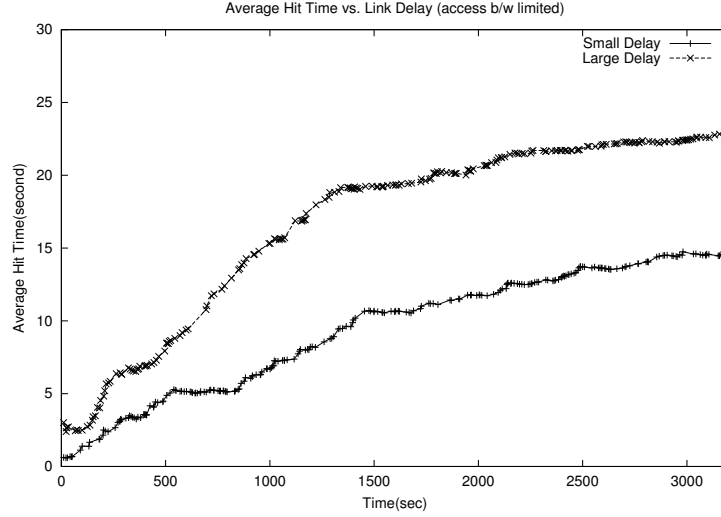


Figure 44: Hit time vs. link delay (2)

5.4.3 Considering the Rest of the Network

Most research on peer-to-peer systems has assumed that the access bandwidth of peers is limiting the network throughput achievable by peers in the system. The assumption enables or simplifies many evaluations, by not requiring the modeling of the rest of the network.

The experiments in this subsection show that when background traffic is added to the backbone in our setup, it can impact various Gnutella performance metrics and the impact tends to differ for different setups. In Figure 45, configuration 1 is the access bandwidth limited configuration we used for the Figure 44 experiment. Configuration 2 has twice the average access bandwidth as configuration 1 and configuration 3 has twice the average link delay as configuration 1. The background traffic consists of bulk TCP transfers starting at random time instances. The numbers shown are relative to the results when there is no background traffic. We can see that different performance metrics can be affected and that the effects vary for different setups. Further analysis is needed to understand how parameters such as the distribution of peer idle time, average wait time before aborting or reattempts impact the packet arrival pattern on the Gnutella TCP flows.

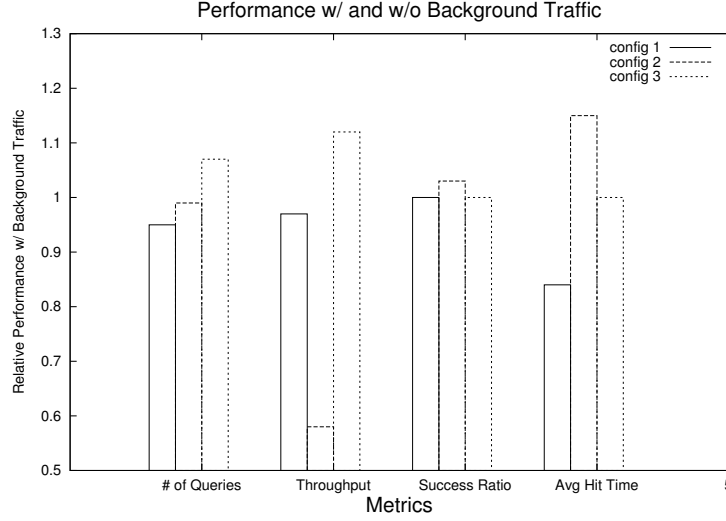


Figure 45: Performance with background traffic

5.5 Large Scale Simulation with PDNS

A Gnutella network can have tens of thousands of peers and run on a network with an even larger number of nodes. The amount of memory required by a simulation environment to model a topology of this size can often be a limiting factor for simulations. In our previous work, we developed a parallelization technique and built a parallel and distributed version of *ns2* called *pdns*[114]. Using *pdns*, a single simulation topology is divided into several pieces, and each of the pieces are simulated in a separate process on individual workstations. By dividing the topology model in this way, we gain a linear increase in the overall size of the topology, based on the number of workstations assigned to the simulation.

However, splitting the topology in this way introduced a number of other issues that required changes in which the *ns2* objects are addressed. We solved this problem by adding the notion of an *IP Address* to the *ns2* node objects, and binding the *ns2* agents to well known port numbers. This enables us to address packets to nodes that are in fact defined and modeled on a different workstation, without requiring knowledge of the details of those nodes. By simply changing the *ns2* addressing format to this IP addressing format used in *pdns*, we ported the *ns2* socket adaptation layer to *pdns* and were able to run the Gnutella simulator on *pdns* without any further changes.

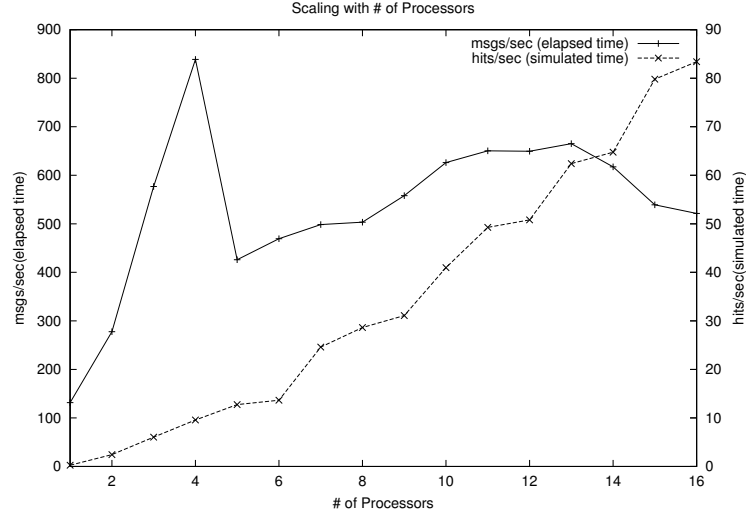


Figure 46: Scaling with number of processors

Using *pdns*, we performed Gnutella simulations using different numbers of processes. In these experiments, we run 512 nodes (distributed across four subnets) and 90 Gnutella servants on every physical machine. The largest experiment, run on 16 machines, simulates a physical network of 8192 nodes with 1440 Gnutella servants. The machines each has 4GB RAM and they are connected using 1Gbit Ethernet. Results show that significant scaling and speedup can be achieved using the parallelization techniques. In Figure 46, we plot the average number of Gnutella protocol messages sent/routed per second of elapsed run time¹ (Y-axis) against the number of processors we use (X-axis). As the number of processors increases, more inter-machine communication is incurred. Therefore, the largest number of messages per second is realized with four processors. On Y2-axis (right Y-axis), we plot the average number of QueryHits per second of *simulated* time, which represents the scale of the peer-to-peer system activity.

Since an important issue in the study of peer-to-peer networks is the effect of network scale on the peer-to-peer performance, we are planning on using *pdns* to create simulation models of 100,000 or more nodes and tens of thousands of Gnutella clients. Our previous

¹Although *message * hop* is a better measure of the simulation workload, we observe that the number of Gnutella protocol messages is a close approximation, given the star-like topology and hence the similar network diameters in the different settings.

work has shown that topologies of this size can easily be simulated using *pdns*.

5.6 Extensible for Different Research Purposes

In this section, we will demonstrate how we can easily extend the Gnutella simulator for use in some of our ongoing research efforts.

5.6.1 GWebCache Simulation

The process for a peer to find and connect to a set of initial peers, often called bootstrapping, has several important user-perceivable performance implications, such as startup latency and the quality of peers connected, which have not been studied previously. Treatment of peer transience may also differentiate one protocol from another in terms of system performance.

There is no standardized protocol for bootstrapping in Gnutella networks. The purpose of this work is to study one of the most popular mechanisms used by Gnutella peers, called GWebCache[7].

To support the GWebCache simulation, the *GnutellaAgent* and *GnutellaApp* is extended to support new interfaces corresponding to the GWebCache protocol, which are simple HTTP GET and HTTP PUT commands and HTTP responses. In addition to these, the *bootstrap()* method of *GnutellaApp* is the only one that needs to be overloaded, to send “HTTP GET hostfile” to preconfigured GWebCache servers. In the upcalls corresponding to the HTTP responses to “HTTP GET hostfile”, the original *GnutellaAgent::Connect()* is called to connect to known servents.

5.6.2 Testbed for Alternative Gnutella Flow Control Mechanisms

People have previously argued that Gnutella network does not scale because of the large network bandwidth it demands as its size increases[116]. We observe in our experiments that there are lots of packet drops on peer nodes. A Gnutella peer drops a packet when the TCP connection on which it is to be sent (or forwarded) is blocked. Both the loss of Query messages and QueryHit messages will degrade the system throughput. It has not been studied previously whether any flow control mechanism at the Gnutella protocol level

will help improving the system throughput. Alternative flow control mechanisms include hop-by-hop back-pressure, buffering and the separation of broadcast and unicast TCP tunnels between neighboring peers. As in physical networks, the performance of a particular flow control mechanism is often dependent on the nature of congestions. In addition, the performance could be dependent on the parameters of the flow control algorithm. E.g., the back-pressure algorithm might take actions only when certain type of messages are blocked, or only when the message TTL satisfy certain conditions.

Using the Gnutella simulator, we are currently evaluating the back-pressure scheme and the buffering scheme, as compared to the simple dropping. The following excerpt of code, *fc_* can be any object that implements a flow control algorithm:

```
int GnutellaAgent::upcall_rcv(PacketData *packet) {
    msgParser->parse(packet);
    .....
    if(forward(packet)==BLOCKED) {
        if(fc_>flowcontrol(packet)==0)
            return 0;
    }
    return packet->size();
    .....
}
```

Back-pressure can be automatically implemented using TCP's flow control mechanism based on advertised receiver window. Since *upcall_rcv()* returns the amount of data is consumed by the socket application, back-pressure will be simply achieved if the *flow-control()* method of the flow control algorithm returns 0.

5.7 Summary

In this chapter, we develop the framework of a platform-independent and extensible packet-level simulator for large-scale peer-to-peer systems. The merits of the framework are demonstrated by the implementation and usages of a Gnutella simulator based on it. Using the Gnutella simulator, we show that peer-to-peer system performance can be impacted by the network characteristics to a large extent, and that the impact is mostly not even revealed by flow-level analysis. We also demonstrate the extensibility of the simulator by describing how

it is used to simulate some schemes proposed to improve the Gnutella system performance. Through our work, we intend to encourage more attention to the detailed network modeling in peer-to-peer research, and we provide a framework of a scalable packet-level peer-to-peer simulator to facilitate that. Part of our future work is to develop second level frameworks for different categories of peer-to-peer systems, e.g., unstructured and structured peer-to-peer systems.

CHAPTER VI

EXPLOITING THE PREDICTABILITY OF TCP STEADY STATE BEHAVIOR TO SPEED UP NETWORK SIMULATION

6.1 *Introduction*

As discussed in Chapter 2, the behavior of a TCP flow is predictable given a steady path condition. It is our goal in this chapter to exploit this fact to estimate the average behavior of a TCP flow over a future period of time where steady-state conditions hold, thus allowing for a reduction (or elimination) of the processing required for packet events during the period. In particular, given the current progress of a TCP data transfer (i.e., the highest sequence number sent or ACKed) and a predicted average throughput over a future steady-state period, we can predict the time a packet with a given sequence number will have been successfully delivered.

The TCP steady-state throughput formula [100] which estimates the average throughput based on RTT and loss ratio has been shown to approximate the real TCP flow throughput closely. We observe that accurate measurement of RTT and loss ratio is quite manageable in a simulation system and the cost saving through throughput prediction could be significant as long as a flow is in steady-state for a long-enough time relative to the time needed to accurately measure loss ratio and RTT. Another way to predict TCP progress is to simply measure the average throughput in the steady-state for a period of time. Note again that these predictions are based on *estimates* of steady-state TCP throughput, which in turn, is based on observations *during* the transfer in steady-state.

In this thesis we design a simulation framework that provides the flexibility to incorporate this method of simulating TCP packet flows. The complexities of such a framework

result from the requirements that: 1) it has to accommodate different network configurations (as heterogeneous as the Internet), on/off flow behaviors and interaction between predicted flows and packet-based flows; 2) it has to preserve the statistical behavior of every entity involved in the system, from hosts to routers to links, so as to maintain the accuracy of the network simulation as a whole.

Our work has the following contributions:

- We design a simulation framework that can utilize prediction to improve the performance of TCP flows simulation, while being able to interact with packet-based flows.
- We evaluate the speedup and approximation of the simulation framework. The two prediction techniques are compared for their approximation to packet-based simulation in different simulation configurations.
- We relate the errors of the prediction-based simulation to the transient TCP behavior and discuss the applicability of different methodologies to improving network simulation for different purposes.

The rest of the chapter is organized as follows. Section 2 details the design of the prediction-based simulation framework with predicted flows only. Section 3 describes the two prediction techniques: formula based prediction and direct measurement prediction. Section 4 extends the simulation framework to accommodate the coexistence of packet-based flows and predicted flows. Section 5 evaluates the performance (speedup and approximation) with an *ns2* implementation, compares the approximation of formula-based prediction and direct measurement prediction for different scenarios. Section 6 relates the approximation of the predicted flows to the transient behavior of TCP and gives suggestions on the use of the simulation technique. Section 7 compares our technique to the fluid flow models. This chapter is concluded in Section 8.

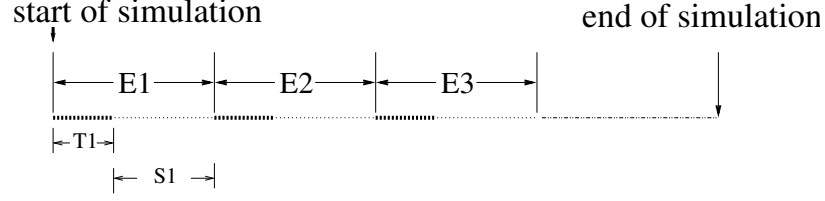


Figure 47: Simulation epochs

6.2 Prediction-based Simulation Design

6.2.1 Simulation Context

To illustrate our simulation scheme, we use the following model to describe our simulation context. In this model, there are n TCP flows and m non-TCP flows. Each flow i ($1 \leq i \leq n + m$) is an on-off flow which can be characterized by a series of (t_{ij}, B_{ij}) , $1 \leq j \leq k_i$, where t_{ij} is the start of the j th on-period of that flow, B_{ij} is the number of bytes sent in that period, and k_i is the number of on-periods of flow i . We call a TCP flow that is simulated with our proposed scheme as a *predicted flow*, as compared to a normal flow which we call a *packet-based flow*. In this section, we will concentrate on predicted flows only, i.e., there is no non-TCP flow and all the TCP flows are predicted flows. In Section 6.4, we will investigate cases where predicted flows interact with packet-based flows.

6.2.2 Simulation Framework Details

During a simulation, the system undergoes a series of epochs E_i as illustrated by Figure 47, where each epoch consists of a *transient period* (T, bold dashed segments) and a *steady-state period* (S, regular dotted segments). In this subsection, we describe the simulation system details of an epoch, including the switch from transient to steady-state period, as well as the transition between successive epochs.

Epochs An epoch represents a certain snapshot configuration of active flows in the system. Transition from one epoch to another is triggered by changes of active flows, which we will detail shortly.

During the initial period of an epoch, TCP flows experience changes in average RTT and loss ratio due to the change of competing flows, thus we call this a *transient period*. It

is during this period that each TCP flow learns its average throughput. Hence all flows¹ in our simulation system send packets during the period. After the transient period, the system switches to the steady-state period, during which each flow is assumed stable and saves runtime by not sending any packet. The length of the transient period should be chosen such that the average throughput seen by each flow has become relatively stable by the end of the transient period. It should be at least longer than T_{ss} , which is defined in [38] as the length of the initial slow start period of a new flow, as we will explain in Section 6.3.

Intra-epoch Transition Intra-epoch transition is the transition from transient period to steady-state period within an epoch and involves two major actions.

One action is for the active TCP flows and the network to checkpoint their current states so that the next time the system switches to transient period, each of them has a starting state that statistically approximates the steady-state of this epoch. TCP flow states include congestion window, RTT, retransmission timeout, and data transfer progress. The network state we checkpoint is the queue occupancy on each router. The checkpointing involves freezing the queue by not allowing any enqueue or dequeue operation, so that the same buffer occupancy can be restored at the next transition.

The other action is for active flows to predict the end of their current on-period. An active flow is assumed to have predicted an average throughput r_p (bytes/sec) by the end of a transient period. Given that an active flow has B_r bytes of data remaining to be sent during its current on-period, we can schedule a *steady-state period* event for the flow at B_r/r_p seconds later, when the on-period is predicted to end. Since a TCP application can pass down additional data during an on-period, the amount of data to be sent within a flow's on-period is not fixed at the time of transition and we need to reschedule the steady-state period event whenever more data is buffered by TCP.

¹In practice, only those flows that are affected by the preceding change of active flows need to switch to transient period. We do not make this distinction in the chapter only for the ease of description.

Inter-epoch Transition Transitions between epochs are triggered by changes of active flows. There are two types of event that result in such changes.

- end of an on-period. The end of a flow's on-period is usually indicated by the execution of its steady-state period event². End of a flow is a special case of this.
- start of an on-period. The start of a flow's on-period is detected when new application data arrives during the flow's off-period. Start of a new flow is a special case of this.

Assume that there are p active flows and the *steady-state period event* of the i th active flow is scheduled to expire at b_i . Further assume that there are q inactive flows and the j th inactive flow will become active at s_j . The next inter-epoch transition will then take place at:

$$\min(b_1, b_2, b_3, \dots, b_p, s_1, s_2, s_3, \dots, s_q)$$

However, neither b_i ($1 \leq i \leq p$) nor s_j ($1 \leq j \leq q$) can be predicted, so that the next epoch transition can only be triggered dynamically as b_i 's expire or as s_j 's take place. Upon an inter-epoch transition, a transition from steady state to transient state, we need to cancel all the scheduled steady-state period events. Also, the active flows and the network need to recover/update states and resume sending/forwarding packets.

When a flow switches from steady state to transient state, it restores all the previously stored TCP states except those related to the progress of data transfer, which need additional attention. In particular, assume that at the time of the last intra-epoch transition, the highest sequence number acknowledged by the other end (*highest_ack*) is ha_o , the current *highest_ack* is updated as follows: $highest_ack = ha_o + r_p * dur_p$, where r_p and dur_p are the predicted throughput and the duration of the steady state in the last epoch. It is also important to restore the steady-state ACK clocking behavior of the TCP flow, which we achieve by appropriately setting relevant state variables of the other end of the TCP connection. When the queues resume operations, residual packets frozen in the queues during the last transition are delivered to their destinations although will be discarded as obsolete data because the receiving TCPs' states have been updated.

²There are cases where an on-period ends during the transient period.

6.3 Predicting Steady-state Throughput

We have described the framework to speed up TCP flow simulation with TCP throughput prediction. This section is dedicated to describing two techniques to predict the average throughput of a steady-state TCP flow.

6.3.1 Summary of TCP behavior

A TCP flow can be characterized as an *initial slow start* phase followed by a *steady-state* phase [38, 100]. TCP tries to capture the maximum available bandwidth during the initial slow start phase with the exponential increase of congestion window on each ACK. At the end of slow start, which is indicated by the first packet loss and the following timeout period, TCP goes into the steady-state. Steady-state of a TCP flow comprises of a series of timeout periods (*TO*), each of which is, in turn, composed of a *slow start* phase and multiple triple duplicate periods (*TDP*). The end of a TDP is marked by triple duplicate ACKs and the end of a TO is marked by a retransmission timeout event. Both the slow start phase and the TDP period strictly follow certain algorithms to adapt the congestion window. Given the deterministic algorithm used, TCP behavior is very predictable given a steady-state path condition, i.e., a path with stationary random losses. In particular, the length of the slow start phase and the length/number of TDPs within a TO, as well as the amount of data transferred during a TO can all be statistically determined.

6.3.2 Measurement Period

Larger speedup can be obtained by using shorter transient periods. On the other hand, a transient period should be long enough to capture the steady-state behavior of each flow. Our experiments show that a measurement period on the order of tens of TDPs gives a close estimate of RTT and loss ratio. However, the question that “how long is long enough” should be answered on a per-simulation basis, which we will elaborate on in Section 6.6.

Prediction should also be based on a period when no flow is still in its initial slow-start stage. We call a period during which the system is actually measuring (predicting) each flow’s throughput a *measurement period*. It should begin after the initial slow-start phase of

the newly (re)started flow in a transient period. The average length of an initial slow-start can be readily evaluated using existing results (see the expression for T_{ss} in [38]).

6.3.3 Direct Throughput Measurement

One approach to predict is to directly measure the average throughput of each flow. To do that, we simply check the amount of data reliably delivered *during* the measurement period. We then derive the average throughput by dividing the amount of data by the length of the measurement period. The accuracy of this prediction depends on whether the measurement period is long enough for the predicted flow to get close to the limiting behavior of its steady-state.

6.3.4 Formula Based Prediction

Our proposed formula-based prediction is based on the model established in [100], where the authors characterize the steady state throughput of a TCP flow as a function of loss ratio, round trip time, b (number of segments ACKed by one ACK) and the maximum window size advertised by the receiver. The $B(loss, RTT, b, W_{max})$ formula applies to a bulk transfer TCP flow, i.e., a flow with an unlimited amount of data to send. Compared to the previous work in [88], this model captures the timeout events in the steady state and hence its applicability is not limited to small packet loss ratios.

With formula-based prediction, we measure for each flow its average RTT and the packet loss probability on its path. We then estimate the average throughput using the TCP throughput formula. The accuracy of the prediction depends both on the accuracy of the measured RTT and loss probability, and on the formula's approximation to real throughput. We obtain the average RTT by averaging all the TCP RTT samples during a measurement period. Our experiments suggest that average RTT converges as the measurement period increases and a period of 10 TDPs is usually long enough to ensure reasonable accuracy. During the measurement period, the simulation counts the number of packets received and dropped on every router a flow traverses. The loss probability of the flow is derived from these drop ratios³. Our experiments suggest that loss probability converges

³Assume that the loss ratio of the on-path routers are $p_1, p_2, \dots, p_k$, the loss probability of the flow is

at approximately the same rate as RTT.

6.4 *Mixing Packet-based and Predicted Flows*

Section 6.2.2 describes the simulation framework with all the flows being predicted TCP flows. The framework speeds up simulation through prediction and aggregation of packet events, i.e., the speedup comes at the cost of losing packet events. This is not desirable for simulations where every packet event is of interest to some flows. It is also possible that TCP flows compete for bandwidth with non-TCP flows, for which the steady-state prediction does not work. Both cases suggest the need to accommodate packet-based flows in the framework, which is the general case described in Section 6.2.1. To that end, we need only to make a few minor changes to the framework, as described below.

Changes to the Framework Within the new framework, the transition between epochs is triggered by the start/end of an on-period of any flow, including packet-based flows. As of a predicted flow, the start of an on-period of a packet-based flow is also detected by observing the arrival of application data during an off-period. The end of an on-period in a packet-based TCP flow, though, is always detected by the vacating of TCP buffer. For a packet-based non-TCP flow, the data source can declare the end of an on-period after observing inactivity for a certain period of time.

Accounting for Network Resources Usage The average RTT and loss ratio a flow experiences are affected by the queue limit and link bandwidth of on-path routers, as well as the additional queuing delay and losses due to competing flows. For each predicted flow, we can account for both the link bandwidth it uses and the queuing delay it causes by adjusting the bandwidth of each link on its path according to its share of the link's throughput. In particular, during a measurement period, we measure the throughput of each link on the flow's path in addition to predicting the flow's average throughput. Suppose that the estimate of the flow's throughput is r_f , the bandwidth and the measured throughput of a link are C and r_l , respectively. When the flow switches to steady-state, we adjust the

approximately $\sum_{i=1}^k p_i - (1 - \sum_{j=1}^{i-1} p_j)$

link bandwidth to $C * (1 - r_f/r_l)$. When a predicted flow switches back to transient state, the link bandwidth is increased according to the flow's share of throughput in the previous epoch. By proportionally reducing and increasing the bandwidth, we statistically maintain the queue occupancy and approximate the queuing delay. We could use a typical queuing model of network routers to explain the rationales behind this approach. A router is often modeled as a single-server queuing system $M/D/1/N$ with Poisson arrival⁴, constant service time and finite capacity. Assume that the packet arrival rate is λ , the service rate is μ , and the queue capacity is N . Using Little's Law, we can derive[119] the limiting, i.e., long-run, probability that the system has n customers (packets):

$$P_n = \frac{(\lambda/\mu)^n (1 - \lambda/\mu)}{1 - (\lambda/\mu)^{N+1}} \quad (1 \leq n \leq N)$$

Notice that packet loss probability is P_N . We could also derive the average waiting time in the system:

$$\begin{aligned} W &= L/\lambda_a \\ &= \frac{[1 + N(\lambda/\mu)^{N+1} - (N+1)(\lambda/\mu)^N]}{(\mu - \lambda)(1 - (\lambda/\mu)^{N+1}) * (1 - P_N)} \end{aligned}$$

Notice that λ_a represents the arrival rate of packets that are not dropped, while λ represents the overall packet arrival rate. Assuming that flows experience packet drops roughly proportional to their bandwidth share, our approach reduces the service rate μ and packet arrival rate λ by the same percentage (the predicted flows' bandwidth share in percentage). Therefore, P_n stays the same as if the predicted flows are still competing for the router resources. Assuming that queuing delay is negligible as compared to propagation delay, which is often an assumption made by fluid flow model analysis[22], or the predicted flows have a small share of bandwidth, our technique of proportional bandwidth reduction introduces negligible error to the one-way delay of packet-based flows.

⁴It is observed that wide-area link-level modeling of aggregate traffic should study the self-similarity of traffic at various levels[103], however, TCP simulation is often used in protocol-specific modeling with individual sources, for which the Poisson packet arrival on routers could be valid.

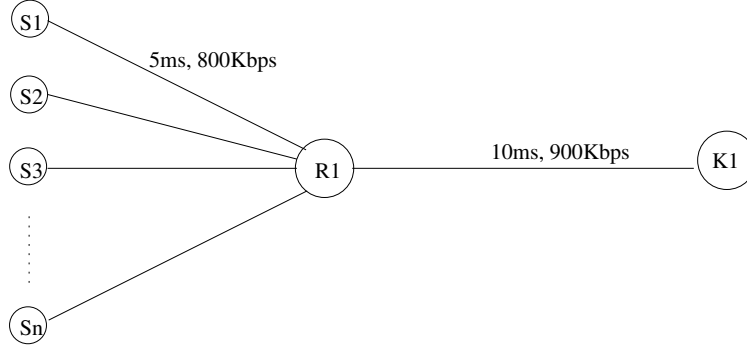


Figure 48: Topology I

6.5 *Experimental Results*

6.5.1 Implementation

We implement the proposed framework and prediction schemes in *ns2* [1, 34]. The predicted flow class inherits the TCP/FullTcp class and has the additional functionalities to switch, measure, predict and identify the changes between on and off periods. We also derive a new class of scheduler that handles all network changes. We perform experiments to primarily answer two categories of questions:

1. how much the predicted flow speeds up the simulation
2. how closely the predicted flow approximates the packet-based flow, in terms of:
 - average flow throughput
 - average network buffer occupancy
 - packet sequence traces of packet-based flows

It should be noted that this section is dedicated to presenting visible results of experiments regarding individual questions above. In Section 6.6, we will analyze these results systematically and summarize our suggestions on the usage of the prediction-based TCP simulation.

Throughout these experiments, flows all perform bulk transfers that are never application-limited, but the system's behavior with on-off flows is easily derived from our experiments with multiple flows that start/finish at different times. Both prediction techniques are

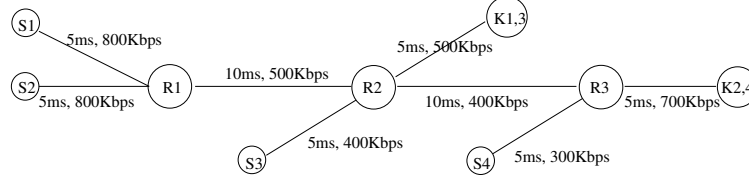


Figure 49: Topology II

tested and compared whenever they might potentially make differences. Most experiments use the network shown in Figure 48. The $\langle \text{propagation delay, bandwidth} \rangle$ on each link S_i-R_1 is $\langle 5\text{ms}, 800\text{Kbps} \rangle$ and the queue size on R_1 is 50 *packets*, unless otherwise noted. There is a TCP connection (flow) and an FTP application between each S_i and K_1 . The network in Figure 49 is used in some experiments. A measurement period of 180 seconds is used unless otherwise specified.

Note that our simulation experiments are relatively low bit-rates but we use standard packet sizes of 536 bytes. We will report our results in the rest of this chapter using “seconds” as our time units. Ultimately what is most important is the relative packet transmission time and not the absolute value of time. While using seconds simplifies the discussion, it should be noted that more realistic bit rates (in the 10’s of Mbps) would make our conclusions apply to much smaller time units in the 10’s or 100’s of milliseconds.

6.5.2 Speedup

From the description of the two prediction techniques, we know that the overhead of prediction is the same for both and they should achieve the same speedup. So we present only the results from the formula-based simulation.

For the results in Figure 50, we use 10 FTP connections from S_i to K_1 in each test, all transferring a file of the same size. Multiple tests are performed with different file sizes. The average throughput is within a 3% difference from a packet-based simulation for each test result presented. Figure 50 plots the relative runtime of a prediction-based simulation to a packet-based simulation, i.e., time used to run the prediction-based simulation normalized by the time used by the corresponding packet-based simulation, against the size of the TCP flow. The average speedup of all flows is presented along with an error bar for a confidence

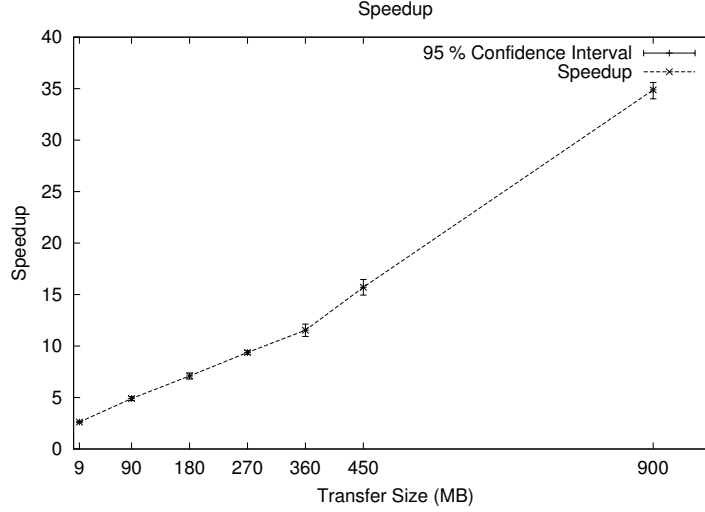


Figure 50: Speedup vs. transfer size

interval of 95%. Speedup with prediction-based simulation increases dramatically with the transfer size, since the same cost of initial measurement is amortized over a longer steady-state period in a large transfer. Although this explanation assumes no additional network changes in a longer transfer, it is conceivable that a larger transfer benefits more as long as the number of network changes does not grow proportionally to the transfer size.

We also find that the relative runtime of the prediction-based simulation corresponds well to the proportion of time it spends measuring, suggesting that the cost of transitions and running in steady-state is negligible.

6.5.3 Approximation

In this subsection, we evaluate how the prediction-based simulation approximates the packet-based simulation, in terms of average flow throughput, network buffer occupancy and packet sequence traces. Comparisons between formula prediction and direct measurement prediction are only necessary in the evaluation of average throughput approximation.

6.5.3.1 Average Throughput

For a predicted TCP flow, we are primarily interested in evaluating its average throughput behavior (or completion time given a transfer size). We want to examine approximations

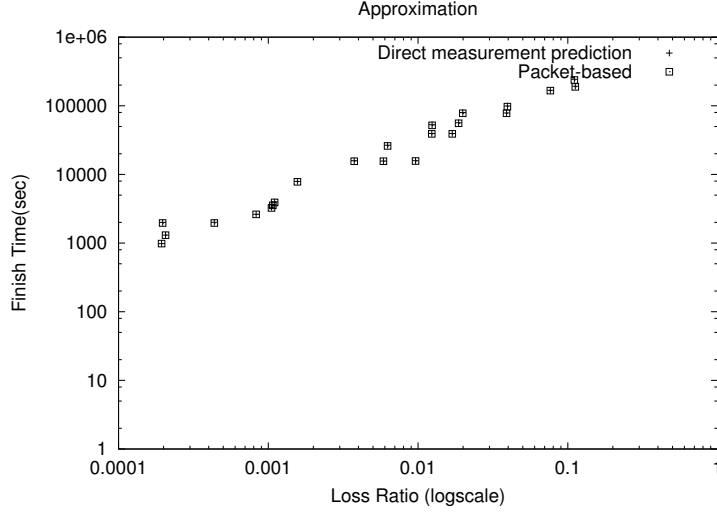


Figure 51: Approximation vs. loss (Direct Measurement)

for: 1) a wide range of RTT and loss ratio values; 2) different queuing disciplines; 3) a different number of flows; 4) situations where interacting flows exhibit path heterogeneity.

Impact of Loss Ratio and RTT In this experiment, we use an FTP connection from S_1 to K_1 in Figure 48. We choose different combinations of queue limit on R_1 and bottleneck (link R_1-K_1) bandwidth to get a wide range of loss ratios and RTTs.

Figure 51 plots the approximation of completion time against loss ratio, when the direct measurement prediction is used. Approximation remains consistently close for the range of loss ratios (up to 13%) we test. Although formula-based prediction also achieves good approximation for this range of loss ratios, it has a trend of increasing error (difference from packet-based simulation) as the loss ratio increases, as is evident in Figure 52. Notice that in Figure 52, we plot the mean prediction error directly, along with a 95% confidence interval. Those larger errors in the formula-based simulations are caused by the inaccuracy of the formula when losses are substantial, which echoes the results in [100]. In Section 6.7, we will relate the inaccuracy to an assumption made by the formula, as studied in [22].

A similar set of experiments exploring the possible relation between RTT and approximation show that for different RTTs but with the same range of loss ratios, the approximation does not change significantly for either prediction. However, with a *fixed* measurement

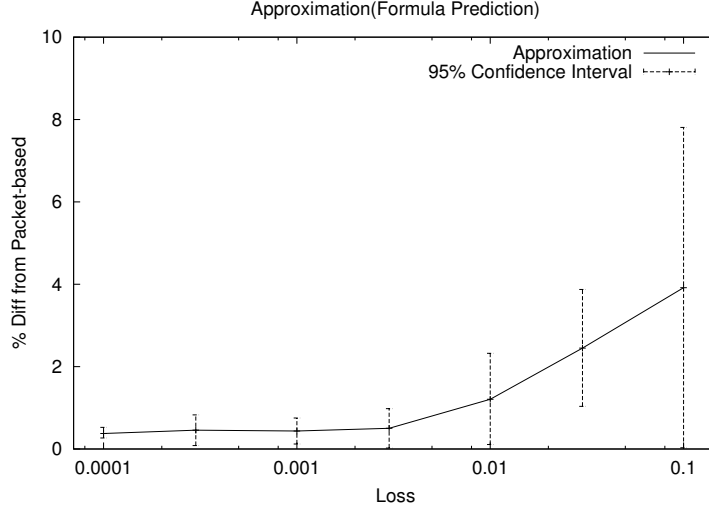


Figure 52: Approximation vs. loss (Formula)

Table 12: Approximation-RED vs. DropTail (% error)

Network Configuration		DropTail	RED
Bandwidth=900Kb Queue Size=50 Packets	1 Flow	0	0.4
	4 Flows	3.9	0.9
	5 Flows	4.3	0.9
	6 Flows	1.7	0.8
Bandwidth=9Mb Queue Size=100 Packets	1 Flow	0	0.3
	4 Flows	3.5	0.4
	5 Flows	1.7	0.2
	6 Flows	1.6	0.6

period we use in these experiments, there is a visible trend of increasing error with larger RTTs using the direct measurement prediction, which we will explain in Section 6.6.

Impact of Queuing Discipline Queuing disciplines interact with end-to-end congestion control and affect the way competing flows share network resources. Table 12 compares the errors of direct measurement prediction with DropTail vs. RED (Random Early Detection) queuing[48]. Using the network topology in Figure 48, we test different settings for the R_1-K_1 link bandwidth, router queuing discipline and queue capacity.

Results show that, all other parameters equal, better approximation is achieved in a RED network than in a DropTail network, except for the case when there is only one flow.

These results are explained by the fact that RED controls the variation in queue occupancy, as we will detail in Section 6.6. We note, however, that formula-based prediction does not closely approximate the packet-based simulation in a RED network. The TCP throughput formula we use assumes that packet losses within a TO round are *correlated*, which is justified only in the case of DropTail queuing [100]. With the same loss rate, a flow incurs much less timeout events with RED than with DropTail, therefore, the formula significantly underestimates the throughput, sometimes by about 30%.

Impact of the Number of Interacting Flows When multiple flows share a path, the increased interaction among flows and the less predictable loss ratio and RTT can stress the accuracy requirement of the simulation framework and the prediction techniques. For the results in Figure 53, we use the setting in Figure 48 with a different number of FTP applications in each test. Initially, we set the queue limit on R_1 to 30 *packets*. With the formula-based scheme, the average approximation remains consistently good (below 4%) until there are 6 flows, and a better approximation is achieved after we change the queue limit to 100 *packets*, which suggests that the increasing error might be caused by the increased loss ratio. Direct measurement does not show an increasing error with more flows, nor is its approximation improved with the increased queue size. These suggest again that there is no observable relationship between loss ratio and the approximation of the direct measurement prediction.

Bandwidth and Delay Heterogeneity The last two experiments in this category are intended to verify that the prediction scheme measures correctly for competing TCP flows that have different path characteristics.

Our first set of experiments show that the upstream bandwidth differences among competing flows do not affect the prediction accuracy discernibly. In the second set of experiments, we test competing flows with different RTTs and find noticeable discrepancies in some situations, while still obtaining a reasonable approximation in some other situations. These situations differ in their transfer sizes.

To explain these phenomena, we use the setup in Figure 49 with two FTP connections,

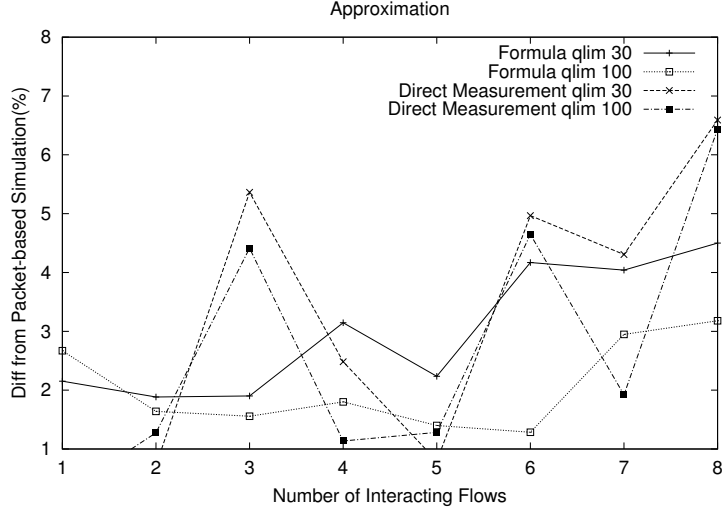


Figure 53: Approximation vs. number of flows

from S_1 and S_2 , respectively. In the first configuration, S_1 sends to K_1 and S_2 sends to K_2 ; in the second configuration, both S_1 and S_2 send to K_1 . The two FTPs have different RTTs in the first case and the same RTTs in the second. Using packet-based simulations, we find that it takes significantly longer for the average throughput to converge in the first configuration than in the second configuration.

Comparing the approximations for the two settings, in Figures 54 and 55 respectively, we observe that: 1) For *slowly converging* configurations like Figure 54, the measurement period required for the direct measurement prediction to obtain a reasonable approximation may be too long to provide significant speedup. Formula-based prediction could achieve a reasonable approximation with a short measurement period; 2) For *fast converging* settings like Figure 55, both predictions approximate well with a short measurement period. However, there are situations where the formula prediction will not approximate well regardless of the measurement period, which we will elaborate on in Sections 6.6 and 6.7.

6.5.3.2 Buffer Occupancy

Buffer occupancy is often an interesting performance metric to the network operators. To obtain a good approximation for this metric, one straightforward approach is to measure the average occupancy of a queue when the path it belongs to is in a measurement period and

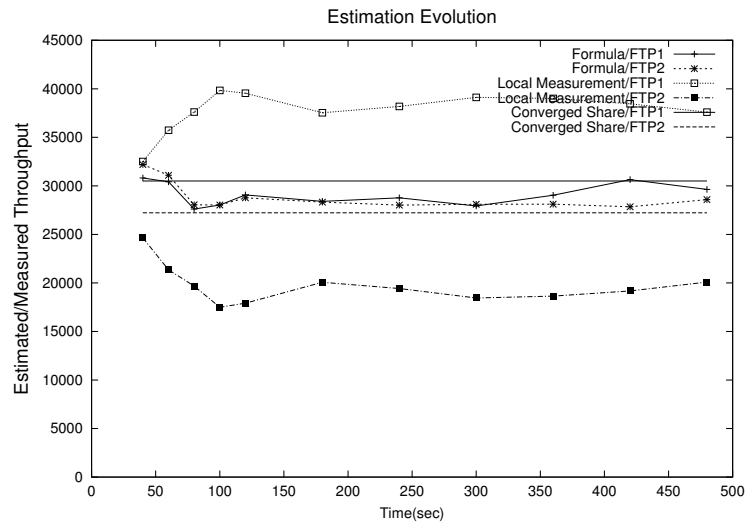


Figure 54: Throughput approximation (1)

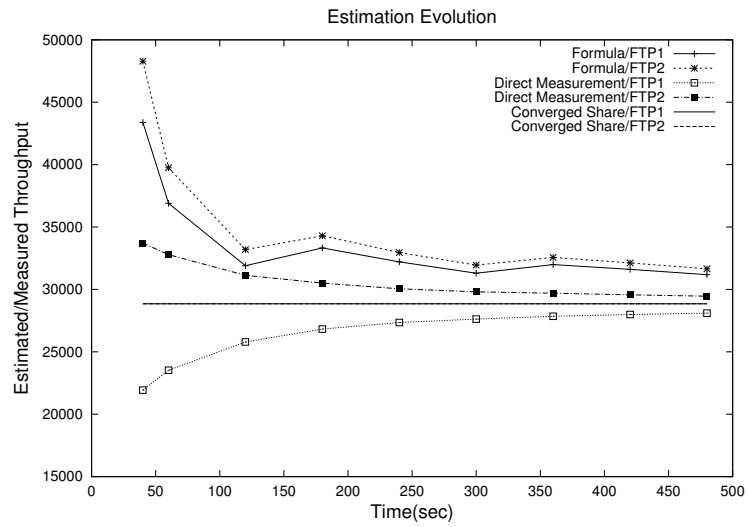


Figure 55: Throughput approximation (2)

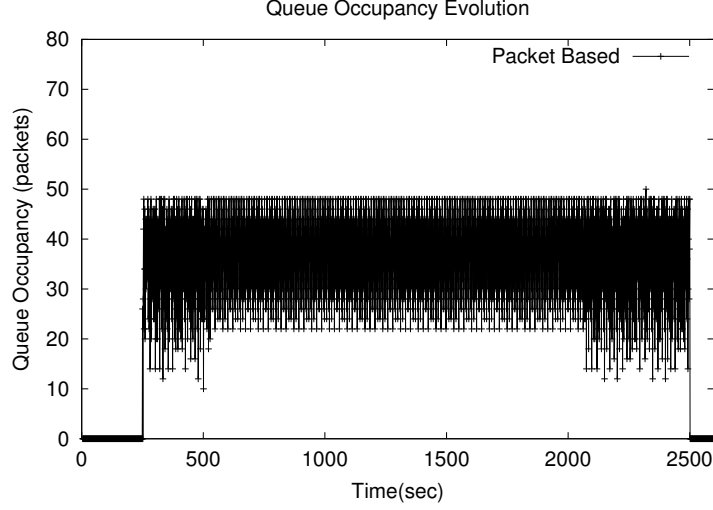


Figure 56: Queue occupancy–packet-based

take the measured average as the average buffer occupancy during the steady-state. The following experiment tests whether the period-wise averages match the measured average buffer occupancies in packet-based simulation: we use 3 FTP flows in Figure 48 and start each at intervals of 250 seconds. Figures 56 and 57 show the instantaneous queue length sampled every 500 changes. In the packet-based simulation, we can see several distinct periods where the buffer occupancy range is stable, which are the epochs in our simulation context. In the prediction-based simulation, a period with constant queue length and a period with fluctuating queue length correspond to the steady-state period and the transient period of an epoch. We can see that each epoch in Figure 56 matches a transient period followed by a steady-state period in Figure 57. The average buffer occupancies over the whole simulation are 30.88 and 29.49, respectively, with a difference of 4.5%.

6.5.3.3 *Mixing Packet-based and Predicted Flows*

Packet Sequence Traces Due to the statistical nature of our approach in determining the instantaneous state during the inter-epoch transitions, we cannot preserve the exact packet sequence. Our experiment is to validate that the average throughput of a packet-based flow which shares path with a predicted flow is reasonably close to what it experiences when the other flow is also simulated with packet-based technique. We use two connections

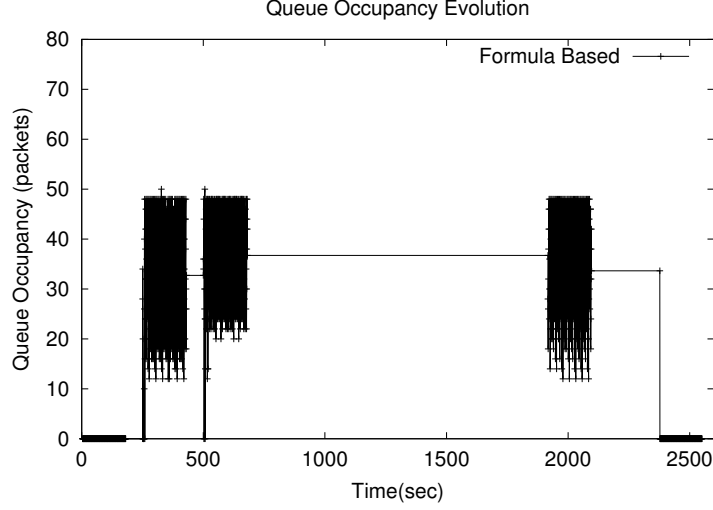


Figure 57: Queue occupancy–formula-based

in Figure 48. In the prediction-based experiment, FTP2 ($S_2 - K_1$) uses a packet-based TCP connection and FTP1 ($S_1 - K_1$) uses a predicted TCP connection. Both flows start at time $t=0$.

Figure 58 shows that the sequence number traces of FTP2 in the prediction-based simulation and in the packet-based simulation match closely.

Buffer Occupancy and Loss Rate Buffer occupancy and loss rate are good indicators of how we approximate the network after applying the technique that proportionally reduces the bandwidth.

Figure 59 shows the evolution of buffer occupancy before and after the predicted flow switches to steady-state at time $t=180$ in the experiment we describe above. Although buffer occupancy changes at a lower rate after the transition, the average queue occupancy remains similar. It is 27.1 and 31.8 respectively, before and after the transition.

Loss rates experienced by FTP2 are 0.39% and 0.35% before and after the transition, respectively.

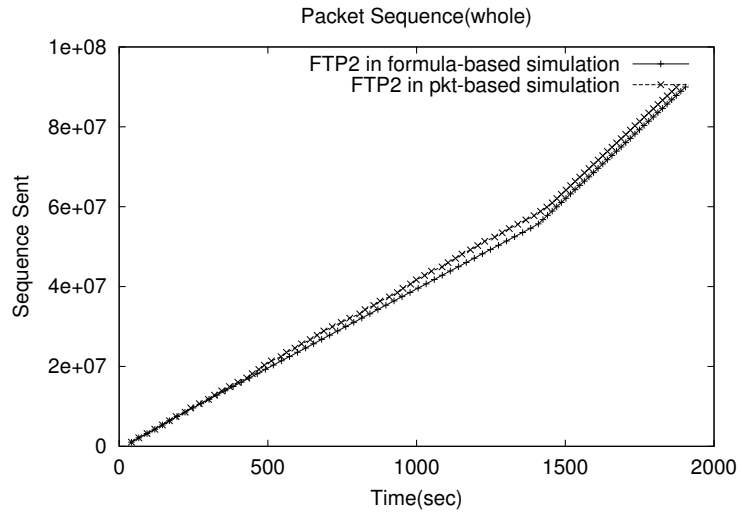


Figure 58: Packet sequence

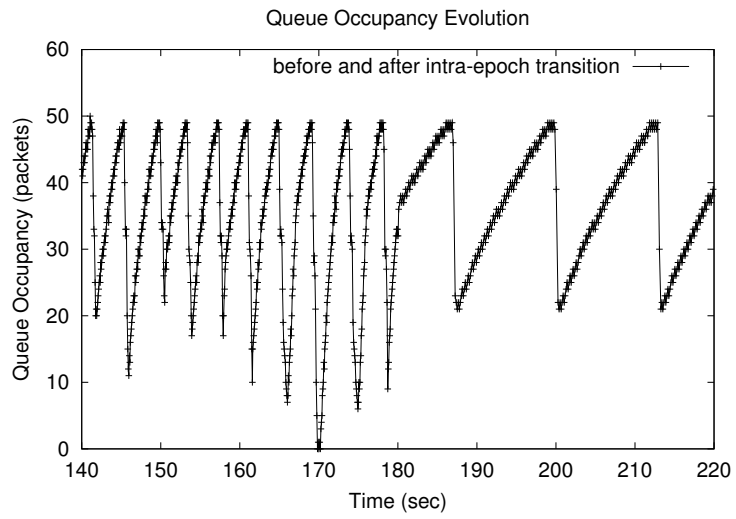


Figure 59: Queue occupancy with mixed simulations

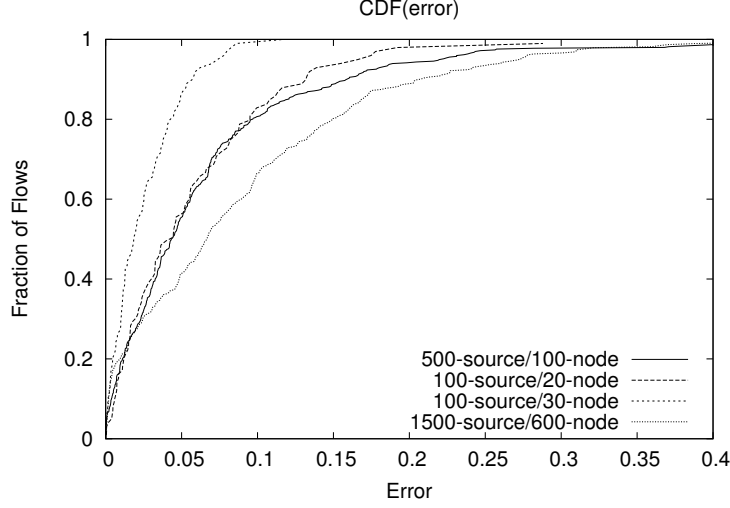


Figure 60: Prediction error vs. network configuration

6.5.4 Experience with Larger IP Networks

In this subsection, we will describe our experience of simulating larger IP networks using the prediction techniques.

We use Internet topologies that are generated with GT-ITM [36]. TCP flows are configured between random pairs of source/destination nodes and are started with a random interval which averages to 500 seconds. Flow transfer sizes follow a Pareto distribution where $\alpha = 1.2$ and the mean is 8M bytes.

6.5.4.1 Approximation

Figure 60 shows the CDF of the throughput prediction error using direct measurement prediction. The three setups have different numbers of nodes and/or TCP connections, as shown on the figure. We observe that in all the experiments, over 80% of the flows have an error under 10%.

Similarly, Figure 61 compares the approximations with different traffic mixes or queuing disciplines. All four experiments use the 100-connection/30-node setup and direct measurement prediction. In this figure, we use “pkt-based” to denote a TCP flow with packet-based simulation, “predicted” to denote a TCP flow with prediction-based simulation, and “CBR” to denote a Constant Bit Rate UDP connection. These results show that: *i*) on average,

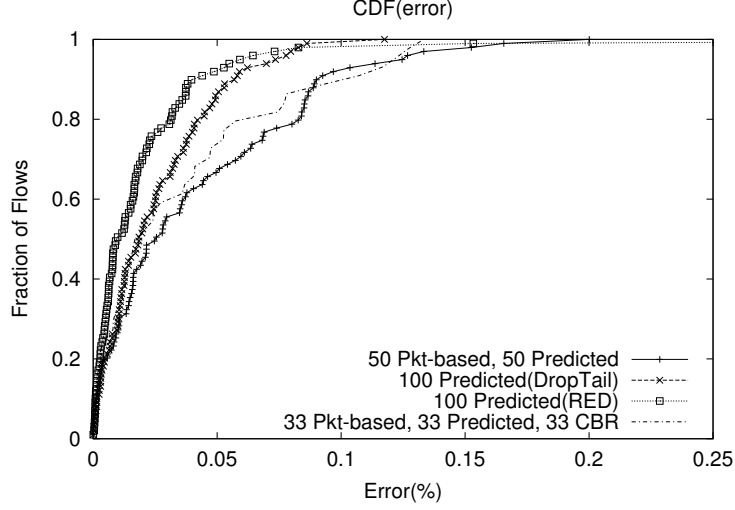


Figure 61: Prediction error vs. traffic mix

prediction achieves better approximation with RED than with DropTail; *ii*) although mixing packet-based and prediction-based simulation degrades the average approximation of predicted flows, over 90% of the flows have an error under 10%; *iii*) mixing UDP traffic does not have noticeable impact on the simulation accuracy.

In Figure 62, we plot the error of individual flows against their sizes⁵. Results show that the approximation has a large variation for flows with very small transfer sizes. In [52], the authors demonstrate that in the Internet environment, TCP throughput for *mice* (small transfers) tends to be highly variable while that for *elephants* (large transfers) is more stable. Back to our experiments, throughput of a small transfer may vary substantially from the prediction-based simulation to the packet-based simulation, thus contributing to the large variations of error.

6.5.4.2 Speedup

As the number of connections increases, the start/finish of flows at different time instances may result in a larger number of transient periods. Nevertheless, speedup is compensated for by savings from packet events of more flows. For large scale Internet simulations with

⁵The result is for one experiment but is representative of many tests, therefore the errors for larger transfer sizes are also representative despite the small number of large transfers.

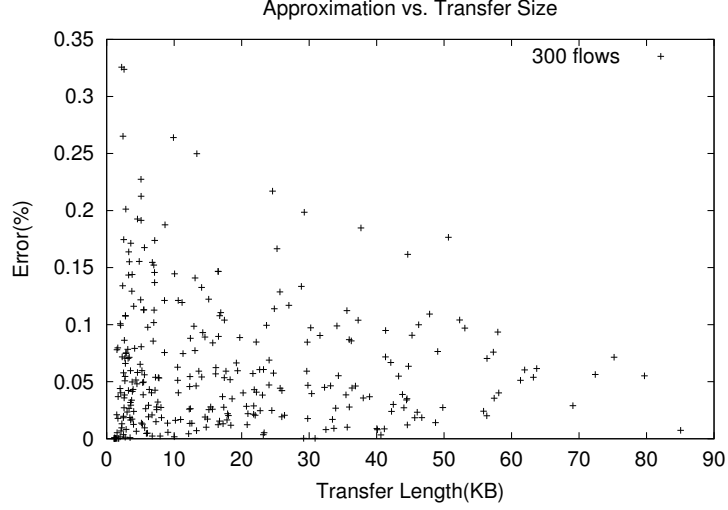


Figure 62: Approximation vs. transfer size

Table 13: Run Time Comparison

Experiment	Prediction	Packet-based
100-conn/30-node	1 hr 20 mins	4 hrs 10 mins
500-conn/100-node	3 hrs 24 mins	9 hrs 12 mins
1500-conn/600-node	9 hrs 29 mins	74 hrs 16 mins

complex flow arrival processes and flow size distributions, it is hard to foretell the speedup that could be offered by the prediction-based simulation. Table 13 lists the time taken by some of the larger scale experiments we perform on a Pentium 2GHz processor with 1GB memory.

We note, however, that on backbone links with a very high degree of multiplexing and rapid changes of active flows, more efficient traffic models and/or simulation techniques could be used. For example, we could associate a loss model with the link and only simulate flows of interest with the prediction technique. It should also be noted that the prediction-based technique does not decrease the memory requirement, as it runs exactly the same packet-based simulation for the measurements.

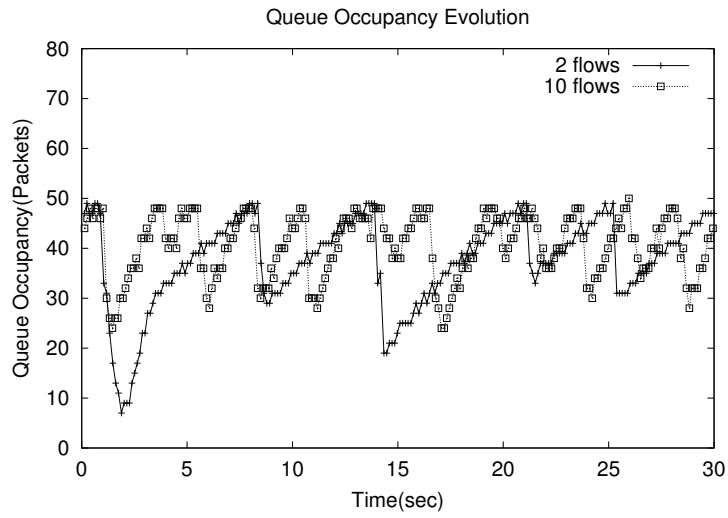


Figure 63: Queue occupancy vs. number of flows

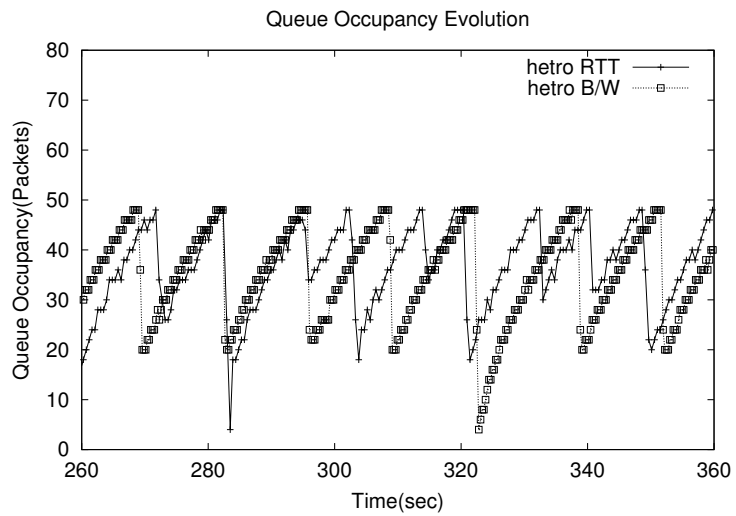


Figure 64: Queue occupancy (heterogeneity)

6.6 *Suggestions for Usage*

In the previous section, we have shown the impact of different system characteristics, such as loss probability, RTT heterogeneity, and queuing disciplines, on the approximation of our simulation technique. While maintaining that prediction is generally applicable to TCP simulation, the goal of this section is to present a systematic view of the causes of the inaccuracies, so that users can decide whether and how the prediction techniques should be used under a particular scenario.

Numerous demonstrations have been performed that show that the throughput of a bulk-transfer TCP flow is predictable given a steady-state path condition, i.e., the path state change can be modeled as a stationary process [22, 52]. On the other hand, in [147], the authors investigate the transient behavior of TCP-friendly congestion control protocols. Part of their conclusions is that TCP sending rate is subject to three types of variations in a stationary environment: smoothness, short-term fairness and long-term fairness. These variations could result in variations of bandwidth sharing over periods of hundreds of RTTs or longer. Their results suggest that the bandwidth we predict based on a flow's experience during the measurement period may differ from its average behavior in the long-run, and that a longer measurement period is required for some simulations than for others.

The analysis in [147] eventually relates the rate variations to the fluctuations of queue occupancy. In the rest of the section, we demonstrate that the buffer variation behavior in some experiments we present in the previous section is consistent with the degree of approximation. Figure 63 shows the evolution of buffer occupancy for the 2-flow experiment and the 10-flow experiment presented in Figure 53. Variation takes place on a smaller time scale in the 10-flow experiment, causing larger variations of TCP sending rate. Similarly, Figure 64 shows that in the experiment with flows of heterogeneous RTTs (Figure 54), buffer occupancy exhibits variation on a smaller time scale than in the case of heterogeneous bandwidth. As illustrated in Figure 65, RED routers keep the average and variation of queue occupancy low. Therefore, prediction could be more accurate for RED routers than for DropTail routers, unless the network is congested free, in which case the queue occupancy is always much lower than its capacity even with DropTail queuing.

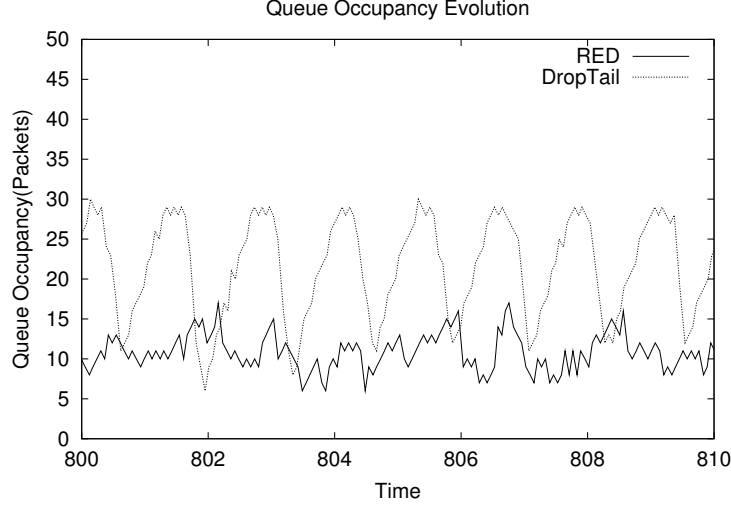


Figure 65: Queue occupancy–RED vs. DropTail

For TCP flows that have less smooth sending rates and a worse short-term fairness index⁶ [147], it takes longer (relative to an RTT) to converge to the steady-state sharing. When using the direct measurement prediction, a longer measurement period should be configured for those flows. Depending on the accuracy requirement of a simulation, a measurement period on the order of hundreds (or over a thousand) of RTTs may be necessary, which may then not provide a good speedup. Formula-based prediction suffers less from the transient behavior and converges faster, since it captures the experience of a path, rather than of individual flows. However, when the simulated environment becomes more complicated, formula-based prediction has the same problem as most fluid models, which we will discuss in Section 6.7. Formula-based prediction achieves similar approximation as the direct measurement prediction for the 100-connection experiments in Figure 60, but not for the 500-connection experiment, as shown in Figure 66. When competing flows are highly heterogeneous and their interactions complex, the inter-loss time observed on a router is far more complicated than the deterministic model assumed by the formula. We also observe that the inter-loss time of individual flows is more predictable than the inter-loss time of the aggregate as observed by routers. Therefore, if we change the formula prediction to

⁶Fairness index is defined as $\frac{(\sum_i x_i)^2}{K \sum_i x_i^2}$, where $x_i (1 \leq i \leq K)$ are the sending rates of K competing flows.

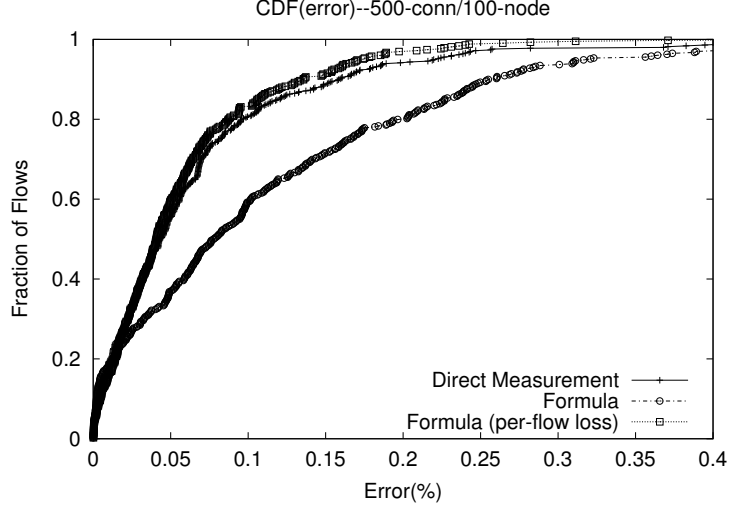


Figure 66: Formula vs. direct measurement

measure and apply the loss ratio of individual flows, we will achieve approximations close to that of the direct measurement prediction, as demonstrated by Figure 66. However, using per-flow loss ratio with the formula is subject to the slow convergence problem as the direct measurement prediction.

Based on the above discussions, we summarize our suggestions on the usage of the prediction techniques:

- *Choice of the prediction technique.* In simple setups, e.g., dumbbell topologies with reasonably provisioned router buffers, where the inter-loss times are approximately constant, formula-based prediction can achieve a good approximation with a shorter measurement period than required for the direct measurement prediction. In more complicated setups where the inter-loss times are far from being deterministic, formula-based prediction usually does not achieve the same degree of approximation as the direct measurement prediction does, and may not meet the accuracy requirement.
- *Length of the measurement period.* The length of measurement period required to obtain a reasonable approximation is dependent on the transient behavior of TCP under a particular path condition. Where there is a better short-term fairness index among

competing flows and the sending rate of individual flows are smoother⁷, e.g., with a small number of homogeneous flows, a shorter measurement is needed. Also note that direct measurement prediction is much more sensitive to the length of measurement period than the formula-based prediction.

Applicability The TCP simulation technique we propose in this work measures the *statistical* steady-state behavior of a flow (notice that our technique is designed to exclude from the measurement the transient TCP behavior resulting from changes of network conditions), and predicts the state of the flow at a future time instance. As such:

- It does not capture the packet-level details of TCP in steady-state. Those details do not affect the statistical behaviors in which we are interested. However, experiments interested in packet-level details should use packet-level simulation.
- In theory, it is not applicable to scenarios where the predictability does not hold. One such example is the mobile and wireless environment, where factors such as random losses and node mobility impair the predictability of TCP behavior. Until we can precisely model these factors, we should examine the significance of errors caused by those unpredictabilities before applying the prediction technique to TCP simulation. Nevertheless, since prediction-based TCP simulation can inter-operate with any technique based on packet-level simulation, for network simulations that integrate wireless and wired subnets, prediction can still be used to simulate flows on the wired subnets.
- Prediction errors are significant in some of the experiments shown in Section 6.5.4, primarily because the measurement period is not long enough to capture the average steady-state behavior. Although we can always improve the approximation by increasing the measurement period, when the network conditions change very frequently (one or more changes per hundreds of RTTs), the measurement period has to be relatively short for our simulation technique to achieve a good speedup. When it is possible to

⁷Smoothness and short-term fairness are shown to be positively correlated in [147].

identify relatively stable vs. dynamic paths in a simulation, prediction-based simulation and packet-based simulation could be applied to flows on these two categories of paths, respectively.

- Overall simulation speedup depends on the percentage of flows to which we are able to apply the prediction techniques, and how the rest of the traffic is simulated. For a large carrier network, e.g., it depends on how much of the traffic is TCP-based and how non-TCP traffic is simulated. It should be noted that the general concept of prediction could also be applied to the simulation of certain types of UDP flow, such as CBR. On the other hand, for large carriers, simulation of traffic at the level of individual sources is often unnecessary. It is likely more efficient to use Wide-Area traffic models, combined with certain simulation techniques, for the simulation of aggregate traffic[19].

6.7 Comparison with Fluid Flow Models

Fluid flow models have been proposed as computation efficient solutions to simulating large scale IP networks. With a fluid model approach, the network is modeled as fluid flowing through pipes, rather than individual packets flowing on communication links. Fluid model for TCP was originally studied on networks with AQM policies, where the packet losses seen by a flow are assumed to be independent [84]. In a more recent paper on fluid flow modeling of TCP/IP [22], the authors show that for a fluid model to accurately approximate the real TCP throughput in general contexts, it has to have a detailed characterization of the correlation and distribution of inter-loss times. They show that models that make simple assumptions about the loss process can be inaccurate on certain paths. The throughput formula we use for prediction, although derived from a packet model, makes the implicit assumption that the time between Timeout events is constant and therefore suffers the same kind of inaccuracy.

Furthermore, the authors of [22] show that their fluid model overestimates TCP throughput on certain paths, e.g., LAN connections, and the overestimation increases with the TCP throughput. Their results show errors up to about 80% from the real throughput for a LAN

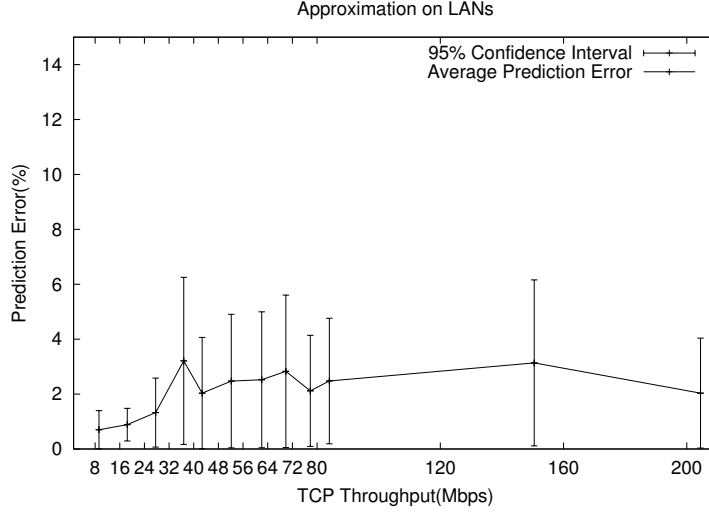


Figure 67: Approximation on LANs

connection. We construct a LAN topology where the links have delays of 1ms or 2ms and bandwidth values ranging from 10Mbps to 300Mbps, and simulate a number of TCP flows using the direct measurement prediction. Figure 67 shows the average flow prediction errors for a number of TCP throughput values, along with the 95% confidence intervals. Not only are the errors very small, but there is no trend of increasing error as the throughput increases.

Fluid flow models always assume an average behavior (mean-value) over certain time horizon called *step size*. All the second order statistics are ignored for their impact on stationary network behavior. In [84], the authors cite a situation where their fluid model is not accurate because of its failure to capture the correlation between TCP's window size and the perceived loss arrival. Figure 68 shows another example where the mean-value fluid model can diverge from the packet-level simulation. This is a case of bad long-term fairness among competing flows. In the experiment, we have a dumbbell topology with a bottleneck link in the center and 50 homogeneous links on each side. There is a TCP flow from each node on the left side to a node on the right side. All the flows have the same transfer length and are started simultaneously. To avoid synchronization [73], we add some small transfers to the bottleneck. The figure shows that the flows differ as much as 80% in the finish time/throughput, which is closely approximated by the direct measurement

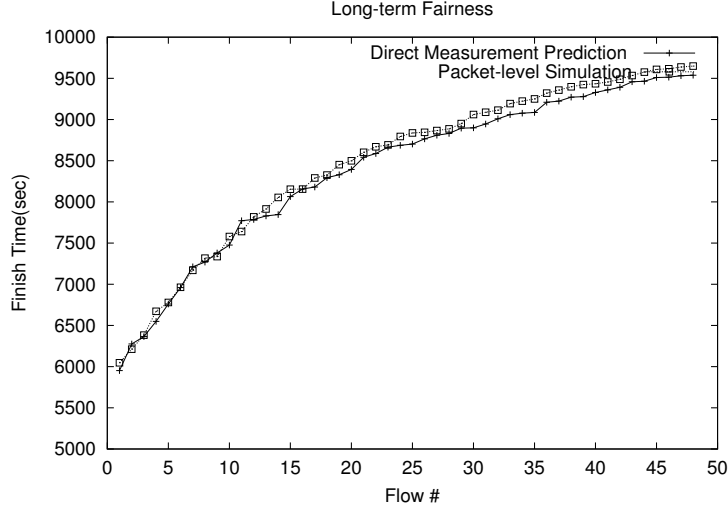


Figure 68: Capturing the transience

prediction but would not have been revealed with fluid models. Since fluid models rely on aggregation of flows and ignore actions specific to individual flows, all the flows will have the same throughput[84, 91].

We have wanted to have a more comprehensive comparison between the fluid models and our technique using the tool described in [91], however, we are not able to obtain the tool at this moment to perform comparative experiments.

6.8 Summary

This chapter proposes to speed up the simulation of TCP flows by predicting and aggregating packet events based on the predictability of steady-state TCP behavior. We design a prediction-based simulation framework to use the predicted flows and describe two techniques to predict the average throughput of a steady-state TCP flow, the direct measurement-based prediction and the TCP throughput formula-based prediction. Our results show that the prediction-based simulation achieves significant speedup over packet-based simulation and the speedup increases dramatically with the size of a transfer. Due to the assumption made about inter-loss times, the formula prediction is not very accurate under complex path conditions. Under simplex conditions, however, formula-based achieves

good approximation with a shorter measurement period than required of the direct measurement prediction, since direct measurement is more sensitive to the transient behavior of TCP congestion control. Compared to fluid model simulations, our prediction-based simulation is easier to be integrated with packet-level simulation, and is more generally applicable. Whereas the aggregation of events in a TCP flow simulation is based on the predictability of TCP steady-state, the idea of aggregating packet events of a flow with predictable behavior is applicable to the simulation of other predictable flows.

CHAPTER VII

OPTIMAL CONFIGURATION OF TCP TRAFFIC IN NETWORK SIMULATIONS UNDER LINK LOAD CONSTRAINTS

7.1 *Introduction*

Network conditions are complex functions of both the offered traffic and the physical configuration of the network, namely the topology, link capacities and router buffer sizes. Whereas models for both traffic and topology exist [30, 36] and are used for simulation, they may lead to unrealistic link conditions if the offered load is not tailored to the network configuration, as noted by some previous work[150]. Unfortunately, as we discuss in Chapters 1 and 2, existing traffic generation methods are in general isolated from the network configuration and lack considerations for the link conditions that will ensue.

In this work, we are concerned with generating traffic for network simulations such that the resulting conditions on certain links, which we call *target links*, are as specified by the simulation users. We propose a methodology for the optimal configuration of traffic under the specified link load constraints. The algorithm takes as input a detailed network configuration and the target link conditions. It then configures the traffic intensity between each pair of source and destination nodes, such that all target link conditions are achieved at the same time.

The traffic generation methodology we propose in this work is for closed-loop network simulations. In particular, it uses a mixture of persistent TCP flows and ON/OFF UDP flows, which is the traffic model many network studies use [27, 153]. The UDP flows generate a small fraction of the traffic (typically less than 10%) and they are intended to simulate applications such as DNS and streaming audio/video. Our goal is then to configure the number of TCP flows between each pair of nodes, such that the user-specified conditions on

all target links are attained. We model the traffic generation problem as a linear program, where each variable represents the number of TCP flows from a particular source to a particular destination, with constraints on the target link conditions. While obtaining the target link conditions, the algorithm generates traffic in such a way that the average flow hop count is as specified, and the memory requirement of the simulation, determined mostly by the number of TCP flows (with a given network topology), is minimized. Our algorithm exploits two basic facts: (1) the throughput of a TCP flow between any two nodes can be derived based on the load and loss rate of the links on its path, in addition to the maximum congestion window of the TCP flow itself [100]; (2) the load on a link, in turn, is the sum of the throughput of all flows traversing it.

The problem stated above is related to the traffic matrix estimation algorithm in network research [90, 156]. The challenge, objectives and context are nonetheless very different. For instance, for the traffic matrix estimation problem, the challenge is in defining the best optimization objective for the linear program. In our context, the challenge resides in relating the conditions of a link to the numbers of TCP flows traversing it between different pairs of source and destination.

As described above, our methodology can be conceived as inferring TCP flow configurations based on the specified link conditions. In that respect, it is also related to the work reported in RAMP [80]. That work recovers path and flow information from packet traces and obtains a full traffic profile including the traffic mix, flow size distribution and arrival rate, which can then be used to populate a network simulation. Clearly, the goal of RAMP is not to configure traffic to match user-specified link conditions, far less to do so for a network of links.

The contributions of this work can be summarized as follows:

- We develop a model to configure traffic for closed-loop network simulations such that specific link conditions are attained on certain links. In doing that, we also maintain an average hop count in the generated traffic, and minimize the number of required TCP flows so as to reduce the memory requirement of the simulation.

- We evaluate and analyze the accuracy of the algorithm and provide insight on the causes of the major errors.

The rest of the chapter is organized as follows. In Section 2, we specify the traffic generation problem we address in this chapter. Section 3 presents a linear program formulation of the problem. Section 4 evaluates the accuracy of the proposed traffic generation scheme. We discuss future work and conclude the chapter in Section 5.

7.2 Problem Statement

In this section, we describe in detail the traffic generation problem that we consider in this work.

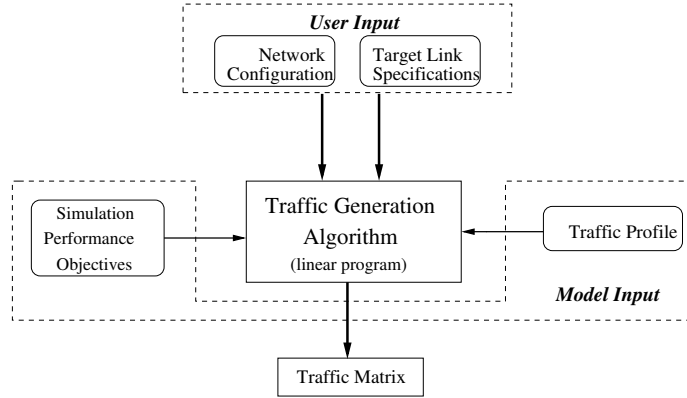


Figure 69: Traffic generation context

7.2.1 Overview of the Traffic Generation Context

To formally define the traffic generation problem, we need to precisely state its context with all the relevant components. The context we propose in this work is outlined in Figure 69. At the heart of this context is a traffic generation algorithm that we will elaborate on in Section 7.3. We distinguish two types of inputs to this algorithm:

- (1) *User inputs* are parameters that concern the simulation user and, in general, they are independent of the traffic generation process. One of the user inputs is the network configuration, and another input is a list of target links in the network with specified link

conditions. They are both required because we aim to match the generated traffic to the network configuration so as to obtain the specified link conditions on the target links.

(2) *Model inputs* are parameters we add to influence certain aspects of the traffic generation process. First, a traffic profile is introduced to add some realism in the generated traffic, and it could reflect characteristics such as the mixture of different types of traffic. Second, we specify a particular aspect of the simulation performance to optimize. One feature of our traffic generation scheme is to optimize the generated traffic to benefit the simulation performance. For instance, if memory cost is the limiting factor to run a simulation, we can minimize the number of objects created for the simulation; if simulation speed is the primary concern, we can minimize the total packet hop count, hence the number of events per simulated unit of time; for parallel and distributed simulation, the objective could be to minimize the cross-partition traffic or to optimally balance the load across partitions.

Collectively, the model inputs and the traffic generation algorithm comprise the scheme we use to generate traffic. The most distinctive facet of this scheme is that it takes as inputs the user specified network configuration and target link conditions. The output of the traffic generation algorithm specifies how many TCP flows to configure between each pair of nodes in the network so as to obtain the user specified link conditions.

7.2.2 Traffic Generation Context Specification

The specific traffic generation context we use in this work can be described as follows:

- **Input 1: Network configuration.** The following parameters of a network are provided as inputs:

(a) Topology and routing. The network consists of V nodes and N links. Network routing is fully specified by the following function:

$$R_{ij}(k) = \begin{cases} 1 & \text{if link } k \text{ is in the path from node } i \text{ to } j \quad (1 \leq i, j \leq V, \quad 1 \leq k \leq N) \\ 0 & \text{otherwise} \end{cases}$$

(b) Detailed configuration of the links. Each link L_k ($1 \leq k \leq N$) is described by a 5-tuple $(S_k, D_k, C_k, B_k, T_k)$, representing source node, destination node, link capacity,

router buffer size (in packets) and propagation delay, respectively. Notice that the links are *directional*.

- **Input 2: Target link specifications.** This input specifies a list of N_g ($N_g \leq N$) target links. The target link conditions we consider include the utilization, *average* loss rate and *maximum* queuing delay. We note that maximum queuing delay on a link is determined to a large degree by the corresponding buffer size B [45]. Therefore, if the user specifies a maximum queuing delay, our scheme will adjust the buffer size accordingly. The delay model (16) in Section 7.3 will show how these two entities are related. For the traffic generation scheme we introduce below, we assume that either this step has already been conducted or the user does not specify any queuing delay requirement, and we focus on the average loss rate and utilization.

In addition to the 5-tuple properties specified for every link, each target link has two additional characteristics: target utilization ρ (in $[0, 1]$) and target loss rate p (in $[0, 1]$). We assume that $\rho < 1$ prescribes $p = 0$, and that $p > 0$ prescribes $\rho = 1$, i.e., lossy links are fully utilized and underutilized links are lossless. These are legitimate assumptions when the router buffers are sufficiently configured [27, 45]¹. There are no specific link conditions for the rest of the links, which we call non-target links, but we make it an implicit specification that the load on each of them is lower than its capacity. This prevents non-target links from limiting the traffic passing through target links.

- **Input 3: Traffic profile.** The traffic profile characterizes the traffic from the following aspects:
 - (a) Traffic mix. The traffic on any link consists of a number of persistent TCP flows and Pareto ON/OFF UDP streams. UDP traffic accounts for a small fraction u (typically less than 10%) of the total load on every link, and it simulates the traffic of applications such as DNS and audio/video streaming. In addition, the presence of

¹A common rule is to configure a router buffer at least twice as large as the bandwidth delay product of the flows with the largest RTT.

UDP traffic eliminates the synchronization among persistent TCP flows, usually an artifact of TCP simulations [74]. This traffic mix does not retain traffic characteristics such as long range dependence. It does, however, represent the most common practice in network simulations [27, 153], because it is sufficient to capture the queue dynamics on shorter time scales and the congestion responsiveness of TCP traffic.

(b) TCP flow characteristics. A flow from node i to j is characterized by the 2-tuple (H_{ij}, W_{ij}) , where H_{ij} is the path hop count determined by routing $\{R_{ij}(k)\}$. W_{ij} is the maximum TCP congestion window configured on a per source/destination pair basis. When the path from i to j is lossless, W_{ij} follows a given distribution; when the path is lossy, W_{ij} is large enough for the flow to be congestion-limited on it. This latter assumption about the configuration needs more explanations, which we defer to the end of Section 7.3.3. Finally, we assume that TCP data segments and UDP packets of all flows have the same size S .

(c) Traffic between any pair of nodes is allowed, subject to the constraint that the average hop count among all TCP flows is H_t .

- **Input 4: Simulation performance objective.** In this work, we minimize the simulation memory requirement. This is motivated by the fact that the large memory requirement of network simulations often preempts the execution of even medium-scale network simulations.
- **Output: a traffic matrix** $\{A\}_{V \times V}$ where A_{ij} represents the number of TCP flows from node i to j . The user will be able to attain the link conditions he/she specifies for the target links, as long as he/she configures A_{ij} TCP flows from node i to node j , for all pairs of i and j .

Problem Statement. With all the components defined above, the traffic generation problem we aim to solve can be stated as follows:

Given a network with V nodes, N edges and specified routing $\{R_{ij}(k)\}$, and a set of N_g target links with specified link utilization and loss rate, generate a matrix of TCP flows

$\{A\}_{V \times V}$ such that 1) the link conditions on all target links are obtained, 2) the average hop count across all flows is as specified by H_t , and that 3) the simulation memory requirement is minimized.

For quick reference, Table 14 lists some notations that we use in the rest of the chapter.

Table 14: Terms and Notations

V : number of nodes	N : number of links	$R_{ij}(k)$: whether link k is in the path from i to j
C : link capacity	T : propagation delay of a link	U_{ij} : throughput of a TCP flow from i to j
B : router buffer (in packets)	$\hat{T}$: RTT of a flow	A_{ij} : number of TCP flows from i to j to be solved for
p : loss rate of a target link	T_q : queuing delay on a lossy link	H_{ij} : hop count of the path from i to j
S : TCP data segment size	ρ : utilization of a target link	u : fraction of UDP traffic on each link
H_t : average flow hop count	b : segments per ACK in TCP	W : TCP maximum congestion window (in packets)

7.3 Traffic Generation Algorithm

In this section, we develop algorithms to solve the traffic generation problem stated in Section 7.2. We start with a simple scenario to illustrate the basic insight on which we build our algorithm, and follow with a full formulation of the problem.

7.3.1 Single Link Example

The traffic generation algorithm we propose in this work is based on two insights: (i) the throughput of a TCP flow can be derived based on the load conditions of its path, in addition to some characteristics, such as maximum congestion window, of the flow itself [100]; (ii) the load of a link, in turn, is the sum of the throughput of all flows traversing the link. We call the second property *traffic balance*. In this subsection, we use a single-link example to demonstrate how we can use these insights to establish the relation between the utilization/loss rate of a link and the number of TCP flows traversing it. This relation enables us to solve for the traffic configuration that will result in the specified utilization/loss rate on the target link, and it serves as a basis for the traffic generation for a network of target links, discussed in Section 7.3.2.

The sample target link is characterized by $(S_0, D_0, C, B, T, \rho, p)$. All flows traversing it are from S_0 to D_0 and have the same maximum congestion window W . The traffic

generation problem in this context is simply to obtain the number of flows from S_0 to D_0 that we need to reach the target ρ or p .

TCP Throughput Model: One of the key premises of our algorithm is that we are able to derive TCP throughput based on certain path conditions. When the path taken by a flow is lossless, the throughput U of the flow is limited by its maximum congestion window W as in:

$$U = \frac{WS}{\hat{T}}, \quad (13)$$

where S is the TCP segment size and $\hat{T}$ is the RTT of the flow.

There are numerous throughput models for a lossy path [28, 89, 100]. The model by Padhye et al. [100] is one of the most cited because it is of closed-form and the required parameters are relatively easy to obtain. That model describes TCP throughput as:

$$U = \min\left\{\frac{S}{\hat{T}\sqrt{\frac{2bp}{3}} + T_0 \min(1, \sqrt{\frac{3bp}{8}})p(1 + 32p^2)}, \frac{WS}{\hat{T}}\right\}, \quad (14)$$

where T_0 is the TCP retransmission timeout period and b is the number of new segments released per new ACK.

Target Utilization: We first consider the flow configuration problem when the target link is lossless and has a target utilization $\rho < 1$. In this case, the TCP flows that constitute its load are window-limited. Notice that the RTT of a flow in this uncongested link case is simply twice the one way link propagation delay T . Therefore, the throughput U of each flow can be derived as:

$$U = \frac{WS}{2T}$$

The number of TCP flows, A , that are needed to reach the target utilization ρ should satisfy the following traffic balance:

$$AU = A\frac{WS}{2T} = C\rho \quad (15)$$

We can then solve (15) for A .

Target Loss Rate: If the target link is lossy with a specified loss rate p , the throughput of a flow traversing it is expressed by (14). The RTT in this case, should include the queuing delay T_q on the congested link, which can be approximated by $\frac{BS}{C}$, because congestion-limited TCP flows tend to fill the buffer of their bottleneck links [45]. The RTT $\hat{T}$ is then modeled as:

$$\hat{T} = 2T + T_q = 2T + \frac{BS}{C} \quad (16)$$

As long as the buffer is sufficiently provisioned, which is the case for most routers on the Internet, a link will be lossy only when it is saturated [27, 45]. In other words, a lossy link can be assumed to be fully utilized. Therefore, the traffic balance equation for a lossy link is as follows:

$$AU = C \quad (17)$$

From (14), (16) and (17), we can solve for the number of flows A to obtain the target loss rate p .

7.3.2 Problem Formulation

Using traffic balance equations similar to those in the previous subsection, we now formulate the problem stated in Section 7.2 as a linear program that consists of the following elements:

Traffic Balance Constraints: A traffic balance constraint is needed for each link to relate the load of the link to the throughput of the flows traversing it. The traffic constraints for different links comprise a system of N_g equations (each corresponding to a target link) and $(N - N_g)$ inequalities (each corresponding to a non-target link) in the following form:

$$\begin{cases} \sum_{i,j} A_{ij} U_{ij} R_{ij}(k) = (1 - u) \rho \cdot C_k & \text{if } k \text{ is a target link } (1 < k < N) \\ \sum_{i,j} A_{ij} U_{ij} R_{ij}(k) < (1 - u) \cdot C_k & \text{if } k \text{ is a non-target link } (1 < k < N) \\ A_{ij|i=j} = 0 \\ A_{ij|i \neq j} \geq 0 \end{cases} \quad (1 \leq i, j \leq V) \quad (18)$$

Note that U_{ij} is the throughput of a TCP flow from node i to j . In principle, the variables A_{ij} , representing numbers of TCP flows, should be integers. Modeling the problem as an integer program, however, could make the problem computationally expensive or infeasible to solve for large-scale networks. Therefore we use a linear program as an approximation. We believe that it is not of interest to most simulation experiment designers to attain the link conditions exactly as specified; instead, a value within a relatively narrow range should be acceptable. We will show in Section 7.4 that the resulting errors are noticeable only to certain links and that they are mostly acceptable.

Simulation Performance Optimization Objective: With a given network topology, the memory cost of a simulation using tools such as *ns2* [34] depends mostly on the number TCP flow objects. Therefore, our objective to minimize simulation memory requirement is translated into minimizing the total number of TCP flows. With the addition of the following optimization objective:

$$\text{Minimize} \quad \sum_{i,j} A_{ij}, \quad (19)$$

we now have a linear optimization program that could render a unique solution:

Additional Constraint on the Average Flow Hop Count: The linear program defined so far by (18) and (19) has no bearing on such flow characteristics as the flow hop count. To preserve a reasonable level of realism in the generated traffic without complicating the problem formulation too much, we add an average flow hop count constraint to the existing linear program. This is expressed as follows:

$$\frac{\sum_{i,j} A_{ij} H_{ij}}{\sum_{i,j} A_{ij}} = H_t, \quad i.e., \quad \sum_{i,j} A_{ij} (H_{ij} - H_t) = 0 \quad (20)$$

Along with (18) and (19), Equation (20) defines the following linear program for the traffic generation in the context we lay out in Section 7.2:

$$\text{Minimize} \quad \sum_{i,j} A_{ij}$$

Subject to:

$$\left\{ \begin{array}{ll} \sum_{i,j} A_{ij} U_{ij} R_{ij}(k) = (1-u) \rho \cdot C_k & (1 < k < N_g) \quad \text{if } k \text{ is a target link} \\ \sum_{i,j} A_{ij} U_{ij} R_{ij}(k) < (1-u) \cdot C_k & (1 < k < N - N_g) \quad \text{if } k \text{ is a non-target link} \\ A_{ij|i=j} = 0 & (1 \leq i, j \leq V) \\ A_{ij|i \neq j} \geq 0 & \\ \sum_{i,j} A_{ij} (H_{ij} - H_t) = 0 & \end{array} \right. \quad (21)$$

7.3.3 Populating the Linear Program

In this subsection, we discuss some issues related to obtaining accurate estimates of the required parameters, mainly $\{U_{ij}\}$, to populate the linear program (21).

Throughput Matrix $\{U_{ij}\}$: To calculate U_{ij} , we need to have accurate estimates of the loss rate and RTT experienced by a flow from i to j . This, however, is not as straightforward as in the single link configuration. The complication is largely due to the presence of congested links, on both the forward and the reverse directions.

- *RTT*. The RTT of a TCP flow should include the queuing delays, if any, both in the forward direction (incurred by data packets) and in the reverse direction (incurred by ACKs). The question is how to obtain an accurate estimate of the queuing delay on a congested link in the context of a network of links.

While the “rule-of-thumb” estimate $\frac{BS}{C}$ is a good approximation of the queuing delay T_q on a congested link when there is no or little TCP data traffic on the reverse link, we found that T_q could be noticeably affected when there is a significant amount of TCP data traffic on the reverse link. Consider a congested link L ($S_0, D_0, C, B, T, \rho_f=1, p > 0$), and suppose the utilization in its reverse direction is ρ_r . We found that T_q can be modeled as follows:

$$T_q = \frac{(r(S + S_{hdr}) + (1-r)S_{ack}) \cdot B \cdot \theta}{C} \quad (22)$$

$$r = \frac{\rho_f}{\rho_f + \frac{\rho_r}{b}} \quad (23)$$

where S_{hdr} and S_{ack} are the sizes of the TCP/IP header and of the TCP ACKs, respectively. The reason to use (22) instead of $T_q = \frac{BS}{C}$ is that there are a significant fraction of packets in the queue of L that do not have a size of S and thus do not contribute a queuing delay of $\frac{S}{C}$. Those are the ACKs for the TCP data traffic in the reverse direction. Given that every b data packets result in one ACK, and that the ratio of TCP packet count on L vs. that on its reverse link is approximated by $\frac{\rho_f}{\rho_r}$, r in (23) approximates the fraction of TCP data packets among all packets on L . Therefore, $r(S + S_{hdr}) + (1 - r)S_{ack}$ approximates the average packet size on L ². In Section 7.4.3, we will demonstrate the accuracy of the queuing delay model using simulations. The term θ in (22) is intended to account for the fact that the queue is not always full. We use $\theta = 0.8$ in the rest of the chapter, although any other value between 0.75 and 1 seems to work similarly well.

For the model with a fraction u of UDP traffic, we can simply replace (23) by the following:

$$r = \frac{\rho_f}{\rho_f + \frac{\rho_r(1-u)}{b}},$$

assuming that the UDP packet size is also S .

- *Loss rate.* We observed that TCP is much less sensitive to ACK losses than to data packet losses, which is consistent with the findings by some previous work [79]. Therefore, we only consider the loss rate for data packets on links in the forward direction. We use an optimistic calculation of the loss rate when multiple bottlenecks exist on a path [126], namely, we take the largest loss rate among the bottlenecks as the loss rate for a flow.

Average Hop Count H_t : Occasionally, we may need to adjust the parameters in the traffic profile based on the user inputs. One such parameter is the target average hop count H_t , for which an improper value could make the linear program (21) infeasible. For instance, if H_t is larger than the length of the longest path in the network, there is no way

²On real-world routers, buffer size is often measured and allocated in terms of bytes. Yet, all the network simulation tools we are aware of allocate buffers in terms of packets.

we can configure the flows such that the average flow hop count is H_t . H_t can be properly configured, e.g., based on the average path length in the network.

Maximum Congestion Window for Flows on Lossy Paths: The second such parameter is the maximum TCP congestion window. With a window configuration that is independent of the user input, not all solutions to the linear program (21) will produce losses as desired of lossy target links. In particular, a solution may consist exclusively of window-limited flows, which do not cause losses yet the aggregate throughput does reach the capacity of a target lossy link. To ensure that the link is in fact lossy, we need to impose additional constraints on the maximum congestion window configuration in the traffic model such that $W > \sqrt{\frac{8}{3bp}}$ (flow is congestion-limited) [100] is satisfied for flows on lossy paths. This has been incorporated in the traffic profile description in Section 7.2.

7.4 *Evaluation*

In this section, we will evaluate the proposed traffic generation algorithm. We refer to the proximity between the yielded link conditions (conditions that result from the generated traffic) and the target link conditions as the *accuracy* of the traffic generation algorithm for a particular set of inputs. Of primary concern is to evaluate the accuracy and to examine the factors that cause significant inaccuracies.

The evaluation of the algorithm for each set of user inputs consists of the following steps:

1. Instantiate a linear program based on the user inputs and the traffic profile, as described in Section 7.3.
2. Solve the linear program for the traffic matrix, i.e., the number of TCP flows to be configured between each pair of nodes, and round the numbers to integers.
3. Set up an *ns2* [34] simulation based on all the inputs; configure TCP flows (TCP Reno, *FullTcp* in *ns2*) according to the traffic matrix (after rounding); configure a Pareto ON/OFF UDP stream on each target link with an average rate $u\rho C$.
4. Run the simulation; for each target link, compare the average utilization and/or loss rate observed in the simulation against the target values. Suppose X_o and X_t are,

respectively, the observed and target values of a link condition, we derive the *relative error* E for this link condition as follows:

$$E = \frac{X_o - X_t}{X_t}$$

In addition, we use the *relative error due to rounding* to assess the error incurred when rounding the traffic matrix solution to integers. More specifically, suppose that the load on a target link calculated from the original solution is L_t , which by the definition of the linear program should be the same as the target load, and that the load calculated after rounding is L_i , then the relative error E_r due to rounding is:

$$E_r = \frac{L_i - L_t}{L_t}$$

We also use the absolute values of E and E_r extensively when we are only interested in the *magnitude* of an error.

In the rest of the section, we first show the accuracy in single link examples, and then evaluate the accuracy for a network of links. We also reveal the causes of the largest errors.

7.4.1 Single Link Example

For the experiments in this subsection, we use a single link L (C=100Mbps, B=400, T=30ms). All TCP flows have a maximum congestion window $W=25$.

The results in Figure 70 show the utilization constraint errors when the target link is lossless. The X-axis represents the target load ($C\rho$) and the left Y-axis represents the observed load. Both relative errors E and E_r are shown (using the right Y-axis) and they almost completely overlap. This indicates that the error in this case can be unilaterally attributed to rounding. Clearly, the error caused by rounding tapers off as the target utilization increases, since the difference is normalized by the target utilization.

Similarly, Figure 71 shows the loss rate approximation in experiments where the target link is lossy. We observe that the error for loss rate is much larger and it does not correspond to rounding error, although all the errors are within the range $[-0.2, 0.7]$. When the target loss rate is very low, the observed loss rate tends to be higher than the target value. When

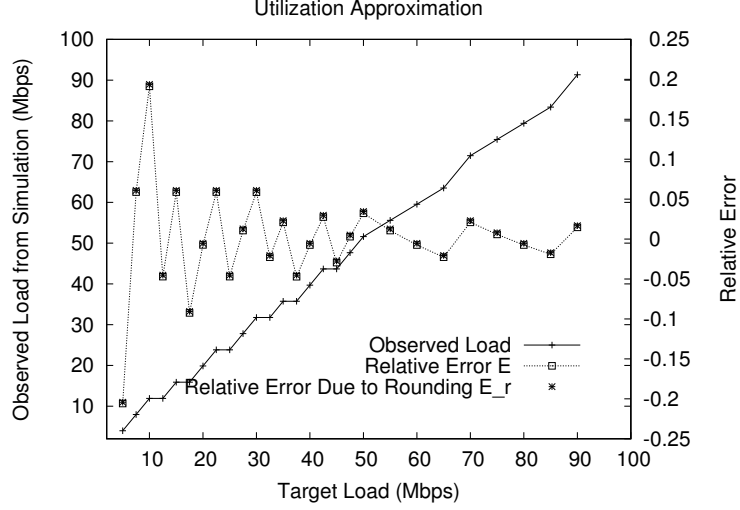


Figure 70: Utilization approximation

the congestion is low, the queue is on average far from being full. As a result, the term $\frac{BS}{C}$ overestimates the average queuing delay, hence the algorithm underestimates the TCP throughput and overestimates the number of TCP flows that are needed to cause a specific loss rate. This results in a higher loss rate than specified. On the other hand, when the target loss rate approaches 0.1, the observed loss rate becomes lower than the target value, which suggests that we may have overestimated the throughput and underestimated the number of flows needed to reach the target loss rate. This overestimation of throughput at high loss rate is inherent to the throughput model we use [100]. In the real Internet, a loss rate exceeding 0.05 is highly unusual.

In summary, the results of this subsection showed that the traffic generation algorithm for a single link is affected by two major factors: (1) accuracy of the throughput estimation, which is better for lossless links than for lossy links, and (2) rounding fractions to integers, which affects the accuracy most significantly when the number of flows required on a link (given by the solution) is small.

7.4.2 Network of Links

Experiments in this subsection are intended to examine the accuracy of the traffic generation algorithm for configurations with a network of links.

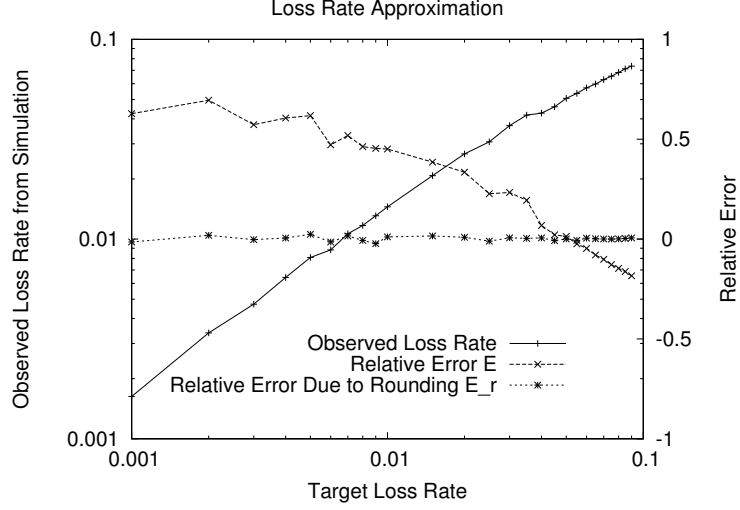


Figure 71: Loss rate approximation

7.4.2.1 Experimental Setup

Unless otherwise noted, the test inputs for our evaluation are described as follows:

- We have experimented with different types and scales of network topologies, and the results we draw are quite consistent. We will present results here based on two topologies. One of them exemplifies a relatively large scale GT-ITM transit-stub *backbone* network [36] with 200 nodes and 698 links, and the link capacities are uniformly distributed between 10Mbps and 100Mbps. The other topology, used in most of the experiments we report, is relatively small-scale with 55 nodes and 120 links, but it exemplifies a network with both *backbone and access* links and is generated as follows:
 - (1) First, a GT-ITM transit-stub topology is generated to serve as the backbone. Transit-transit links all have a capacity of 100Mbps. The capacities of transit-stub and stub-stub links range from 10Mbps to 80Mbps following a uniform distribution.
 - (2) A random number (uniformly distributed in the range $[0, 4]$) of leaf nodes are attached to each stub router. To model different types of access links, the link capacity between a stub router and a leaf node follows a discrete distribution, and takes one of the following values: 1.5Mbps (30%), 10Mbps (50%), 100Mbps (20%).
 - (3) The link propagation delays follow a uniform distribution between 10ms and

100ms. An average link delay T_a is calculated for the given network once the link delays are configured. Based on the target value of average flow hop count H_t , we calculate a typical RTT in the network as $\hat{T}_t = 2H_t \times T_a$. We then configure the router buffer for each link as the bandwidth delay product: $C \times \hat{T}_t$.

- We select target links randomly with a certain probability, which varies across experiments. The target utilization for a non-lossy link follows a uniform distribution ranging from 0.2 to 0.8. The target loss rates on lossy links are uniformly distributed between 0.001 and 0.03. We limit the selection of lossy target links to those with capacity $C < 35\text{Mbps}$, to reflect the fact that losses occur mostly on lower capacity links.
- The maximum TCP congestion window size follows a uniform distribution between 15 and 45 packets. The TCP data segment size S is 1448 bytes in our simulation.
- We use $u = 0.1$, $\alpha = 1.7$ and $\beta = 1$ throughout the experiments, where α and β define the Pareto distribution of the UDP ON/OFF intervals. We use $H_t = 2.5$ unless otherwise noted.

For each set of inputs, we perform five experiments with different random seeds to derive the results presented.

7.4.2.2 Utilization Approximation

We first examine the impact of different factors on utilization approximation.

Impact of Network Topology: As noted above, we did not observe a noticeable relationship between the accuracy of our algorithm and the scale and type of the input network topology. Due to space limitations, we do not show results for multiple network topologies. In Figure 72, we use the backbone topology (with 200 nodes and 698 links) as described in the experimental setup. Figure 72 shows the CDFs of the magnitude of E for three configurations with different numbers of target links and lossy links. We observe that for about 90% of the target links, the relative error is smaller than 0.1 in these experiments.

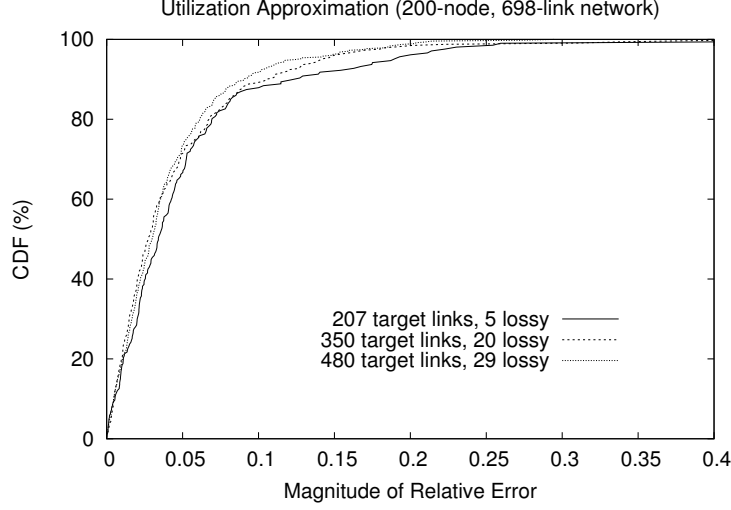


Figure 72: CDF of the magnitude of relative error (backbone topology)

In the rest of this subsection, we use the backbone and access network topology described above to explore the impact of other factors.

Impact of Integer Rounding: We first examine how much of the error in a typical setting is due to rounding.

Figures 73 and 74 show the relative error for utilization constraints in experiments involving 70 target links, among which 5 are lossy. Figure 73 reveals that the largest errors mostly occur on links with a small target load, $L = C\rho < 8Mbps$ in this case, and that for the rest of the links the relative error E is bounded between -20% and 20% except for a couple of outliers. This tendency of decreasing relative error with larger target load is consistent with the impact of rounding. Figure 74 further confirms this observation. We can see that the distributions of the magnitude of E and of E_r are almost the same, indicating that rounding is almost solely responsible for the errors in this configuration.

Figure 75 shows similar results as Figure 74, but for a configuration with 14 lossy links and the same total number of target links. Figure 75 shows that, on average, the error of the algorithm is larger than what can be explained by rounding, especially for errors up to 0.3. This indicates that there are other factors contributing to the errors. In the following several experiments, we will investigate the impact of lossy links and the interactions among

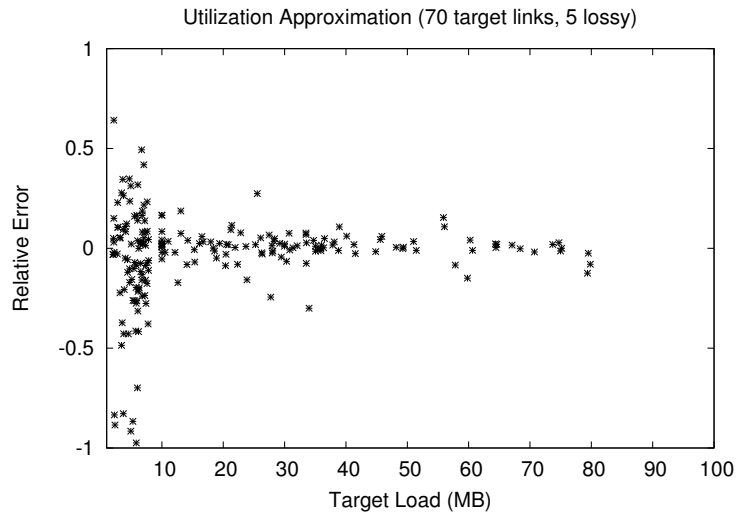


Figure 73: Relative error vs. target load

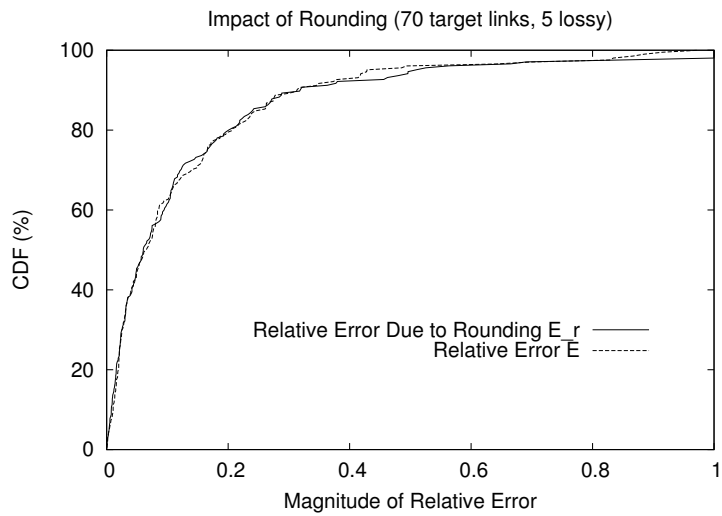


Figure 74: E vs. E_r (5 lossy links, 70 target links)

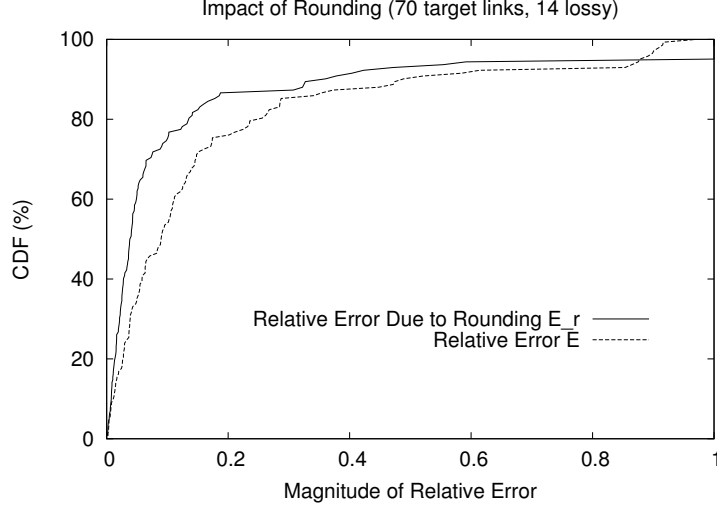


Figure 75: E vs. E_r (14 lossy links, 70 target links)

target links, both of which we believe stress the accuracy of throughput estimation.

In terms of the overall distribution, both Figures 74 and 75 show that about 80% of the target links incur an error below 0.2. The slightly worse approximation, as compared to Figure 72, is explained by the fact there are access links in this topology with very low target loads (and hence very few flows), for which the impact of rounding could be significant.

Impact of the Number of Target Links and Lossy Links: When there are more target links in the same network, it is expected that more target links will be affected by the traffic on a particular target link, and vice versa. Whether this increased interaction among target links causes increased errors is a question relevant to the scalability of the traffic generation algorithm. We found that the accuracy is not noticeably affected by the number of target links, provided that the number of lossy links does not vary significantly. This is demonstrated by Figure 76, which shows the CDFs of relative errors for four configurations that have different numbers of target links but a similar number of lossy links. We can see that the CDFs are very close among the four configurations. Notice that starting from Figure 76 and for the rest of the section, we report results only for links with a relatively high target load ($> 8\text{Mbps}$), on which the errors are not dominated by the rounding error E_r .

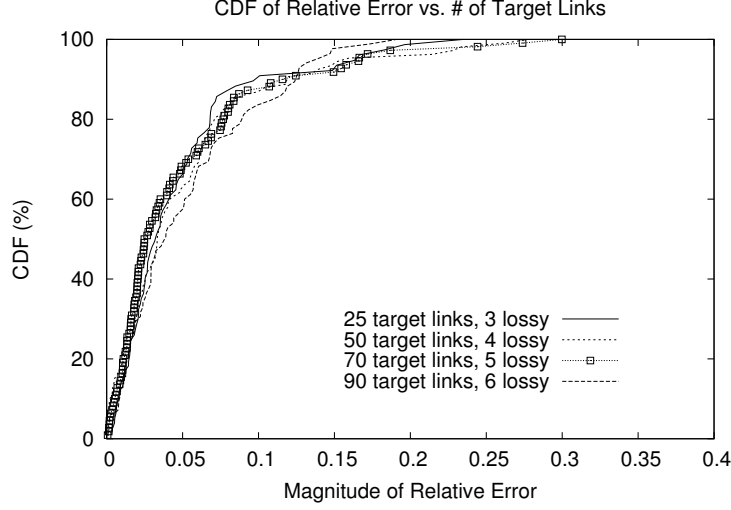


Figure 76: Impact of the number of target links

On the other hand, increasing the number of lossy links noticeably affects the overall accuracy. Figure 77 shows the CDFs of errors for four configurations with different numbers of lossy links and the same number of target links. Clearly, the average error increases with the number of lossy links. Explaining this behavior entails an exposition of the impact of lossy links on other links and we defer it to the next subsection.

Impact of Average Flow Hop Count: Increasing the average flow hop count while maintaining other configurations will also increase the interaction among target links, and potentially the approximation error.

Figure 78 compares the accuracy for three configurations, of which the only difference is the specified average flow hop count H_t . There are 7 lossy links in each of these configurations, out of a total of 70 target links. Figure 79 shows a similar comparison for three configurations, each with 14 lossy links out of 70 target links. Whereas Figure 78 shows that average hop count does not affect the error considerably, Figure 79 clearly shows that the errors increase with the average hop count. Based on the differences between the two figures, it is again suggested that the degree of interactions among links bears on the accuracy of the traffic generation only when there is a relatively large fraction of lossy links.

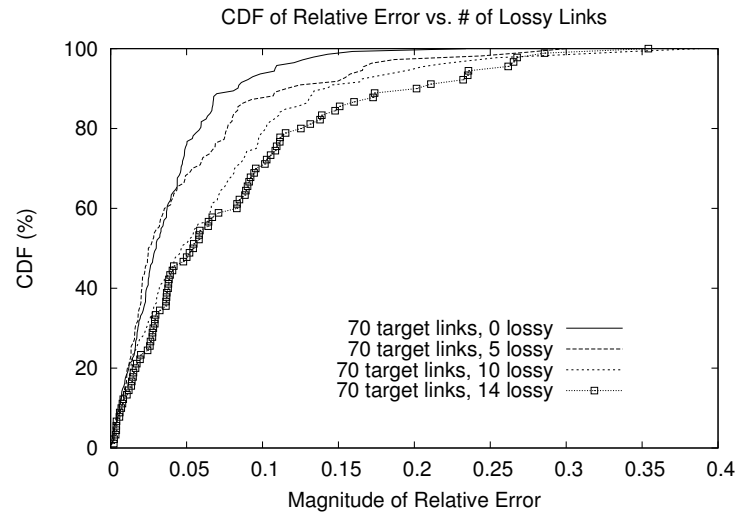


Figure 77: Impact of the number of lossy links

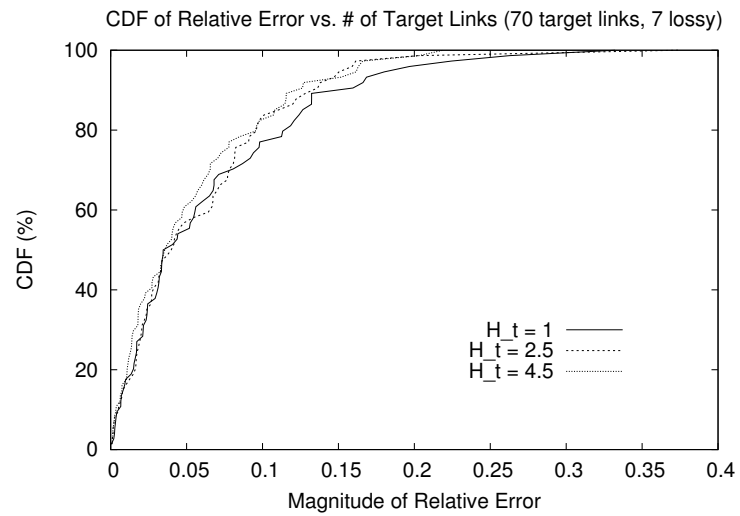


Figure 78: Impact of average hop count (7 lossy links)

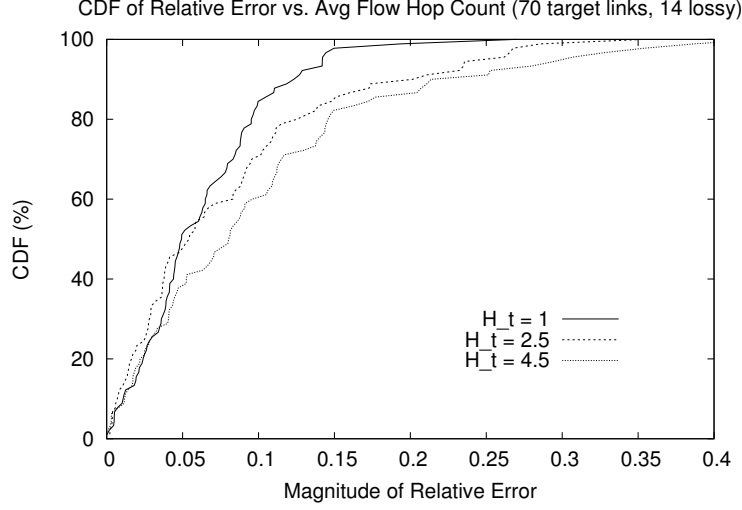


Figure 79: Impact of average hop count (14 lossy links)

7.4.2.3 Loss Rate Approximation

Unlike the approximation of utilization, which is to certain degree dependent on the input, we did not find the average loss rate approximation to be better for some inputs than for others. Also, errors for the loss rate approximation span a larger range. Figure 80 is a scatter-plot of the target loss rate vs. observed loss rate on different links. These results were collected from the set of experiments that are presented in Section 7.4.2.2. For most of the samples, the ratio between the observed and the target loss rates is within the range $[0.5, 3]$.

One may think that the error for loss rate approximation, at this level, would significantly affect the utilization approximation of a lossy link. Clearly, if the average loss rate for TCP flows that traverse a link is twice the target value, the throughput of each flow will be approximately $\frac{1}{\sqrt{2}}$ of the estimated, and the same will be true for the utilization of the link. Fortunately, we find that the *average loss rate of TCP flows* is much closer to the target value than is the *average loss rate*. This is explained by the fact that UDP flows, which are not adaptive to congestion and packet losses [102], experience much higher loss rates than TCP flows. This causes the majority of the lossy target links to have an average packet loss rate that is larger than the target value.

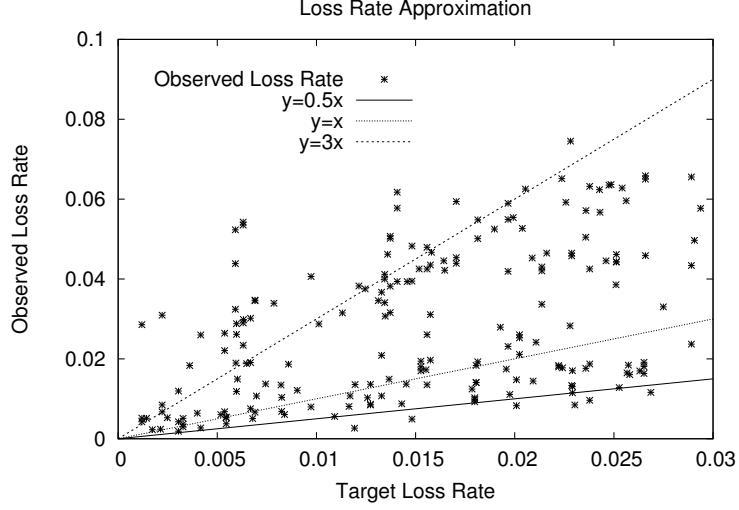


Figure 80: Loss rate approximation (scatter plot)

7.4.3 Explaining the Impact of Lossy Links

Section 7.4.2.2 identifies lossy target links as one major factor that affects the accuracy of the utilization optimization. In this subsection, we aim to understand the cause of this impact by examining the effect of congestion on the RTT and loss rate of TCP flows.

RTT: We first demonstrate that $T_q = \frac{BS}{C}$ and the model (22) in Section 7.3 are good approximations of the queuing delay on a congested link, for cases without and with TCP data traffic in the reverse direction, respectively.

Figure 81 shows the distribution of packet queuing delay for a highly congested link L ($C=45\text{Mbps}$, $B=200$, $\rho=1$). A data packet is 1492 bytes with a 44-byte header, and an ACK is 44 bytes. When there is no traffic in the reverse direction of the link, the estimate $\frac{BS}{C}$ yields a maximum queuing delay $\frac{1492 \times 8 \times 200}{45 \times 10^6} = 53\text{msec}$. This is very close to the observed delay in this scenario, shown as “data packets (w/o reverse traffic)”.

We then add data flows in the reverse direction, with a link utilization $\rho_r = 0.65$. In Figure 81, “data pkts (w/ reverse traffic)” and “ACK pkts (w/ reverse traffic)” show that the queuing delay distributions for data packets and for ACKs are very similar (notice that the ACKs are for the data traffic in the reverse direction). What is important to note, though, is that the maximum queuing delay in this case is approximately 36msec , about

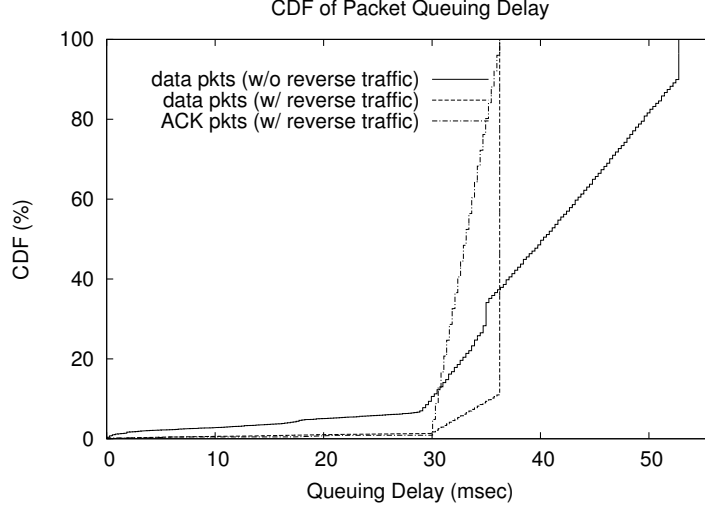


Figure 81: CDF of packet queuing delay

68% of the 53msec maximum queuing delay in the previous case. This is exactly what the delay model (22) would have calculated without the factor θ , which is added to capture the average instead of maximum queuing delay.

Loss Rate: Further analysis of packet-level traces reveals that the largest errors in our experiments take place exclusively on links of which the *reverse* direction is congested. In addition, the congestion in the reverse direction causes data packet losses in the forward direction of each of these links, which is not congested by its own TCP data load. This causes a significant error in the throughput estimate of a TCP data flow traversing the forward direction of the link.

The cause of packet losses on such an uncongested link is the queuing of ACKs in the reverse direction, which leads to a phenomenon known as *ACK compression*. The negative impact of ACK compression on TCP performance has been documented by previous work [79, 154], and it can be summarized as follows. ACKs that are queued arrive at the TCP sender at intervals of $\frac{S_{ack}}{C}$, which leads to the sender transmitting data packets, each $S + S_{hdr}$ bytes, at those intervals. Since data packets are usually much larger than ACKs, this will build up queues in the forward direction and may cause losses. It is, however, not easy to model the loss probability that will result from this phenomenon [79].

7.4.4 Summary of the Evaluation Results

In this section, we have evaluated the accuracy of the proposed traffic generation algorithm in attaining the target link conditions. There are two major factors that affect the utilization approximation. First, rounding to integer numbers of flows can cause errors, especially in links with a small number of flows. Second, lossy target links could cause ACK compression, which eventually affects the estimate of loss rate for flows in the reverse direction; the impact of lossy links is more pronounced when there is more dependence, in terms of traffic, among links. In most of our experiments, the utilization error is smaller than 20% for as many as 80% of the target links, and rounding only affects links with a very small number (between 1 and 3) of flows. Loss rate accuracy is in general not as good as utilization accuracy, and the ratio between the observed and the target loss rates is mostly within the range $[0.5, 3]$. However, we believe that loss rate approximations in this range are acceptable to most target applications of the proposed traffic generation method.

7.5 *Summary*

In this paper, we developed and evaluated a traffic generation algorithm for TCP simulations that matches the generated traffic to the network configuration such that the resulting link conditions are as specified by the user. The algorithm enables users to obtain realistic and explicitly controlled link conditions through background traffic simulation. This was achieved in a trial-and-error manner in previous work and it was possible only for very simple networks. While attaining target link conditions, our algorithm also minimizes the simulation memory requirement and conforms to certain realism considerations, such as the traffic mix and the average flow hop count. Our evaluations show that link conditions realized by the generated traffic are very close to the target conditions, and that the algorithm scales well with the size of the network topology and the number of target links. For future work, we intend to use the traffic generation algorithm to study in depth how sensitive different simulation studies are to certain characteristics of the background traffic. In addition, we will explore the application of this methodology with alternative traffic models or optimization objectives.

CHAPTER VIII

CONCLUSION AND FUTURE WORK

8.1 Research Summary

This thesis addresses two types of performance evaluation problems that are related to a seemingly familiar topic, TCP-based services and networks. The major contributions of this thesis are demonstrated by the success stories summarized below.

Performance Evaluation-Assisted Design of Advanced TCP-based Services This thesis studies two design problems motivated by the emerging intelligent routing and peer-to-peer resource sharing systems. The approaches we took rely heavily on detailed performance evaluation in order to understand the nature of the target problem, the performance of alternative solutions that exist, and the components towards better solutions.

- **Throughput prediction for large TCP transfers.** This thesis presents a systematic evaluation of the accuracy of two categories of TCP throughput prediction methods: Formula-based (FB) and History-based (HB) methods. The evaluation is based on (1) carefully designed Internet measurement, and (2) simple but explanatory analytical studies. While the evaluation methodology itself can be used for similar studies, the most important contributions of this study, as we believe, are the empirical and analytical findings that offer insights to the design of better throughput prediction methods. FB prediction is an attractive option, given that it does not require intrusive measurements or any history of prior TCP transfers. We demonstrated however that it can be inaccurate, especially when the transfer attempts to saturate the path, and we explained the reasons for these errors. HB prediction, on the other hand, is quite accurate but it is feasible only when there is a history of previous TCP transfers in the same path. Although the accuracy of HB prediction does not depend so much on the actual predictor, it does depend on the transfer's maximum congestion window

size and on the underlying path. We explained the path dependency based on two factors: the load and the degree of multiplexing on the bottleneck link of the path.

- Congestion control for message-oriented peer-to-peer overlay networks. This thesis evaluates the congestion control and message loss behavior in the Gnutella network. It also exposes the design issues of congestion control for such an overlay network. Our investigation reveals the heterogeneous congestion control strategies existing among different Gnutella server implementations and that most of the congestion control mechanisms are derived from those originally designed for native network routers in the context of packet forwarding network. Through measurement, we show the different Ultrapeer message loss behavior resulting from this heterogeneity. To understand the overall user-perceived performance and network congestion level with alternative congestion control mechanisms, we perform packet-level simulations of the Gnutella network. We show that the usage and performance of these mechanisms are strongly affected by the characteristics of the Gnutella network, among which are the lack of a strong notion of “flow”, and the heterogeneity of congestion control mechanisms. In particular, we show that: (1) the design of an optimal priority policy for the priority queuing scheme is challenged by multiple dimensions of performance requirement and the dynamic workload composition; (2) TCP backpressure, which is hardly adopted by the numerous server implementations, might deliver a superior performance, although its performance is sensitive to the traffic load; (3) rate limiting reduces congestion effectively only at the cost of throughput degradation; (4) for each of the above schemes, heterogeneity precludes the full realization of its full advantage. The results of this work can be used to guide the design of better congestion control in message-oriented peer-to-peer overlay networks.
- Packet-level simulation of peer-to-peer systems: GnutellaSim. We design and develop a framework for an extensible and scalable peer-to-peer simulation environment that can be built on top of existing packet-level network simulators. The simulation environment is portable to different network simulators, which enables us to simulate a

realistic large scale peer-to-peer system using existing parallelization techniques. We demonstrate the use of the simulator for some simple experiments that show how Gnutella system performance can be impacted by the network characteristics. With simple extensions, the simulator also enables us to study a particular peer-to-peer research problem at the packet level—evaluation of the different Gnutella congestion control schemes that are currently in use in the Gnutella network. Many researchers around the world have been using GnutellaSim for their work.

Efficient TCP/IP Network Simulations Using Detailed Knowledge of TCP This thesis designs and implements two algorithms, to speed up network simulations, and to generate realistic workload for large scale network simulations, respectively. Unlike traditional approaches to these two issues, we utilize much knowledge of TCP and of network traffic in our designs.

- Exploiting the predictability of steady-state TCP behavior to speed up network simulation. We propose to speed up the simulation of TCP flows by predicting and aggregating packet events based on the predictability of steady-state TCP behavior. We design a prediction-based simulation framework to use the predicted flows and describe two techniques to predict the average throughput of a steady-state TCP flow, the direct measurement-based prediction and the TCP throughput formula-based prediction. Our results show that the prediction-based simulation achieves significant speedup over packet-based simulation and the speedup increases dramatically with the size of a transfer. Due to the assumption made about inter-loss times, the formula prediction is not very accurate under complex path conditions. Under simplex conditions, however, formula-based achieves good approximation with a shorter measurement period than required of the direct measurement prediction, since direct measurement is more sensitive to the transient behavior of TCP congestion control. Compared to fluid model simulations, our prediction-based simulation is easier to be integrated with packet-level simulation, and is more generally applicable. Whereas the aggregation of events in a TCP flow simulation is based on the predictability of

TCP steady-state, the idea of aggregating packet events of a flow with predictable behavior is applicable to the simulation of other predictable flows.

- Traffic generation under load constraints. We developed and evaluated a traffic generation algorithm for TCP simulations that matches the generated traffic to the network configuration such that the resulting link conditions are as specified by the user. The algorithm enables users to obtain realistic and explicitly controlled link conditions through background traffic simulation. This was achieved in a trial-and-error manner in previous work and it was possible only for very simple networks. While attaining target link conditions, our algorithm also minimizes the simulation memory requirement and conforms to certain realism considerations, such as the traffic mix and the average flow hop count. Our evaluations show that link conditions realized by the generated traffic are very close to the target conditions, and that the algorithm scales well with the size of the network topology and the number of target links.

8.2 *Future Directions*

This thesis suggests several topics for future work.

First, on the throughput prediction for large TCP transfers, our findings suggest that predictive models (unlike the existing descriptive models) of TCP throughput will be an interesting area to explore for better FB predictions. We also suggest that such a model should account for the load that will be exerted by the target flow, and the impact of that load on the queueing delays and losses in the path. More information about the underlying path, such as the capacity, available bandwidth, buffer size, and number of competing flows, may help achieving this goal. For HB prediction, one interesting problem is how to extend its usage. In particular, efficient mechanisms to acquire, share and reuse throughput history can be investigated. For example, hosts in the same subnet may share information on the throughput to/from the hosts in a target subnet.

Second, research in workload generation for large scale network simulations should be keen on incorporating realistic traffic matrix models when such models become available. While work on TCP workload generation provides a framework to incorporate traffic matrix

models (by the use of additional constraints and parameters), establishing models for intra-AS and inter-AS traffic matrices is known to be very difficult and the research in this area is far from mature. We intend to use the traffic generation algorithm to study in depth how sensitive different simulation studies are to certain characteristics of the background traffic. In addition, we will explore the application of this methodology with alternative traffic models or optimization objectives. More specifically, we are interested in Web-like traffic models, and in optimizing the load balance for parallel and distributed network simulations.

Third, while the research on peer-to-peer file sharing systems has abated quite a bit near the end of this thesis work, the usage of these systems remains high and has never slipped. Therefore, optimizing the user-perceived experience and managing the resource usage of these systems are bound to be lingering issues of interest. Much of the research in this area has been invested in elements that have direct implication for these issues, such as searching/locating schemes and peer selection schemes. Our study of the Gnutella systems shows the indirect impact of searching overlay congestion control mechanisms on these performance metrics, as some other works have revealed. It will be interesting to see measurement studies that quantify this impact in the real network, especially if such a study is conducted such that a comparison with the optimization in searching scheme can be drawn. While the two elements of peer-to-peer systems are orthogonal, it is our belief that the importance of the congestion control mechanisms has been relative insufficiently addressed.

Fourth, we believe that the impact of traffic models on performance evaluation, e.g., simulation, has not been fully examined. On one hand, there are many dimensions along which a traffic model can be categorized, each affecting the network behavior in different ways; on the other hand, different applications can be affected by traffic models in different ways. A systematic analysis of the factors that affect the basic characteristics of network traffic will be useful as a guide to the performance evaluation and design of new network applications and systems. We believe at least the following dimensions should be used to fully specify a traffic model:

- Open-loop vs. Closed-loop.

- Persistent TCP flows vs. Stream of TCP flows.
- Finite population (closed-loop) vs. Infinite population (open-loop).
- Window-limited TCP flows vs. Congestion-limited TCP flows.
- Unidirectional vs. Bidirectional traffic.
- Topology: dumbbell vs. Internet topology generators.
- Access link bandwidth relative to backbone bandwidth.

Finally, one of the general themes this thesis tries to bring out is the persisting importance of performance evaluation of a subject that seems so familiar to us. Floyd recently announced a new IRTF group (Transport Models Research Group) on the evaluation of transport protocols [16, 17]. The group is chartered to do research specifically in the area of modeling and evaluation of transport protocols, rather than in the design new congestion control protocols. Echoing what we stated at the beginning of this thesis, we see as an important mission the continued efforts in performance evaluation as new applications and technologies emerge and as the Internet usage grows.

REFERENCES

- [1] “Network Simulator.” <http://www.isi.edu/nsnam/ns>, Jan 2000.
- [2] “To the Bandwidth Barrier and Beyond.” <http://dss.clip2.com>, 2000.
- [3] “Gnutella Protocol Specification v0.4.” http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf, 2001.
- [4] “FreeNet.” <http://freenet.sourceforge.org>, Feb 2003.
- [5] “GnutellaSim.” <http://www.cc.gatech.edu/computing/compass/gnutella>, Sep 2003.
- [6] “Gtk-gnutella.” <http://gtk-gnutella.sourceforge.net>, Feb 2003.
- [7] “GWebCache.” <http://www.gnucleus.net/gwebcache/>, Feb 2003.
- [8] “Limewire.” <http://www.limewire.org>, Feb 2003.
- [9] “Mutella.” <http://mutella.sourceforge.net>, Feb 2003.
- [10] “Phex.” <http://phex.sourceforge.net>, Feb 2003.
- [11] “Qtella.” <http://qtella.sourceforge.net>, Feb 2003.
- [12] “SETI@Home.” <http://setiathome.ssl.berkeley.edu>, Mar 2003.
- [13] “Ultrapeer Specification.” <http://www.limewire.com/developer/Ultrapeers.html>, April 2003.
- [14] “Traffic Generators.” <http://icir.org/models/trafficgenerators.html>, Jan 2004.
- [15] “Hardware in the Loop/Man in the Loop.” http://www.scalable-networks.com/pdf/HITL_Case_Study.pdf, Jun 2005.
- [16] “IRTF TMRG.” <http://www.postel.org/pipermail/end2end-interest/2005-May/004953.html>, Jun 2005.
- [17] “Metrics for the Evaluation of Congestion Control Mechanisms.” draft-floyd-transport-metrics-00.txt, May 2005.
- [18] “Scalable Network Technologies.” <http://www.qualnet.com/products/developers/simulator.php>, Jun 2005.
- [19] ADAS, A., “Traffic Models in Broadband Networks,” *IEEE Communications Magazine*, July 1997.
- [20] AKELLA, A., MAGGS, B., SESHAN, S., SHAIKH, A., and SITARAMAN, R., “A Measurement-Based Analysis of Multihoming,” in *Proceedings of ACM SIGCOMM*, 2003.

- [21] AKELLA, A., PANG, J., SHAIKH, A., MAGGS, B., and SESHAN, S., "A Comparison of Overlay Routing and Multihoming Route Control," in *Proceedings of ACM SIGCOMM*, 2004.
- [22] ALTMAN, E., AVRACHENKOV, K., and BARAKAT, C., "A Stochastic Model of TCP/IP with Stationary Random Losses," in *Proceedings of ACM SIGCOMM*, 2000.
- [23] ANDD G. HE, Y. G. and HOU, J., "On Exploiting Traffic Predictability in Active Queue Management," in *Proceedings of INFOCOM*, 2002.
- [24] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., and MORRIS, R., "Resilient Overlay Networks," in *Proceedings of ACM Symposium on Operating Systems Principles*, 2001.
- [25] ANDERSEN, D. G., SNOEREN, A. C., and BALAKRISHNAN, H., "Best-Path vs. Multi-Path Overlay Routing," in *Proceedings Internet Measurement Conference (IMC)*, 2003.
- [26] ANDERSON, D., BALAKRISHNAN, H., KAASHOEK, M., and MORRIS, R., "Resilient Overlay Networks," in *Proceedings of SOSP*, 2001.
- [27] APPENZELLER, G., KESLASSY, I., and MCKEOWN, N., "Sizing Router Buffers," in *SIGCOMM*, 2004.
- [28] BACCELLI, F. and KIM, K. B., "TCP Throughput Analysis under Transmission Error and Congestion Losses," in *Proceedings of IEEE INFOCOM*, 2004.
- [29] BALAKRISHNAN, H., PADMANABHAN, V., SESHAN, S., and KATZ, R., "A comparison of mechanisms for improving TCP performance over wireless links," *ACM/IEEE Transactions on Networking*, vol. 5, no. 6, pp. 756–769, 1997.
- [30] BARFORD, P. and CROVELLA, M., "Generating Representative Web Workloads for Network and Server Performance Evaluation," in *Proceedings of SIGMETRICS*, 1998.
- [31] BASIOUNI, M., CHIU, M., LOPER, M., GARNSEY, M., and WILLIAMS, J., "Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation," *ACM Transactions on Modeling and Computer Simulations (TOMACS)*, vol. 7, no. 3, pp. 293–331, 1997.
- [32] BLAKE, S., BLACK, D., CARLSON, M., DAVIES, E., WANG, Z., and WEISS, W., "An Architecture for Differentiated Services, rfc2475," 1998.
- [33] BOX, G. E. P., JENKINS, G. M., and REINSEL, G. C., *Time Series Analysis: Forecasting and Control (3rd edition)*. Prentice-Hall, 1994.
- [34] BRESLAU, L., ESTRIN, D., FALL, K., FLOYD, S., and ET.AL, "Advances in Network Simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59–67, 2000.
- [35] CAI, W., LEE, F., and CHEN, L., "An Auto-adaptive Dead Reckoning Algorithm for Distributed Interactive Simulation," in *Proceedings of Workshop on Parallel and Distribution Simulation (PADS)*, 1999.
- [36] CALVERT, K., DOAR, M., and ZEGURA, E. W., "Modeling Internet Topology," *IEEE Communications Magazine*, Jun 1997.

- [37] CAO, J., CLEVELAND, W., GAO, Y., JEFFAY, K., SMITH, F. D., and WEIGLE, M., "Stochastic Models for Generating Synthetic HTTP Source Traffic," in *Proceedings of Infocom*, 2004.
- [38] CARDWELL, N., SAVAGE, S., and ANDERSON, T., "Modeling TCP Latency," in *Proceedings of IEEE Infocom*, 2000.
- [39] CARSON, M. and SANTAY, D., "NIST Net - A Linux-based Network Emulation Tool," *ACM Computer Communication Review*, vol. 33, no. 6, pp. 111–126, 2003.
- [40] CHASE, J., GALLATIN, A., and YOCUM, K., "End-System Optimizations for High-Speed TCP," *IEEE Communications Magazine*, Jun 2000.
- [41] CHAWATHE, Y., RATNASAMY, S., L.BRESLAU, LANHAM, N., and SHENKER, S., "Making Gnutella-like P2P Systems Scalable," in *SIGCOMM*, 2003.
- [42] CHAWATHE, Y., RATNASAMY, S., L.BRESLAU, LANHAM, N., and SHENKER, S., "Making Gnutella-like P2P Systems Scalable," in *Proceedings of SIGCOMM*, 2003.
- [43] CHU, Y.-H., RAO, S. G., SESHAN, S., and ZHANG, H., "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture," in *Proceedings of SIGCOMM*, Aug. 2001.
- [44] COHEN, B., "Incentive Build Robustness in BitTorrent," 2003.
- [45] DHAMDHERE, A., JIANG, H., and DOVROLIS, C., "Buffer Sizing for Congested Internet Links," in *Proceedings of IEEE Infocom*, 2005.
- [46] ERIKSSON, H., "MBONE-The Multicast Backbone," *Communications of the ACM*, vol. 37, no. 8, pp. 54–60, 1993.
- [47] FLOYD, S., HANDLEY, M., PADHYE, J., and WIDMER, J., "Equation-Based Congestion Control for Unicast Applications," in *Proceedings of SIGCOMM*, 2000.
- [48] FLOYD, S. and JACOBSON, V., "Random Early Detection Gateways for Congestion Avoidance," *ACM/IEEE Transaction on Networking*, vol. 1, no. 4, pp. 397–413, 1993.
- [49] FLOYD, S., JACOBSON, V., LIU, C., MCCANNE, S., and ZHANG, L., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *IEEE/ACM Transactions on Networking*, vol. 55555, no. 6, pp. 784–803, 1997.
- [50] FLOYD, S. and KOHLER, E., "Internet Research Needs Better Models," in *Proceedings of HotNets*, 2002.
- [51] FOSTER, I., *The Grid: Blueprinting for a New Computing Infrastructure*. Morgan Kaufmann, 2004 (2nd edition).
- [52] FREDJ, S. B., BONALD, T., PROUTIERE, A., REGNIE, G., and ROBERTS, J. W., "Statistical Bandwidth Sharing: A Study of Congestion at Flow Level," in *Proceedings of ACM SIGCOMM*, Aug. 2001.
- [53] FUJIMOTO, R., PERUMALLA, K., PARK, A., WU, H., AMMAR, M., and RILEY, G., "Large Scale Simulation: How Big? How Fast?," in *Proceedings of MASCOTS*, 2003.

- [54] GARETOO, M., CIGNO, R. L., MEO, M., and MARSON, M., "A Detailed and Accurate Closed Queueing Network Model of Many Interacting Flows," in *Proceedings of IEEE Infocom*, 2001.
- [55] GE, Z., FIGUEIREDO, D., JAISWAL, S., KUROSE, J., and TOWSLEY, D., "Modeling Peer-Peer File Sharing Systems," in *Proceedings of IEEE Infocom*, 2003.
- [56] GNUCLEUS. <http://www.gnucleus.com>, Feb 2003.
- [57] GOYAL, M., GUERIN, R., and RAJAN, R., "Predicting TCP Throughput From Non-invasive Network Sampling," in *Proceedings of IEEE Infocom*, 2002.
- [58] GROSSGLAUSER, M. and BOLOT, J.-C., "On the Relevance of Long-Range Dependence in Network Traffic," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 629–640, 1999.
- [59] GU, Y. and FUJIMOTO, R., "A Flexible Architecture for Remote Server-Based Emulation," in *Proceedings of MASCOTS*, 2004.
- [60] HACKER, T., NOBLE, B., and ATHEY, B., "Improving Throughput and Maintaining Fairness using Parallel TCP," in *Proceedings of IEEE Infocom*, 2004.
- [61] HARRISON, D., KALYANARAMAN, S., and RAMAKRISHNAN, S., "Overlay Bandwidth Services: Basic Framework and Edge-to-Edge Closed-Loop Building Block, (preprint)," 2001.
- [62] HE, G. and HOU, J. C., "On Exploiting Long Range Dependence of Network Traffic in Measuring Cross Traffic on an End-to-End Basis," in *Proceedings of IEEE INFOCOM*, Apr. 2003.
- [63] HE, Q. and AMMAR, M., "Congestion Control and Message Loss in Gnutella Networks," in *Proceedings of SPIE Conference on Multimedia Computing and Networking (MMCN)*, 2004.
- [64] HE, Q., AMMAR, M., RAJ, H., RILEY, G., and FUJIMOTO, R., "Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems," in *Proceedings of MASCOTS*, 2003.
- [65] HE, Q., AMMAR, M., RILEY, G., and FUJIMOTO, R., "Exploiting the Predictability of Steady-state TCP Behavior to Speed Up Network Simulations," *Performance Evaluation*, vol. 58/2-3, 2004.
- [66] HE, Q., DOVROLIS, C., and AMMAR, M., "On the Predictability of Large Transfer TCP Throughput," in *In Proceedings of ACM SIGCOMM (to appear)*, Aug 2005.
- [67] HE, Q., DOVROLIS, C., and AMMAR, M., "Prediction of TCP Throughput: Formula-based and History-based Methods," in *In Proceedings of ACM SIGMETRICS conference (extended abstract)*, 2005.
- [68] HU, N. and STEENKISTE, P., "Evaluation and Characterization of Available Bandwidth Probing Techniques," *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 879–894, Aug. 2003.

- [69] “Iperf.” <http://dast.nlanr.net/Projects/Iperf/>, Mar 2004.
- [70] JAIN, M. and DOVROLIS, C., “End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput,” *IEEE/ACM Transactions on Networking*, vol. 11, pp. 537–549, Aug 2003.
- [71] JIN, C., WEI, D., and LOW, S., “FAST TCP: Motivation, Architecture, Algorithms, and Performance,” in *Proceedings of IEEE Infocom*, 2004.
- [72] JIN, S., GUO, L., MATTA, I., and BESTAVROS, A., “A Spectrum of TCP-friendly Window-based Congestion Control Algorithms,” *IEEE/ACM Transactions on Networking*, 2003.
- [73] JOO, Y., RIBEIRO, V., FELDMANN, A., GILBERT, A., and WILLINGER, W., “On the Impact of Variability on the Buffer Dynamics in IP Networks,” in *Proceedings of 39th Annual Allerton Conference on Communication*, 2001.
- [74] JOO, Y., RIBEIRO, V., FELDMANN, A., GILBERT, A., and WILLINGER, W., “TCP/IP Traffic Dynamics and Network Performance: A Lesson in Workload Modeling, Flow Control, and Trace-driven Simulations,” *ACM Computer Communication Review*, vol. 31, April 2001.
- [75] KANT, K. and IYER, R., “A Performance Model for Peer to Peer File-Sharing Services.” Intel Corporation, Nov 2001.
- [76] KARBHARI, P., AMMAR, M., DHAMDHARE, A., RAJ, H., RILEY, G., and ZEGURA, E., “Bootstrapping in Gnutella: A Measurement Study,” in *Passive and Active Measurements Workshop*, 2004.
- [77] KRISHNAMURTHY, B., WILLS, C., and ZHANG, Y., “On the Use and Performance of Content Distribution Networks,” in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001.
- [78] KWON, G. and BYERS, J., “Reliable Overlay Multicast with Loosely Coupled TCP Connections,” in *Proceedings of IEEE Infocom*, 2004.
- [79] LAKSHMAN, T., MADHOW, U., and SUTER, B., “TCP/IP Performance with Random Loss and Bidirectional Congestion,” *ACM/IEEE Transactions on Networking*, vol. 8, no. 5, pp. 541–555, 2000.
- [80] LAN, K. and HEIDEMANN, J., “Rapid Model Parameterization from Traffic Measurements,” *ACM Transactions on Modeling and Computer Simulations (TOMACS)*, vol. 12, no. 3, pp. 201–229, 2002.
- [81] LE, L., AIKAT, J., JEFFAY, K., and SMITH, F. D., “The Effects of Active Queue Management on Web Performance,” in *Proceedings of SIGCOMM*, 2003.
- [82] LENITRA, L., AMMAR, M., ZEGURA, E. W., and CLARK, R., “Posting Protocol for Improved Keyword Search Success in Peer-to-Peer File Sharing Systems,” in *Proceedings of Multimedia Computing and Networking (MMCN)*, 2003.
- [83] LILJENSTAM, M., LIU, J., NICOL, D., YUAN, Y., YAN, G., and GRIER, C., “RINSE: the Real-Time Immersive Network Simulation Environment for Network Security Exercises,” in *Proceedings of PADS*, 2005.

- [84] LIU, Y., PRESTI, F., MISRA, V., TOWSLEY, D., and GU, Y., "Fluid Models and Solutions for Large-Scale IP Networks," in *Proceedings of SIGMETRICS*, 2003.
- [85] LOW, S., PETERSON, L., and WANG, L., "Understanding TCP Vegas: A Duality Model," in *Proceedings of SIGMETRICS*, 2001.
- [86] LUCAS, M., DEMPSEY, B., WREGE, D., and WEAVER, A., " (M, P, S) -An Efficient Background Traffic Model for Wide-Area Network Simulation," in *Proceedings of GlobeCom*, 1997.
- [87] MAH, B., SHOLANDAR, P., MARTINEZ, L., and TOLENDINO, L., "IPB: An Internet Protocol Benchmark Using Simulated Traffic," in *Proceedings of MASCOTS*, 1998.
- [88] MAHDAVI, J. and FLOYD, S., "TCP-Friendly Unicast Rate-Based Flow Control." Note sent to end2end-interest mailing list, Jan 1997.
- [89] MATHIS, M., SEMKE, J., and MADHAVI, J., "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM Computer Communications Review*, vol. 27, pp. 67–82, July 1997.
- [90] MEDINA, A., TAFT, N., SALAMATIAN, K., BHATTACHARYYA, S., and DIOT, C., "Traffic Matrix Estimation: Existing Techniques and New Directions," in *SIGCOMM*, 2002.
- [91] MELAMED, B., PAN, S., and WARDI, Y., "Hybrid Discrete-Continuous Fluid-flow Simulation," in *Proceedings of ITCOM*, 2001.
- [92] MISRA, V., GONG, W., and TOWSLEY, D., "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," in *Proceedings of ACM SIGCOMM*, 2000.
- [93] MORRIS, R., "TCP Behavior with Many Flows," in *Proceedings of ICNP*, 1997.
- [94] NG, T. S. E., CHU, Y., S.RAO, SRIPANIDKULCHAI, K., and ZHANG, H., "Measurement-based Optimization Techniques for Bandwidth-demanding Peer-to-Peer Systems," in *Proceedings of IEEE Infocom*, 2003.
- [95] NOGUEIRA, D., ROCHA, L., SANTOS, J., ARAUJO, P., ALMEIDA, V., and W.MEIRA, "A Methodology for Workload Characterization of File-Sharing Peer-to-Peer Networks," in *Proceedings of Annual Workshop of Workload Characterization*, 2002.
- [96] OHSAKI, H., MORITA, M., and MURATA, M., "On Modeling Round-Trip Time Dynamics of the Internet using System Identification," in *Proceedings of International Conference on Information Networking*, 2002.
- [97] OHSAKI, H., TAKAGAKI, K., and MURATA, M., "A Control Theoretical Analysis of a Window-Based Flow Control Mechanism for TCP Connections with Different Propagation Delays," in *Proceedings of SPIE ITCOM*, 2000.
- [98] OSOKINE, S., "The Flow Control Algorithm for the Distributed Broadcast-Route Networks with Reliable Transport Links." <http://www.grouter.net/gnutella>.
- [99] OTT, D., SPARKS, T., and MAYER-PATEL, K., "Aggregate Congestion Control for Distributed Multimedia Applications," in *Proceedings of IEEE Infocom*, 2004.

- [100] PADHYE, J., V.FIROIU, D.TOWSLEY, and KUROSE, J., "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 133–145, Apr. 2000.
- [101] PARK, K. and TUAN, T., "Performance Evaluation of Multiple Time Scale TCP Under Self-Similar Traffic Conditions," *Modeling and Computer Simulation*, vol. 10, no. 2, pp. 152–177, 2000.
- [102] PAXSON, V., "End-to-End Internet Packet Dynamics," *IEEE/ACM Transaction on Networking*, vol. 7, pp. 277–292, Jun 1999.
- [103] PAXSON, V. and S.FLOYD, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 226–244, June 1995.
- [104] PAZOS, C., AGRELO, J. S., and GERLA, M., "Using Back-Pressure to Improve TCP Performance with Many Flows," in *Proceedings of Infocom*, 1999.
- [105] PETERSON, L., SHENKER, S., and TURNER, J., "Overcoming the Internet Impasse through Virtualization," in *Proceedings of HotNets*, 2004.
- [106] POURAHMADI, M., *Foundations of Time Series Analysis and Prediction Theory*. John Wiley and Sons, 2001.
- [107] PRASAD, R., JAIN, M., and DOVROLIS, C., "Socket Buffer Auto-Sizing for High-Performance Data Transfers," tech. rep., GIT-CERCS, 2004.
- [108] QIAO, Y., SKICEWICZ, J., and DINDA, P., "An Empirical Study of the Multiscale Predictability of Network Traffic," in *Proceedings of IEEE HPDC*, 2003.
- [109] QIU, L., PADMANABHAN, V. N., and VOELKER, G. M., "On the Placement of Web Server Replicas," in *Proceedings of IEEE INFOCOM*, pp. 1587–1596, 2001.
- [110] RIBEIRO, V., RIEDI, R., BARANIUK, R., NAVRATIL, J., and COTTRELL, L., "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," in *Proceedings of Passive and Active Measurements (PAM) workshop*, 2003.
- [111] RILEY, G., "The Georgia Tech Network Simulator," in *Proceedings of MoMeTools*, 2003.
- [112] RILEY, G. and AMMAR, M., "Simulating Large Networks: How Big is Big Enough," in *Proceedings of First International Conference on Grand Challenges for Modeling and Simulations*, 2002.
- [113] RILEY, G., AMMAR, M., and ZEGURA, E., "Efficient Routing With Nix-Vectors," in *Proceedings of HPSR*, 2001.
- [114] RILEY, G., FUJIMOTO, R., and AMMAR, M., "A Generic Framework for Parallelization of Network Simulations," in *Proceedings of MASCOTS*, 1999.
- [115] RIPEANU, M., FOSTER, I., , and IAMNITCHI, A., "Mapping the Gnutella Network: Properties of Large-scale Peer-to-Peer systems and Implications for System Design," *Internet Computing Journal*, vol. 6, no. 1, 2002.
- [116] RITTER, J., "Why Gnutella Doesn't Scale. No, really," 2001.

- [117] ROHRS, C., “Improving Gnutella’s Ping/Pong Scheme,” 2002.
- [118] ROS, D. and MARIE, R., “Estimation of End-to-End Delay in High-speed Networks by Means of Fluid Model Simulations,” in *Proceedings of 13th European Simulation Multiconference*, 1999.
- [119] ROSS, S. M., *Introduction to Probability Models*. Harcourt Academic Press, seventh ed., 2000.
- [120] R.S.TSAY, “Outliers, Level Shifts, and Variance Changes in Time Series,” *Journal of Forecasting*, 1988.
- [121] S. BANERJEE, B. BHATTACHARJEE, C. K., “Scalable Application Layer Multicast,” in *Proceedings of SIGCOMM*, 2002.
- [122] SACHRIFC, “<http://www.limewire.com/developer/sachrffc.txt>,” Mar 2003.
- [123] SANG, A. and LI, S., “A Predictability Analysis of Network Traffic,” *Computer Networks*, vol. 39, pp. 329 – 345, Jul 2002.
- [124] SAROIU, S., GUMMADI, P. K., DUNN, R., GRIBBLE, S., and LEVY, H., “An Analysis of Internet Content Delivery Systems,” in *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI)*, 2002.
- [125] SAROIU, S., GUMMADI, P. K., and GRIBBLE, S., “A Measurement Study of Peer-to-Peer File Sharing Systems,” in *Proceedings of Multimedia Computing and Networking (MMCN)*, 2002.
- [126] SAVAGE, S., COLLINS, A., HOFFMAN, E., SNELL, J., and ANDERSON, T., “The End-to-End Effects of Internet Path Selection,” in *Proceedings of SIGCOMM*, 1999.
- [127] SEN, S. and WANG, J., “Analyzing Peer-to-Peer Traffic Across Large Networks,” *ACM/IEEE Transactions on Networking*, vol. 12, no. 2, pp. 219–232, 2004.
- [128] SIKDAR, B., KALYANARAMAN, S., and VASTOLA, K. S., “Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno and SACK,” *IEEE/ACM Transactions on Networking*, vol. 11, pp. 959–971, Dec. 2003.
- [129] SOMMERS, J., KIM, H., and BARFORD, P., “Harpoon: A Flow-Level Traffic Generator for Router and Network Tests,” in *Proceedings of SIGMETRICS (extended abstract and poster)*, 2004.
- [130] SPRINT, “IP Traffic Monitor.” <http://ipmon.sprint.com>, Apr 2002.
- [131] STRAUSS, J., KATABI, D., and KAASHOEK, F., “A Measurement Study of Available Bandwidth Estimation Tools,” in *Proceedings of ACM Internet Measurement Conference*, 2003.
- [132] SUBRAMANIAN, L., STOICA, I., BALAKRISHNAN, H., and KATZ, R. H., “OverQoS: Offering Internet QoS Using Overlays,” *ACM Computer Communication Review*, vol. 33, no. 1, pp. 11–16, 2004.
- [133] SWANY, M. and WOLSKI, R., “Multivariate Resource Performance Forecasting in the Network Weather Service,” in *Proceedings of Supercomputing*, 2002.

- [134] SYLVIA RATNASAMY AND MARK HANDLEY AND RICHARD KARP AND SCOTT SHENKER, "Topologically-Aware Overlay Construction and Server Selection," in *Proceedings of IEEE Infocom*, 2002.
- [135] TIAN, X., WU, J., and JI, C., "A Unified Framework for Understanding Network Traffic Using Independent Wavelet Models," in *Proceedings of IEEE Infocom*, 2002.
- [136] TOUCH, J., "Dynamic Internet Overlay Deployment and Management Using the X-Bone," in *Proceedings of ICNP*, 2001.
- [137] TUAN, T. and PARK, K., "Multiple Time Scale Congestion Control for Self-Similar Network Traffic," *Performance Evaluation*, vol. 36, pp. 359–386, 1999.
- [138] URVOY-KELLER, G. and BIRSACK, E. W., "A Congestion Control Model for Multicast Overlay Networks and its Performance," in *Proceedings of NGC*, 2002.
- [139] VAZHKUDAI, S., SCHOPF, J., and FOSTER, I., "Predicting the Performance of Wide Area Data Transfers," in *Proceedings of IEEE IPDPS*, 2002.
- [140] VAZHKUDAI, S. and SCHOPT, J., "Predicting Sporadic Grid Data Transfers," in *Proceedings of IEEE HPDC*, 2002.
- [141] VILLAMIZAR, C. and SONG, C., "High Performance TCP in ANSNET," *ACM Computer Communication Review (CCR)*, vol. 24, Oct 1994.
- [142] WOLSKI, R., SPRING, N., and HAYES, J., "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Journal of Future Generation Computing Systems*, vol. 15, no. 5-6, pp. 757–768, 1998.
- [143] WOLSKI, R., SPRING, N., and PETERSON, C., "Implementing a Performance Forecasting System for Metacomputing: the Network Weather Service," in *Proceedings of Supercomputing*, 1997.
- [144] XU, L., HARFOUSH, K., and RHEE, I., "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," in *Proceedings of IEEE Infocom*, 2004.
- [145] Y. CUI, Y. X. and NAHRSTEDT, K., "Optimal Resource Allocation in Overlay Multicast," in *Proceedings of ICNP*, 2003.
- [146] YAN, A. and GONG, W., "Time-driven Fluid Simulation for High-speed Networks," Tech. Rep. TR-96-CCS-1, Dept. of ECE, Univ. of Massachusetts, Dec 1999.
- [147] YANG, Y., KIM, M., and LAM, S., "Transient Behaviors of TCP-friendly Congestion Control Protocols," in *Proceedings of IEEE Infocom*, 2001.
- [148] YAUN, G., CAROTHERS, C. D., and KALYANARAMAN, S., "Large-Scale TCP Models Using Optimistic Parallel Simulation," in *Proceedings of PADS*, 2003.
- [149] YI, Y. and SHAKKOTTAI, S., "Hop-by-hop Congestion Control over a Wireless Multi-hop Network," in *Proceedings of IEEE Infocom*, 2004.
- [150] YUKSEL, M., SIKDAR, B., VASTOLA, K., and SZYMANSKI, B., "Workload Generation for *ns* Simulations of Wide Area Networks and the Internet," in *Proceedings of CNDC*, 2000.

- [151] ZEGURA, E., AMMAR, M., FEI, Z., and BHATTACHARJEE, S., "Application-Layer Anycasting: A Server Selection Architecture and Use in a Replicated Web Service," *IEEE/ACM Transactions on Networking*, vol. 8, p. 455, 2000.
- [152] ZHANG, H., "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks," *Proceedings of the IEEE*, vol. 83, Oct 1995.
- [153] ZHANG, H., LIU, B., and DOU, W., "Design of A Robust Active Queue Management Algorithm Based on Feedback Compensation," in *Proceedings of SIGCOMM*, 2003.
- [154] ZHANG, L., SHENKER, S., and CLARK, D., "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," in *Proceedings of SIGCOMM*, 1991.
- [155] ZHANG, Y., DUFFIELD, N., PAXSON, V., and SHENKER, S., "On the Constancy of Internet Path Properties," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001.
- [156] ZHANG, Y., ROUGHAN, M., LUND, C., and DONOHO, D., "An Information-Theoretic Approach to Traffic Matrix Estimation," in *Proceedings of SIGCOMM*, 2003.
- [157] ZOU, L. and AMMAR, M., "A File-Centric Model for Peer-to-Peer File-Sharing Systems," in *Proceedings of IEEE International Conference on Network Protocols*, 2003.
- [158] ZUKERMAN, M., NEAME, T., and ADDIE, R., "Internet Traffic Modeling and Future Technology Implications," in *Proceedings of IEEE Infocom*, 2003.