Virtual InterNetwork Testbed (VINT): methods and systems

Final Report

For

Contract DABT63-96-C-0105

Prepared by

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Sponsored by

Defense Advanced Research Projects Agency
3701 N. Fairfax Drive
Arlington, VA 22203-1714

Monitoring Agency

Directorate of Contracting
ATZS-DKO-I
P.O. Box 12748
Fort Huachuca, AZ 85670-2748

February 21, 2000

REPORT DOCUMENTION PAGE

Form Approved
OMB No. 0704.0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	21 February 2000	Final Report 9-26-96 to 2-21-00

4. TITLE AND SUBTITLE
Virtual Internetwork Testbed (VINT): Methods and Systems

5. FUNDING NUMBERS
C: DABT63-96-C-0105

6. AUTHOR(S)
Lee Breslau

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Directorate of Contracting
Attn: ATZS-DKO-I
Post Office Box 12748
Fort Huachuca, AZ 85670-0414

10. SPONSORING/MENTORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release, Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The goal of this project, as outlined in our original proposal, was to enhance the state of the art of network protocol design through the creation of a network simulator and related tools, and to validate this software by using it in our ongoing research. As described below, we have met these objectives. This VINT Project has made several releases of the ns simulator, the nam network animator and other related tools (e.g., topology generators). In addition, this simulator has been used in our ongoing research program, resulting in several technical papers on a range of topics. More importantly, the simulator has been widely adapted by a broad community of networking researchers and has been used in a wide range of research reflected in scores of technical papers.

14. SUBJECT TERMS
ns simulator, Asymptotic Behavior of SRM, Measurement-Based Admission Control, Core-Stateless Fair Queueing, Web Caching

15. NUMBER OF PAGES
95

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

VINT Project Final Report

Xerox PARC

February 21, 2000

Introduction

This document and references herein constitute a final report of the DARPA-funded VINT Project (Virtual InterNetwork Testbed) at the Xerox Palo Alto Research Center. This project has been a collaboration between Xerox PARC, USC Information Sciences Institute and the Lawrence Berekeley National Lab. While much of the work was accomplished jointly, this report will focus primarily on contributions at Xerox.

The goal of this project, as outlined in our original proposal, was to enhance the state of the art of network protocol design through the creation of a network simulator and related tools, and to validate this software by using it in our ongoing research. As described below, we have met these objectives. The VINT Project has made several releases of the *ns* simulator, the *nam* network animator and other related tools (e.g., topology generators). In addition, this simulator has been used in our ongoing research program, resulting in several technical papers on a range of topics. More importantly, the simulator has been widely adopted by a broad community of networking researchers and has been used in a wide range of research reflected in scores of technical papers.

ns simulator

The primary development effort of the VINT project has been focused on the *ns* network simulator. Version 1 of the simulator was developed at the University of California at Berkeley prior to the start of the VINT project. This version, which had been used in important network research had two major drawbacks. First, the set of modules supporting network protocols and algorithms was fairly limited because it had only been used in the study of a limited number of research problems. Second, the simulator architecture was not ideal for future extensibility.

Under the auspices of the VINT project *ns* was re-architected, reflecting a more modular and extensible design. This new architecure is embodied in version 2 of the simulator. In addition, the functionality of the simulator, reflected in the number and kinds of network algorithms and protocols it implements, has been expanded dramatically. Additional functionality includes (but is by no means limited to) several routing protocols (both unicast and multicast), many variants of TCP, new router scheduling algorithms, multicast transport protocols, wireless networking and web caching. Several ancillary tools have also been

released to support such things as network animation, scenario generation and topology generation. Specific contributions at Xerox to the simulator include modules supporting model-based and trace-based generation of simulator traffic, router scheduling algorithms (e.g., DRR), web caching functionality (server, proxy and client modules), and functionality for support of real-time services (signalling protocol, scheduling and admission control algorithms, token bucket filters). In addition we made contributions to the core simulator infrastructure in areas such as event scheduling, random number generation, random variable support and scaling, and we contributed to the development of the network animator *nam*.

The VINT project has made several releases of *ns* version 2 (the most recent being version 2.1b6 in January of this year.) It has gained widespread acceptance among the networking research community and it has had a broad impact on the work of this community. Anecdotally, it is by far the most commonly used simulator by researchers interested in the design and testing of new protocols and algorithms for the Internet. This is reflected in the following:

- The simulator has been downloaded by hundreds of sites worldwide.
- The mailing list for users of the simulator (ns-users@mash.cs.berkeley.edu) generates several hundred messages per month.
- It is by far the most commonly used simulator in papers submitted to major academic conferences (such as Sigcomm and Infocom.)
- Many pieces of contributed code have been produced by the user community, further enhanching the functionality of the simulator.

The *ns* simulator is available at http://www-mash.cs.berkeley.edu/ns/. Additional information about the simulator and the broader contributions of the project is available in [1]. The network animator, *nam*, is described in [5].

Research Using *ns*

A critical factor in the success of the VINT project is that the software it has produced has been employed to support ongoing network research of the project members. Subjecting the simulator to the stress of daily use has served to validate its design, highlight weaknesses, and inform continuing improvements and development of the simulator. This strategy is largely responsible for producing a piece of software that has widespread use and impact. In this section we describe some of the ways in which *ns* has been used in research. We confine our discussion to work performed at Xerox PARC. More complete listings of research making use of *ns* can be found at http://www-mash.cs.berkeley.edu/ns/ns-research.html and http://netweb.usc.edu/vint/publications.html.

Measurement-Based Admission Control

The purpose of this research was to compare the performance of several measurement-based admission control algorithms that had been proposed in the literature. Little comparison

among these algorithms existed previously. This study required the addition of new functionality to *ns* to implement a scheduling algorithm for real-time services, a signalling protocol to support admission control, and the admission control algorithms themselves. The architecture embodied in version 2 of the simulator greatly facilitated this task. Specifically, the modular architecture enabled clean separation of functionality where appropriate (e.g., between measurement-based admission control algorithms and load estimators) and the object-oriented nature of the simulator provided a convenient platform on which to program several different admission control algorithms. This study is described in [4]. In addition to the valuable insights learned from this study, we hope that by implementing the existing algorithms in the simulator, it will be much easier for researchers proposing new algorithms to evaluate their algorithms.

Core-Stateless Fair Queueing

Core-Stateless Fair Queueing (CSFQ) is an algorithm whose aim is to provide isolation from misbehaving flows and fairness between flows without requiring core routers to maintain per flow information. This provides the benefits of Fair Queueing while overcoming its scalability problems. CSFQ was evaluated using simulation to compare its performance to that of FIFO scheduling with tail dropping, FIFO scheduling with RED, and DRR. This simulation study exposed one of the key benefits of *ns*. We were able to leverage the existing broad range of functionality available in *ns* (e.g., TCP, DRR, RED, UDP) and avoid duplication of work. Hence, researcher effort could be focused on implementing and studying a new algorithm (CSFQ) while avoiding duplication of previous implementation efforts. The results of the CSFQ study are presented in [7].

Web Caching

We also used *ns* in a study of web caching. The goal of this research was to design a scalable web cache consistency architecture. The simulation evaluation consisted of a comparison of the proposed architecture to other schemes, such as Time-To-Live based algorithms and traditional invalidation approaches. As in the admission control work described above, the modular nature of the simulator facilitated the implementation of these competing designs (e.g., as different variants of web caches.) This work resulted in significant new web-related functionality in *ns*, including support for web servers, proxy caches and clients. A more complete description is available in [8].

Asymptotic Behavior of SRM

In another study, *ns* was used to study the global loss recovery in Scalable Reliable Multicast (SRM). One interesting aspect of the simulations in this study is that they did not use the existing node and routing structures in *ns*. This study was interested in scaling behavior and at the time it was performed, many of the subsequent enhancement to *ns* that improved its scaling were not yet available. Nonetheless, the simulator was beneficial to this study as the node and routing structures could be replaced easily with components specially tailored for the problem at hand while retaining the core simulator event handling

mechanism. This demonstrates that as important as the particular protocol and algorithms implemented in the simulator is the extensible framework that allows the simulator to be modified to support virtually any problem requiring event driven simulation. This study of scaling in SRM is reported in [6].

Service Priority and Adaptive Applications

This study examined the impact of multiple levels of scheduling priority on delay adaptive applications (e.g., VAT audio tool). As with our other work, having a simulator with the existing functionality of *ns* limited the amount of work needed to perform the study. In this case, we only needed to add different receiver behaviors to the simulator, taking advantage of existing traffic generation, router and link functionality. This work is reported in [2].

Priority Dropping and Layered Video

While the prior work looked at scheduling priority, the final study we mention examined the effect of drop priority on layered video applications. Prior work had posited that priority dropping had poor incentive properties while providing good performance. In this study, with the aid of simulation, we showed that the performance of priority dropping was not as good as expected while the incentive properties were better than anticipated. This work is described in [3].

References

[1] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving simulation for network research. To appear in IEEE Computer, March 1999.

[2] Sandeep Bajaj, Lee Breslau, and Scott Shenker. Is service priority useful in networks? *ACM Sigmetrics*, pages 66–77, June 1998.

[3] Sandeep Bajaj, Lee Breslau, and Scott Shenker. Uniform versus priority dropping for layered video. In *Proceedings of ACM Sigcomm*, pages 131–143, September 1998.

[4] Lee Breslau, Sugih Jamin, and Scott Shenker. Comments on the performance of measurement-based admission control algorithms. In *IEEE Infocom*, March 2000.

[5] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network visualization with the VINT network animator nam. To appear in IEEE Computer, March 1999.

[6] Suchitra Raman, Steve McCanne, and Scott Shenker. Asymptotic behavior of global recover in SRM. In *ACM Sigmetrics*, pages 90–99, June 1998.

[7] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of ACM Sigcomm*, pages 118–130, September 1998.

[8] Haobo Yu, Lee Breslau, and Scott Shenker. A scalable web cache consistency architecture. In *Proceedings of ACM Sigcomm*, pages 163–174, September 1999.

Improving Simulation for Network Research *

Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall,
Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy,
John Heidemann, Polly Huang, Satish Kumar, Steven McCanne,
Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu,
Haobo Yu, Daniel Zappala

USC Computer Science Department Technical Report 99-702b

March 1999 (revised September 1999)†

Abstract

New protocols and algorithms are being developed to meet changing operational requirements in the Internet. Simulation is a vital tool to quickly and inexpensively explore the behavior of these new protocol across the range of topologies, cross-traffic, and interactions that might occur in the Internet. This paper describes ns, a widely used, multi-protocol network simulator designed to address the needs of networking researchers. Ns provides multiple levels of abstraction to permit simulations to span a wide-range of scales, emulation, where real-world packets can enter the simulator. We describe the ns architecture and examine the range of ways simulation and ns are used in networking research.

Keywords: network protocol design, simulation, Internet protocols, split-language programming, ns, nam

1 Introduction

In recent years, the Internet has grown significantly in size and scope, and as a result new protocols and algorithms are being developed to meet changing operational requirements in the Internet. Examples of such requirements include quality of service support, multicast transport, security, mobile networking, and policy management. Development and evaluation of protocols and algorithms for these domains requires answering many design questions. Although small-scale evaluation in a lab, wide-area testbeds, and custom simulators can all be valuable, each has significant shortcomings. These approaches often lack the wide mix of traffic and topologies found in real networks, they can incur substantial expense, and repetition of experiments under controlled conditions can be difficult.

Multi-protocol network simulators can provide a rich environment for experimentation at low cost. A common simulation environment used across disparate research efforts can provide substantial benefits to the networking community. These benefits include improved validation of the behavior of existing protocols, a rich infrastructure for developing new protocols, the opportunity to study large-scale protocol interaction in a controlled environment, and easier comparison of results across research efforts.

The VINT project is attempting to facilitate the design and deployment of new wide area Internet protocols by providing network researchers with an improved set of simulation tools. This paper presents the VINT simulation framework and describes how it aims to meet many of the simulation needs of the network research community. We begin by identifying the requirements of a multi-protocol network simulator, after which we describe how VINT's ns simulator addresses these requirements. We then present the software architecture of ns, which provides an extensible framework within which new protocols can be developed. We then show several examples of ways in which ns has been used in protocol design and development, and

*This research is supported by the Defense Advanced Research Projects Agency (DARPA) through the VINT project at LBL under DARPA Order E243, at USC/ISI under DARPA grant ABT63-96-C-0054, at Xerox PARC under DARPA grant DABT63-96-C-0105.

†Originally published in March, 1999, this technical report was updated in September, 1999 (one section was moved and a number of typos were fixed). This technical report has been accepted to appear in IEEE Computer Magazine.

we evaluate the success and shortcomings of the VINT effort. We conclude by discussing previous work on network simulation and related topics, and by describing future challenges. A companion paper describes nam, the network animation companion to ns [12].

Ns is publicly available at `http://www-mash.cs.berkeley.edu/ns/` and has been widely used by network researchers.

Alternatives to a Common Simulator (sidebar)

Testbeds and laboratory experiments are also important approaches to network research. Since they use real code, experiments run in testbeds or labs automatically capture important details that might be missed in a simulation. This approach also has drawbacks; testbeds are expensive to build, testbeds and labs can be difficult to reconfigure and share, and they have limited flexibility. In addition, some networking phenomena such as wireless radio interference can be difficult to reproduce experimentally, thus making it difficult to compare or evaluate protocol designs.

Protocol design using simulation usually begins with an individual investigator's simulations of isolated protocol elements using small-scale topologies and simplified/static assumptions about higher and lower level protocols. Because the startup costs are so high, no individual group has the resources to create a comprehensive and advanced networking simulation environment, leading to a lack of standardization and reproducibility of simulations constructed by different groups of designers. In the current paradigm, directly comparable data would be available only if each individual designer implemented, within their own simulator, all of the competing mechanisms. Very few research groups have the resources to do this, and it is often most effective to have a simulation component constructed by those who know most about the particular protocol represented by the component.

Related Work (sidebar)

Network Simulators Network simulation has a very long history. Ns itself is derived from REAL [22], which is derived from NEST [11]. Although we cannot list all relevant network simulators here, this section describes distinguishing features of network simulators and compares prominent examples with ns.

Simulators have widely varying focuses. Many target a specific area of research interest such as a particular network type or protocol like ATM or PIM multicast. Others, including ns, REAL, OPNET [10], and INSANE [25] target a wider range of protocols. The most general of these provide a simulation language with network protocol libraries (e.g. Maisie [3] and OPNET [10]). Very focused simulators model only the details relevant to the developer.

The engine of ns and other network simulators is a discrete event processor. Several complementary approaches have been taken to improve accuracy, performance, or scaling. Some simulators augment event processing with analytic models of traffic flow or queueing behavior (for example, OO [29] and fluid network approximations [23]) for better performance or accuracy.

Parallel and distributed simulation is a second way to improve performance. Several simulators support multiprocessors or networks of workstations [22, 3, 31]. Although ns is focused only on sequential simulation, the TeD effort has parallelized some ns modules [31].

Abstraction is a final common approach to improving simulator performance. All simulators adopt some level of abstraction when choosing what to simulate. FlowSim was the first network simulator to make this trade-off explicit [2]. As discussed in "Abstracting Simulation", ns supports several levels of abstraction.

A number of different simulation interfaces are possible, including programming in a high-level scripting language, a more traditional systems language [3], or sometimes both [10]. Some systems focus on allowing the same code to run in simulation and a live network (for example, x-Sim [6] and Maisie [3]). Most systems augment programming with a GUI shell of some kind Ns provides a split-level programming model where packet processing is done in a systems language while simulation setup is done in a scripting language. Nam [12] provides visualization output and is currently being enhanced to support simple scenario editing.

Network Emulation. Early work in network emulation included the use of "flakeways" (gateways that could alter or drop packets) and were used for early TCP/IP tests. More recent work has included special purpose stand-alone network emulators supporting packet delays and drops [1, 33]. These systems are usually implemented as kernel drop-in modules that intercept the IP layer packet forwarding path and thus appear to end stations as routers. Their capabilities are generally limited to simple packet manipulations and don't provide for interference from simulated cross traffic. Moreover, these systems do not include a general simulation capability as provided by ns.

2 Simulation Needs of Researchers

Simulation allows the evaluation of network protocols under varying network conditions. Studying protocols, both individually and as they interact with other protocols, under a wide range of conditions is critical to explore and understand the behavior and characteristics of these protocols. The VINT project, through the ns simulator and related software, provides several critical innovations that broaden the range of conditions under which protocols can be evaluated while making this experimentation tractable:

- **Abstraction:** Varying simulation granularity allows a single simulator to accommodate both detailed and high-level simulations. Networking protocols are studied at many levels, both at the detail of an individual protocol, and in the aggregation of many data flows and interaction of many protocols. The abstraction mechanisms in ns allow researchers to examine both of these issues without changing simulators, and to validate abstractions by comparing detailed and abstract results.

- **Emulation:** Most simulation experiments are confined to a single simulated world including only those protocols and algorithms included in the simulator. However, emulation, which allows a running simulator to interact with operational network nodes, can be a powerful tool in protocol design.

- **Scenario generation:** Testing protocols under an appropriate set of network conditions is critical to achieve valid and useful results. Automatic creation of complex traffic patterns, topologies, and dynamic events (i.e., link failures) can help generate such appropriate scenarios.

- **Visualization:** Tools that allow researchers to understand more easily the complex behavior in a network simulation are needed. Given the complex range of behaviors, and the large scale of the networks involved, merely providing tables of summary performance numbers does not adequately describe the behavior of the network. Visualization adds a dynamic representation to network behaviors, allowing researchers to develop better protocol intuition and aiding protocol debugging. Nam, a network animation tool, is described in a companion paper [12].

- **Extensibility:** The simulator must be easy to extend in order to add new functionality, explore a range of scenarios, and study new protocols. Ns employs a split programming model designed to make scripts easy to write and new protocols efficient to run.

In addition to these innovations, several engineering issues have substantial impact on a simulator's usability. First among these is the availability of a wide range of protocol modules in the simulator. This allows easy comparison of different approaches. It also reduces simulation development time enabling the researcher to focus on those aspects of the simulation relevant to the design question being studied. Second, validated protocols against which new variants can be compared are needed. Validation of TCP is illustrated in a separate paper [15]. Other protocols are validated in ns to the degree warranted by their maturity. Finally, given the significant number of protocol modules in ns and the interactions among them, mechanisms to prevent modifications in one module from breaking functionality in another are needed. To this end, ns includes many automated test suites that keep unintentional changes in behavior from creeping into the simulator.

In the following sections we expand on the innovations in ns and we describe its innovative software architecture.

3 VINT and the *ns* Simulator

The VINT project aims to bring a change in current protocol engineering practices by enabling the study of protocol interactions and scaling using a common simulation framework with advanced features. The public distribution of our system has helped to reduce the duplication of effort expended in the networking research and development community.

As mentioned above, the ns simulator includes several special features targeted at supporting large scale, multi-protocol simulations. These features include an alternative configuration for large-scale simulations, a capability to interface the simulator to a live network, automated simulation scenario generation facilities, and visualization. In the remainder of this section we describe the first three of these features. Visualization is described in a companion paper [12].

3.1 Abstracting Simulation

Computer resource limitations such as memory and processing time often constrain the number of net-

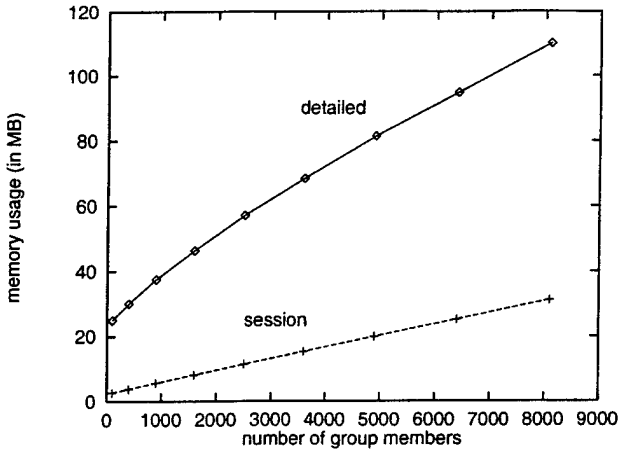


Figure 1: Session-level abstraction allows substantially larger numbers of multicast group members in the same amount of memory.

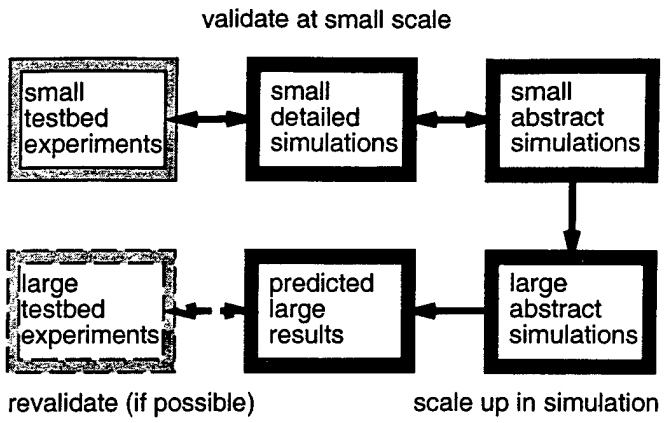


Figure 2: Validation of abstract simulations.

work objects (nodes, links and protocol agents) that can be simulated in a packet-level simulation. A scalable network simulator accommodates wide ranges of variation in each kind of network object, data in transit, and information collected. There are three complementary approaches to scaling a simulator: tuning the implementation, removing unnecessary simulation detail, and supporting parallelism. Other researchers have successfully explored parallel network simulation, and multiple efforts to parallelize ns are currently underway elsewhere (see "Related Work" for references to these approaches). Our efforts are focused on tuning our implementation and providing multiple levels of protocol abstraction. By eliminating less important details, substantial savings can be realized while preserving the basic validity of the model.

VINT provides several levels of abstraction in ns. The default simulator provides a *detailed* model with hop-by-hop packet forwarding and dynamic routing updates. *Centralized routing* replaces routing messages with a centralized computation, saving processing time and memory in exchange for slightly different timing in routing changes. *Session-level* packet forwarding replaces hop-by-hop packet flow with a precomputed propagation delay [21]. *Algorithmic routing* replaces shortest-path routing with tree-based routing, transforming $O(n \log n)$ memory requirements to $O(n)$. Each abstraction sacrifices some details to save memory, so abstractions must be applied only when appropriate.

By adjusting the simulation abstraction level, a user is able to trade off simulator performance versus packet-level accuracy. Increasing the level of abstraction provides the ability to perform increasingly large simulations, while decreasing the level of abstraction provides for a more realistic simulation. The session level simulator can abstract many details of links, nodes, and cross-traffic. Simulations can be run in both detailed and session level mode side-by-side to compare the performance and accuracy across the different levels of abstraction. Figure 1 shows the memory savings possible from session-level simulations for a particular scenario with large multicast groups.

The cost of abstraction is simulation accuracy. The degree to which accuracy is sacrificed, and the impact of this sacrifice on the validity of the results, varies greatly between simulation scenarios. For example, while the details of a particular media's approach to segmentation and reassembly are important for LAN simulations, they can be reflected adequately in the link's packet loss rate for higher-level WAN simulations.

To insure that abstraction does not substantially alter simulation results, Figure 2 shows how we validate simulations at small scale before projecting results at larger scales [21]. A quantitative analysis of SRM performance across detailed and session simulations suggests that while the timing of individual SRM events does vary, average aggregate behavior changes by only 3–9% in the cases we examined. Finally, we are also working on hybrid abstractions in which different portions of the same simulation operate in detailed and session levels of abstraction.

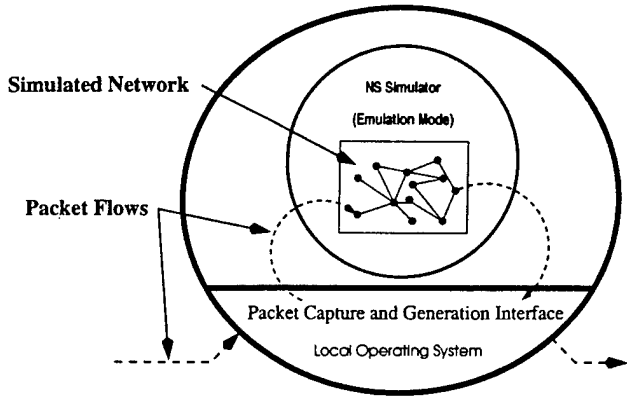


Figure 3: Emulation: live network traffic passes through simulated topology and cross-traffic.

3.2 Emulation interface

Ns includes an *emulation interface* which provides a method for network traffic to pass between real-world network nodes and the simulator. In combination with the simulator's tracing and visualization facilities, emulation provides a powerful analysis tool for evaluating the dynamic behavior of protocols and their implementations in end systems. An emulation scenario is constructed by placing the simulator as an intermediate node (or end node) along an end-to-end network path, as illustrated in Figure 3. The simulator contains a simulated network, and passes live network traffic through the simulation, subjecting it to the dynamics of the simulated network. The simulator's scheduler is synchronized with real-time, allowing the simulated network to emulate its real-world equivalent so long as the simulated network can keep pace with the real world events.

Emulation is useful beyond conventional simulation in evaluating both end system and network element behavior. With emulation, end system protocol implementations can be subjected to packet dynamics (e.g. drops, re-ordering, delays) that are difficult to reproduce reliably in a live network. Furthermore, by capturing traffic traces of live traffic injected into the simulation environment, visualization tools may be employed to evaluate the end system's dynamic responses. In the converse situation, network element behavior (e.g., a queueing or packet scheduling discipline) may be evaluated in relation to live traffic generated by real-world end stations. Such simulations are useful in identifying undesirable network element behavior prior to deployment in live networks.

The ns emulation facility is currently under development, but an experimental version has already proven useful in diagnosing errors in protocol implementation. For example, researchers at UC Berkeley have developed the MediaBoard, a shared whiteboard application using a version of the SRM protocol supported in the MASH toolkit [27]. The simulator is placed between groups of live end stations communicating using SRM. Multicast traffic passing between groups must traverse the simulator, and is subject to the dynamics of its simulated network. Visualization of traces taken within the simulation environment reveals end station retransmissions triggered by packets dropped or delayed within the simulated network. This use of emulation has helped to pinpoint time-dependent behaviors of the MediaBoard that are otherwise very difficult to diagnose.

3.3 Scenario Generation

In ns, a *simulation scenario* defines the input configuration for a simulation run. Scenarios are made up of several components: a *network topology*, including the physical interconnects between nodes and the static characteristics of links and nodes, *traffic models* which define the network usage patterns and locations of unicast and multicast senders, and *test generation*, which creates events such as multicast group distributions (receivers joining and leaving) and *network dynamics* (node and link failures) designed to stress an implementation. Automated generation of scenarios is important in the evaluation of protocol robustness. It allows researchers to cover much larger portions of the operational space than is possible through manual reconfiguration. Furthermore, by subjecting competing protocols to identical scenarios, meaningful comparative studies can be performed.

Topology Ns supports both pre-defined and automatically generated network topologies. Pre-defined topologies may be created manually or chosen from a *topology library* ranging from simple topologies to the topologies of real operational networks. Tools that automatically generate topologies provide the ability to create random topologies according to a set of specified parameters such as degree of connectivity, levels of hierarchy, and other features. Rather than create our own topology generation tools from scratch, we support the Georgia Tech Internetwork Topology Models (GT-ITM) package which creates flat random networks using a variety of edge distribution models, as well as hierarchical and transit-stub networks. In addition, the *tiers* system can be used to create three-level hier-

archical topologies similar to the transit-stub GT-ITM topologies [7].

The key challenge in topology generation is coming up with topologies that embody relevant characteristics of real networks. Once accomplished, the ns framework easily allows simulation of any generated topology. Hence, if new and better topology generation tools are developed in the future, using their output in ns likely requires at most a simple format conversion program.

Traffic Models Traffic generation support or *load libraries* provide a synthetic application workload model. For example, application traffic generation, call patterns, and multicast group membership dynamics may be included in a load library. As with the topology libraries, load libraries may be derived from empirical data, analytic models, or generated randomly to allow "what-if" investigation of particular parts of the operating region, even if that region is not currently observable in operational networks.

Ns provides a wide variety of source models that can be used in conjunction with both unicast and multicast transport protocols. At present, supported protocols include reliable delivery transport (e.g. several TCP variants, SRM), and unreliable transports with various semantics (e.g. RTP and UDP). For simulations of TCP, both bulk data and interactive sources are available. The former can model an FTP application while the latter, based in part on a model developed from traffic traces [8], models Telnet-like applications. We simulate web traffic with models based on Mah's measurements [24]. Other source models are available for non-flow controlled applications. These include a constant bit rate source, on-off sources using either exponential or Pareto distributions (the latter useful in generating self-similar traffic [37]), and a source that generates traffic from a trace file.

The composable framework of ns makes adding new traffic models fairly easy, and encourages construction of compound models out of the individual component. In simulations of Receiver-driven Layered Multicast (RLM), for example, a multi-layered video source was created by combining several CBR streams [28]. A similar approach was used to incorporate correlations of burstiness across layers in another study involving layered video [4].

In creating a simulation scenario, specifying individual traffic sources generated by the source models provided by ns is not always sufficient. Instead, in large network simulations, configuring a set of sources that in the aggregate generate suitable background traffic with desired characteristics (e.g., aggregate bandwidth, burstiness, self-similarity, etc.) is a challenge. Developing tools to help users synthesize simulation scenarios is an area of ongoing work in the VINT project.

Test Generation Choosing an appropriate set of test conditions for a simulation experiment is never simple, and evaluating the correctness of a protocol can be a daunting task. We developed a framework for *S*ystematic *T*esting of Protocol *R*obustness by *E*valuation of *S*ynthesized *S*cenarios (STRESS) [19, 20] in order to reduce the effort needed to identify pathological cases of protocol behavior. As the name implies, this framework integrates systematic synthesis of test scenarios with the VINT simulation environment of ns. We are in the process of developing automatic test generation algorithms for multicast protocols. These methods were applied to multicast routing protocol studies in ns. Several design errors were discovered and corrected with the aid of STRESS; the detailed results are presented in [19].

Future work in this area will consider the effect of a wider range of network failures on multicast routing. We will also investigate systematic methods for performance evaluation and sensitivity analysis of end-to-end protocols such as multicast transport. In addition, we plan to use the emulation interface in ns to conduct systematic conformance testing and performance profiling of actual protocol implementations.

4 Software Architecture

The ns software is constructed in a way intended to promote extension by users. The fundamental abstraction provided by the software architecture is "programmable composability". In this model, simulation configurations are expressed as a *program* rather than as a static configuration or through a schematic capture system. A simulation program *composes* objects dynamically into arbitrary configurations to effect a simulation configuration. By adopting a full fledged programming model for simulation configuration, the experimentalist is free to extend the simulator with new primitives or "program in" dynamic simulation "event handlers" that interact with a running simulation to change its course as desired.

Rather than adopt a single programming language that defines a monolithic simulation environment, we have found that different simulation functions require different programming models to provide adequate flexibility without unduly constraining performance. In particular, tasks like low-level event processing or

packet forwarding through a simulated router require high performance and are modified infrequently once put into place. Thus, they are best served by an implementation expressed in a compiled language like C++. On the other hand, tasks like the dynamic configuration of protocol objects and the specification and placement of traffic sources are often iteratively refined and undergo frequent change as the research task unfolds. Thus, they are best served by an implementation in a flexible and interactive scripting language like Tcl [30].

To this end, ns exploits a *split programming model*, where the simulation kernel—i.e., the core set of high-performance simulation primitives—is implemented in a compiled language (C++) while simulations are defined, configured, and controlled by writing an "ns simulation program" expressed in the Tcl scripting language. This approach can be a boon to long-term productivity because it cleanly separates the burden of simulator design, maintenance, extension, and debugging from the goal of simulation itself—the actual research experiments—by providing the simulation programmer with an easy to use, reconfigurable, and programmable simulation environment. Moreover, it encourages a programming style that leads to an important separation of mechanism and policy: core objects that represent simple and pure operations are free of built-in control policies and semantics and can thus be easily reused.

In our split programming model, fine-grained simulation objects are implemented in C++ and are combined with Tcl scripts to effect more powerful, higher-level "macro-objects". For example, a simulated router is composed of demultiplexers, queues, packet schedulers, and so forth. By implementing each primitive in C++ and composing them using Tcl a range of routers can be simulated faithfully. We can string together the low-level demultiplexers, queues, and schedulers to model an IP router perhaps with multicast forwarding support, or instead arrange them into a configuration that models a high speed switch with a new scheduling discipline. In the latter case, the switch could be easily extended with protocol agents (implemented entirely in Tcl) that modeled an experimental signaling protocol. Performance also guides our split programming model. Low-level event-level operations like route lookups, packet forwarding, and TCP protocol processing are implemented in C++, while high-level control operations like aggregate statistics collection, modeling of link failures, route changes, and low-rate control protocols are implemented in Tcl. Careful design is necessary to obtain a desirable trade-off between performance and flexibility, and this division often migrates during the course of a protocol investigation.

This composable macro-object model is naturally expressed using object-oriented design, but unfortunately, at the time we designed ns, Tcl did not provide support for object-oriented programming constructs nor did it provide very effective programming constructs for building reusable modules. Thus, we adopted an object-oriented extension of Tcl. Of the several Tcl object extensions available at the time, we chose the Object Tcl (OTcl) system from MIT [36] because it required no changes to the Tcl core and had a particularly elegant yet simple design. We further adopted a simple extension to OTcl called *TclCL* (for Tcl with classes) that provides object scaffolding between C++ and OTcl and thereby allows an object's implementation to be split across the two languages in congruence with our split programming model [27].

With the OTcl programming model in place, each macro-object becomes an OTcl class and its complexity is hidden behind a simple-to-use set of object methods. Moreover, macro-objects can be embedded within other macro-objects, leading to a hierarchical architecture that supports multiple levels of abstraction. As an example, high-level objects might represent an entire network topology and set of workloads, while the low-level objects represent components like demultiplexers and queues. As a result, the simulation designer is free to operate at a high level (e.g., by simply creating and configuring existing macro-objects) at a middle level (e.g., by modifying the behavior of an existing macro-object in a derived subclass) or at a low level of abstraction (e.g., by introducing new macro-objects or split objects into the ns core). Finally, class hierarchies allow users to specialize implementations at any one of these levels, for example extending a "vanilla TCP" class to implement "TCP Reno". The net effect is that simulation users can implement their simulation at the highest level of abstraction that supports the level of flexibility required, thus minimizing exposure to and the burden associated with unnecessary details.

5 Research with Ns

Network research simulations can often be categorized into one (or more) of a few broad themes. These include selecting a mechanism among several options, exploring complex behavior, and investigating unforeseen multiple protocol interaction. This section uses examples from the broad base of ns-based simulations in the networking community to demonstrate instances of each theme.

Selecting a Mechanism As in most design activities, much of the time spent in protocol design, redesign, and debugging concerns evaluation of the various alternatives to accomplishing a goal. Ns has seen broad use in developing TCP variants and extensions, exploring reliable multicast protocols, and in considering packet scheduling algorithm in routers.

As an example, ns has been used to explore several TCP variants and extensions such as selective acknowledgments [13], forward acknowledgments [26], explicit congestion notification (ECN) [14], and pacing [35]. These efforts were aided by the existence of a simulator-specific TCP implementation. By omitting application-specific baggage such as memory management and IP fragmentation, ns users were able to focus on the research issues such as packet retransmission policies and throughput.

Exploring Complex Behavior Complex behavior often takes the form of unexpected self-organization of dynamic systems. Examples include synchronization of periodic network traffic such as routing updates, TCP "ACK compression" in asymmetric or congested networks, undesired or unpredicted differential treatment of TCP flows due to RTT variations, contention for bandwidth reservations, and "ACK implosion" for large-scale reliable multicast protocols. In each of these domains, simulation has been a useful tool in helping to identify and understand these phenomena.

Error recovery in the Scalable Reliable Multicast (SRM) [17] is an example of exploration of complex behavior with ns. SRM was designed to support reliable group communication for large group sizes. It uses a probabilistic-based NACK protocol to achieve reliability. A receiver detecting a loss multicasts negative acknowledgement to the group. Each group member who has the missing data prepares to repair the error. To avoid repair implosion (everyone sending the repair at once), repairs are delayed by a random amount proportional to the estimated distance between the participants. While the original simulations of SRM were done in a stand-alone simulation tool, an SRM implementation has been added to ns, where it has been widely used to study SRM recovery behavior over a wide range of topologies [32] and variants [34]. This research was enabled by the public availability of of SRM in a well-documented simulator.

Comparing Research Results: A common research challenge is comparing a new protocol design against existing protocols. Comparisons of full protocols are often difficult because they may require a particular operating system or may not be widely available. By providing a publicly available simulator with a large protocol library, ns has become an ideal "virtual testbed" for comparing protocols.

The reliable multicast community have used ns widely for protocol comparison. In addition to the SRM variants previously described, Hänle used ns to compared the Multicast File Transfer Protocol [18], and DeLucia considered representative-based congestion control [9].

Multi-protocol interactions Multiple protocol interactions include the impact of protocol operation at one layer upon another layer (e.g. http on TCP, reservations on datagram delivery) or the interaction of unrelated protocols (e.g. the effect of uncontrolled traffic sources on congestion-controlled traffic flows or routing stability on transport layer performance). The problem with studying protocol interactions is that it requires twice the effort of studying a single protocol: the designer must understand and implement protocols at all the relevant layers. Ns reduces this effort by providing a validated library of important protocols.

RED and TCP snooping are two examples where ns greatly aided protocol studies exploring interactions between TCP and router queueing policies (RED) and TCP and wireless networking (Snoop). Random Early Detection (RED) queue management suggests that routers should detects incipient congestion (before running out of buffer capacity) and signal the source [16]. Early work on RED began on an ancestor of ns; RED is now a standard part of the simulator. Snooping proposes that TCP performance can be improved if routers replay TCP segments lost due to transmission failure over a wireless hop [5]. Both of these approaches benefitted from the rich ns protocol library.

Protocols Investigated With Ns (sidebar)

Ns has been used to develop and investigate a number of protocols:

- TCP behavior: selective acknowledgements, forward acknowledgments, explicit congestion notification, rate-based pacing, over asymmetric links (satellite)
- router queuing policies: random early detection, explicit congestion notification, class based queueing

- multicast transport: Scalable Reliable Multicast (SRM) and variants (RPM, scalable session messages), PIM variants, router support for multicast, congestion control, protocol validation and testing, reliable multicast
- multimedia: layered video (RLM), audio and video quality-of-service, transcoding
- wireless networking: SNOOP and split-connection TCP, multi-hop routing protocols
- protocol response to topology changes
- application-level protocols: web cache consistency protocols

References to some specific papers can be found in the text and at the web page http://www-mash.cs.berkeley.edu/ns/ns-research.html. As an example of ns's use in networking community, it was the most commonly used simulator at SIGCOMM '98.

6 Evaluation

The VINT effort has benefited from the contributions of a wide number of users. The project itself spans four geographically-dispersed groups of developers, and the user community includes more than 200 institutions world-wide (based on messages posted to the mailing list). Ns includes a large amount of code contributed from this user community. Currently, we have two mechanisms for adding contributed code from users: we can point to the contribution on a "Contributed Code" web page, or we can incorporate the contributed code into the main ns distribution (typically with documentation and a validation test program). Code integrated into the main distribution will track ns as it evolves; experience stresses the importance of the automated validation tests in this process.

Although the ns user community has been steadily growing, there will always be times when a researcher finds it more convenient to write stand-alone code or to choose an alternative general-purpose simulator. A custom simulator can address exactly the problem faced by a researcher. Even though ns's abstraction techniques allow two orders of magnitude scaling, a researcher's custom simulator can get exactly the correct scaling behavior. Finally, a new simulator will avoid the cost of learning ns. However, we have found that researchers often underestimate the amount of infrastructure required to build a new simulator and interpret its results.

Wide use of a common simulation platform provides some very serendipitous effects, however. By providing a rich collection of alternatives and variants for frequently used functionality (e.g. for TCP and queueing variants), ns encourages researchers to incorporate these alternatives into the parameter space of their own simulations. Without the infrastructure of ns or a similar environment, it seems unlikely researchers would be able to cover such a rich parameter space due to the additional cost of developing such infrastructure. This is particularly true of experimental new approaches. For example, RED queue management in ns has been widely used in a range of simulations well before it was standardized and available in products. This availability has helped understanding and acceptance of RED and helped other researchers anticipate how their protocols will behave in future networks.

A disadvantage of ns is that it is a large system with a relatively steep initial learning curve. Availability of a tutorial (contributed by Marc Greis) and continuing evolution of the ns documentation has improved the situation, but ns's split programming model remains a barrier to some developers. As described in "Software Architecture", the choice of the fine-grain object decomposition is intentional because it allows two levels of programming. Simple scripts, topology layout, and parameter variation can often be done exclusively in OTcl. Although C++ is required to implement most new protocols, ns's object-oriented structure makes it fairly easy to implement variants of existing protocols. For completely new protocols, the large set of existing modules promotes re-use by the advanced programmer as is evident in ns' existing protocols and classes.

7 Conclusions

Simulation in network research plays the valuable role of providing an environment in which to develop and test new network technologies without the high cost and complexity of constructing testbeds. While not a complete replacement for testbeds, a standard framework for simulation used by a diverse set of researchers increases the reliability and acceptance of simulation results. Despite the benefits of a common framework, the network research community has largely developed individual simulations targeted at specific studies due to the considerable effort required to construct a general-purpose simulator. Because of the special purpose nature of such simulators, studies based on them often do not reflect the richness of experience derived from experimentation with a more extensive set of traffic sources, queuing techniques, and protocol models.

The VINT project, using ns as its simulator base and nam as its visualization tool, has constructed a common simulator containing a large set of models for use in network research. By including algorithms still in the research phase of development, users of the simulator are able to explore how their particular work interacts with these future techniques. Furthermore, because of the many protocols and models included with the system, researchers are often able to modify and construct their own simulations based on the provided models with relative ease. In several cases, modules developed outside the VINT project have been incorporated as a standard component to the simulator. We intend to further foster such contributions, and expect them to increase in the future.

While the VINT project so far has been relatively successful in achieving its goals, it remains to be seen how well the VINT project and the ns simulator will address the challenges of building on this success. The VINT project is an ongoing experiment in providing and using a multi-protocol simulator that allows researchers in the network research community to more easily build on each others' work. Future challenges for the VINT project include the development of mechanisms for the successful integration of code contributed by the user community, reducing the learning curve for using ns, further developing tools for large-scale simulations with a diverse traffic mix, and providing tools for newer areas of research such as mobility and higher-level protocols.

References

[1] AHN, J., DANZIG, P. B., LIU, Z., AND YAN, L. Evaluation of TCP Vegas: Emulation and experiment. In *Proceedings of the ACM SIGCOMM* (Cambridge, Massachusetts, Aug. 1995), ACM, pp. 185–195.

[2] AHN, J.-S., DANZIG, P., ESTRIN, D., AND TIMMERMAN, B. Hybrid technique for simulating high bandwidth delay computer networks. In *Proceedings of the ACM SIGMETRICS* (Santa Clara, CA, USA, May 1993), ACM, pp. 260–261.

[3] BAGRODIA, R. L., AND LIAO, W.-T. Maisie: A language for the design of efficient discrete-event simulations. *IEEE Transactions on Software Engineering 20*, 4 (Apr. 1994), 225–238.

[4] BAJAJ, S., BRESLAU, L., AND SHENKER, S. Uniform versus priority dropping for layered video. In *ACM SIGCOMM* (Sept. 1998), pp. 131–143.

[5] BALAKRISHNAN, H., PADMANABHAN, V., SESHAN, S., AND KATZ, R. A comparison of mechanisms for improving TCP performance over wireless links. In *Proceedings of the ACM SIGCOMM '96* (Stanford, CA, Aug. 1996), ACM, pp. 256–269.

[6] BRAKMO, L., AND PETERSON, L. Experiences with network simulation. In *Proceedings of the ACM SIGMETRICS* (1996), ACM.

[7] CALVERT, K., DOAR, M., AND ZEGURA, E. W. Modeling Internet topology. *IEEE Communications Magazine 35*, 6 (June 1997), 160–163.

[8] DANZIG, P. B., JAMIN, S., CÁCERES, R., MITZEL, D. J., AND ESTRIN, D. An empirical workload model for driving wide-area TCP/IP network simulations. *Journal of Internetworking: Research and Experience 3*, 1 (Mar. 1992), 1–26.

[9] DELUCIA, D., AND OBRACZKA, K. A multicast congestion control mechanism using representatives. Tech. Rep. USC-CS TR 97-651, Department of Computer Science, University of Southern California, May 1997.

[10] DESBRANDES, F., BERTOLOTTI, S., AND DUNAND, L. OPNET 2.4: an environment for communication network modeling and simulation. In *Proceedings of the European Simulation Symposium* (Delft, Netherlands, Oct. 1993), Society for Computer Simulation, pp. 609–614.

[11] DUPUY, A., SCHWARTZ, J., YEMINI, Y., AND BACON, D. NEST: A network simulation and prototyping testbed. *Communications of the ACM 33*, 10 (Oct. 1990), 64–74.

[12] ESTRIN, D., HANDLEY, M., HEIDEMANN, J., MCCANNE, S., XU, Y., AND YU, H. Network visualization with the VINT network animator nam. Tech. Rep. 99-703, University of Southern California, Mar. 1999.

[13] FALL, K., AND FLOYD, S. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review 26*, 3 (July 1996).

[14] FLOYD, S. TCP and explicit congestion notification. *ACM Computer Communication Review 24*, 5 (Oct. 1994), 10–23.

[15] FLOYD, S. Simulator tests. From ftp://ftp.ee.lbl.gov/papers/simtests.ps.Z, Oct 1996.

[16] FLOYD, S., AND JACOBSON, V. Random early detection gateways for congestion avoidance. *ACM/IEEE Transactions on Networking 1*, 4 (Aug. 1993), 397–413.

[17] FLOYD, S., JACOBSON, V., LIU, C.-G., MCCANNE, S., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. *ACM/IEEE Transactions on Networking 5*, 6 (Dec. 1997).

[18] HÄNLE, C. A comparison of architecture and performance between reliable multicast protocols over the MBone. Master's thesis, Institute of Telematics, University of Karlsrühe, 1997.

[19] HELMY, A., AND ESTRIN, D. Simulation-based 'STRESS' Testing Case Study: A Multicast Routing Protocol. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Montreal, Canada, July 1998), IEEE, pp. 36–43.

[20] HELMY, A., ESTRIN, D., AND GUPTA, S. Fault-oriented test generation for multicast routing protocol design. *Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE/PSTV'98), IFIP TC6/WG6.1 Join International Conference* (Nov. 1998), 93–109.

[21] HUANG, P., ESTRIN, D., AND HEIDEMANN, J. Enabling large-scale simulations: selective abstraction approach to the study of multicast protocols. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Montreal, Canada, July 1998), IEEE, pp. 241–248.

[22] KESHAV, S. REAL: a network simulator. Tech. Rep. 88/472, University of California, Berkeley, Dec. 1988.

[23] KESIDIS, G., AND WALRAND, J. Quick simulation of atm buffers with on-off multiclass markov fluid sources. *ACM Transactions on Modeling and Computer Simulations 3*, 3 (July 1993), 269–276.

[24] MAH, B. An empirical model of HTTP network traffic. In *Proceedings of the IEEE Infocom* (Kobe, Japan, Apr. 1997), IEEE.

[25] MAH, B. A. *INSANE Users Manual.* The Tenet Group Computer Science Division, University of California, Berkeley 94720, may 1996. `http://HTTP.CS.Berkeley.EDU/~bmah/Software/Insane/InsaneMan.ps`.

[26] MATHIS, M., AND MAHDAVI, J. Forward acknowledgement: Refining TCP congestion control. In *Proceedings of the ACM SIGCOMM '96* (Stanford, CA, Aug. 1996), ACM, pp. 281–291.

[27] MCCANNE, S., BREWER, E., KATZ, R., ROWE, L., AMIR, E., CHAWATHE, Y., COOPERSMITH, A., MAYER-PATEL, K., RAMAN, S., SCHUETT, A., SIMPSON, D., SWAN, A., TUNG, T.-L., WU, D., AND SMITH, B. Toward a common infrastructure for multimedia-networking middleware. In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (St. Louis, Missouri, May 1997), IEEE, pp. 39–49.

[28] MCCANNE, S., JACOBSON, V., AND VETTERLI, M. Receiver-driven layered multicast. In *ACM SIGCOMM* (Stanford, CA, U.S.A., Aug. 1996), pp. 117–130.

[29] MIKLER, A. R., WONG, J. S. K., AND HONAVAR, V. An object oriented approach to simualting large communication networks. *Journal of Systems Software 40* (1998), 151–164. huang folder: general simulator.

[30] OUSTERHOUT, J. K. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, MA, 1994.

[31] PERUMALLA, K., FUJIMOTA, R., AND OGIELSKI, A. TED—a language for modeling telecommunication networks. *ACM SIGMETRICS Performance Evaluation Review 25*, 4 (Mar. 1998).

[32] RAMAN, S., MCCANNE, S., AND SHENKER, S. Asysmptotic scaling behavior of global recovery in SRM. In *Proceedings of the ACM SIGMETRICS* (Madison, WI, USA, June 1998), ACM, pp. 90–99.

[33] RIZZO, L. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review 27*, 1 (Jan. 1997).

[34] SHARMA, P., ESTRING, D., FLOYD, S., AND ZHANG, L. Scalable session messages in SRM. Sumbitted to infocom '98?, USC technical report, Aug. 1997.

[35] VISWESWARAIAH, V., AND HEIDEMANN, J. Improving restart of idle TCP connections. Tech. Rep. 97-661, University of Southern California, Nov. 1997.

[36] WETHERALL, D., AND LINBLAD, C. J. Extending Tcl for dynamic object-oriented programming. In *Proceedings of the USENIX Tcl/Tk Workshop* (Toronto, Ontario, July 1995), USENIX, p. 288.

[37] WILLINGER, W., TAQQU, M., SHERMAN, R., AND WILSON, D. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. In *Proceedings of the ACM SIGCOMM* (Cambridge, Massachusetts, Aug. 1995), ACM, pp. 100–113.

Is Service Priority Useful in Networks?

Sandeep Bajaj Lee Breslau Scott Shenker*
{bajaj,breslau,shenker}@parc.xerox.com
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Abstract

A key question in the definition of new services for the Internet is whether to provide a single class of relaxed real-time service or multiple levels differentiated by their delay characteristics. In that context we pose the question: is service priority useful in networks? We argue that, contrary to some of our earlier work, to properly address this question one cannot just consider raw network-centric performance numbers, such as the delay distribution. Rather, one must incorporate two new elements into the analysis: the utility functions of the applications (how application performance depends on network service), and the adaptive nature of applications (how applications react to changing network service). This last point is especially crucial; modern Internet applications are designed to tolerate a wide range of network service quality, and they do so by adapting to the current network conditions. Most previous investigations of network performance have neglected to include this adaptive behavior.

In this paper we present an analysis of service priority in the context of audio applications embodying these two elements: utility functions and adaptation. Our investigation is far from conclusive. The definitive answer to the question depends on many factors that are outside the scope of this paper and are, at present, unknowable, such as the burstiness of future Internet traffic and the relative offered loads of best-effort and real-time applications. Despite these shortcomings, our analysis illustrates this new approach to evaluating network design decisions, and sheds some light on the properties of adaptive applications.

*This research was supported in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contracts DABT63-94-C-0073 and DABT63-96-C-0105. The views expressed here do not reflect the position or policy of the U.S. government.

1 Introduction

The Internet has traditionally provided applications with a single class of best-effort service. The performance requirements of elastic applications, such as file transfer and electronic mail, allow them to adapt to the changing delays and bandwidth provided by this service. More recently, the increasing bandwidth of Internet links as well as the increasing processing power of end hosts has focused widespread attention on the desire to use the Internet for the transport of real-time multimedia content, such as audio and video. The architecture and protocols needed to support the more stringent requirements of these applications have been the subject of considerable research and discussion in recent years. New resource reservation protocols, scheduling algorithms and admission control algorithms have been proposed in the academic literature.[1] Recently, two components of a new Internet architecture have been moved to Proposed Standard in the Internet Engineering Task Force (IETF): the reservation protocol RSVP [3, 17] and new network element services Guaranteed [12] and Controlled-Load [16] service (see also [2, 13] for overviews of this architecture). The key break from best-effort service, where sources need not notify the network before transmitting packets, is that for these real-time services flows must request service from the network – specifying their desired quality of service and their proposed traffic characterization – and the network can accept or reject their requests.

One issue that arose in the discussion of the Controlled-Load[2] service (and in the earlier Controlled-Delay service, which was supplanted by the Controlled-Load service) is whether or not to offer more than one priority level of service; that is, whether to have multiple levels of scheduling priority within the Controlled-Load service.[3] The current service definition offers only a

[1] The literature is far too vast to review here, but see [4, 5, 6, 7, 9, 15, 18] and references therein for a few representative examples.

[2] Controlled-Load service is a relaxed real-time service that provides low delay and low loss, but does not provide delay bounds.

[3] The analogous question remains even if real-time applications are supported by a best-effort network service, a solution advocated by some in the community. That is, would these applications be better suited by a single class of best-effort service (as exists today) or by multiple levels of service differentiated by delay? Our treatment remains valid, although we

single level of service, but the question remains as to what benefits offering additional levels of service would provide.[4] Offering multiple levels of service carries with it the cost of additional complexity to deal with signaling the priority level, merging reservations with different priority levels, and scheduling overhead. We do not discuss those costs here, but instead ask only how large a benefit multiple priority levels might offer. Clearly if such benefits are minimal then there is no need to incur the additional complexity; if the benefits are significant then one needs to more carefully assess the complexity costs.

At first glance, the benefit of multiple priority levels seems obvious. After all, applications have a wide spectrum of delay constraints, from interactive conferencing with its need for small network delays to playback of stored video which can easily tolerate large network delays. Given this wide disparity in delay requirements, it seems only natural that one can increase the overall welfare by offering different levels of service. One can model this analytically (following a very similar model in [14]).

In the following simple example with two kinds of applications we compare a network with two priority classes to one with a single class of FIFO service. Let U_i denote the performance level, or utility, of application i as a function of average network delay. Let V denote the total utility, or total value, of the applications: $V = U_1 + U_2$. Consider a network with a single link modeled by an exponential server (of rate $\mu = 1$) and flows modeled by Poisson arrival processes. Consider two types of network clients, with Poisson arrival rates $r = 0.25$ and with $U_1 = 21 - 10d_1$ and $U_2 = 2 - d_2$ where d_i represents the average queueing delay delivered to client i.[5] Thus, we have two clients with different sensitivities to delay. If we use FIFO service in the network, then $d_1 = d_2 = \frac{1}{(1-0.5)} = 2$ and so $V^{FIFO} = 1$. If we use strict priority service, with preemption, and give client 1 priority, then $d_1 = \frac{1}{(1-0.25)} = 4/3$ and $d_2 = \frac{1}{(1-0.25)(1-0.5)} = 8/3$ and $V^{priority} = 7$. Thus, the strict priority scheduling algorithm is more efficient – delivers a higher value of V at the same bandwidth – than FIFO. In fact, when compared to all possible scheduling algorithms, the strict priority scheduling algorithm gives the most efficient feasible allocation of delay for this simple example.

However, this model ignores an important aspect of the problem. Specifically, network utility does not depend only on the characteristics of the packet delivery services provided, but also on how applications deal with different levels of network service. Modern network applications, in contrast to the *rigid* audio and video applications designed for more predictable data delivery services such as the telephone network or cable-TV transmission infrastructures, are *adaptive*; that is, they adapt to the current network conditions. This adaptation can take on several forms; in this paper we consider a class of applications known as *delay adaptive*. We describe these applications in more detail in Section 2. Such adaptivity is now a central piece of the accepted design philosophy in the Internet. The ability of application adaptivity to cope with changing network conditions has strong bearing on the question we ask here. After all, if adaptive applications can adjust essentially without degradation under any reasonable network conditions, there would never be any need for multiple levels of service. In fact, some have made precisely this claim when arguing for a single level of service.

Whatever the extent of adaptivity's ability to mask network delay and jitter (i.e., changes in delay), it is certainly clear that because of this active adjustment, the dependence of an adaptive application's utility on the network service is quite complicated. Simply put, such adaptivity renders simplistic analyses such as the one above invalid. How an application reacts to the network service determines how its performance depends on the network service, and the application's performance sensitivities (to delay, loss of fidelity, or both) determines what adaptation algorithm is most appropriate. As we shall see, for a given packet delivery service the delay experienced can be less in a very delay-sensitive application than in a delay-insensitive one, because the former will use an adaptation algorithm that aggressively attempts to reduce delays. Yet, despite adaptivity's centrality as an Internet application design paradigm, most performance analyses of network designs are performed without careful attention to the adaptive nature of applications.[6] The central purpose of this work is to illustrate how one can incorporate the behavior of adaptive applications into the performance analysis of a network design decision. It turns out that differences in delay (and jitter) in network service can largely be masked by this adaptive behavior for applications that are sensitive to only one of delay or fidelity. Thus, the simplistic analyses based on rigid applications are misleading. However, applications that are sensitive to both delay and fidelity achieve significant performance benefits from additional priority levels under some traffic loads. Therefore, while adaptation is very effective, it is not a universal panacea.

These results do not translate into a facile answer to the question of whether or not to offer multiple levels of Controlled-Load service. Instead, they serve as a cautionary note against simplistic conclusions based on the analysis of more static applications, and also against the ability of adaptivity to remove all sensitivity to performance variations. Our study also highlights our current state of ignorance about how the perceived performance of audio and video applications depend on the underlying network dynamics. We hope that by clarifying the gaps in the current understanding future work can begin closing them.

would need to consider different mixtures of traffic including best-effort as well as real-time applications.

[4] Offering service priority is one form of *service discrimination* within the Internet, where different packets receive different service. Service discrimination can take several other forms; a network may provide unreserved or reserved service, service may be differentiated by dropping priority, or a network may provide pre-emptable and non-pre-emptable reserved services.

[5] Recall that the average delay in the M/M/1 queueing network considered here is just $d = \frac{1}{(\mu - r)}$. If we have two priority levels, with arrival rates r_1 and r_2 respectively, then the delays are given by $d_1 = \frac{1}{(\mu - r_1)}$ and $d_2 = \frac{\mu}{(\mu - r_1)(\mu - r_1 - r_2)}$.

[6] Some work, such as in [8, 10], do analyze different adaptation algorithms, but their purpose was to refine the adaptation algorithm, not ask what implication adaptation had for network design.

The remainder of this paper is organized as follows. Section 2 describes the class of applications we consider in this paper, and then discusses several forms of adaptive behavior. Section 3 presents the results of simulation experiments that study the impact of different classes of delay on the performance of applications. We conclude in Section 4 with a discussion of the implications of our findings.

2 Adaptive Applications

In this section, we describe the class of applications that motivates our work. We begin by describing what we refer to as *adaptive* applications.[7] Then we describe two adaptation algorithms, appropriate for use by audio applications, that we use in our later simulations. We focus on these audio applications because it is in this domain that adaptive algorithms have been most widely utilized. The extent to which delay adaptation is applied to video remains to be seen, but we expect the methodology we use here could be applied to video algorithms as well. Finally, we present our model of utility functions and describe the four classes of applications we consider.

2.1 Delay Adaptation

Consider a real-time audio or video application in the Internet. Such an application will typically sample its media source (e.g., an audio input device or video frame grabber) and then send packetized data over the network. Each packet experiences a variable amount of queueing delay in the network, in addition to the fixed propagation and transmission delays (assuming all packets follow the same path). Thus, the packet stream generated by the source arrives at the destination perturbed by the variable network delay. The receiver can remove some or all of the jitter induced by the network by buffering packets for later playback. We refer to the time for which a packet is buffered at the receiver as its *playback delay*. We refer to the *playback point* as the total delay from when a packet is sent until it is played at the receiver. For real-time data, such as audio or video, if a constant playback point is maintained for all packets then there is no loss of fidelity. Otherwise, the incoming signal is distorted and so there is a loss of fidelity in the application.

Determining the playback point for each packet is a key issue in the design of these applications. Any playback strategy can make use of timestamps in packets, such as those provided by the Real-time Transport Protocol (RTP) [11], to determine the relative send times of successive packets, and thus need not assume synchronized clocks at the sender and receiver. If the receiving application knows *a priori* the maximum possible delay experienced in the network it can buffer the first packet for this maximum before playing it. This will enable the receiver to remove all jitter from the signal, since all subsequent packets (other than those that may be lost in the network) will arrive before their playback points, thereby maintaining the proper offset from the previous packet. However, neither the current Internet best-effort service, nor the proposed Controlled-Load real-time service provides applications with information about maximum network delays. While the proposed Guaranteed Service does provide delay bounds, it is an expensive service to provision (precisely because it provides delay bounds), and therefore is not likely to be widely utilized.

Since these applications must operate in environments where no end-to-end delay bound is known, they must be prepared to adjust the playback point of packets based on changing network conditions. That is, the application determines dynamically (in ways we describe below) how long to buffer each packet before playing it out. Buffering will remove some of the jitter introduced by the network, but periodic adjustments to the playback point will cause some distortion in the received signal. Hence, the application's performance is not merely a function of the service provided by the network. Rather, it is also a function of both the total delay in playing back the data (including network and playback delays) and the distortion incurred by varying the playback point over time.

Different applications will have different levels of sensitivity to these performance measures. Throughout this paper we characterize applications by the degree to which they care or do not care about each of delay and distortion. We simplify our study by considering four prototypical applications: those that care about both delay and distortion, those that care about delay only, those that care about distortion only, and those that care about neither. The notion of "caring" or "not caring" (or "sensitivity" and "insensitivity" which we use equivalently) are relative terms. For instance, even a delay insensitive application, such as the playback of recorded audio, has some delay constraints dictated by the user (e.g., delays of minutes, or several seconds, might not be tolerable). Similarly, a distortion insensitive application, such as an interactive session in which some distortion can be tolerated, also has limits to this tolerance (e.g., the speech need not be faithfully reproduced, but it must at least be intelligible). Our point, when using the terms "not caring" or "insensitive", is that these applications will be able to tolerate larger delays, or larger distortions, than other applications while still achieving acceptable performance.

2.2 Adaptation Algorithms

We expect the particular adaptation algorithms employed by delay adaptive applications to vary. For example, an interactive application may employ an adaptation algorithm that attempts to reduce the playback delay (and hence the total delay). Such a strategy, which we will refer to as *aggressive* adaptation, increases the risk that some packets will arrive after their scheduled playback points, in which case they will have to be dropped or the playback point will have to be adjusted. In either case, the resulting signal is significantly distorted. Alternatively, a non-interactive application, such as playback of recorded content or a one-way broadcast, may employ a more conservative adaptation algorithm, choosing larger playback delays and reducing the probability that packets arrive after

[7] Application adaptivity can actually take several forms. In this paper we consider the specific class of *delay-adaptive* applications. *Rate-adaptive* applications vary their sending rate in response to changing network conditions.

their playback points.

We now describe two adaptation algorithms, which we refer to as *conservative* and *aggressive*, that we use later in our simulations. These algorithms are appropriate for use by audio applications that generate blocks of data interspersed with periods of silence (as would be generated by a silence suppression mechanism). The general strategy they employ is to pick a playback point for the first packet in each talkspurt such that all packets within the talkspurt will (ideally) arrive before their respective playback points. When a packet arrives late (i.e., after it should have been played) the adaptation algorithm has two choices. It can discard the packet, or it can play the packet and adjust the playback points of subsequent packets in the talkspurt. It is unclear in general which strategy is better. For the purposes of this study we adopt the latter strategy based on previous studies (e.g., [1]) that have observed correlations in packets with large delays and on our own simulations that have shown that late packets generally arrive in bursts; given the choice between discarding several packets or introducing some jitter, the latter seems preferable.

The conservative algorithm fixes the playback point of the first packet to a predetermined value. All subsequent packets maintain the same playback point assuming they arrive in time. When a packet arrives late, the playback point is doubled and this new playback point is used for all subsequent packets. Hence, the playback point is adjusted upward but never downward. This algorithm attempts to maintain fidelity at the expense of higher delay. The second algorithm is more aggressive, yielding lower delay at the expense of increased distortion. It is taken from the adaptation algorithm in the *Visual Audio Tool (VAT)* developed at Lawrence Berkeley National Laboratory with minor modifications.[8] This algorithm estimates a measure of variance based on the difference in delay between successive packets. At the start of each talkspurt a new playback delay is computed using the previous offset and the estimate of variance.

2.3 Performance Measures and Utility Functions

The performance of an adaptive application can be characterized by two measures: delay and distortion. These measures are a function of both the packet delivery service and the adaptation algorithm. Delay includes both delays experienced in the network as well as playback delays. Distortion captures changes in the playback point. For our delay measure, we use the average of the delay experienced by each packet. Thus,

$$Delay = \frac{\sum_i d_i}{n}$$

where d_i is the delay experienced by packet i and n is the total number of packets. For distortion, an individual distortion value is first computed for each packet as follows:

$$dist_i = min(\frac{d_i - d_{i-1}}{t_i - t_{i-1}}, thresh)$$

[8]Source code for the VAT application, including its adaptation algorithm, is available at http://www-nrg.ee.lbl.gov/vat.

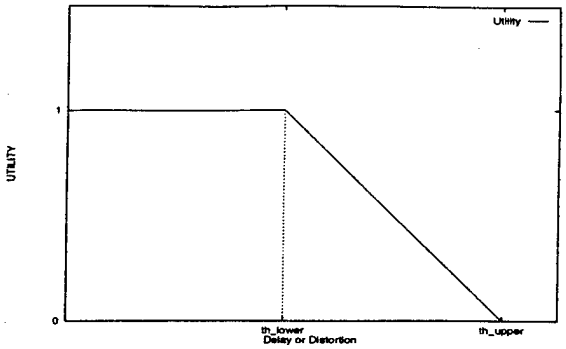


Figure 1: General form of utility functions for delay and distortion.

where t_i is the send time of packet i, and $thresh$ is a constant, set to 2 in our experiments. Including the difference in send times of successive packets in the denominator gives higher weight to intra- rather than inter-talkspurt adjustments in the playback point. $thresh$ bounds the maximum per packet distortion penalty (at 40 ms since inter-packet times, $t_i - t_{i-1}$, are 20 ms within talkspurts in our source model.) The overall measure of distortion is merely the average of the per-packet measures:

$$Distortion = \frac{\sum_i dist_i}{n}$$

We normalize these values so that they are reported in milliseconds of distortion per packet.

From the delay and distortion performance measures, we derive measures of application performance or utility using *utility* functions. The general form of the utility functions we use (for both delay and distortion) is shown in Figure 1. These functions have the following characteristics. First, below some threshold (th_{lower} in the figure), applications do not suffer any perceptible effects from delay or distortion. Second, above another threshold (th_{upper}), applications derive no utility. Finally, between th_{lower} and th_{upper}, utility degrades linearly. Total utility for an application is merely the product of its individual delay and distortion utility values:

$$U_{tot} = U_{del} \times U_{dis}$$

An application must receive good performance on both measures to achieve high overall utility, and poor performance on either leads to overall unhappiness. For each of the utility functions, we vary the values of th_{lower} and th_{upper} to capture the relative sensitivity or insensitivity of applications to each of delay and distortion. Thus, for sensitive applications, th_{lower} and th_{upper} will be set to lower values than for insensitive ones.

The relationship between performance measures and application utility is certainly not a simple as the model we use. For instance, actual functions are likely not linear, may depend on how performance varies in time rather than on static measures, and may involve subtle interactions between delay and distortion. However, we believe that our simple model captures the

most important aspects of performance and utility, and at the very least is sufficient for this initial investigation. Subsequent research into the true nature of these utility functions would provide useful guidance for our modeling; at present, the relevant literature is quite sparse.

3 Simulations

We used discrete event simulation to study the effects of service priority on application utility given our model of applications and their utility described above. Our simulation environment built on version 2 of the *ns* simulator developed at the University of California at Berkeley. To the base simulator, which provides event management, measurement functions, packet transmission and traffic generation, we added additional functionality, such as adaptation algorithms, utility functions and priority queueing, needed to carry out our experiments.[9] In this section we first describe our simulation methodology, and then report our results.

3.1 Simulation Model

The purpose of our simulation experiments was to compare the utility of a network providing a single level of service for real-time applications to one providing two levels of service in the simplest possible network context. The simulation topology consisted of a single 2Mbps link connecting two nodes.[10] Each simulation consisted of a set of source/receiver pairs generating background load on the network and test applications whose performance and utility was measured. This study is concerned with real-time applications, so we assume the existence of real-time services in the network. However, since we directly control the level of offered load in our experiments (by adjusting the number of source/receiver pairs in the network), we did not need to model resource reservation or admission control functions explicitly in the simulated network. Instead, we assume that all traffic in the network has passed an admission control test, has an installed reservation, and is receiving real-time service. No best-effort traffic was included in the simulations. We discuss the implications of this later in Section 4. When testing a single level of service, all packets are served in a single FIFO queue. For priority service, we used two FIFO queues served in strict priority order (without pre-emption). When reporting our results, we refer to these as the FIFO and Priority tests, respectively. We will also sometimes refer to the high priority service in the Priority tests as Level 1 service, and the low priority as Level 2. In all experiments, offered load was controlled and enough buffers provisioned so that there were no dropped packets.

Each experiment was repeated with the test applications using the conservative and aggressive adaptation algorithms described in Section 2.2. In addition, experiments were run with a non-adaptive, or *rigid*, receiver algorithm, which we describe in Section 3.2.2. At a given level of offered load, measures of delay and distortion were computed for each algorithm (conservative, aggressive, rigid) and for each network service (FIFO, Priority Level 1, Priority Level 2). The performance measures were mapped into application utility as follows. First, we chose an appropriate adaptation algorithm for each of the four types of applications (recall the two by two taxonomy of applications based on their level of sensitivity to each of delay and distortion.) Applications that were sensitive to both delay and distortion and applications that were sensitive to delay only used the aggressive algorithm.[11] Applications that were sensitive to distortion only, and those that were sensitive to neither, used the conservative algorithm.

Given an application's performance sensitivities and adaptation algorithm, utility values for each kind of service were computed. The following values of th_{lower} and th_{upper} were used. For delay sensitive utility, we used values of $th_{lower} = 50$ ms and $th_{upper} = 100$ ms. For delay insensitive utility, we set $th_{lower} = 1000$ ms and $th_{upper} = 2000$ ms. For distortion sensitive applications we used $th_{lower} = .25$ ms/pkt and $th_{upper} = 1.0$ ms/pkt. For distortion insensitive applications, we set $th_{lower} = 2.5$ ms/pkt and $th_{upper} = 10.0$ ms/pkt. The delay sensitive values are set to represent tolerances for interactive applications.[12] The delay insensitive utility is appropriate for non-interactive playback applications, but where response time does matter to the user (i.e., pointing and clicking and receiving stored audio over the network). Deciding on distortion values for utility was difficult without a better sense of the actual effect of playback distortion on users. We chose values such that distortion sensitive and insensitive applications perceived distortion in very different manners.

Two different kinds of source models were used in the simulations. Test sources were represented by an on/off source model that generates "talkspurts" and idle periods like those generated by voice data with silence suppression. Sources transmit 200 byte packets at a rate of 80kbps during "on" periods and are silent during "off" periods. These parameters are consistent with 8 KHz 8-bit mu-law PCM audio sent in 20 ms frames with 40 bytes of overhead per packet. Both the on and off times were taken from exponential distributions with a 500 ms average.

Background traffic was generated by capturing a trace of low frame rate video taken of one of the authors during a network videoconference. The trace,

[9] The *ns* simulator is available at http://www-mash.cs.berkeley.edu/ns. Our extensions to the simulator, and the simulation scripts we ran to generate the results in this paper can be found at ftp://ftp.parc.xerox.com/pub/net-research/sigmetrics98.

[10] If, as many have claimed, there is a single bottleneck link on any network path, then the simple topology is sufficient to understand the behavior of application adaptation algorithms. The verification of this claim, or a better understanding of the effect of queueing delays at multiple hops, is a subject for future study.

[11] While it should come as no surprise that this algorithm is appropriate for delay sensitive distortion insensitive applications, it is not clear a priori which algorithm is better for applications that are sensitive to both delay and distortion. We determined, through experimentation, that the aggressive adaptation algorithm was more effective than the conservative adaptation algorithm for these applications.

[12] Note that in addition to the variable delay captured by these utility functions, there will be other fixed sources of delay.

which lasts for approximately 1,400 seconds and has an average rate of 32kbps, was produced by the *vic* video program.[13] Within a single simulation run, multiple sources sending from this trace started at random points in the trace file to avoid synchronization. This background traffic is more bursty than the traffic generated by the on/off source model. Space prevents us from presenting data using additional kinds of background traffic, such as other source models or video traces produced with different codecs or content. However, as we discuss in Section 4, additional traffic models would not provide us with a more definitive answer to our question.

Each data point in the graphs below is an average of 20 simulation runs each with different seeds to the random number generator. Individual runs lasted for 5,000 simulation seconds.

For each scenario (adaptation algorithm, service discipline) the number of sources generating background traffic was varied to generate different load levels. Our results are generally reported as a function of utilization, with each point on the x-axis representing a fixed number of background sources. These values are reported in terms of percentage of the link bandwidth generated by the background and test sources together. For the Priority experiments, 25% of the background traffic was in Level 1 (high priority) and 75% was in Level 2. All traffic in our experiments represents real-time traffic.

In reality, best-effort traffic will continue to make up an important part of Internet traffic. Our analysis does not suffer by omitting best-effort traffic from the model, since we assume it would receive lower priority than real-time traffic, and therefore would not impact the delays seen by real-time traffic. However, the presence of best-effort traffic would impact the amount of real-time traffic in the network. If one assumes, for instance, that 20% of network traffic will be best-effort, then utilization levels higher than 80% in our experiments fall outside of expected operating conditions. Thus, an important, but unanswerable, question in analyzing our results is how much of the link bandwidth will be taken by best-effort traffic. If it is a large percentage, then one need only consider fairly low levels of real-time utilization, and there the comparison of two levels of priority versus one is quite different than at higher levels of utilization.

3.2 Results

We present our results in three stages. First, we present "raw" data of queueing delays and jitter induced by the network. This shows the service provided by the network, before any processing by the applications. Then we add the application performance and utility to our analysis in the context of non-adaptive applications. Finally, we present results of experiments using the adaptive algorithms described earlier. This incremental approach demonstrates the importance and impact of the specific characteristics of applications (i.e., their utility and adaptation algorithms) we consider.

[13]The *vic* program is available at http://www-nrg.ee.lbl.gov/vic/.

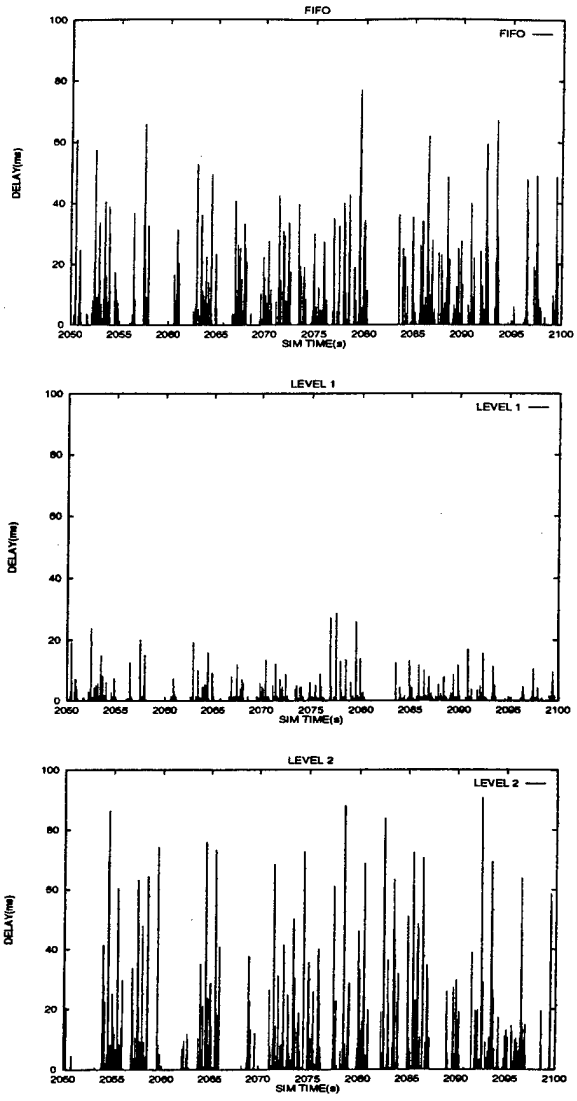


Figure 2: Delay trace at 55% utilization

3.2.1 Raw Network Performance

Figure 2 shows a plot of delay versus time over a 50 second simulation interval for a single test source in the FIFO case and for test sources in each level in the Priority case. Average utilization is 55% in both experiments. Histograms of delay (over a 500 second simulation interval) are shown in Figure 3. These graphs depict, as expected, that the service provided by Level 1 is better than that of Level 2 and of FIFO, and that FIFO was better than level 2 (although we were surprised by how small this latter difference was in the histograms). The unanswered question is whether or not these performance differences matter significantly to applications. Consider first the average delays: 0.71 ms for Level 1 and 6.73 ms for Level 2. While in absolute terms, this difference is significant, it is likely to be dwarfed by other sources of delay in the network, such as propagation time. Hence, if average delay matters, then one may conclude that multiple levels of service

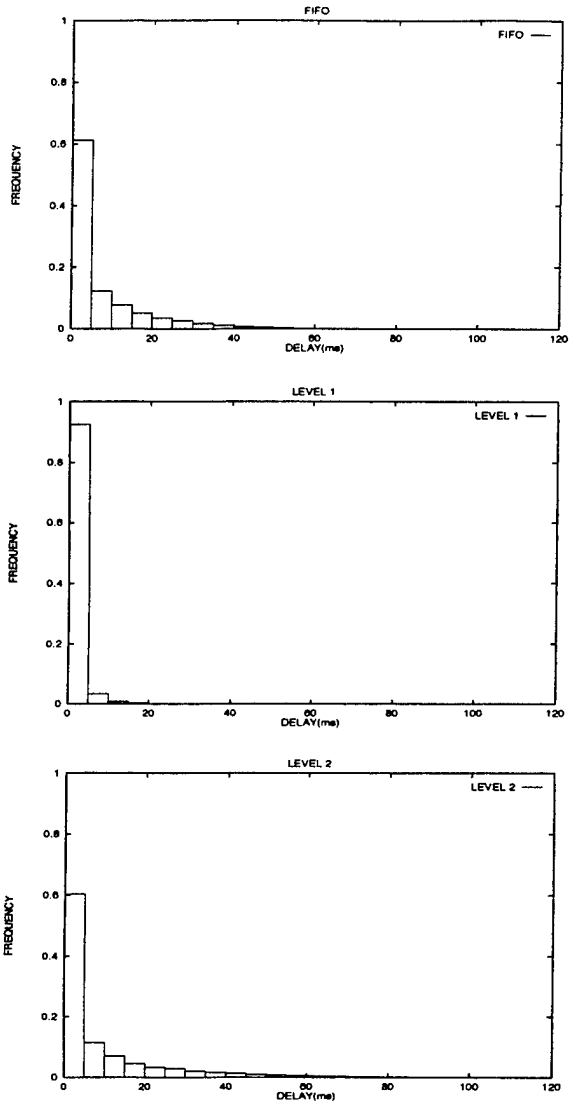


Figure 3: Delay histogram at 55% utilization.

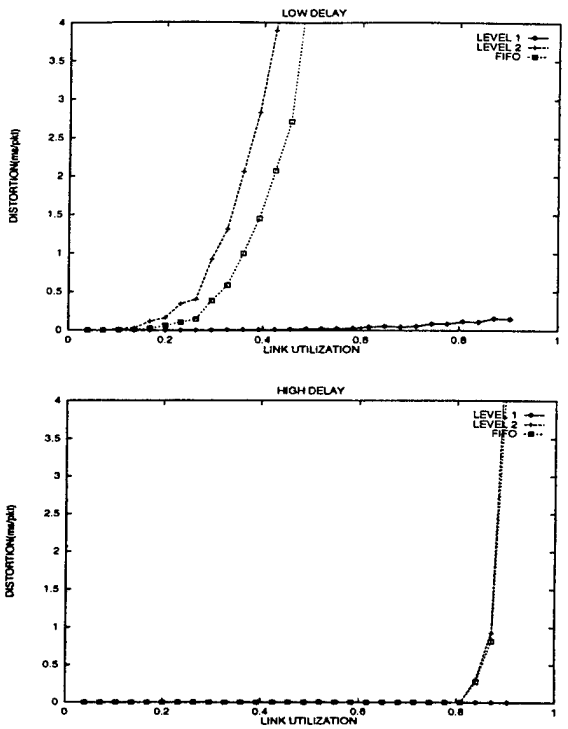


Figure 4: Distortion for rigid applications.

does not provide significant benefits to applications. However, the tails of the delay distributions are dramatically different. For example, the maximum delay experienced for Level 1 and Level 2 are 30 and 100 ms, respectively. In contrast to the averages, the differences between these figures are likely to be significant to some applications (e.g., interactive ones). Hence, it is apparent that one cannot address the design issue we raise here without considering the effect of the network service on the applications that use it. Specifically, how do applications adapt to the service, and how do they ultimately perceive the service?

3.2.2 Rigid Application Performance

We first consider the relationship between network service and application performance in the context of rigid applications that do *not* adapt to current network conditions. Rigid applications remove network jitter by maintaining a constant playback point for all packets. That is, all packets are buffered so that the sum of their network and playback delays are equal. Packets that arrive after their playback points must be discarded. This receiver behavior maintains perfect fidelity as long as packets arrive "in time", but degrades when packets arrive late. While such an application is impractical for the Internet (because applications have no way of knowing where to set the playback point when the first packet arrives in a way that will produce an acceptable level of distortion), we consider its behavior here to motivate the need to include adaptation in our analysis.

For rigid applications, the delay performance measure is merely the fixed delay experienced by all packets. To measure distortion, we assign a penalty of 120 ms (or three times the maximum penalty incurred by the adaptive algorithms) for each packet that arrives late and is dropped by the application.[14] The playback point for a rigid application is determined by the utility functions for delay. A delay sensitive application using the rigid playback algorithm sets its playback delay to 25 ms, half the delay threshold at which utility starts to degrade, while delay insensitive applications set the playback delay of the first packet to 500 ms.[15]

[14] Relating the distortion measure of rigid and adaptive applications is problematic, as it involves comparing the cost of late packets dropped by the application to the cost of adjusting the playback algorithm. Given our performance measures and utility functions, utility starts to degrade at .2% packet loss and utility is zero when packet loss reaches .8% for distortion sensitive rigid applications. For distortion insensitive applications, the corresponding thresholds are 2% and 8%.

[15] Choosing the playback point for rigid applications is also problematic. If the application has knowledge about the queueing delay of the first packet it receives, it could set the playback

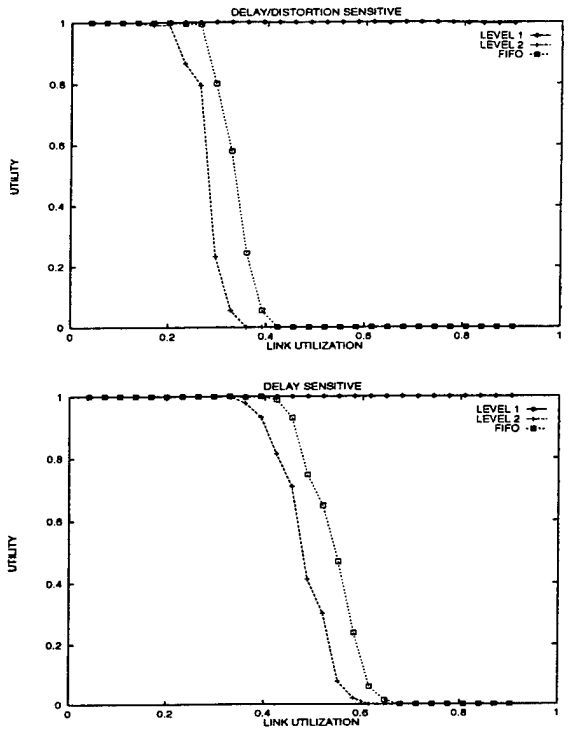


Figure 5: Rigid application utility for delay/distortion sensitive and delay sensitive applications.

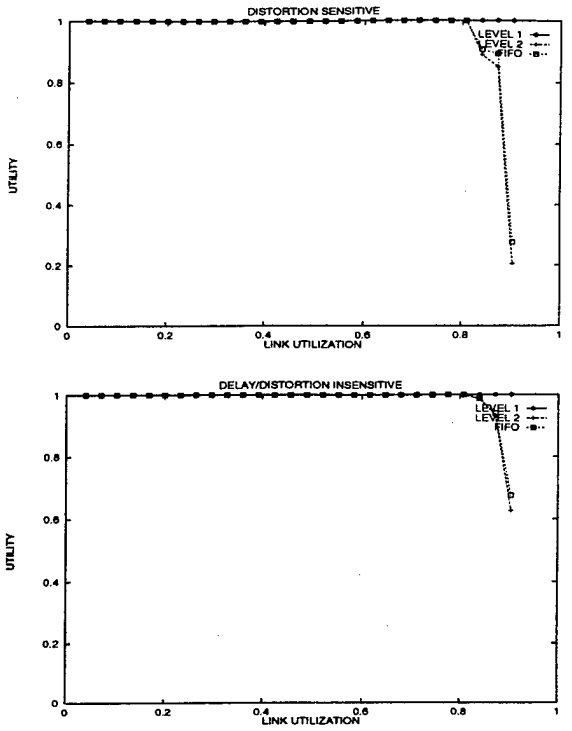


Figure 6: Rigid application utility for distortion sensitive and delay/distortion insensitive applications.

Figure 4 shows distortion as a function of offered load for rigid applications. For both delay sensitive and insensitive applications, data are shown for FIFO service (all traffic in a single service class) and for each of two levels in the Priority service case. With a low delay threshold (set for a delay sensitive application) there is no distortion up to about 20% utilization for the FIFO case. Beyond that, distortion starts to increase, deteriorating rapidly beyond 40% utilization. In the case of priority service, the Level 1 traffic experiences negligible distortion up to levels of utilization exceeding 80%.[16] The distortion of the Level 2 traffic is similar to the distortion of the FIFO service with the increases occurring at slightly lower levels of load. When the playback point of the rigid application is set to satisfy delay insensitive applications, no distortion is experienced (except at very high loads with Level 2 and FIFO service) as the playback point is large enough to enable almost all packets to arrive before their playback times.

These figures indicate how much distortion (resulting from discarded late packets) applications experience. However, they do not provide any indication about the effect that this distortion has on application performance. We employ the utility functions described in Section 2.3 to each of 4 classes of applications (characterized by their relative sensitivity or insensitivity to each of delay and distortion) for the FIFO case and for each level of service in the priority case. The results are shown in Figures 5 and 6. Applications that are sensitive to both delay and distortion achieve high utility up to about 25% load with FIFO service, then performance deteriorates rapidly. Applications that are only sensitive to delay and not to distortion experience this performance degradation at higher levels of load (45%). Applications that are sensitive to distortion only and applications that are not sensitive to either performance measure achieve high utility, except at the very highest levels of load.

In the Priority case, Level 1 service allows all applications to achieve high utility at all levels of load. Above load levels of 45%, Level 2 is only useful for applications that are insensitive to delay.

3.2.3 Adaptive Application Performance

We now consider the impact of the aggressive and conservative playback algorithms described in Section 2.2. To make comparisons between these algorithms as fair as possible, the minimum playback delay at the start of a talkspurt was set to 25 ms in the aggressive algorithm. This is consistent with the playback delay in the rigid algorithm. Within a talkspurt, when a packet arrived late and the playback point was adjusted, the minimum additional playback delay was set to 5 ms. The initial playback delay for the conservative algorithm was also 25 ms. We repeated our simulation

point optimally (i.e., to the delay value at which utility starts to degrade) and minimize packet loss. However, without this information, it can only guess. We used half the optimal value as the playback point, and set the minimal playback point of the adaptive algorithms to the same value to make the performance comparisons fair.

[16] Recall, the X axis is total offered load; hence in the Priority experiments, 80% load consists of 20% high priority traffic and 60% low priority traffic since we hold the ratio of high to low priority fixed at 1:3.

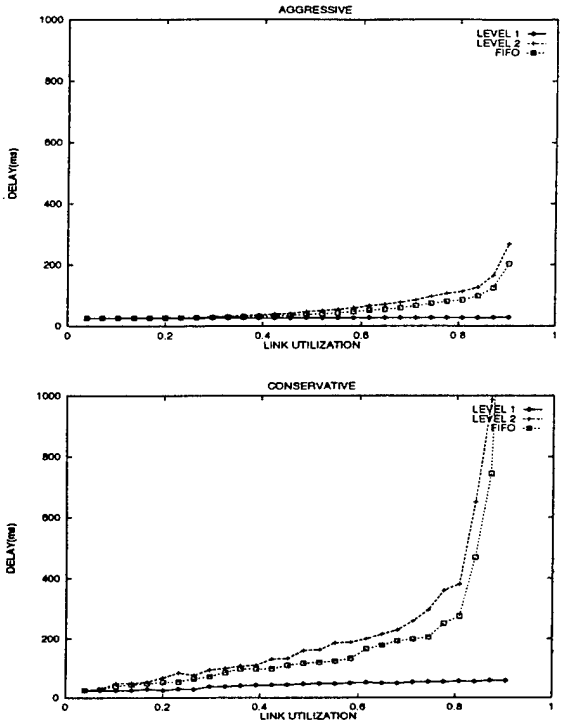


Figure 7: Delay for adaptive applications.

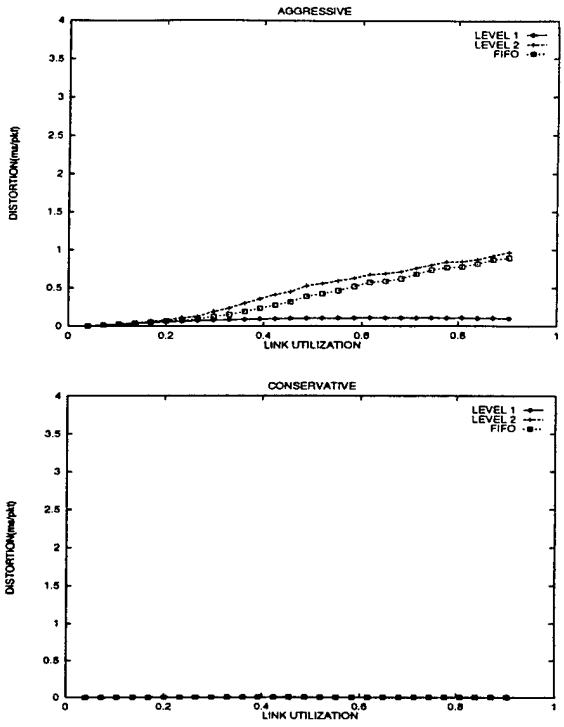


Figure 8: Distortion for adaptive applications.

experiments, computing delay and distortion for each algorithm in the FIFO test and for both levels of service in the Priority case. Delay measures as a function of load are shown in Figure 7 and distortion measures are shown in Figure 8.

These figures demonstrate that the two adaptation algorithms offer tradeoffs of delay for distortion. The aggressive receiver gives lower delays and higher distortion than the conservative algorithm at equivalent load levels. For example, with FIFO service, the aggressive receiver's delay is 51.2 ms while the conservative algorithm yields an average delay of 165.8 ms at 60% utilization (Figure 7). At the same load, the algorithms yield distortion values of .57 and .00071, respectively (Figure 8). However, these figures also show that while adaptation can reduce one quantity at the expense of the other, there is little adaptation can do to reduce both. So, as we shall see, applications that are sensitive to both delay and distortion are the most vulnerable to network service variations.

When two levels of service are available, the higher priority level always gives applications better service at high levels of load. However, the relative difference between Level 1 and Level 2 service depends on the adaptation algorithm. For instance, the difference between Level 1 and Level 2 delays is smaller with the aggressive algorithm than with the conservative algorithm. Conversely, the difference in distortion between Level 1 and Level 2 is small with the conservative algorithm and large with the aggressive algorithm. These differences affect the relative utility applications receive in different service levels. We next look at application utility as a function of the application's performance sensitivities.

Figures 9 and 10 show utility as a function of offered load for different types of applications. Each application uses the adaptation algorithm that is best suited for it.[17] With FIFO service, applications that are sensitive to both delay and distortion receive good service (i.e., high utility) only at low levels of load. Utility starts to decrease at utilization levels of about 40% of the link bandwidth as the adaptation algorithm is unable to meet both the delay and distortion requirements of the application, simultaneously. When high priority service is used, performance does not deteriorate. Applications that are sensitive to delay and not distortion maintain high utility up to 60% utilization with FIFO service, as the adaptation algorithm can optimize the performance measure about which the application cares the most. At higher loads (> 60%), high priority service does improve the performance of these applications. The applications that are only sensitive to distortion use the conservative adaptation algorithm to minimize distortion and achieve high utility at all but the highest levels of offered load with FIFO service. Hence, these applications derive no benefit from priority service, except in extreme conditions. Finally, applications that are insensitive to both delay and distortion also do not benefit from priority service (except at very high loads), given that their performance requirements are such that they are satisfied at most load levels with FIFO service.

[17] That is, the two kinds of applications that are sensitive to delay use the aggressive algorithm and the other two use the conservative algorithm.

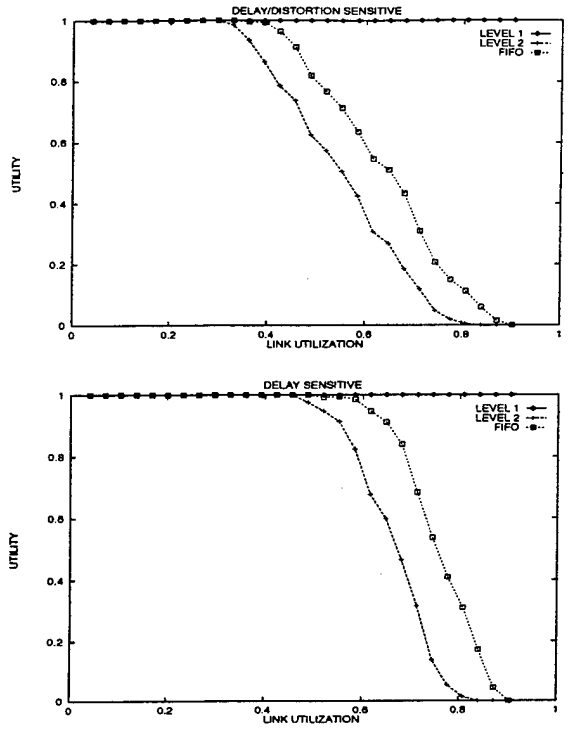


Figure 9: Adaptive application utility for delay/distortion sensitive and delay sensitive applications.

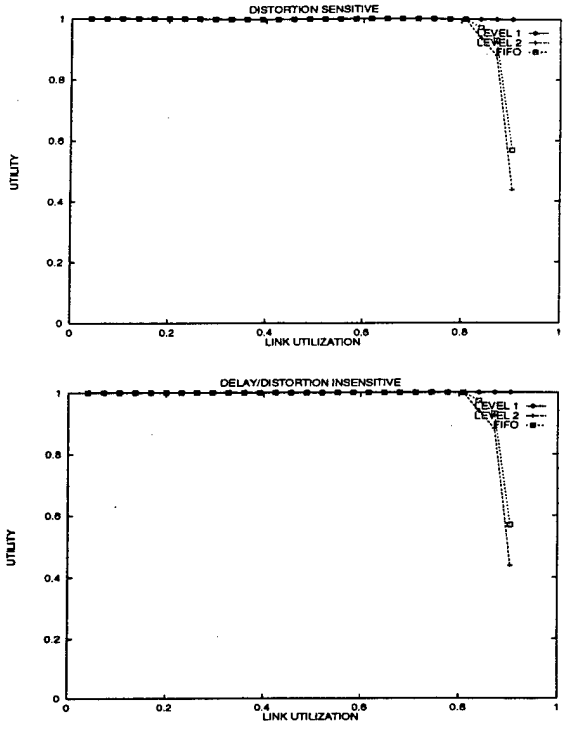


Figure 10: Adaptive application utility for distortion sensitive and delay/distortion insensitive applications.

3.2.4 One or Two Levels

The previous results showed, not surprisingly, that some applications achieve higher utility with high priority service than without it, and the magnitude of the difference depends on the characteristics of the application (its performance sensitivities) and on the level of the ambient traffic. However, by itself, this does not provide an answer to the question of the number of service levels that should be offered in the network. After all, better service always helps some applications, but at the same time it gives worse service to other applications. Hence, the answer depends on how much better two levels of service makes the overall network service. There is not a single best way to answer this question. We consider two alternatives.

The first, and perhaps most obvious approach is to consider the impact of multiple levels of service on total network utility. This method is fraught with problems. For instance, it depends on the mix of different kinds of applications in the network, the absolute value of utility achievable by each application, and on an incentive mechanism that impacts the mapping of application to service level. Nonetheless we proceed forward using the results of our previous experiment. We assume that there are equal amounts of each kind of application, that only the applications that are sensitive to both delay and distortion use the higher priority service, and that all applications have the same maximum utility.[18] Figure 11 shows the average utility per application as a function of offered load. Below 40% utilization, two levels of service offers no increase in utility. Above 40% load, service priority does offer modest advantages, with the benefit increasing with load. In contrast, Figure 12 shows total utility for rigid applications. Relative to the adaptive applications, these results present a stronger, but possibly misleading, case for multiple levels of service.

In addition to looking at the impact on total network utility, it is important to ask what effect service levels has on particular classes of applications. That is, independent of total utility, it may be important for a network to make sure that it serves all classes of applications adequately. In this case, the relevant question to ask is, for a given service discipline, at what level of offered load is an application no longer well-served by the network. We can ask this question in the context of our earlier results (see Figure 9). When only a single FIFO service class is offered, applications that are sensitive to delay and distortion start to suffer a loss in utility at utilization levels of 40%. When these applications use priority service, the network provides them with useful service at very high load. Assuming the other application types use the lower priority service, it is important to consider the levels of utilization at which the network no longer satisfies these applications. As is evident from the previous graphs, the delay insensitive applications do not suffer by using the lower priority service. The delay sensitive and distortion in-

[18] The actual mapping of applications to service levels depends on the incentives of using each class, which are not included in our analysis. However, given our previous results, we assume that if priority service is available, it will be used only by applications sensitive to both delay and distortion, since these are the applications that derive the most benefit from it.

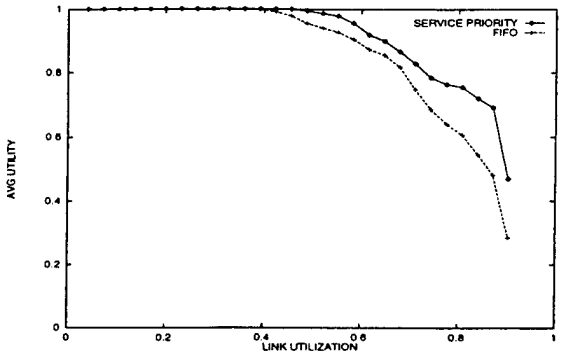


Figure 11: Average utility for adaptive applications.

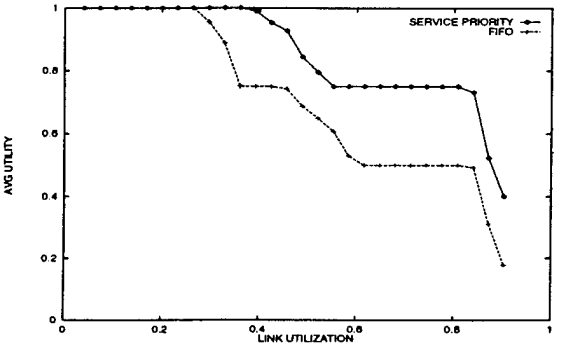


Figure 12: Average utility for rigid applications.

sensitive applications do suffer a bit when using the lower priority service: the level of utilization at which they no longer achieve utility of 1 decreases from over 60% to 50%. Nonetheless, when looking at all application classes, multiple levels of service increases the load levels at which the network can satisfy all of them. This suggests that multiple service levels may be worth deploying according to this criterion.

4 Discussion

Previous studies of application adaptivity have compared algorithms, focusing on such performance measures as the percentage of late packets for a given packet stream and adaptation algorithm.[8, 10] We believe ours is the first study to incorporate this adaptive behavior into consideration of a network design question. Our study provides no definitive answer to the question of whether multiple service priorities should be provided for real-time traffic in the Internet. Our results showed that with a single level of service, performance of some applications starts to degrade at utilization levels below 50%. These applications would benefit from two levels of service, and other applications are able to tolerate a lower priority service, yielding higher total network utility. However, the ultimate answer to our question depends on several characteristics of the future Internet, about which we are uncertain. First, how bursty will aggregate traffic be? We have presented results for a single kind of background traffic produced by multiplexing a moderate number of low frame rate video sources. Results from additional experiments, not shown here due to space limitations, showed that the burstiness of background traffic can be viewed as a knob that can be varied. With smoother background traffic and FIFO service, performance does not degrade until higher levels of utilization are reached. Therefore, the relative benefits of priority service are smaller and only occur at higher levels of utilization, making a weaker case for multiple levels of service. The converse is true with burstier background traffic. One should not take too seriously the absolute values of the utilization levels presented here; by making the traffic even burstier, one could make the levels of utilization at which performance degrades, arbitrarily small. We do not yet know if future Internet traffic will be smooth enough everywhere to obviate the need for multiple levels of service, or will the levels of burstiness be such that multiple service priorities are unambiguously desirable. In any event, additional simulations of either less bursty or more bursty traffic will not resolve this.

Second, what will the ratio of best-effort to real-time traffic be in the network? For example, if future network traffic consists of 90% best-effort traffic then the relevant utilization levels for real-time traffic in our simulations would only be 10%. At this level of load, for all but the burstiest traffic imaginable, real-time traffic always receives low delay and distortion, even with a single level of service. Delay and distortion would be absorbed by best-effort applications, which are well-suited to handle the performance degradation. Just as a few years ago one would not have predicted the tidal wave of web traffic, we cannot, at this point, predict the extent to which the Internet will be used for real-time applications.

Finally, what are the nature of utility functions of real-time applications? While we believe we have captured the essential characteristics of these functions, actual thresholds and functions may be significantly different. Ultimately, our simulation model can provide an answer to the larger question given an adequate set of parameters, but the model itself cannot resolve these questions. Hence, a more definitive answer requires a much better understanding of network applications, traffic mix and utility functions.

While we do not provide an unambiguous answer to the initial question we posed, our results do yield other key observations. First, application adaptivity is not a panacea. Our simulations showed that adaptive algorithms can very successfully remove distortion at the receiver, or they can reduce delay. However, achieving both low delay and low distortion is difficult under moderate load with bursty traffic. Hence, if there are applications that are sensitive to both performance measures, then under certain traffic conditions, adaptivity may not be enough. In this case, service discrimination inside the network is needed to provide these applications an acceptable level of performance.

Finally, we obtained different results for rigid and adaptive applications, further emphasizing the importance of including realistic application behavior in the analysis. While we demonstrate this point in the context of one specific network design question, we believe it has widespread applicability. For example, when considering other questions, such as whether or not the network should provide multiple levels of dropping pri-

ority, appropriate models of applications are needed. Conversely, if nothing else, this study has shown us that previous research (such as [14]) that models applications as rigid and does not take application adaptivity into account can lead to incorrect or misleading results. In the Internet, design analyses must incorporate the adaptive nature of applications.

References

[1] Jean-Chrysostome Bolot. End-to-end packet delay and loss behavior in the Internet. In *Proceedings of ACM Sigcomm*, pages 289–298, San Francisco, California, September 1993. ACM. also in *Computer Communication Review* 23 (4), Oct. 1992.

[2] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. RFC 1633, Internet Engineering Task Force, June 1994.

[3] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation protocol (RSVP) – version 1 functional specification. Technical Report RFC 2205, Internet Engineering Task Force, September 1997.

[4] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM Sigcomm*, pages 14–26, August 1992.

[5] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. Network support for multimedia: A discussion of the Tenet approach. *Computer Networks and ISDN Systems*, 10:1267–1280, July 1994.

[6] Sally Floyd. Comments on measurement-based admissions control for controlled-load services. submitted to CCR, July 1996.

[7] Sugih Jamin, Peter B. Danzig, Scott J. Shenker, and Lixia Zhang. A measurement-based admission control algorithm for integrated services packet networks. *IEEE/ACM Transactions on Networking*, 5(1):56–70, February 1997.

[8] Sue B. Moon, Jim Kurose, and Don Towsley. Packet audio playout delay adjustment algorithms: performance bounds and algorithms. Research report, Department of Computer Science, University of Massachusetts at Amherst, Amherst, Massachusetts, August 1995.

[9] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[10] Ramachandran Ramjee, Jim Kurose, Don Towsley, and Henning Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *IEEE Infocomm*, pages 680–688. IEEE, 1994.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. RFC 1889, SRI Network Information Center, January 1996.

[12] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. RFC 2212, Internet Engineering Task Force, September 1997.

[13] S. Shenker and J. Wroclawski. Network element service specification template. Technical Report RFC 2216, Internet Engineering Task Force, September 1997.

[14] Scott Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communications*, 13(7), September 1995.

[15] C. Topolcic. Experimental internet stream protocol, version 2 (ST-II). RFC 1190, SRI Network Information Center, October 1990.

[16] J. Wroclawski. Specification of the controlled-load network element service. RFC 2211, Internet Engineering Task Force, September 1997.

[17] J. Wroclawski. The use of RSVP with IETF integrated services. Technical Report RFC 2210, Internet Engineering Task Force, September 1997.

[18] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A new resource reservation protocol. *IEEE Network Magazine*, 7(5):8–18, September 1993.

Uniform versus Priority Dropping for Layered Video*

Sandeep Bajaj Lee Breslau Scott Shenker
{bajaj,breslau,shenker}@parc.xerox.com
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Abstract

In this paper, we analyze the relative merits of uniform versus priority dropping for the transmission of layered video. We first present our original intuitions about these two approaches, and then investigate the issue more thoroughly through simulations and analysis in which we explicitly model the performance of layered video applications. We compare both their performance characteristics and incentive properties, and find that the performance benefit of priority dropping is smaller than we expected, while uniform dropping has worse incentive properties than we previously believed.

1 Introduction

A common question facing network designers is what functionality the Internet should offer, and whether to place that functionality in the interior of the network (in network routers) or at its edges (in hosts). When evaluating new network control mechanisms, it is not enough to merely consider network-centric criteria, such as the local effects on queue sizes at routers, or the end-to-end delays provided by a particular network architecture. Rather, because the ultimate purpose of the network is to support applications, one must evaluate the impact of the proposed mechanisms on application performance.[1] If the proposed mechanisms require significant additional complexity in the network infrastructure, their adoption should only be considered if they provide significant benefits to a large (or extremely important) segment of applications.[2]

For example, a priority dropping mechanism will increase the dropping probability of low priority packets while protecting high priority packets from frequent loss. However, in and of itself, the change in drop rates for different traffic does not indicate whether the control mechanism significantly improves the performance of applications. The desirability of drop priority depends crucially on its impact on application performance. That aspect of the issue, which has received comparatively little attention so far, is the focus of our paper.

This paper is devoted to the question of drop priority for the transmission of layered video in contexts where packet drops, rather than packet delays, are the primary determinant of application performance. In such cases, we ask: should the Internet retain the currently predominant *uniform* dropping mechanism, where all data packets are treated equally with respect to dropping, or should the Internet adopt a *priority* dropping mechanism where lower priority packets are dropped before higher priority ones? We specifically focus on the implications of these network dropping mechanisms for application performance.[3] To this end, we consider a class of layered video applications and introduce a simplified model of their performance; this model is extremely primitive, but nonetheless highlights several open questions about the characteristics of network video applications. Using both discrete event simulation and analytical models, we assess the impact of different network dropping schemes on application performance and on their incentive properties.

*This research was supported in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contracts DABT63-94-C-0073 and DABT63-96-C-0105. The views expressed here do not reflect the position or policy of the U.S. government.

[1] We will use the terms application performance and application utility interchangeably.

[2] Alternatively, one would consider adopting network designs that allowed a significantly higher level of link utilization without significant application performance degradation; in this case no single user is substantially better off in terms of application performance, but the increased utilization implies that there are many more such satisfied users. Of course, the desirability of any new network level mechanism depends on the level of mechanistic complexity introduced; however, we adhere to the philosophy that one should only consider adoption after the significance of the benefit is demonstrated, and so the consideration of implementation complexity comes after the analysis of the potential benefits. We only address the latter (the potential benefits) and not the former (the implementation complexity) in this paper.

[3] We do not discuss the mechanisms for implementing drop priority, such as whether these priority levels are indicated in packets in TOS bits, or signaled as part of the multicast join message.

Much to our chagrin, we find that many of our previously held opinions and assumptions about the performance of packet dropping algorithms and the incentives they provide to applications are either wrong, or are only true in the context of specific network conditions. In particular, we find that while priority dropping performs better in general than uniform dropping, the magnitude of the difference is smaller than we expected. At the same time, we find that uniform dropping has worse incentive properties than we thought.

We hasten to note that we do not offer specific conclusions about whether or not the Internet should offer drop priority for layered video applications. Our results are ambiguous on this point – there are some conditions where drop priority yields significant benefits, and other conditions where it doesn't– and there are many relevant issues (such as implementation complexity) that are outside of our ken. Instead of making a specific recommendation about priority dropping, our goal is to illustrate more generally the basic approach of considering application performance in conjunction with network control mechanisms.

In the next section we provide background about the question of network dropping algorithms that motivated this work, and identify three central questions to be addressed. In Section 3 we outline our general model of application utility, along with the simulation and theoretical frameworks used in this study. In Sections 4 and 5 we present the results of our investigations into the performance and incentive aspects of priority and uniform dropping. We conclude with a discussion in Section 6.

2 Background

A key challenge in sending video over the Internet is matching the transmission rate to the currently available bandwidth. One approach is to employ rate adaptive coding algorithms [2, 3] in which the parameters of the coding algorithm can be adjusted periodically to match the available network bandwidth. However, this strategy is problematic for multicast transmission [12, 14] since, in general, there is not a single bottleneck bandwidth available between the source and all receivers; some paths will have more available bandwidth than others. Choosing a single transmission rate (at any instance in time) will either unnecessarily constrain those receivers with higher bandwidth paths or congest the lower bandwidth paths leaving some receivers with high loss rates. In response, Deering [5] and others [9, 11, 15, 16] proposed using layered coding algorithms[4] to transmit video data, striping different layers across different IP multicast groups [4]. In this scheme, a receiver only subscribes to those groups for which sufficient network capacity exists. Thus, in a heterogeneous network, each receiver is able to adjust its level of subscription to receive an appropriate amount of traffic.

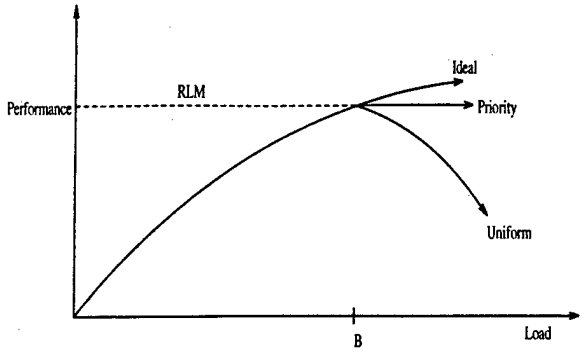


Figure 1: A representation of application performance as a function of load with perfectly smooth traffic. The maximal performances of priority and uniform dropping are the same, and RLM achieves this optimal level. The uppermost curve depicts the performance level achieved if the link had infinite bandwidth. The load level marked by B denotes the bandwidth of the bottleneck link.

Such a layered encoding and transmission scheme lends itself to the use of priority dropping. Packets belonging to the base layer of a hierarchically encoded video stream can be marked as high priority while packets belonging to each successive enhancement layer can be marked as successively lower priority. During times of congestion, the network can preferentially drop the low priority packets, protecting the base layer from significant loss. Shifting loss away from more important and towards less important packets improves the picture quality. For this reason, priority dropping (for layered encodings) was considered by many in the Internet community to be an extremely promising approach for digital video.

However, in their recent work on Receiver-driven Layered Multicast [12] (RLM), McCanne *et al.* argue against this use of priority dropping.[5] Their argument is best illustrated by Figure 1 (based on a similar figure in their paper) which plots the application performance (which, in the case of digital video, is synonymous with picture quality) as a function of offered load for uniform and priority dropping when capacity is fixed. In addition, the uppermost curve shows the quality of the picture as a function of load with unlimited link capacity (and therefore no packet drops.) Underlying this figure is the implicit assumption that there is a bottleneck rate below which there is no dropping, and above which all excess packets are dropped. That is, the output rate equals the input rate whenever the input rate is below the bottleneck rate, and the output rate is equal to the bottleneck rate whenever the input rate is greater than the bottleneck rate. As the figure

[4] Layered coding algorithms, such as described in [9, 11], partition the encoded signal into several *layers*. The base layer encodes a fairly primitive rendering of the image, and higher layers encode increasingly finer enhancements to the image. Layered coding algorithms thus provide several different levels of encoding simultaneously. A receiver can choose how may layers to receive, and the more layers that are available to decode, the higher the resulting picture quality.

[5] While the introduction of this paper makes remarks about the inadvisability of priority dropping (as we outline below), we should note that in this paper RLM is also described as a way to cope with the current uniform dropping infrastructure (regardless of what one thinks of priority dropping). Moreover, the bulk of the paper is devoted to the design of the RLM algorithm itself – an admirable exercise in elegance and scalability – and the discussion of the relative merits of uniform and priority dropping is clearly not the central focus of the paper.

shows, up to the bottleneck load the application performance of the two dropping schemes is equal because no packets are dropped in either case. Beyond the bottleneck load, the performance of uniform dropping degrades because no additional packets are transmitted, and packets are dropped uniformly from all layers; the performance of priority dropping remains constant since only packets from the lower priority levels are dropped. Thus, as illustrated by Figure 1, uniform and priority dropping both attain the same maximal performance, and the performance curves only differ when excess load is offered.

As discussed by McCanne *et al.* [12], this behavior has implications for the incentives faced by users. Under uniform dropping, users maximizing their own performance would restrain their usage to the bottleneck rate. Priority dropping, on the other hand, does not provide any penalty for sending low priority packets that are dropped inside the network, and so users (at least based solely on application performance) would not mind sending faster than the bottleneck rate.[6] Thus, uniform dropping provides better performance-based incentives than priority dropping.

Priority dropping, however, has the advantage that the application need not determine the bottleneck rate precisely to achieve optimal performance. In contrast, to achieve the optimal performance with uniform dropping, the application must identify the bottleneck rate precisely because performance degrades if the transmission rate is higher (or lower) than the bottleneck rate.

RLM [12] is a host-based algorithm that responds to current network conditions (as measured by packet drops) to achieve the appropriate bandwidth level. Members leave a level (*i.e.*, leave the multicast group associated with that level) when their overall drop rate (across all levels) becomes too high; in addition, they periodically perform experiments by joining a level to test if the drop rate with the addition of the new level is sufficiently low. RLM's innovative design coordinates the activities among the various members of the multicast group so that these experiments don't interfere with each other. Under the conditions explored in [12], where the traffic is rather smooth (*i.e.*, CBR-like), RLM successfully achieves the bottleneck bandwidth, and does so in a scalable manner for large multicast groups. We have indicated this on Figure 1 by marking RLM's performance level at the peak of the priority and uniform dropping curves.

One can summarize (with appropriate additional caveats) the three basic tenets of the arguments against priority-dropping: (1) uniform dropping achieves (essentially) the same optimal level of performance as priority dropping (*i.e.*, the peaks of the curves are nearly the same), (2) RLM achieves (close to) optimal performance, and (3) uniform dropping provides incentives for users to reach the optimal operating point whereas

[6]The impact of users not constraining their usage of lower priority packets is not clear, nor can it be evaluated in the simple settings we investigate here. If there is only a single bottleneck in the network, then there are no ill-effects of sending lower priority packets that will be dropped at that bottleneck. Negative consequences only arise in more complicated settings where these destined-to-be-dropped packets congest one or more links upstream of the bottleneck. It is not clear how significant a phenomena this is.

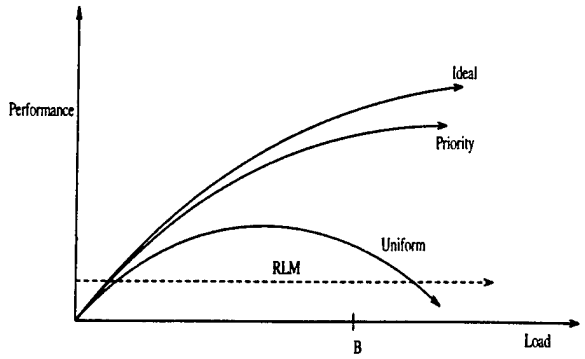


Figure 2: A representation of our intuition about application performance as a function of average load with bursty traffic. Priority dropping achieves a significantly higher maximal performance level than uniform dropping. Moreover, RLM's performance is significantly below the maximal performance achievable under uniform dropping. As before, the uppermost curve depicts the performance level achieved if the link had infinite bandwidth, and the load level marked by B denotes the bandwidth of the bottleneck link. Note that since the x-axis represents average load and traffic is bursty, the performance of priority dropping can increase beyond the bottleneck rate.

priority dropping does not provide performance-based incentives to constrain usage. While these arguments certainly have an initial appeal, we must admit that after reading [12] we were not entirely convinced by them. In the spirit of full disclosure we now discuss the intuition that (mis)guided us as we embarked on our research program.

The assumption underlying Figure 1 is that packet dropping occurs only when the bottleneck rate is exceeded, and then all excess packets (*i.e.*, all packets exceeding the bottleneck rate) are dropped. This is a reasonably good approximation when network traffic is quite smooth, like CBR traffic, but Internet traffic is not typically CBR-like. In fact, Internet traffic is characteristically quite bursty [8, 10], and bursty traffic results in packet drops even when long-term average transmission rates are well below bottleneck rates. With bursty traffic, there will be some packet drops at all significant utilization levels, thereby producing a performance gap between priority and uniform dropping even below the bottleneck rate. Consequently, our initial hypothesis when undertaking this work was that in the presence of bursty traffic (similar to what we envision is present in the Internet), priority dropping would significantly outperform uniform dropping. By this we mean the optimal performance achievable with priority dropping would be much higher than the highest performance achievable with uniform dropping; in essence, we imagined that with bursty traffic the performance curves would look more like Figure 2 than like Figure 1.

RLM uses packet drops as a signal of congestion, joining and leaving levels based on the current dropping rate. In this manner, RLM attempts to match its transmission rate to the bottleneck bandwidth. When traffic is bursty, however, the bandwidth available at

the bottleneck (and perhaps even the location of the bottleneck itself) can fluctuate significantly. Moreover, these fluctuations occur on time scales much shorter than the response times of host-based mechanisms (which require at least the round trip time between the bottleneck and the receiver to respond). We assumed that any attempt to adjust, at the endpoint, to rapid fluctuations at the router was doomed to fail. Consequently, our initial hypothesis was that, under bursty conditions, adaptive mechanisms implemented at the endpoints would perform significantly worse than the optimal performance under uniform dropping, and certainly much worse than the performance achieved with priority dropping. As shown in Figure 2, our initial assumption was that RLM, or indeed any similar adaptive algorithm used with uniform dropping, would result in performance significantly below the peak of the uniform dropping curve.

Lastly, there is the issue of incentives. The performance curves in Figures 1 and 2 depict the performance of a single application given a fixed (smooth in Figure 1 and bursty in Figure 2) traffic load. The peaks of these curves identify points that are optimal for that individual application. However, in constructing network mechanisms we are typically more interested in achieving global or social optima, where the total performance – the performance of all applications – is optimized. There are many models of congested systems where the individually optimal point, often called the Nash equilibrium [7], is quite far from the socially optimal outcome [13]; the reasoning in [12] about the incentives provided by uniform dropping did not make the necessary distinction between individually and socially optimal outcomes. While uniform dropping provides performance-based incentives to constrain usage (and priority dropping did not), this does not at all imply that under uniform dropping the joint behavior resulting from applications seeking their individually optimal points will result in socially optimal, or close to socially optimal, outcomes. Despite this fact, our initial assumption about incentives was that uniform dropping's incentive properties were still far superior to those of priority dropping.

Thus, our initial assumptions were quite different from those embedded in the arguments against priority dropping advanced by McCanne *et al.* in [12]. To resolve these fundamentally conflicting intuitions and arguments, we set out to answer three basic questions about uniform and priority dropping:

1. Do uniform and priority dropping achieve the same optimal performance? To what extent does the answer depend on the burstiness of traffic, and is the difference ever significant?

2. Can RLM (as an example of a control mechanism used with uniform dropping) achieve the optimal performance under uniform dropping? Again, does the answer depend on the burstiness of traffic?

3. What incentives do these different dropping mechanisms present to individual applications? Do operating points where each user is individually optimal differ from the socially optimal operating point?

To address these questions, we use simulation and analysis of simple models of layered video applications transmitting over a network. The simulation models are, by necessity, crude approximations because we know little about the traffic and performance characteristics of future layered video sources, and even less about the likely nature of future background traffic. The analytical models suffer from these same drawbacks but, in addition, tractability requires even further simplification. Thus, we make no pretense that the models we used are *realistic* in any precise sense; they merely illustrate some of the basic issues involved, and highlight areas where future research is needed. These simulation and analytical models are presented in the following section.

3 Simulation and Analytical Models

In this section we describe our simulation and analytical models of applications and the network. For our purposes here, an application has two somewhat distinct aspects: its offered load (the nature of the packet stream the application transmits over the network) and its performance characteristics (how application performance depends on the network service those packets receive). The basic assumptions about the performance characteristics embedded in the analytical and simulation models are quite similar in spirit, and we describe them first. We then describe the network and traffic aspects of the simulation model, and then those of the analytical model.

3.1 Basic Assumptions about Performance of Layered Video

We represent the performance of an application by a *utility function*. This function maps the service received into some performance (or utility) level delivered to the end user. Below we describe a model for layered video, our canonical application, but it may well apply to other layered applications.

The canonical representation of layered video we use throughout this paper consists of some fixed number L of layers. Each layer is characterized by a traffic stream and a potential value it provides to the application. One expects the *per bit* value to decrease with additional layers; the most value is derived from the information encoded in the base layer, and the relative value of bits in the enhancement layers decreases. For each layer l let $a(l)$ denote the bandwidth and $f(l)$ denote the per-bit value. We assume that $f(l)$ is a non-increasing function.[7] If all packets are successfully received (we assume delay is not an issue, only dropping, so we ask only *if*, not *when*, packets are received), then the total utility is merely $\sum_{l=1}^{L} a(l)f(l)$. Define $F(l) = \sum_{k=1}^{l} f(k)a(k)$ to be the total utility of all layers up to l.

Our utility functions describe, in the presence of loss, how much value an application derives from the set of packets it actually receives. Since utility is a subjective measure that depends in large part on human

[7] As l increases, the value of layer l decreases. Hence, higher layers (in terms of their index) will have lower drop priority and lower layers will have higher drop priority.

perception (so it may vary from person to person) and on the characteristics of the coding algorithm used (so it may vary from implementation to implementation if different encoding schemes are used), the range of applicability of any particular utility function is limited. Rather than choose a single utility function and claim that it accurately represents the truth about application utility, we instead examine a family of extremely simple utility functions in an attempt to understand what impact different utility functions have on the results of our study.

We assume that the utility of each layer is independent of the other layers' utilities.[8] The utility of a given layer is a function of the loss experienced in that layer; we represent this by a non-decreasing function $g(z)$ with $g(0) = 0$ and $g(1) = 1$, where z is the fraction of packets received in the layer. Thus, if $d(l)$ is the fraction of packets dropped in layer l then the total utility is given by:

$$\sum_l a(l)f(l)g(1 - d(l)) \quad (1)$$

To explore the impact of different utility functions, we use functions of the form $g(z) = z^m$, $m > 0$. The nonlinearities for $m \neq 1$ could be due to characteristics of the coding algorithm, human perceptual factors, or both.

3.2 Simulation Framework

We use discrete event simulation to study the performance of uniform and priority dropping. Our simulator, which used version 2 of the *ns* network simulator as a starting point and added new functionality as needed, incorporates the utility functions described above, as well as the relevant source models and network control mechanisms.[9] Below we describe the application source models, the router queueing and dropping algorithms, and the network topology.

Source models: We use an abstract layered source model that captures two essential characteristics of layered video traffic: (1) the instantaneous traffic in each layer varies over time, and (2) there is high correlation between the instantaneous traffic in each layer (one might expect factors such as motion or scene change that lead to changes in bit rate to have impact across layers). The layered source model is built out of individual layers. We first describe the traffic in the base layer ($l = 1$) and then describe the traffic generated by the higher layers.

We divide time into discrete intervals of length Δ_1, and let t be the index of the time intervals. Let n_t be the number of packets sent in time interval t. All n_t packets are sent back-to-back at a starting time chosen at random from a uniform distribution within the interval. In every time interval t, the rate n_t is selected independently from the following random distribution: $n_t = 1$ with probability $\frac{P-1}{P}$, and $n_t = PA+1-P$ with probability $\frac{1}{P}$. This model produces hi-low sources that generate either $n_t = 1$ packet per interval or $n_t = PA + 1 - P$ packets per interval (and A continues to describe the average number of packets per interval). Note that when $P = 1$, this model produces CBR-like traffic with $n_t = A$. Increasing P yields increasingly bursty traffic. Throughout our simulations, we use $\Delta_1 = 1$ second, and $A = 4$ packets per interval. All packets have size $s = 1000$ bytes.

The higher layers are slight modifications of this model. We impose the requirement that $f(l)a(l) = f(1)a(1)$ so that all layers contribute equal value; for convenience, we assume $\frac{f(1)}{f(l)}$ is an integer. Then, for each time interval of the base layer (of length Δ_1), we create $\frac{f(1)}{f(l)}$ subintervals of length $\Delta_l = \Delta_1 \frac{f(l)}{f(1)}$. As in the base layer, a certain number n of packets are sent back-to-back in each of these subintervals, starting at some uniformly distributed starting time. Inter-layer correlations are captured by using the same value of n_t in each subinterval Δ_l (for all layers l) of a given base interval Δ_1; that is, a number n_t is chosen for each time interval t for the base layer, and this n_t is used to govern the transmissions in each subinterval (for each higher layer) of the interval t. Thus, as n_t varies randomly, all layers adjust their sending rates in concert.

For our simulations, we typically use $a(l) = 2^{l-1}a(1)$ and $f(l) = 2^{1-l}f(1)$, so each layer potentially contributes a unit of value.[10] With $\Delta_1 = 1$ second, $A = 4$ packets per interval, and $s = 1000$ bytes, we have average transmission rates per layer of 32kbps, 64kbps and 128kbps, etc.

Dropping and Scheduling Algorithms: Under uniform dropping, all packets have equal drop probability. We use tail drop in most of our experiments; a packet arriving at a full queue is dropped, otherwise it is queued for later transmission. We also use Random Early Detection [6] (RED) as our dropping algorithm in some experiments.

The priority dropping algorithm is slightly more complex and involves dropping on both input and on output. When a packet arrives and the queue is full, rather than dropping the arriving packet (as with drop tail) the arriving packet is queued and a packet (the latest in the queue) of the lowest priority among those packets in the queue is dropped. Dropping on output is also performed in order to prevent transmission of already queued low priority packets from causing later drops to higher priority packets. Dropping on output uses a threshold parameter ζ, with $0 \leq \zeta \leq 1$. A packet in layer l at the front of the queue is dropped if the total number of packets in the queue of higher priority than l is greater than ζ times the total number of buffers; otherwise, the packet is transmitted. This heuristic allows low priority packets to be sent when the probability is small that they will cause subsequently arriving higher priority packets to arrive at

[8] In [1] we also consider utility models that capture dependency between layers, where the dropping rate in one layer may effect the utility of another layer. Such dependencies arise in many coding algorithms.

[9] The *ns* simulator is available at http://www-mash.cs.berkeley.edu/ns. Our extensions to the simulator, and the simulation scripts we ran to generate the results in this paper can be found at ftp://ftp.parc.xerox.com/pub/net-research/breslau/dropping.

[10] The decreasing per-bit value per layer is fundamental to our model. However, the grouping of bits into layers is arbitrary. We chose to hold the per-*layer* value constant, implying an increasing rate per layer. Our analytical model is not burdened by this somewhat arbitrary decision.

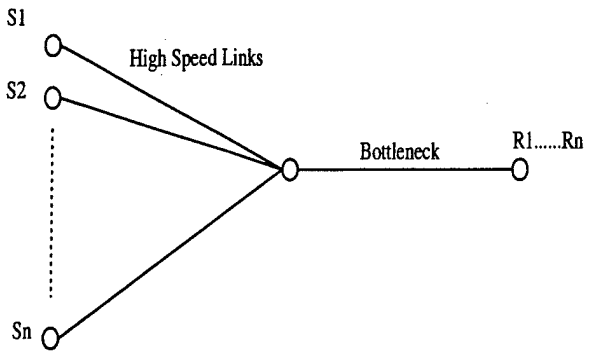


Figure 3: Simulation Topology

a full queue. We used the value $\zeta = 0.6$ in our simulations (both here, and in the subsequent places where parameter ζ is invoked).[11]

In the experiments described in this paper, we used a simple FIFO scheduling algorithm with the two dropping algorithms described above. We have also simulated uniform and priority dropping with Fair Queueing. Results of these experiments are reported in [1].

Topology: We use a very simple topology in our experiments (shown in Figure 3) to assess the impact of different dropping strategies across a single bottleneck link. There are n sources and receivers. There is a single bottleneck link, and high speed links (10 Mbps) connect each source to the bottleneck. All dropping occurs at the bottleneck link. Note that we do not address the issue of multiple receivers per source since it is irrelevant to priority dropping, and RLM deals gracefully with large groups for uniform dropping. Thus, for clarity, we reduce the problem to its bare essentials by focusing on single receiver groups.

Inputs for each experiment include the number of sources, the traffic parameter P used for each source, the bottleneck link bandwidth, and the number of buffers b at the bottleneck link. Unless otherwise stated, our simulations use eight sources and a bottleneck bandwidth of 4 Mbps. We use $P = 1$ or $P = 5$ to produce smooth or bursty traffic, respectively, and $b = 60$ or $b = 20$ to test the effect of buffer size on our results.

3.3 Theoretical Framework

We augment our simulation study with theoretical analysis. We employ an extremely crude model that captures some of the essential features but leaves out many details. The model complements the simulations since the theoretical model can incorporate a wider variety of traffic models (in terms of their drop rates as a function of load) and the layers are infinitesimally small (avoiding effects due to the large size of the lower priority layers). Below we describe the model; the results of the analysis are presented in the following sections.

[11] Designing an ideal priority dropping mechanism was not a goal of this research. We fully expect that one could improve on the priority dropping algorithm outlined here. However, experimentation has shown that this algorithm does accomplish our main goal which is to achieve high throughput while protecting higher priority packets from loss resulting from the transmission of lower priority packets.

Source Models: Rather than having a finite number of discrete layers (labeled by l), we model each flow as having a continuum of layers (labeled by x), each with an infinitesimal unit of bandwidth (*i.e.*, $a(x) = 1$). Let r_i denote the highest layer being sent by flow i. Since each layer consumes a unit of bandwidth, r_i is also the total bandwidth sent by the i'th flow. The assumption that different flows send the same amount of bandwidth in each layer is clearly limiting, but it greatly simplifies our analysis and does not directly undercut our central concern, that being the tradeoff between priority and uniform dropping. As before, $f(x)$ is the potential value for each layer (if all packets are received in that layer), and $F(x) = \int_0^x dy f(y)$ denotes the total potential value for all layers up to x.[12]

The performance of the network, from the perspective of the i'th flow, is characterized by $d_i(\vec{r}, x)$ which is the drop fraction of packets in flow i, layer x, and where $\vec{r}$ describes the transmission rates of all flows. Note that the quantity $d_i(\vec{r}, x)$ can be interpreted as the drop rate (packets dropped per unit time), or the drop fraction (fraction of packets dropped), given that each layer consumes a single unit of bandwidth. The continuum version of the utility function given by Equation 1 is:

$$u_i = \int_0^{r_i} dx f(x) g(1 - d_i(\vec{r}, x)) \qquad (2)$$

Dropping Algorithms: We consider two forms of dropping behavior: uniform and priority dropping. For uniform dropping the basic principle is that there are no distinctions between layers as far as dropping is concerned: $d_i(\vec{r}, x) = d_i(\vec{r}, y)$ for all x, y. For priority dropping the basic assumption we make is that the dropping rate for higher priority packets is completely unaffected by the presence of lower priority packets. In reality, priority dropping schemes will never achieve this perfection, but this assumption makes the model tractable. This means that $d_i(\vec{r}, x)$ is independent of all r_j as long as $r_j \geq x$ (and this applies to $j = i$ as well) and so, in particular, $d_i(\vec{r}, x) = d_i(\vec{r} \wedge x, x)$ (using the notation that $a \wedge b = \min[a, b]$ and that $(\vec{a} \wedge b)_i = \min[a_i, b]$).

Scheduling Algorithms: The basic behavior of the queueing system is represented by an increasing and convex ($D'' \geq 0$) function $D : \Re_+ \mapsto \Re_+$. $D(T)$ is the drop rate (packets dropped per unit time) resulting from a total traffic load of T; the drop fraction is given by $\frac{D(T)}{T}$. We make the basic approximation that the total drop rate depends only on the total traffic load (and is not a function of the individual flow rates).[13] Thus, we must have:

$$\sum_i \int_0^{r_i} dx \; d_i(\vec{r}, x) = D(\sum_i r_i)$$

The canonical example we use for the function D is that of an M/M/1/b queue with unit service rate

[12] Note that since $a(x) = 1$, $f(x)$ is both the per-bit value and the per-layer value and we can leave out the term $a(x)$ in the expression for $F(x)$.

[13] Note that since the throughput is bounded, we must have $\lim_{T \to \infty} D'(T) = 1$, and so $0 \leq D'(T) \leq 1$ for all T; this fact will be relevant later in the paper.

(modeling a bottleneck bandwidth of 1) and b buffers, so $D(z) = z^{b+1}\frac{1-z}{1-z^{b+1}}$. In the limit of infinite b, this reduces to the perfect bottleneck model where $D(z) = (z-1)_+$ (where we use the notation $x_+ = \max[0, x]$). The buffer size parameter b can also be seen as describing the smoothness of the traffic, with infinite b describing infinitely smooth traffic.[14]

We now compute the functions $d_i(\vec{r}, x)$ for uniform and priority dropping assuming FIFO scheduling.[15] For uniform dropping, the loss rates are given by:

$$d_i(\vec{r}, x) = \frac{D(\sum_i r_i)}{\sum_i r_i}$$

For priority dropping the loss rates are:

$$d_i(\vec{r}, x) = D'(\sum_j r_j \wedge x)$$

The drop fraction of each layer (which is independent of the flow) is given by the incremental increase in the total drop rate when that layer is added on (ignoring all lower priority layers).

We now use these simulation and theoretical frameworks to address our three key questions.

4 Performance

In this section we address the two questions concerning performance: (1) Do uniform and priority dropping achieve the same optimal performance? and (2) Can RLM achieve the optimal performance level under uniform dropping? We begin with results from our simulation experiments.

4.1 Simulation Results

Most of our simulation studies of performance are presented in the following form. For a given choice of utility function and network scenario, we simulated the network with each source sending up to level l for $l = 1, \ldots L$ under both priority and uniform dropping. In addition, we simulated all sources using RLM (with uniform drop). All simulations were run for 600 simulation seconds; data collected during an initial warmup period of 160 seconds was discarded. Per source utility was computed using as input the percentage of those packets sent during the 440 second period that were delivered to the receiver (in other words, we set $(1-d) = \frac{r}{s}$ where s is the number of packets sent in the layer and r is the number received). The data is presented on a single graph, with one curve describing uniform drop, one curve describing priority drop, and a horizontal line depicting the performance level achieved by RLM. The horizontal axis indicates the number of levels sent by each source (for the uniform and priority drop tests). Average utility per source is plotted on the vertical axis. Below we present the main results from our simulations experiments.

[14] Increasing the buffer size and decreasing the burstiness of traffic are roughly equivalent; they both decrease the dropping rate at a given level of throughput. In our theoretical model, we only vary the parameter b, but in our simulations we separately vary buffers and burstiness.

[15] See [1] for the analogous treatment of the Fair Queueing scheduling algorithm.

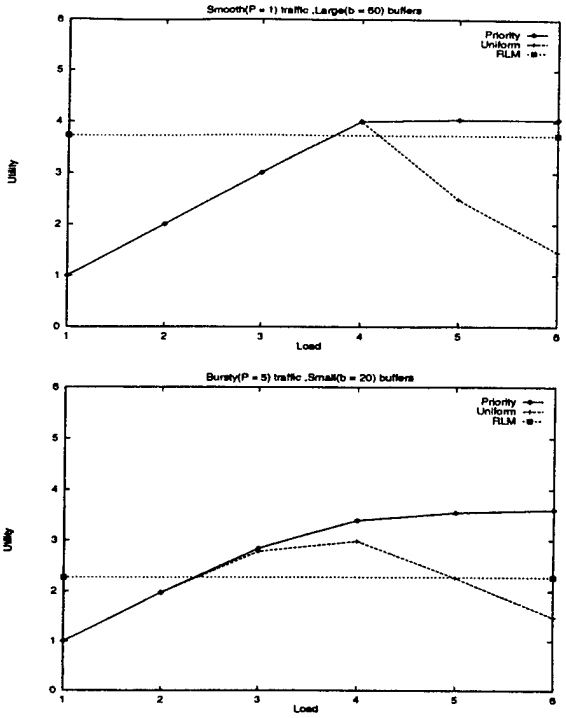


Figure 4: The top graph depicts the results with smooth ($P = 1$) traffic and large ($b = 60$) buffers. The peaks of the priority and uniform dropping curves are 4.04 and 4.00, respectively. RLM achieves a performance of 3.73. The bottom graph depicts the results with bursty ($P = 5$) traffic and small ($b = 20$) buffers. The peaks of the priority and uniform dropping curves are, respectively, 3.60 (priority) and 2.98 (uniform). RLM achieves a performance of 2.27.

4.1.1 Burstiness and Buffer Size

We first explore the effect of burstiness and buffer size on performance, using a linear utility function ($g(z) = z$). We vary the parameter P in our source model to control the burstiness of the traffic, and we vary the number of buffers, b, at the bottleneck link (4Mb) to control the ability of the switch to absorb bursts of packets. Figure 4 shows the results of two experiments, one where the number of packets sent per layer per interval is constant (*i.e.*, $P = 1$) and the buffers are large ($b = 60$, corresponding to 120 msec of buffering), and another where the traffic is burstier ($P = 5$) and the buffers smaller ($b = 20$).

For the case where the traffic is smooth and the buffers are large, the results are very much like those predicted in Figure 1. When offered load is less than the bottleneck link rate, uniform and priority dropping achieve the same utility. They both achieve the same maximum values, and then the performance of uniform dropping degrades as load increases, while the performance of priority dropping does not. RLM achieves nearly the same maximum utility.[16]

[16] The slight difference between RLM's utility and the maximum utility achieved by uniform and priority dropping is due to the absence of a mechanism to insure fairness in FIFO scheduling, and different flows can send at different rates (that is, the

Burstier data coupled with smaller buffers leads to markedly different results (as shown in the bottom graph in Figure 4). In this case, priority dropping achieves significantly higher average utility than uniform dropping (3.60 vs. 2.98).[17] The other striking difference between the smooth and bursty scenarios is the relative performance of RLM. Its utility with bursty traffic and smaller buffers is 2.27, around two-thirds of priority dropping and about three quarters of uniform dropping. With bursty traffic, there are enough losses, even at relatively lower levels of utilization to prevent RLM join experiments from succeeding. Thus, as we suspected, the results with bursty traffic look more like Figure 2 than like Figure 1.

One can relate these performance numbers to the equivalent amount of throughput being wasted. That is, with the exponential bandwidth per layer function $a(l) = 2^{l-1}a(1)$ we use in these simulations, the performance differential of between priority (3.60), uniform (2.98) and RLM (2.27) can be viewed as achieving useful throughputs of, respectively, 11.8, 6.92, and 4.08 (in units of the throughput of the base layer). The performance increase of 58% between RLM and priority dropping reflects roughly 1.3 additional layers (from $3.60 - 2.27 = 1.33$), which is equivalent to almost tripling the effective throughput. That is, while the priority dropping has a much higher effective throughput, the worth of these additional bits falls off so rapidly that the increase in utility is only 58%. If we use a linearly increasing bandwidth per layer $a(l) = l$ (see Figure 5) the performance numbers are 4.5 (priority), 3.71 (uniform), and 2.5 (RLM), corresponding to effective throughputs of 12.5, 8.84, and 4.50, respectively, again reflecting nearly a tripling of the effective throughput (and yielding an 80% increase in utility) when comparing priority to RLM.

The impact of burstiness on RLM's ability to adapt can be further demonstrated with the following contrived experiment where a single receiver performing RLM is adapting to available bandwidth in the face of bursty cross-traffic generated by a single on-off source with a sending rate of 6 Mbps, and exponentially distributed on and off times with average 100 msec. In this case, even though the utilization is only 65%, RLM is unable to utilize the available bandwidth and achieves a performance figure of only 1.19.

For completeness, we ran simulation of the two remaining combinations of buffer size and burstiness ($P = 1$ and $b = 20$; $P = 5$ and $b = 60$). The results (not shown here for lack of space) show that it is indeed *both* the burstiness of traffic and the size of buffers that account for the results described above. Bursty traffic presents opportunities for priority dropping to perform better than uniform dropping and also makes it more

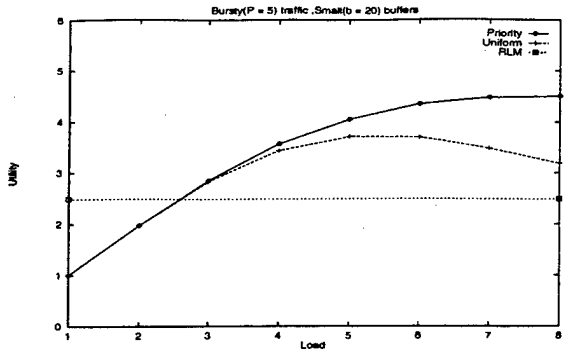


Figure 5: This graph depicts the results with bursty ($P = 5$) traffic and small ($b = 20$) buffers with $a(l) = l$. The peaks of the priority and uniform dropping curves are, respectively, 4.50 (priority) and 3.71 (uniform). RLM achieves a performance of 2.50.

difficult for the RLM adaptation algorithm to find the optimal operating level. As expected, large buffers can absorb burstiness and mitigate its ill-effects on the performance of uniform dropping and RLM.

4.1.2 Other Experiments

We conducted many additional simulation experiments to explore the impact of various parameters on our results. In particular we ran simulations with non-linear utility functions, different scheduling and dropping algorithms, varying bandwidth on the bottleneck link, and varying degrees of multiplexing. We briefly summarize those results here. A more detailed description is presented in [1].

To briefly summarize, in all of our experiments, priority dropping performed better than uniform dropping. As expected, this performance advantage increased with increased burstiness, decreased buffer sizes, decreased multiplexing, and decreased bandwidth per flow; the effect of nonlinear utility functions was ambiguous, sometimes increasing and sometimes decreasing the ratio. Using Fair Queueing instead of FIFO scheduling had little effect on results, since our simulations had fairly homogeneous flows; this is not a scenario in which fairness mechanisms can be expected to have an impact. In addition, we tested the RLM mechanism with RED [6], with little impact on results.

While we anticipated the direction of these effects of various parameters on the results, we thought their magnitudes would be much bigger. The performance advantage of priority dropping over the optimal uniform performance was quite modest; the biggest percentile advantage was roughly 27% (achieved in the case of reduced multiplexing). Results in the next section shed further light on the limits of the difference.

The performance advantage of priority dropping over RLM was somewhat more significant. For instance, in the case where the bottleneck capacity was reduced, priority dropping offered almost twice the performance of RLM. The performance advantage of priority over RLM had the same dependence on parameters as did the priority versus optimal uniform dropping, except that the effect of nonlinearities was

receivers are not all subscribed to the same levels). This unequal allocation of resources leads to a slight decrease in total utility, as bits sent in lower priority layers by one source could be replaced by more valuable bits of another source. This performance degradation suffered by RLM disappears when a scheduling mechanism that allocates equal shares per flow, such as Fair Queueing is used.

[17] Note that this difference may be somewhat overstated in the sense that we only performed the uniform and priority tests with all senders sending the same number of layers. The best possible performance of uniform dropping may in fact occur when some flows are sending 3 layers and some are sending 4.

no longer ambiguous. When the nonlinearity was quite extreme ($m = 100$) the performance advantage was essentially infinite (2.37 versus 0.02), but under more moderate nonlinearities ($m = 4$) the performance advantage was only 24%. While RLM did suffer under extremely bursty conditions, it was far more resilient than we had expected. This is partially due to the very rapid decrease in $f(l)$; for instance, if the per-flow bandwidth allowed roughly four levels per flow, then if RLM could utilize only 50% of the available bandwidth its performance disadvantage was a mere 25% (because it would get three out of the available four layers). When, as in Figure 5, the function $f(l)$ decreased less rapidly, the relative performance of RLM was significantly poorer.

4.2 Theory

We now turn to our theoretical model to provide additional insight about the performance results obtained through simulation. In particular, we derive results that are not bound to the particular layered source model used in simulations. To address the relative performance of uniform versus priority dropping, we first consider a single flow. The utility for uniform dropping is given by:

$$u(r) = g(1 - \frac{D(r)}{r})F(r) \tag{3}$$

The utility for priority dropping is given by:

$$u(r) = \int_0^r dx f(x) g(1 - D'(x)) \tag{4}$$

We start with a case that we can solve exactly: an M/M/1/b queue with $b = 1$, $g(z) = z^m$ for $m > 0$, and $f(x) = \frac{p}{(1+x)^{p+1}}$ for $p > 0$ (so $F(x) = 1 - \frac{1}{(1+x)^p}$). Here, $1 - d(r, x) = \frac{1}{1+r}$ in the uniform case, and $1 - d(r, x) = \frac{1}{(1+x)^2}$ in the priority case. Letting $u(r)$ denote the utility at a given sending rate r, in the uniform case we have $u(r) = (1+r)^{-m}(1 - \frac{1}{(1+r)^p})$ and so the maximizing r value is given by $1 + r = (1 + \frac{p}{m})^{\frac{1}{p}}$ and the maximal utility is $u = \frac{p}{m}(1+\frac{p}{m})^{-(\frac{m}{p}+1)}$. In the priority case, we have $u(r) = \int_0^r dx p(1+x)^{-(p+1)}(1+x)^{-2m} = \frac{p}{2m+p}(1 - (1+r)^{-(p+2m)})$, and so the maximal utility is just $u = \frac{p}{2m+p}$. Let R denote the ratio of maximal priority utility u^{pri} to maximal uniform utility u^{uni}. Setting $w = \frac{p}{m}$ yields:

$$R = \frac{u^{pri}}{u^{uni}} = \frac{(1+w)^{(1+w^{-1})}}{2+w}$$

This is a decreasing function in w so R attains its maximal value in the limit $\frac{p}{m} \to 0^+$, and there $R = \frac{e}{2} \approx 1.359\ldots$.[18]

We can numerically compute the ratio R for more general cases, varying b, m, and p. We also consider the function $f(x) = \beta e^{-\beta x}$ and do similar computations. The results are summarized below.

Typically R decreases with b; the performance advantage of priority dropping decreases as the traffic gets smoother (or, equivalently, the buffering gets larger). When varying the rate of decrease in $f(x)$ we find that the ratio decreases with p (when $f(x) = \frac{p}{(1+x)^{p+1}}$) as long as $p \geq 0$ but has a peak at intermediate values of β (when $f(x) = \beta e^{-\beta x}$). This is consistent with the intuition that priority dropping is of no use when the values $f(x)$ drop off too fast (so only the base layer is of significant value) or too slowly (so discriminating between layers is of little use). The dependence on m is somewhat more subtle; for $b = 1$ the maximal ratio occurs for large m, but for greater values of b the maximal ratio occurs for small m; we do not have an explanation for this behavior.

Note that in no case do we attain a higher value for R than $\frac{e}{2}$. While the roughly 36% increase in performance achieved by priority dropping is certainly significant, we must admit that we had expected priority dropping to achieve higher levels of improvement under at least some conditions. We now conjecture that as long as F and D are smooth around the origin, this ratio is the highest possible. Our reasoning is as follows. We assume that for a given g and F, the ratio R is highest when $D(z) \approx z^2$ for small z. This is reasonable since higher powers (*e.g.*, $D(z) \approx z^3$) give lower ratios (as shown by our data for higher values of b), fractional powers (*e.g.*, $D(z) \approx z^{1.5}$) mean D is not smooth near the origin contrary to our assumption, and $D(z) \approx z$ produces a smaller ratio (since $D'(z) \approx \frac{D(z)}{z}$ is roughly constant for small z if the leading term in $D(z)$ is linear, and so the ratio is increased if we remove the linear term from D). Moreover, we assume that for such initially quadratic $D(z)$, the ratio R increases when we substitute $(g(z))^2$ for $g(z)$, as suggested by our data for increasing m. Lastly, for $g(z) = z^m$ the performance for large m does not depend on F (as long as it is smooth near the origin, so $\frac{F(r)}{r}$ has a finite nonzero limit as $r \to 0^+$). Thus, it appears that the result that $R_{max} = \frac{e}{2}$ does not depend on the details of D (aside from the leading term around the origin) or F, and is reached in the limit of $g(z) = z^m$ for diverging m. We are not able to prove this conjecture; whether or not $R_{max} = \frac{e}{2}$ remains the most interesting open theory question arising from our study.[19]

Thus, one of our preconceived notions, that of priority being able to achieve extremely large improvements over uniform dropping, is likely wrong. Further, using the same model, we find that priority dropping does not outperform uniform dropping in call cases. It is straightforward to show that if g is a step function ($g(z) = 0$ if $x < \gamma$ and $g(z) = 1$ if $x \geq \gamma$ for some threshold $0 < \gamma < 1$) then uniform dropping outperforms priority dropping. However, one can show that

[18] Note that the case of $p = 0$ is ambiguous, so we only consider the limit. Setting $p = 0$ in the formula for F yields a different answer than setting $p = 0$ in the formula for f and then integrating; the limit is consistent with the formula derived from f, not F, in the case of $p = 0$.

[19] This bound is violated if we allow $D(z)$ to have a singularity at the origin, such as $D(z) \approx z^{1.5}$. In particular, when we look at the M/M/1/b formulae for $0 < b < 1$, where $D(z) \approx z^{1+b}$, we find that R diverges as m becomes infinite and b vanishes. We don't know what such singular $D(z)$ functions would signify (perhaps extreme burstiness), or if they are accurate representations of reality, but we do not address such singular cases in this paper.

if g is concave then priority dropping outperforms uniform dropping.

So far our theory has compared uniform and priority drop for only a single flow. We can extend our analysis to the multiple flow case without much additional complication. Consider the case where there are n flows; when $\vec{r}$ denotes the transmission rates then the utilities are given by equations 3 and 4. It is straightforward to show that, in the uniform case, the total utility $\sum_{i=1}^{n} u_i$ is maximized when $r_i = r_j$ for all i, j. Similarly, in the priority dropping case the total utility is maximized when all r_i are infinite. Consequently, the n-flow maximization problem with dropping function $D(z)$ yields the same optimality results, for both the uniform and priority dropping cases, as the 1-flow maximization problem with dropping function $\tilde{D}(z) = \frac{D(nz)}{n}$. Thus, the generalized form of our conjecture is that with an arbitrary number of flows (with the same conditions of smoothness on D and F), the maximal ratio of the total utilities of priority dropping to uniform dropping is bounded above by $\frac{e}{2}$.

5 Incentives

The previous results compared the performance of priority dropping to the optimal uniform dropping performance, and to that achieved by RLM. However, in this discussion we assumed that system-wide optimality was the only goal. We now remove that assumption and address our third question: what incentives do different dropping mechanisms present to applications? This issue has two aspects: (1) the properties of Nash equilibria in the presence of performance-based incentives, and (2) the effect of nonperformance incentives on these Nash equilibria. We now discuss these two aspects in turn.

5.1 Nash Equilibria

We first assume that there are no usage incentives other than performance. When using priority drop users have no performance-based incentive to constrain usage, whereas they do when using uniform drop. In this section we demonstrate that this does not necessarily imply that uniform drop naturally leads to socially optimal operating points. We return to our theoretical model, initially considering FIFO scheduling, and investigate what happens when there are n individually optimizing users; that is, we assume that users adjust their transmission level so as to maximize their own utility.

Consider the case of perfectly smooth traffic (modeled by setting $b = \infty$ in the M/M/1/b model), so $D(z) = (z-1)_+$. Then the maximal utility for a given user with priority dropping is at least $F(\frac{1}{n})$, with equality holding if all other users have $r_j \geq \frac{1}{n}$. Thus, the socially optimal point, and the Nash equilibrium point, is any vector $\vec{r}$ such that $r_j \geq \frac{1}{n}$ for all j.

For uniform dropping, the utility for a given user sending at rate x, with r sent by everyone else, is $u(x) = g(\min[1, \frac{1}{x+(n-1)r}])F(x)$. First we compute the socially optimal outcome (in which we know $r_i = r_j$ as we argued earlier), so we maximize the function $u(r) = g(\min[1, \frac{1}{nr}])F(r)$. Note that for $nr < 1$, $u(r) = F(r)$ and $F(r)$ is nondecreasing, so a maximal point must exist with $nr \geq 1$. For $nr > 1$, assuming all first derivatives exist, we can calculate $u'(r) = g(\frac{1}{nr})F'(r) - g'(\frac{1}{nr})F(r)\frac{1}{nr^2}$. Note that $rF'(r) \leq F(r)$. If g is concave, then we also have $g'(\frac{1}{nr}) \leq nrg(\frac{1}{nr})$ and thus $u'(r) \leq 0$ for $nr > 1$ and so at least one socially optimal operating point has $nr = 1$. However, there are cases with nonconcave g where the socially optimal operating point has $nr > 1$. Consider, for instance, the case where $g(z)$ is a step function (with the step at $z = \gamma < 1$). Then, the socially optimal value is given by $nr = \frac{1}{\gamma} > 1$ with per-flow utility $F(\frac{1}{n\gamma})$.

Next, we compute the Nash equilibrium. The equilibrium occurs when $\frac{du}{dx} = 0$ when $x = r$. Thus, at the equilibrium value r we must have $g(\frac{1}{nr})F'(r) = g'(\frac{1}{nr})F(r)\frac{1}{(nr)^2}$. Consider the case where $g(z) = z^m$. Then, the socially optimal value is $nr = 1$ for all $m \geq 1$ and the Nash equilibrium is reached when $\frac{rf(r)}{F(r)} = \frac{m}{n}$. For the case where $f(x) = (1-x)_+$, the Nash equilibrium is $r = \frac{1-\frac{m}{n}}{1-\frac{m}{2n}}$. The per-flow utility at this Nash equilibrium is $n^{-m}\frac{\frac{m}{n}}{2(1-\frac{m}{2n})}(\frac{1-\frac{m}{2n}}{1-\frac{m}{n}})^m$. This per-flow utility, and the total utility, vanish in the limit of large n or large m. The socially optimal per-flow utility is $\frac{1}{n}(1-\frac{1}{2n})$ for $m \geq 1$, and so the total utility $(1-\frac{1}{2n})$ approaches unity in the large n limit.

These results show that even when traffic is completely smooth (*i.e.*, in the infinite b limit), the Nash equilibria and the socially optimal operating points can be quite different; in particular, the Nash equilibrium can asymptotically (in the limit of large n) have zero total utility whereas the socially optimal point has full unit total utility. Thus, the performance-based incentives provided by uniform dropping do not lead to socially optimal, or even adequate, outcomes with FIFO scheduling. While we cannot solve these models exactly for burstier traffic, numerical computations for various choices of b and m for $f(x) = (1-x)_+$, $f(x) = \frac{p}{(1+x)^{p+1}}$ and $f(x) = \beta e^{-\beta x}$ show that these results continue to hold.

The intuition behind these results is that when a single user increases her usage, the penalty of that increased usage (in terms of an increased drop rate) is spread among all users, but the benefit (in terms of increased throughput) goes exclusively to the increasing user. Thus, the equilibrium occurs not when the benefit equals the penalty (which is the socially optimal point), but when $\frac{1}{n}$'th of the penalty equals the benefit, and this occurs only for much larger load levels (and this effect is magnified for larger n).

The above results show that the distinction between Nash and socially optimal operating points is significant when FIFO scheduling is used with uniform dropping. However, when uniform dropping is used with Fair Queueing, the first-order conditions for Nash equilibrium become identical to those for the socially optimal outcome. This is because the penalty for increased usage is born exclusively by the user with the highest sending rate, and so this user makes the socially optimal penalty/benefit tradeoff. Thus, with Fair Queueing, uniform dropping does indeed provide the proper

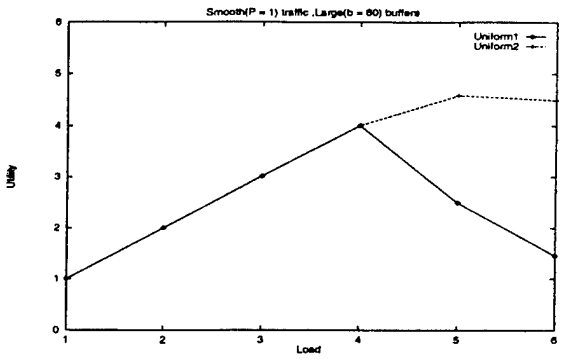


Figure 6: This graph compares the individually optimal and socially optimal operating points with $P = 5$ and $b = 60$. The curve 'Uniform1' describes the case when all 8 flows are sending the same number of levels (determined by the reading on the horizontal axis). Clearly the socially optimal operating point is when all flows are sending the first four levels. The curve "Uniform2' depicts the results when the 7 background flows are sending the first four levels, but the single test flow is sending up to the level determined by the reading on the horizontal axis. The utility of this flow is maximized when sending the first five levels. Thus, the socially optimal level is not a Nash equilibrium.

incentives.[20]

The simulations do not have exactly analogous behavior because the large granularity of the flow layers; each level consumes a significant amount of bandwidth, particularly the higher levels, and so typically one finds multiple equilibria rather than the single equilibrium points found in the simple theory models. Nonetheless, we can observe, as shown in Figure 6, that the socially optimal operating points are not Nash equilibria. The socially optimal result is to have each flow sending the first four levels, but when the background flows are held fixed at that level, a single test flow maximizes its utility by sending the first five levels.

One response to these dire results about Nash equilibria is that users won't be selfish and will just use RLM, which seems to achieve fairly reasonable results even for large n. This may very well be true. If such compliance is to be expected, then one can also expect users to use RLM with priority dropping to restrain usage, thereby removing the one problem with priority dropping. For instance, in the scenario with $P = 5$ and $b = 60$, RLM with priority dropping achieves a performance of 3.19 (versus 2.96 with uniform dropping) and constrains usage to only sending levels 1 through 3, plus occasionally level 4. The performance of RLM with priority dropping can be further improved by adjusting the dropping level at which RLM leaves a level (or considers a join experiment to have failed), thereby tuning the tradeoff between performance and bandwidth usage.

The behavior of RLM in the context of priority dropping is relatively insensitive to the burstiness of the traffic because the operating point need not be precisely at the bottleneck bandwidth. Thus, priority dropping is only a problem if we are worried about incentives, and in that case both priority and uniform dropping have problems (with FIFO scheduling).

Another response to the results about Nash equilibria is that there may be other sources of incentives that might make the Nash equilibria more palatable. These other incentives also presumably apply to priority dropping, and so usage would be constrained in both cases. We now address these other incentives.

5.2 Nonperformance Incentives

Above we assumed that the only incentives were related to the performance of the application itself. However, users clearly face other forms of incentives. The total bandwidth available to a user is limited to the speed of their access line, and for many users this access line is fairly slow (either a modem, or a shared T1); in addition, some hosts are quite limited in their network I/O. For any of these reasons, traffic from one application can have a performance impact on the user's other applications; *e.g.*, one's video traffic might congest one's own web traffic. Moreover, in some cases the Internet access charges are based, at least in part, on usage. Thus, we assume that there is some slight disincentive to send traffic. This seems compatible with reality given that most users do not usually send or receive traffic that they know is worthless. It is important to note that these incentives can either be applied to the sender or the receiver of the traffic, depending on the application (who is deriving value from the application) and the context (whose access line is congested).

We model this usage-constraining incentive by applying a cost c for each unit of bandwidth. While this conjures up the idea of a per-bit price, that is not what is implied; in *reality* this usage-constraining incentive can come from any one of a number of nonmonetary sources, but it is most easily *modeled* by assigning a cost for transmission. If the incentive is placed on the sender, the cost is applied for each unit of bandwidth *sent*, and if the incentive is placed on the receiver, the cost is applied for each unit of bandwidth *received* (*i.e.*, we assume that there is no receiver disincentive for packets that never arrive). A user maximizes utility minus cost. In the case of perfectly smooth traffic, an infinitesimally small c applied to the sender will provide incentive to operate at the bottleneck throughput; the excess traffic is completely worthless, and any nonzero disincentive is enough to throttle throughput back to the bottleneck capacity. However, if the incentive is present at the receiver, there is no constraint on usage, since no additional traffic arrives. Thus, applying incentives at the receiver involves somewhat of a paradox; we wish to prevent receivers from joining layers that will be mostly dropped, but need to do so by applying incentives only for those packets that are *not* dropped.

We can extend our theory model to look at this question for more general traffic loads, still with a single flow. We first consider only priority dropping, and assume that c is the cost of the incremental layer r (it

[20] While typically Fair Queueing does provide better incentives than FIFO, in general when users are heterogeneous the Nash equilibria under Fair Queueing are not socially optimal. This result holds in our example because users have the same utilities and flow structure. See [13] for a fuller discussion.

doesn't matter what the "charge" is for other layers, that will just be a fixed cost in these equations, ignoring income effects[21]). If the incentives are applied at the sender, the maximality condition is $u'(r) = c$ but if incentives are applied at the receiver, the maximality condition is $u'(r) = c(1 - D'(r))$. From the expression in equation 4, we find that the resulting equation is:

Sender Incentives $f(r)g(1 - D'(r)) = c$

Receiver Incentives $f(r)g(1-D'(r)) = c(1-D'(r))$

The relevant question is whether or not usage is stabilized for any nonzero c; by stabilized we mean that all solutions for r are finite for any nonzero c. When the sender is "charged" usage is stabilized if either $\lim_{r\to\infty} f(r) = 0$ or $\lim_{z\to 0+} g(z) = 0$. When the receiver is "charged" usage is stabilized if $\lim_{z\to 0+} \frac{g(z)}{z} = 0$ and $D'(z) < 1$ for all finite z. Note that these are sufficient but not necessary conditions, but we expect them (or other sufficient conditions) to hold quite widely.

For the example treated in Section 4 with $b = 1$ and $f(x) = p(1 + r)^{-(p+1)}$ and $g(z) = z^m$, we find:

Sender Incentives $1 + r = c^{\frac{-1}{p+2m+1}}$

Receiver Incentives $1 + r = c^{\frac{-1}{p+2m-1}}$

If we have $p \geq 1$ and $m > 0$ then usage is always stable in both the sender and receiver incentive cases; the usage levels decrease with increasing p and m.

Thus, for a wide range of conditions, usage stabilizes even if incentives apply only for received packets (although at higher levels of usage than if incentives apply for sent packets). While there is clearly no stabilization for $b = \infty$ when receivers are "charged" (this case violates the $D'(z) < 1$ assumption), numerical computations show that stabilization occurs for all finite b, and for other choices for f and g. The point is that with even slightly bursty traffic, some fraction of packets get through and so joining a layer that is almost completely useless (because of the high drop rates) is discouraged because the user is "charged" for the packets that do get through.

These nonperformance incentives also improve the nature of the Nash equilibrium under uniform dropping. We can return to our simple model and compute the Nash equilibrium after adding a small "charge" c for usage. One way to compare the relative effectiveness of these nonperformance incentives in the priority and uniform dropping cases is to look at the values of c for which they achieve r values that are comparable to the socially optimal r value for $c = 0$.[22] In the cases we computed, for sender "charging", priority dropping required lower levels of nonperformance incentives to restrain usage to these levels. Thus, these incidental sender incentives can more easily restrain usage in priority dropping than prevent the poor Nash equilibria under FIFO service with uniform dropping. When receivers incur these nonperformance incentives, the usage levels of uniform and priority dropping are roughly comparable (with the c values needed to restrain usage for priority dropping being slightly higher than those needed for uniform dropping). Note that, as expected, both uniform and priority dropping required significantly higher levels of these incentives when the receiver incurred them.

In light of these results, at the very least both uniform and priority dropping have problems with incentives, and one might make a reasonable case that priority dropping, because it is more easily restrained by sender-incurred nonperformance incentives, actually has better incentive properties than uniform dropping.

6 Discussion

This paper is devoted to a comparison of uniform and priority dropping in the fairly narrow context of layered video applications. Our results are both humbling and ambiguous. "Humbling" because some of our preconceived notions were wrong, and "ambiguous" because the results do not provide a clear answer to whether adopting priority dropping would provide significant benefit to layered digital video applications. Priority dropping certainly does result in higher performance than uniform dropping when g, the utility function, is concave. However, contrary to our expectations, the performance advantage of priority dropping over the optimal uniform dropping performance is quite modest. Moreover, we conjecture that there is a universal upper bound of roughly 36% on this performance gap.

However, the real point of comparison for uniform dropping is not the optimal point on the uniform dropping curve, which is an ideal point that we cannot reliably achieve in practice, but instead is the performance level achieved by an actual endpoint adaptation algorithm. For the comparisons in this paper, we used the RLM algorithm as an example of such an algorithm since we are aware of no algorithm that performs better. Here too we were surprised; the RLM adaptation algorithm is far more resilient than we expected. Under some rather extreme conditions – very bursty background traffic or high degrees of nonlinearity – RLM performed quite poorly; however, contrary to our initial expectations, RLM managed to achieve fairly adequate performance in a very broad range of less extreme conditions. Our current explanation for this is that when $f(l)$ decreases rapidly (as was the case in our simulations), one can use only a small fraction of the available bandwidth and still attain reasonable performance, since most of the value lies in the first few layers. While our expectations about the performance differences were irrationally exuberant, we do not want to minimize the fact that priority dropping achieved performance improvements of 50% to 100% over RLM in many settings. Thus, the performance improvements offered by priority dropping are indeed significant, and the conjectured bound of roughly 36% is not an indication of the relative performance of priority dropping to RLM, or to any other endpoint adaptation algorithm.

[21] Income effects are where the marginal utility of money depends on the total amount.

[22] This is a somewhat arbitrary comparison, but it is asking how large do these nonperformance incentives have to be to restrain usage to a given level, and choosing the socially optimal level for $c = 0$ as that level seems like a reasonable choice.

While the performance properties of uniform drop were unexpectedly good, the incentive properties of uniform drop with FIFO service were, at least in theory, surprisingly poor. Moreover, these incentive properties, or more particularly the performance at the Nash equilibrium, were especially bad with a large population of flows. Using Fair Queueing instead of FIFO largely alleviates these incentive problems for uniform dropping.

In contrast, the incentive aspects of priority dropping may not be as bad as advertised. While priority dropping does provide poor performance-based incentives, even minimal amounts of nonperformance incentives will rectify the situation. In fact, when these usage-constraining incentives are incurred at the sender and when FIFO service is used, usage is more easily constrained in the priority dropping case than in the uniform dropping case.

Thus, we end up with somewhat of a paradox. If one takes the incentive issues seriously, then priority dropping may be better than uniform dropping, at least with FIFO service. But if one conjectures that instead users will be well-behaved and use RLM with uniform dropping in spite of their own personal incentives, then one could just as easily conjecture that users would use RLM (or some similar protocol) with priority dropping, which would yield better overall performance at the same level of bandwidth consumption. Thus, the argument that incentives are the reason to prefer uniform dropping to priority dropping is only valid if one believes that Fair Queueing (or some other protocol that enforces fairness) is used, or that the sender-based nonperformance usage constraining incentives, whatever their origin, are vanishingly small.

We hasten to note that this is only an initial study of this rather fundamental question. There are many unresolved questions about the performance and incentive properties of uniform and priority dropping. However, we think the most pressing open questions left by our study are those concerning the nature of application utility functions. We do not, nor does the research community we suspect, have a sense of whether our utility models capture the essential aspects of reality. Knowing how to best model application utility would provide much-needed guidance to network designers in their analysis of network performance, and may lead to more definitive answers to the questions posed here.

References

[1] Sandeep Bajaj, Lee Breslau, and Scott Shenker. Uniform versus priority dropping for layered video - extended version. Preprint, June 1998.

[2] Jean-Chrysostome Bolot and Thierry Turletti. A rate control mechanism for packet video in the internet. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Toronto, Canada, June 1994.

[3] Jean-Chrysostome Bolot, Thierry Turletti, and Ian Wakeman. Scalable feedback control for multicast video distribution in the internet. In *Proceedings of ACM Sigcomm*, pages 58–67, London, England, August 1994. ACM.

[4] Stephen Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[5] Steve Deering. Internet multicast routing: State of the art and open research issues. *Multimedia Integrated Conferencing for Europe (MICE) Seminar at the Swedish Institute of Computer Science, Stockholm*, October 1993.

[6] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[7] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, Massacachusetts, 1991.

[8] M. W. Garrett and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. *Computer Communications Review*, 24(4), October 1994. SIGCOMM '94 Symposium.

[9] D. Hoffman and M. Speer. Hierarchical video distribution over internet style networks. *Proceedings of IEEE International Conference on Image Processing, Lausanne, Switzerland*, pages 5–8, September 1996.

[10] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In Deepinder P. Sidhu, editor, *Proceedings of ACM Sigcomm*, pages 183–193, San Francisco, California, September 1993. ACM. also in *Computer Communication Review* 23 (4), Oct. 1992.

[11] S. McCanne and M. Vetterli. Joint source/channel coding for multicast packet video. *Proceedings of IEEE International Conference on Image Processing, Washington, DC*, pages 25–28, October 1995.

[12] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *Proceedings of ACM Sigcomm*, pages 117–130, Palo Alto, California, August 1996.

[13] Scott Shenker. A game theoretic analysis of switch service disciplines. *Transactions on Networks*, 3(6):819–831, 1995.

[14] Thierry Turletti. The INRIA videoconferencing system IVS. *Connexions*, 8(10):20–24, October 1994.

[15] Thierry Turletti and Jean-Chrysostome Bolot. Issues with multicast video distribution in heterogenous packet networks. *Proceedings of Sixth International Workshop on Packet Video Portland, OR*, September 1994.

[16] M. Vishwanath and P. Chou. An efficient algorithm for hierarchical compression of video. *Proceedings of IEEE Interantional Conference on Image Processing, Austin, TX*, November 1994.

Comments on the Performance of Measurement-Based Admission Control Algorithms

Lee Breslau
AT&T Labs - Research
75 Willow Road
Menlo Park, CA 94025
breslau@research.att.com

Sugih Jamin
EECS Department
University of Michigan
Ann Arbor, MI 48109-2122
jamin@eecs.umich.edu

Scott Shenker
International Computer Science Institute
1947 Center Street
Berkeley, CA 94704-1198
shenker@icsi.berkeley.edu

Abstract—**Relaxed real-time services that do not provide guaranteed loss rates or delay bounds are of considerable interest in the Internet, since these services can achieve higher utilization than hard real-time services while still providing adequate service to adaptive real-time applications. Achieving this higher level of utilization depends on an admission control algorithm that does not rely on worst-case bounds to guide its admission decisions. Measurement-based admission control is one such approach, and several measurement-based admission control algorithms have been proposed in the literature. In this paper, we use simulation to compare the performance of several of these algorithms. We find that all of them achieve nearly the same utilization for a given packet loss rate, and that none of them are capable of accurately meeting loss targets.**

I. Introduction

In an effort to better support applications with real-time constraints, several new per-flow packet delivery services have been proposed for the Internet (e.g., [24], [26]).[1] Lying between the extremes of *hard* real-time services (which provide worst-case guarantees) and the vagaries of the current best-effort service are *soft* real-time services that provide an enhanced quality of service without making hard guarantees. Specifications for these services might provide a delay *target*, rather than a bound, and permit periodic excursions above this target [6], or they might specify that the service provides low delay and low loss without quantifying actual performance [26].

One key difference between hard and soft real-time services is the nature of their admission control algorithms. Hard real-time services necessarily use *parameter-based* admission control algorithms that are based on worst case bounds derived from the parameters describing the flow; these algorithms typically result in low network utilization in the face of bursty network traffic. Soft real-time services can use less stringent admission control algorithms. It has long been recognized that *measurement-based* admission control algorithms (MBACs) are more appropriate for these soft real-time services [6], [18]. Because they base admission control decisions on measurements of existing traffic rather than on worst-case bounds about traffic behavior, MBACs can achieve much higher network utilization than parameter-based algorithms while still providing acceptable service [19]. Of course, traffic measurements are not always good predictors of future behavior, and so the measurement-based approach to admission control can lead to occasional packet losses or delays that exceed desired levels. However, such occasional service failures are acceptable given the relaxed nature of the service commitment provided by soft real-time services.

In designing a measurement-based admission control algorithm, one can conceivably have two goals. One is to provide a parameter that accurately estimates *a priori* the level of service failures that will result. The other is to achieve the highest possible utilization for a given level of service failures. Several measurement-based admission control algorithms have been proposed in the literature (see, for example, [7], [10], [11], [13], [14], [15], [16], [17], [19], [20], [21]) and they implicitly or explicitly seek to achieve one or both of these design goals.

The proposed algorithms, although embracing similar goals, differ in four important ways. First, some algorithms are *principled*, based on solid mathematical foundations such as Large Deviation theory, and others are *ad hoc*, in that they lack a theoretical underpinning. Second, the specific equations used in making admission decisions are quite different. Third, while all algorithms have a parameter that varies the level of achieved performance and utilization (by making the algorithm more or less aggressive), some algorithms attempt to calibrate this parameter and have it serve as an accurate estimate of the resulting performance, while others leave the parameter uncalibrated; in the latter case it is assumed the network operator will learn appropriate parameter settings over time. Fourth, the measurement processes used to produce an estimate of network load are very

This work was begun while Lee Breslau and Scott Shenker were with the Xerox Palo Alto Research Center. At Xerox, this work was supported in part by the Defense Advanced Research Projects Agency, monitored by Fort Huachuca under contract DABT63-96-C-0105. Sugih Jamin's research is supported in part by the NSF CAREER Award ANI-9734145 and the Presidential Early Career Award for Scientists and Engineers (PECASE) 1998. Additional funding is provided by MCI WorldCom, Lucent Bell-Labs, and Fujitsu Laboratories America, and by equipment grants from Sun Microsystems Inc. and Compaq Corp.

[1] Here we are restricting our attention to services that make per-flow assurances; we are not addressing services that only give aggregate service assurances, such as the recent Differentiated Services proposals [5], [22], since they do not rely on per-flow admission control.

different; they range from a simple point sample estimate, to an exponentially weighted average, to estimates based on both the mean and variance of measured load. Thus, the space of measurement-based admission control algorithms is both heavily and broadly populated.

Somewhat surprisingly, given the number of papers on the subject, no comprehensive comparison of these algorithms exists. Previous comparisons (including our own previous work on the subject) look only at a few test cases, and then only for a few of the algorithms [20], [21]. In this paper we extend this previous work by considering more (although by no means all) of the proposed algorithms, and by subjecting them to more extensive tests. In all of these tests we use packet losses as the definition of a service failure.[2] We evaluate the algorithms according to how well they are able to meet the two goals of MBACs. First, we compare the *performance frontier* or *loss-load curve* (we will use these terms interchangeably) achieved by each algorithm; the loss-load curve depicts the rate of losses that occur at a given level of utilization. Second, for those algorithms that attempt to predict the resulting level of losses, how close is the resulting performance to the target?

On the first goal, we find that even though the algorithms are derived from diverse motivations and theories, they all produce essentially the same performance frontier. The particular theory upon which they are based and the specific admission equations they use seem to be of little consequence. Regarding the second goal, we find that none of the algorithms achieve the specified performance targets consistently. However, some algorithms do somewhat better than others; whether these differences are important, and whether future algorithms can do better, remains an open question.

The remainder of this paper is organized as follows. In the next section we describe the algorithms we include in our study and briefly review previous performance comparisons of the algorithms. In Section III we describe our simulation methodology and present experimental results comparing the performance frontiers of the various algorithms. In Section IV we study the extent to which algorithms can accurately predict the resulting loss level. We summarize our findings in Section V.

II. Measurement-Based Admission Control Algorithms

To give the context necessary for discussing our results, in this section we very briefly describe the six admission control algorithms whose performance we study. These algorithms represent a broad, though not complete, sample of existing MBACs. Each algorithm has two key components: a measurement process that produces an estimate of network load, and a decision algorithm that uses this load estimate to make admission control decisions. After presenting each of the six algorithms, we elaborate on some common features of the algorithms.

For the purposes of this study, we assume that applications use a signaling protocol, such as RSVP [3], to make their requests for service to the network. These service requests contain a traffic descriptor describing the worst case behavior of the application traffic. The traffic descriptor takes the form of a token bucket with parameters r and b denoting the token rate and bucket depth, respectively.[3] We measure the quality of the service delivered in terms of packet drops. Soft real-time services are typically intended to be scalable, therefore we only consider MBACs that require no per-flow state; that is, the measurements are taken on the aggregate traffic, not on individual flows. Since measurement is done on the aggregate and admission control decisions are made on a per flow, rather than a per packet basis, implementation overhead is not critical [20] and is not explored in this paper.

Some admission control algorithms do not fit within the framework we consider and are excluded from our study. For example, we do not include one of the MBACs described in [13] because it depends on per-flow (rather than aggregate) measurements. In addition to excluding algorithms that require per-flow measurements, we also do not consider algorithms that make any assumptions, either implicitly or explicitly, about the average behavior of flows. For example, we do not include the MBAC presented in [16] because it computes a per-flow average estimate and assumes that all arriving and departing flows conform to that average. We only consider algorithms that make no assumption about what a flow's contribution will be to aggregate load beyond the worst case parameters supplied by the flow. Similarly, when a flow departs the network, its prior contribution to aggregate load can only be determined by measuring subsequent aggregate load.

Following are brief sketches of the six admission control algorithms we compare:

- **Measured Sum (MS).** The Measured Sum algorithm [20] admits a new flow if the sum of the token rate of the new flow and the estimated rate of existing flows is less than a utilization target times the link bandwidth. A time window estimator is used to derive the estimated rate of existing flows.
- **Hoeffding Bounds (HB).** The admission control algorithm described in [11] computes the equivalent bandwidth for a set of flows using the Hoeffding bounds. A new flow is admitted if the sum of the peak rate of the new flow and the measured equivalent bandwidth is less than the link utilization. An exponential averaging measurement mechanism is used to produce the load estimate.
- **Tangent at Peak (TP).** Four measurement-based admission control algorithms are presented in [13]. The first algorithm, based on the tangent at the peak of an equivalent bandwidth curve computed from the Chernoff Bounds, admits a new flow if the following condition is met:

$$np(1 - e^{-sp}) + e^{-sp}\widehat{\nu} \leq \mu, \tag{1}$$

where n is the number of admitted flows, p is the peak rate of the flows, s is the space parameter of the Chernoff Bound, $\widehat{\nu}$ is the estimate of current load, and μ is the link bandwidth. This algorithm uses a point sample measurement process.
- **Tangent at Origin (TO).** A second algorithm presented in [13] uses a tangent to the equivalent bandwidth curve at the origin. Here, a new flow is admitted if the following equation is

[2] Violations of a delay target may also be a relevant characteristic. However, for the fixed buffer regime we study, this is sufficiently similar to loss and so we do not treat it separately.

[3] Some of the admission control algorithms require a peak rate p. Following [11], the peak rate is computed from the token bucket parameters as $p = r + b/T$, where T is the basic measurement interval used by the algorithm.

satisfied:

$$e^{sp}\widehat{\nu} \leq \mu. \quad (2)$$

This admission control algorithm also uses the point sample measurement process.[4]

• **Measure CAC (MC).** The *Measure* admission control algorithm [7], which is based on large deviation theory, admits a new flow if the sum of the peak rate of the flow and the estimated bandwidth of existing flows is less than the link bandwidth. The estimated bandwidth takes as input a target loss rate and makes use of the scaled cumulant generating function of the arrival process.

• **Aggregate Traffic Envelopes (TE).** The admission control algorithm in [21] uses measurements of the maximal traffic envelopes of the aggregate traffic, capturing variability on different time scales. Both the average and variance of these traffic envelopes, as well as a target loss rate, are used as input into the admission algorithm.

The brief descriptions presented above ignore the details of the individual algorithms, but the key point is that the algorithms differ both in their underlying theory and in the specific measurement and admission control equations they use. While these differences are what we seek to understand in this paper, certain similarities are worth noting. For instance, each of these algorithms has one component that derives a load estimate based on measured traffic and another component that makes an admission decision using this load estimate. Rather than treating each algorithm as a monolithic block, it is possible in some cases to pair the estimation process of one algorithm with the decision process of another. This allows us to ask whether differences in performance derive from the estimation process, the decision process, or both. We undertake this "mix and match" analysis in Section III.

In addition to the equations that form the basis of the algorithms described above, there are also certain MBAC features that address specific practical concerns. For instance, when a new flow is admitted to the network, the existing load estimates will not immediately reflect the presence of the new flow. In such a case, the network runs the risk of admitting too many flows before recognizing that load has increased. To prevent this situation, some of the algorithms (MS, HB, MC) artificially increase the load estimate to account for a newly admitted flow. This feature, while included in the specifications of three algorithms, can be seen as an independent mechanism that can be applied to any of them. We eliminate this feature as a source of performance differences between algorithms by including it in all of the algorithms in our performance comparison.[5]

A final observation is that each of the admission control equations has one or more parameters that control their operation. For example, the MS algorithm has a *utilization target* that affects how many flows will be admitted, the MC and TE algorithms use a *target loss rate*, and the HB algorithm has a parameter that indicates the probability that the actual bandwidth requirement exceeds the estimates. While these parameters were not all intended as tuning parameters by the designers of the algorithms, adjusting these parameters will make the algorithms either more conservative or more aggressive with regard to the number of flows they admit. Hence, instead of providing a single level of performance, each algorithm enables a range of loss rates and utilizations depending on the values of these parameters. Thus, we describe the utilization performance of these algorithms by their loss-load curves or performance frontiers.

This paper is an extension of our earlier work [20]. In that paper we compared three different measurement-based admission control algorithms (MS, HB, and an acceptance region based MBAC from [14] which was later generalized in [13]) and one simple parameter-based admission control algorithm. These algorithms were compared for several different traffic loads (similar to those we use here, to be described in Section III-A) and on single link and multiple link network topologies (as we discuss in Section III-A, we only use a single link network topology in this paper). The simulation results in the earlier paper were deficient in several respects. The algorithms were only tested at one parameter value setting. Such *point comparisons* cannot describe the entire performance frontier provided by an admission control algorithm, and so do not adequately characterize the performance of an MBAC. Moreover, for the particular parameter values and traffic models used in [20], the admission control algorithms recorded no losses, so only the utilization figures could be compared. Also, there was no attempt to compare the target loss rate with the actual loss rates, so there are no results analogous to those in Section IV. Thus, this previous work did not adequately answer the relevant question: how well do the various MBACs satisfy the two goals of measurement-based admission control?

There have been few other attempts to systematically compare the performance of measurement-based admission control algorithms. The closest work is [21], in which the performance of the TE algorithm is compared to that of HB and the algorithm specified in [19].[6] The authors of [21] compare utilization achieved for particular quality of service targets, and do not compare the performance frontiers of the algorithms; however, the main thrust of [21] is on achieving accurate loss estimates, and to evaluate success along that dimension it is not necessary to investigate the entire performance frontier.

In one other related piece of work, in a short (three page) discussion paper [4] we briefly review some of the research presented here and then use that to argue that the research agenda in measurement-based admission control should address certain policy issues (such as how to allocate admission between large and small flows, and between flows traveling many hops and those traveling fewer hops).

III. Performance Frontiers

In this section we evaluate how well each of the six algorithms performs with respect to the first goal: achieving high network utilization and low packet loss. We first describe our simulation methodology and present our basic results for the MBACs with

[4] A third algorithm presented in [13] is equivalent to the HB algorithm. As described above, the fourth algorithm is excluded because it depends on per-flow measurements.

[5] Results of simulations not included in this paper show the importance of this feature. Under highly dynamic conditions, performance can degrade if estimation algorithms do not account for the presence of newly admitted flows.

[6] Based on communication with the author of [11], we do not interpret the parameter in HB as a performance target; however, one could easily make that interpretation, and that is what is done in [21].

several different source models. We then focus on three specific issues: the impact of heterogeneous traffic, a comparison between MBACs and an ideal parameter-based algorithm, and implications of long range dependent traffic on measurement-based admission control. Throughout the discussion and accompanying figures, we refer to the algorithms by the abbreviations introduced in the previous section: MS, HB, TO, TP, MC, TE.

A. Simulation Methodology

We use discrete event simulation to generate performance frontiers for each algorithm. Simulations were carried out using the *ns* network simulator.[7] In order to understand the behavior of the algorithms in the most simple case, we used a simple topology in which admission control was employed on a single bottleneck link. While interesting issues may arise when studying admission control in a multi-link scenario, the basic performance aspects of these algorithms are most easily revealed in this simpler one-link configuration, particularly since the admission control decisions for each of the algorithms are made on a link-by-link basis. Further, we expect that issues arising in a multi-link scenario (e.g., discrimination against larger flows and flows traversing longer paths [4], [19]) are independent of the particular algorithms and are, therefore, orthogonal to the questions we ask here.[8]

A simulation experiment consists of a random process of flow arrivals. Each flow requests service from the network using a simple resource reservation protocol, and it is admitted or rejected according to the specifics of the algorithm in question. A rejected flow departs the network without sending any data packets and does not retry its service request again. A flow that is accepted sends data packets for a flow lifetime chosen from a random distribution. Packets are generated according to a source model selected for the flow when it is created.

We use two kinds of source models in our experiments. The first is an ON/OFF source, in which the source transmits at a constant rate during a randomly chosen ON period, and then remains idle for a randomly chosen OFF time. The second kind of source model uses a trace of video traffic to drive the simulation. The specific parameters are described below. Packets generated by a source are subject to policing by a token bucket filter. The token bucket parameters (rate and bucket depth) are included in the reservation request that is handed to the admission control module.

For each simulation, the average utilization and packet loss rate are measured. Data collected during an initial warmup period are discarded. All simulations were repeated using different seeds to the random number generator. The number of repetitions and the length of each simulation were varied depending on the underlying variability of the source model and offered load used in each experiment. The averages across all repetitions are reported in our results.

In all experiments, the bottleneck link bandwidth is 10 Mbps. Unless otherwise noted, packets are 128 bytes long, and there is buffering for 160 packets at the bottleneck link. In most of our experiments, the total offered load (in terms of the number of flows requesting service) is high, leading to a high call rejection rate. While the actual rejection rates may be unrealistically high, it is in the regime of overload that the behavior of the admission control algorithms is most interesting.

Each of the algorithms has several parameters that control how much history is maintained by the estimation algorithm. We tried, when possible, to use parameter settings suggested in the original references. However, in some cases we found that changing these values yielded better performance. We suspect that this is due to differences between our source models and offered load and those used by other researchers. In all cases, we used those parameter values that yielded the best performance in our experiments.

B. Results

Our first experiments use homogeneous on/off sources with exponentially distributed on and off times (325ms average). The transmission rate during on periods is 64kbps, making the average rate 32 kbps. The token rate and bucket depth are set to 64 kbps and 1 packet, respectively (assuring no loss at the token bucket filter). These parameters are consistent with PCM coded voice that might be produced by an IP telephony application. On average each source consumes about .3% of the link bandwidth. Flow inter-arrival times are exponentially distributed with a mean of 400 ms. Flow lifetimes, which are also exponentially distributed, have a mean of 300 seconds. We refer to this traffic model as the EXP1 source. Simulations were run for 6000 simulation seconds; data collected during the first 1500 seconds was discarded. Each simulation was repeated 5 times with different seeds to the random number generator.

Results for this experiment are shown in Figure 1. This graph plots the packet loss rate on a log scale as a function of link utilization. A performance frontier is shown for each of the six algorithms.[9] It is difficult to distinguish between the performance frontiers in the graph, indicating that all of the algorithms yield very similar performance. That is, they all permit essentially the same choices in the tradeoff between loss rate and utilization. Further, the very slight differences in performance are not of practical importance, because even if one algorithm yields a marginally higher loss rate than another at a given level of utilization, the loss rates can be made equivalent with extremely small changes in utilization. Because there is variance in both the x and y values in the figure (i.e., a given MBAC input parameter determines both the utilization and packet loss rate), and these variations are highly correlated and not normally distributed, we do not depict these variations as error bars in our graphs. However, the variance across simulation runs is small.

In some cases, the interfaces between the estimation and decision components of each algorithm are such that the estimation process of one can be used with the decision process of another. When this was possible, we "mixed and matched" the various components. Specifically, the MS, HB, TO and TP decision al-

[7] http://www-mash.cs.berkeley.edu/ns/.

[8] This is not to say we don't think these issues are interesting. In fact, given the results we present here, we make the case in [4] that these issues of discrimination mentioned above should be considered more seriously by MBAC researchers.

[9] Because utilization is not an independent variable in these experiments, data points are not plotted for the same x values for each algorithm. The actual number of points plotted varies across algorithms, but we have covered an overlapping range on the x axis for each curve.

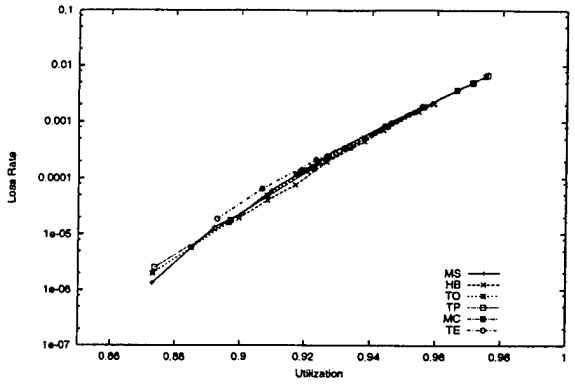


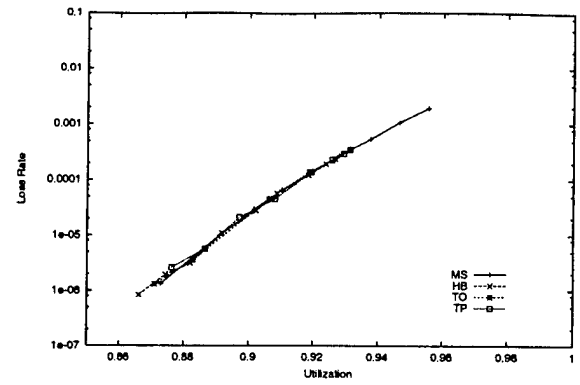
Fig. 1. Performance frontiers of measurement-based admission control algorithms with EXP1 traffic

gorithms were run with the Time Window, Exponential Averaging, and Point Sample estimators in order to understand the degree to which each component impacts the results. Figure 2 shows the results for these three estimators with the four different decision algorithms. Relative to Figure 1, the slight variations across algorithms have been reduced. This result demonstrates two things. First, the conclusion above that each algorithm has nearly the same performance frontier does not depend on any particular coupling between estimation and decision processes. Second, the reduced variance indicates that it is the estimation process, and not the decision algorithm that is responsible for the slight variations in Figure 1.

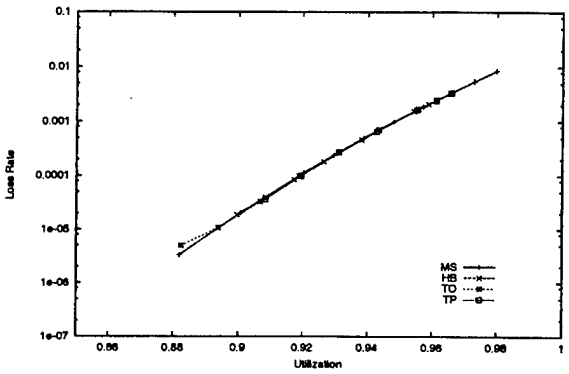
We performed additional experiments using the following source models:

- EXP2 – in this source model, the peak rate is increased by a factor of 10 (640 kbps versus 64 kbps) relative to the EXP1 source while the average rate is held constant, leading to a burstier source model.
- POO1 – this is an on/off source with the same averages as the EXP1 source. However, they are taken from a Pareto distribution. Flow lifetimes are taken from a lognormal distribution with a median of 300 seconds following [2], [9]. The aggregation of these sources produces traffic that is long range dependent [8], [25].
- STARWARS – this source model is taken from a trace file produced by an MPEG encoding of the Star Wars motion picture [12]. Each source starts from a random place within the trace file in order to avoid correlation among the sources. This source model differs from the previous ones in that it has a higher average rate (350kbps vs 32kbps) resulting in a lower degree of multiplexing, and it is characteristic of traffic produced by a video source rather than an on/off model. With this source model, packets are 200 bytes long and there are 500 packet buffers at the bottleneck link.
- HET – this experiment consists of a mix of six different on/off sources, with varying average rates, idle times and burst times. Each arriving flow chooses from among these source models at random. All flows have the same leaky bucket parameters, so they appear identical to the admission control algorithm.

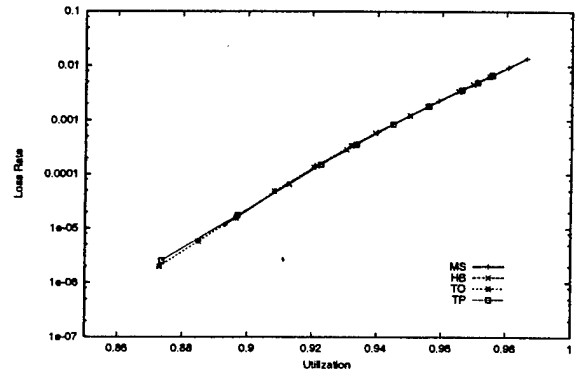
The results from these experiments (not shown here) reveal that our basic result holds across different traffic models. That is,



(a) Time Window



(b) Exponential Averaging



(c) Point Sample

Fig. 2. Decision algorithms paired with different estimators for the EXP1 source model

in the presence of burstier sources, long range dependent traffic, lower multiplexing, traffic derived from a video trace, and heterogeneous traffic (with identical token bucket parameters), all of the algorithms achieve roughly the same performance frontier. In addition, we repeated the experiments with the EXP1 traffic source and more moderate offered load (yielding a lower call rejection rate.) The essential results were unchanged under these conditions.

C. More on Heterogeneous Traffic

We now briefly return to the issue of heterogeneous traffic. In the simulation with heterogeneous traffic described above, all flows had identical token bucket parameters, and so were indis-

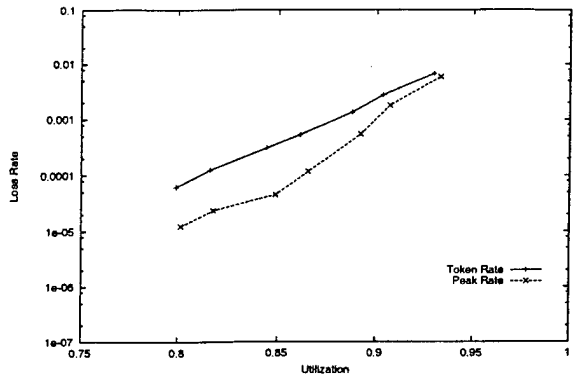


Fig. 3. Peak rate versus token rate versions of the MS admission control algorithm with heterogeneous traffic

tinguishable to the admission control algorithms. The results in this case were consistent with those in the homogeneous experiments. We now ask what happens when the token bucket parameters are no longer identical, allowing the admission control algorithms to admit them differentially. Not surprisingly, when the flows are distinguishable, different admission control algorithms lead to different mixtures of traffic, and hence to different performance frontiers. To illustrate this, we consider an experiment where each arriving flow used one of the following two source models, chosen with equal probability. The first source model was the Star Wars trace introduced above. This trace had an average rate of approximately 350 kbps. In order to accommodate its burstiness, the token bucket parameters are $r = 800kbps$ and $b = 200kb$. The second source model was a Constant Bit Rate (CBR) source sending at 800 kbps. The token bucket parameters for this source are $r = 800kbps$ and $b = 1.6kb$ (to hold a single packet).

Figure 3 shows results for this heterogeneous traffic mix with 2 admission control algorithms. The first is the Measured Sum (MS) algorithm, which uses the token rate of the new flow. The second algorithm is a variant of Measured Sum using the peak rate (computed as $p = r + b/T$, with $T = 500ms$ in our experiments) of the incoming flow, rather than its token rate, in the admission control equation. The first version of the MS algorithm does not discriminate between the two kinds of flows because they have the same token rates. This leads to a traffic mix that is made up of roughly equivalent numbers of the two kinds of flows. The peak rate algorithm, on the other hand, discriminates against the trace driven flows, as they have a higher peak rate ($1200kbps$ vs. $800kbps$). This leads to a traffic mix in which the CBR sources outnumber the video sources by a ratio of approximately 3:1. Consequently, the peak rate algorithm has a better performance frontier than the token rate algorithm. We introduced the peak rate version of the MS admission control algorithm to accentuate the extent of discrimination. One finds similar, but less extreme, results when comparing the six admission control algorithms we have discussed in this paper under heterogeneous traffic loads with distinguishable flows.

Note that the traffic mix admitted by the peak-rate algorithm is, in the aggregate, less bursty than the one admitted by the token rate algorithm; thus, the loss rate experienced at an equivalent utilization is lower than is experienced with the token rate admission control algorithm. In general, when admission control algorithms admit different mixtures of flows, the aggregate traffic will have different degrees of burstiness, and so the performance frontiers will no longer be the same. Thus, in the face of heterogeneous and distinguishable flows, MBACs don't necessarily produce the same performance frontier.

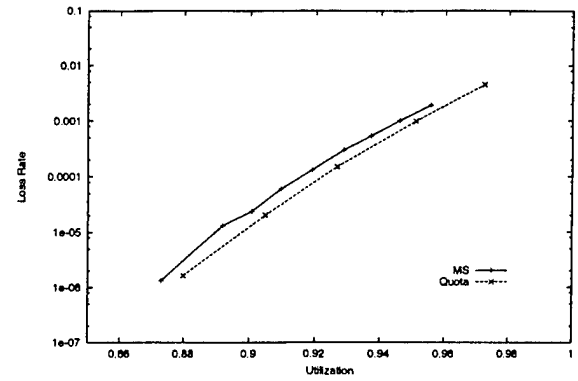
One might think that this would undercut our observation about the equivalence between various MBACs. However, we think that the question of which traffic mixture should be admitted is one of policy, not efficiency. Clearly one could minimize the loss rates by admitting only CBR-like flows, but such a limitation would be unwise as it would preclude bursty sources from obtaining reasonable service. Admission control algorithms that happen to pick less bursty flows to admit, while providing superior performance frontiers (in the presence of heterogeneous and distinguishable traffic) are not necessarily more desirable and in fact have only made one particular policy choice out of a broad range of possible choices.

D. Comparison with an Ideal Algorithm

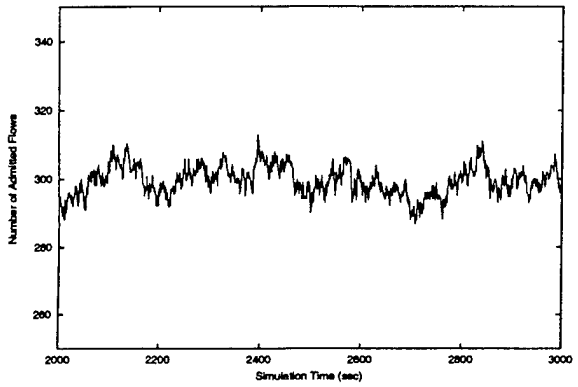
We now elaborate on our result that all of the algorithms have similar performance frontiers. With so much effort going into the design of measurement-based admission control algorithms, one might have assumed that the effort would lead to improved performance. Our simulations suggest quite the opposite, that even very simple *ad hoc* algorithms achieve the same performance frontier as more complicated and more principled ones. Given this, we ask two questions. First, why are the differences in performance between the algorithms so small? Second, are there untapped advantages not yet realized by any of these algorithms or are they in fact all performing at or near some optimal level? To answer these questions, we construct an "ideal" algorithm.

Consider our initial experiment with the EXP1 traffic source. In this simulation, all flows in the network were homogeneous exponential on/off sources. The aggregate traffic generated by these sources has no long term correlation. Further, the time scale at which individual sources change between the idle state and the active state (100s ms) is shorter than the time scale at which new flows are admitted to the network (seconds). Thus, it is impractical for the admission control algorithm to attempt to adjust to short term fluctuations in traffic (i.e., on the time scale of bursts). Given that there are no long term correlations in the aggregate traffic, the ideal strategy for admission control is to keep long term average load constant. While this might present a challenge in reality, it is trivial in our simulation environment when we have homogeneous flows with no long term correlations. Hence, for present purposes we define the *Quota* algorithm, which does not depend on measurements. This simple algorithm admits a newly arriving flow if there are less than n flows currently receiving service, and rejects the flow otherwise. The parameter n controls how conservative or aggressive the algorithm is. While this algorithm is helpful in better understanding the limits of the performance of MBACs, it is impractical in any real setting since it requires homogeneous flows.

Figure 4a plots the performance frontiers for the Quota algorithm and for one of the measurement-based algorithms (MS)



(a)



(b)

Fig. 4. (a) Performance frontiers for MS and Quota algorithms with EXP1 traffic (b) Number of admitted flows as a function of time for the MS algorithm

with the EXP1 traffic. As the figure shows, the Quota algorithm outperforms the measurement-based algorithm; across the load levels tested, the loss rate for the measurement-based algorithm is between 50% and 250% higher than that of the Quota algorithm. Figure 4b plots the number of admitted flows as a function of time for one simulation with the Measured Sum algorithm; a similar plot for the Quota algorithm yields an essentially straight line (the offered load is sufficiently high so that a new flow arrives very soon after a flow leaves, making the admitted load very close to constant). The MS algorithm mimics the Quota algorithm fairly well, but there is significant variation in the number of admitted flows. Similar variations in load occur when using the other MBACs we evaluated. Note that for the same average utilization, increased variability in load leads to higher loss rates. Thus, with the EXP1 traffic model, it is precisely these variations in admitted load that leads to the worse performance frontier for the measurement-based admission control algorithm. Is this variation inevitable, or can MBACs eventually match the performance of the Quota algorithm?

There are two distinct causes for this variation leading to the performance degradation relative to the Quota algorithm. The first is the way that the measurement-based algorithms must deal with the arrival and departure of flows. Because the measurement-based algorithms we consider use aggregate rather than per-flow measurements, they do not know how much a departing flow was contributing to the previous estimate of load.[10] Measurement-based algorithms must therefore wait before admitting a new flow until new measurements reflect the departure of the previous flow. During this time, additional flows may depart, and the number of flows in the system may drop. The Quota algorithm on the other hand, with its perfect but unrealistic knowledge of the departing flow, can immediately admit a new flow. Similarly, when a new flow is admitted to the system, measurement-based algorithms must assume worst case behavior about the new flow until new measurements reflect its presence. In contrast, the Quota algorithm can admit flows based on their average behavior and need not delay further admissions.

The second factor leading to variation in the number of admitted flows is that measurement-based admission control algorithms, by their reliance on measurements of current traffic, must necessarily respond to significant fluctuations in the load even when the number of flows has not changed. That is, the MBAC cannot distinguish between having too many flows admitted and a long fluctuation to a higher level of aggregate traffic by a fixed set of flows; not being able to detect the difference, the MBAC is forced to turn away flows during such a fluctuation even when there are too few flows present and similarly, if the current flows fluctuate to a lower level of traffic, the MBAC is forced to admit flows even when too many are already present.

Note that there is an inherent tension between the two factors that cause MBAC performance to degrade relative to the Quota algorithm. To avoid adapting to short term fluctuations in load, longer measurement intervals are suggested [15], [19]. Longer measurement intervals, on the other hand, will only slow down the reaction of the measurement-based algorithms to the departure and arrival of flows. Therefore, it is likely that these two factors will prevent any measurement-based algorithm from ever performing as well as the Quota algorithm.

If MBACs could emulate the Quota algorithm, then they would all have the same performance frontier, and our results in Section III-B would be rendered obvious. However, the discussion above shows that MBACs cannot accurately emulate the Quota algorithm. The surprise in our results in Section III-B is that the set of MBACs we tested all had such similar deviations from the *ideal* behavior of the Quota algorithm. One might have thought (indeed, we did think) that different admission control equations and different measurement procedures would make a difference in how well this ideal was followed; our results suggest that this is not the case.[11]

E. Long Range Dependence

Before turning to the second goal of MBACs (performance targets) we briefly discuss long range dependence and its effect on admission control. Long range dependence has been observed in video traffic [1], [12] and may also arise from the

[10] In addition, some signaling protocols may not even provide explicit *teardown* messages, exacerbating the problem of updating estimates when flows depart the network.

[11] Our fuller set of simulations (not presented here) suggest that the length of the averaging periods, and the way in which new flows are treated, are much more important than the equations themselves in determining how close MBACs come to the performance frontier of the Quota algorithm. This is consistent with the observations above about the two causes of the variations, since they both relate to measurement intervals and the treatment of new flows, and are orthogonal to the specific equations used in the admission decision.

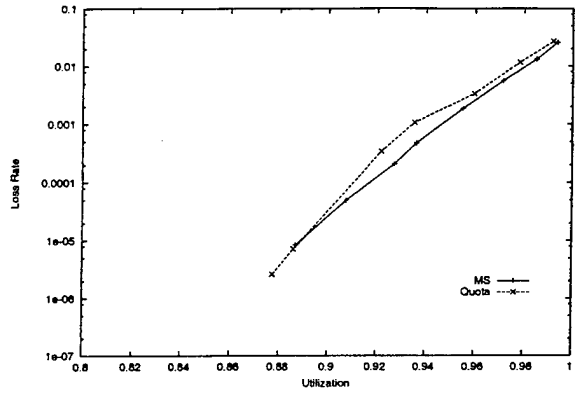


Fig. 5. Measurement-based algorithm vs. Quota Algorithm for long range dependent traffic

aggregation of audio traffic [11], two traffic classes that may be subject to admission control. Results for the POO1 traffic source showed that our basic result, that the various MBACs have similar performance frontiers, remains unchanged in the face of long range dependent traffic. However, the relative performance of the Quota algorithm, held up as an *ideal* algorithm in the previous Section, is quite altered by the presence of long range dependent traffic.

Above we showed that the Quota algorithm, which admits a fixed number of flows, performs better than the measurement-based algorithms with the EXP1 traffic source (which does not give rise to long range dependent aggregate traffic.) We repeated these experiments using the POO1 traffic source. Results are shown in Figure 5, and for clarity we again show only the Measured Sum algorithm and the Quota algorithm. With long range dependent traffic, we see the opposite results. In this case, the measurement-based algorithm performs *better* than the Quota algorithm (which one can think of as a simple, if unrealistic, parameter-based algorithm). The explanation for this is straightforward. The long range dependent traffic exhibits variations over long time scales. By keeping the number of flows fixed, the Quota algorithm does nothing to smooth these variations. The measurement-based algorithm, on the other hand, is able to adjust the number of flows admitted in response to the variations. As the aggregate load increases, departing flows need not be replaced by new ones, and when load decreases additional flows can be admitted.

We believe the implications of this are important. The original arguments for using measurement-based admission control claimed that the worst-case behavior of bursty traffic is far worse than the average case, and that it is hard *a priori* to know the average behavior of a bursty traffic flow. Since the average behavior is unknown, any parameter-based algorithm must be based on worst-case parameters, leading to low network utilization. However, our results here indicate that that argument should be taken one step further. Even if the *average* behavior of traffic flows were known, the existence of long range dependent traffic would still mandate the need for measurement-based admission control in order to adapt to these long time scale fluctuations. Thus, while it has previously been suggested that long range dependence may present certain challenges for measurement-based admission control [11], [23] (and we do not disagree with those arguments), we believe that long range dependence also provides additional motivation for the use of measurement-based admission control. When the time scale of flow arrivals and departures is shorter than that of the ebb and rise of traffic, measurement-based admission control enables the network to react to these traffic fluctuations.

These results on long range dependence also shed light on another issue. Some have argued that our basic result—that the performance frontiers of MBACs are very similar—follows quite directly from the observation that all algorithms seek to mimic the Quota algorithm. In Section III-B we found that there are inherent limitations to how closely any MBAC can mimic the Quota algorithm. Our results about long range dependence further show that mimicking the Quota algorithm is not always the optimal behavior.

IV. Performance Targets

Results in the previous section showed that all the measurement-based algorithms are capable of making the same tradeoff between utilization and loss. However, network operators who will deploy these algorithms may be interested in more than just knowing that the algorithms achieve the same tradeoff. Rather, it may be important for a network operator to know *how* to end up at a particular point on the performance frontier, so that a desired loss rate can be achieved. When comparing algorithms, it is important to ask to what extent their input parameters are useful in predicting actual performance. An algorithm that allows an operator to control resulting performance will be preferred over one that does not.

We note that not all of the designers of the algorithms we study intended their algorithms to be tunable, nor did they all make claims about how well the algorithms were able to meet a particular performance target. Hence, we undertake this evaluation not to judge whether a particular algorithm meets its design objectives. Rather, we begin with the observation that each algorithm has one or more parameters that can be adjusted to control performance. We ask whether these parameters are able to provide functionality that network operators may find useful.

The tuning parameter in the TP and TO algorithms represents the space parameter of the Chernoff Bound used to compute the equivalent bandwidth curve upon which the algorithms are based. As such, this parameter does not represent a meaningful performance target. One may then ask whether this parameter can be mapped into a useful performance value in a deterministic way. For instance, if a particular parameter value in the TO algorithm always yields the same loss rate, then the parameter can be useful in predicting actual performance. However, a review of our simulation results shows that this is not the case. As an example, with the TP algorithm, a parameter value of $4.0e^{-7}$ yields loss rates of .0098, .0018 and less than 10^{-7} with the POO1, EXP1 and Star Wars sources, respectively. These kinds of inconsistencies were also observed with the TO algorithm. Thus, the tuning parameter in the TO and TP algorithms can not be used to predict actual performance.

The MS algorithm has a parameter, v, which represents a cap on the fraction of the link bandwidth that can be used by traffic subject to admission control. As such, its semantics are easily

understood, and we can ask whether it is useful as a utilization target. Simulation results indicate that it is not. For example, with the EXP1 traffic source, when $v = 1.0$, average utilization is 94% of the link bandwidth. With the EXP2 traffic source, utilization is only 75% of the link bandwidth with the same value of v. Further, even if the utilization target was consistently met, we question the value of this parameter as a performance target. We expect loss rate to be a more relevant parameter, since loss rate directly affects user performance.

The HB algorithm uses a parameter, ϵ, to represent the probability that the stationary bandwidth requirement of a set of flows exceeds the computed equivalent bandwidth of the flows. In practice, this does not turn out to be a useful predictor of loss. For example, in the simulations shown previously, we typically use values of ϵ above .9. Further, these values do not map into actual loss in any consistent manner. For example, with $\epsilon = .9$, the loss rates are .00045, .005 and less than 10^{-7} with the EXP1, POO1, and Star Wars source models, respectively.

Algorithm	Source Model	Target Loss Rate	Actual Loss Rate
TE	EXP1	10^{-6}	1.9×10^{-5}
TE	EXP1	10^{-2}	4.8×10^{-2}
TE	Star Wars	10^{-6}	5.5×10^{-4}
TE	Star Wars	10^{-2}	4.4×10^{-3}
TE	EXP2	10^{-6}	3.1×10^{-5}
TE	EXP2	10^{-2}	1.8×10^{-3}
TE	POO1	10^{-6}	1.3×10^{-2}
TE	POO1	10^{-2}	4.1×10^{-2}
MC	EXP1	10^{-6}	1.1×10^{-4}
MC	EXP1	10^{-2}	2.4×10^{-4}
MC	Star Wars	10^{-6}	3.0×10^{-3}
MC	Star Wars	10^{-2}	4.5×10^{-3}
MC	EXP2	10^{-6}	1.7×10^{-4}
MC	EXP2	10^{-2}	2.0×10^{-4}
MC	POO1	10^{-6}	1.2×10^{-2}
MC	POO1	10^{-2}	1.6×10^{-2}

TABLE I
TARGETED VERSUS ACTUAL LOSS RATES FOR THE TE AND MC ALGORITHMS

The final two algorithms, TE and MC, use target loss rate as a tuning parameter. Table I shows both the target and actual loss rates for both algorithms and several traffic sources. These data show that the algorithms are unable to achieve performance close to their targeted performance in a consistent manner. Indeed, for each algorithm the table shows examples in which the actual loss rate is both higher and lower than the target, sometimes by 2 or 3 orders of magnitude. While the TE algorithm comes closer to its targets in general, it still misses by a couple of orders of magnitude in some cases. As such, even though the targets are achieved under certain scenarios, they do not predict performance reliably.

In sum, none of the algorithms provide tuning parameters that are useful as performance targets. At best, these parameters can be seen as largely uncalibrated knobs that can increase or decrease utilization and loss.

V. CONCLUSIONS

In this paper we compared several different measurement-based admission control algorithms. We evaluated the algorithms according to two criteria. First, what tradeoff of loss and load do they each achieve? This criterion shows how well the algorithms are able to balance the conflicting goals of providing good quality of service to individual users and achieving high network utilization (i.e., satisfying many users). Here our results were unambiguous. Across a range of traffic sources, all the algorithms, whether *ad hoc* or principled, achieved nearly identical performance. This result argues that there is no particular performance benefit of one over the others. Our study also yielded several additional insights about measurement-based admission control. First, we showed that for many algorithms, the measurement estimation and admission decision processes can be decoupled. Second, differences in performance caused by flow heterogeneity are a matter to be addressed by policy, rather than by algorithmic differences. Third, simulation results showed that measurement-based admission control algorithms not only cope well with long range dependence in traffic, in some circumstances they are more adept at handling it than are parameter-based algorithms.

The second criterion we used to evaluate the algorithms was the extent to which they provided performance tuning knobs that allow network operators to set a target performance level for the network. Such a knob would allow the network operator to decide where on the performance frontier the network should operate. Here the results were less impressive. None of the algorithms was able to reliably match actual performance to targeted performance levels. Thus, we believe that for any of these algorithms, network operators will need to monitor actual performance in order to learn appropriate parameter settings. On the other hand, some algorithms did better than others in this regard in the sense that they tended to get closer to targets on average than others. While the magnitude of the errors was in all cases large enough to call into question the value of the knobs as performance targets, whether or not this difference is important is a subject of debate. The ability of future algorithms to improve in this regard is an open question.

ACKNOWLEDGEMENTS

We had many helpful conversations with Nick Duffield, Sally Floyd, Richard Gibbens, Matt Grossglauser, Frank Kelly, Ed Knightly, Andrew Moore and Jingyu Qui about the measurement-based admission control algorithms discussed in this paper. Patrick McDaniel provided valuable feedback on an earlier version of this paper. We would like to thank Sandeep Bajaj for his assistance with the implementation of some of the admission control algorithms in the *ns* simulator. Sugih Jamin would like to thank Don Hoffman, Debasis Mitra, Jeff Mogul and Chuck Song for their support of this work.

REFERENCES

[1] BERAN, J., SHERMAN, R., TAQQU, M. S., AND WILLINGER, W. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications 43*, 2 (Feb. 1995), 1566–1579.

[2] BOLOTIN, V. "Modeling Call Holding Time Distributions for CCS Network Design and Performance Analysis". *IEEE Journal on Selected Areas in Communications 12*, 3 (Apr. 1994), 433–438.

[3] BRADEN, R., ED., ZHANG, L., BERSON, S., HERZOG, S., AND JAMIN, S. Resource ReSerVation protocol (RSVP) – version 1 functional specification. Tech. Rep. RFC 2205, Internet Engineering Task Force, Sept. 1997.

[4] BRESLAU, L., JAMIN, S., AND SHENKER, S. Measurement-based admission control: What is the research agenda? In *Proc. of IEEE/IFIP Seventh International Workshop on Quality of Service (IWQOS '97)* (London, England, 1999).

[5] CLARK, D., AND WROCLAWSKI, J. An approach to service allocation in the Internet. Internet draft, Internet Engineering Task Force, July 1997.

[6] CLARK, D. D., SHENKER, S., AND ZHANG, L. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM Sigcomm* (Aug. 1992), pp. 14–26.

[7] CROSBY, S., LESLIE, I., MCGURK, B., LEWIS, J. T., RUSSELL, R., AND TOOMEY, F. Statistical properties of a near-optimal measurement-based cac algorithm. In *Proceedings IEEE ATM '97* (June 1997).

[8] CROVELLA, M., AND BESTAVROS, A. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking 5*, 6 (Dec. 1997), 835–846.

[9] DUFFY, D., MCINTOSH, A., ROSENSTEIN, M., AND WILLINGER, W. "Statistical Analysis of CCSN/SS7 Traffic Data from Working CCS Subnetworks". *IEEE Journal on Selected Areas in Communications 12*, 3 (Apr. 1994), 544–551.

[10] DZIONG, Z., JUDA, M., AND MASON, L. A framework for bandwidth management in ATM networks – aggregate equivalent bandwidth estimation approach. *IEEE/ACM Transactions on Networking 5*, 1 (Feb. 1997), 134–147.

[11] FLOYD, S. Comments on measurement-based admissions control for controlled-load services. Technical report, Lawrence Berkeley Laboratory, July 1996.

[12] GARRETT, M. W., AND WILLINGER, W. Analysis, modeling and generation of self-similar VBR video traffic. *Computer Communications Review 24*, 4 (Oct. 1994). SIGCOMM '94 Symposium.

[13] GIBBENS, R., AND KELLY, F. "Measurement-Based Connection Admission Control". *15th International Teletraffic Congress* (Jun. 1997).

[14] GIBBENS, R. J., KELLY, F. P., AND KEY, P. B. A decision-theoretic approach to call admission control in ATM networks. *IEEE Journal on Selected Areas in Communications SAC-13*, 6 (1995), 1101–1113.

[15] GROSSGLAUSER, M., AND TSE, D. A framework for robust measurement-based admission control. *Computer Communications Review 27*, 4 (Oct. 1997), 237–248. ACM SIGCOMM'97, Sept. 1997.

[16] GROSSGLAUSER, M., AND TSE, D. N. C. A time-scale decomposition approach to measurement-based admission control. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)* (New York, Mar. 1999).

[17] GROSSGLAUSER, M., TSE, D. N. C. C., KUROSE, J., AND TOWSLEY, D. A new algorithm for measurement-based admission control in integrated services packet networks. In *Proceedings of the Fifth International Workshop on Protocols for High-Speed Networks* (Antipolis, France, Oct. 1996).

[18] JAMIN, S., DANZIG, P., SHENKER, S., AND ZHANG, L. A measurement-based admission control algorithm for integrated services packet networks. In *Proceedings of ACM Sigcomm* (Sept. 1995).

[19] JAMIN, S., DANZIG, P. B., SHENKER, S. J., AND ZHANG, L. A measurement-based admission control algorithm for integrated services packet networks. *IEEE/ACM Transactions on Networking 5*, 1 (Feb. 1997), 56–70.

[20] JAMIN, S., SHENKER, S., AND DANZIG, P. "Comparison of Measurement-based Admission Control Algorithms for Controlled-Load Service". *Proceedings of the Conference on Computer Communications (IEEE Infocom)'97* (Apr. 1997).

[21] KNIGHTLY, E. W., AND QIU, J. Measurement-based admission control with aggregate traffic envelopes. In *IEEE ITWDC '98* (Ischa, Italy, September 1998).

[22] NICHOLS, K., JACOBSON, V., AND ZHANG, L. A two-bit differentiated services architecture for the Internet. Internet Draft, Internet Engineering Task Force, May 1999. Work in progress.

[23] PAXSON, V., AND FLOYD, S. Wide area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking 3*, 3 (June 1995).

[24] SHENKER, S., PARTRIDGE, C., AND GUERIN, R. Specification of guaranteed quality of service. RFC 2212, Internet Engineering Task Force, Sept. 1997.

[25] WILLINGER, W., TAQQU, M., SHERMAN, R., AND WILSON, D. "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level". *Proceedings of ACM Sigcomm'95* (Aug. 1995), 100–113.

[26] WROCLAWSKI, J. Specification of the controlled-load network element service. RFC 2211, Internet Engineering Task Force, Sept. 1997.

Network Visualization with the VINT Network Animator Nam *

Deborah Estrin, Mark Handley, John Heidemann,
Steven McCanne, Ya Xu, Haobo Yu

USC Computer Science Department Technical Report 99-703b

March 1999 (revised November 1999)†

Abstract

Protocol design requires understanding state distributed across many nodes, complex message exchanges, and with competing traffic. Traditional analysis tools (such as packet traces) too often hide protocol dynamics in a mass of extraneous detail.

This paper presents *nam* , a network animator that provides packet-level animation and protocol-specific graphs to aid the design and debugging of new network protocols. Taking data from network simulators (such as ns) or live networks, nam was one of the first tools to provide general purpose, packet-level, network animation. Nam now integrates traditional time-event plots of protocol actions and scenario editing capabilities. We describe how nam visualizes protocol and network dynamics.

Keywords: network protocol visualization, packet-level animation, Internet protocol design, network simulation, ns, nam

1 Introduction

Designers of network protocols face many difficult tasks, including simultaneous monitoring of the state of a potentially large number of nodes (for example, in multipoint protocols), understanding and analyzing complex message exchange, and characterizing dynamic interactions with competing traffic.

Traditionally, packet traces have been used to accomplish these tasks. However, packet traces have two major drawbacks: they present an incredible amount of detail, which challenges the designer's ability to comprehend the data, and they are static, which hides an important dimension of protocol behavior. As a result, detailed analysis frequently becomes tedious and error-prone. Although network simulators such as ns [2] can easily generate numerous detailed traces, they provide limited help in analyzing and understanding the data.

Network-specific visualization tools address this problem, allowing the user to take in large amounts of information quickly, to visually identifying patterns in communication, and to better understand causality and interaction. This paper presents *nam* , a network animator that provides packet-level animation and protocol-specific graphs to aid the design and debugging of new network protocols (Figure 1). Nam was one of the first tools to provide general purpose, packet-level, network animation. Recent work has integrated traditional time-event plots of protocol actions and added scenario editing capabilities. Nam benefits from a close relationship with ns, the VINT project's network ns [2] which can collect detailed protocol information from a simulation. With some pre-processing, nam can also be used to visualize data taken directly from real network traces.

Related Work (sidebar)

Network protocol visualization has been explored in many contexts, beginning with static protocol graphs, and visualization of large-scale traffic, more recently including simulation visualizations and editors.

Graphs of packet exchanges are very useful at understanding cause-and-effect in complex protocols like TCP. Work at MIT [10] and the University of Arizona [3] is typical: graphs show time against TCP se-

*This research is supported by the Defense Advanced Research Projects Agency (DARPA) through the VINT project at LBL under DARPA Order E243, at USC/ISI under DARPA grant ABT63-96-C-0054, at Xerox PARC under DARPA grant DABT63-96-C-0105.

†Originally published in March, 1999, this technical report was updated in November, 1999 (one section was moved, some text was added and rewritten, and a number of typos were fixed). This technical report has been accepted to appear in IEEE Computer Magazine.

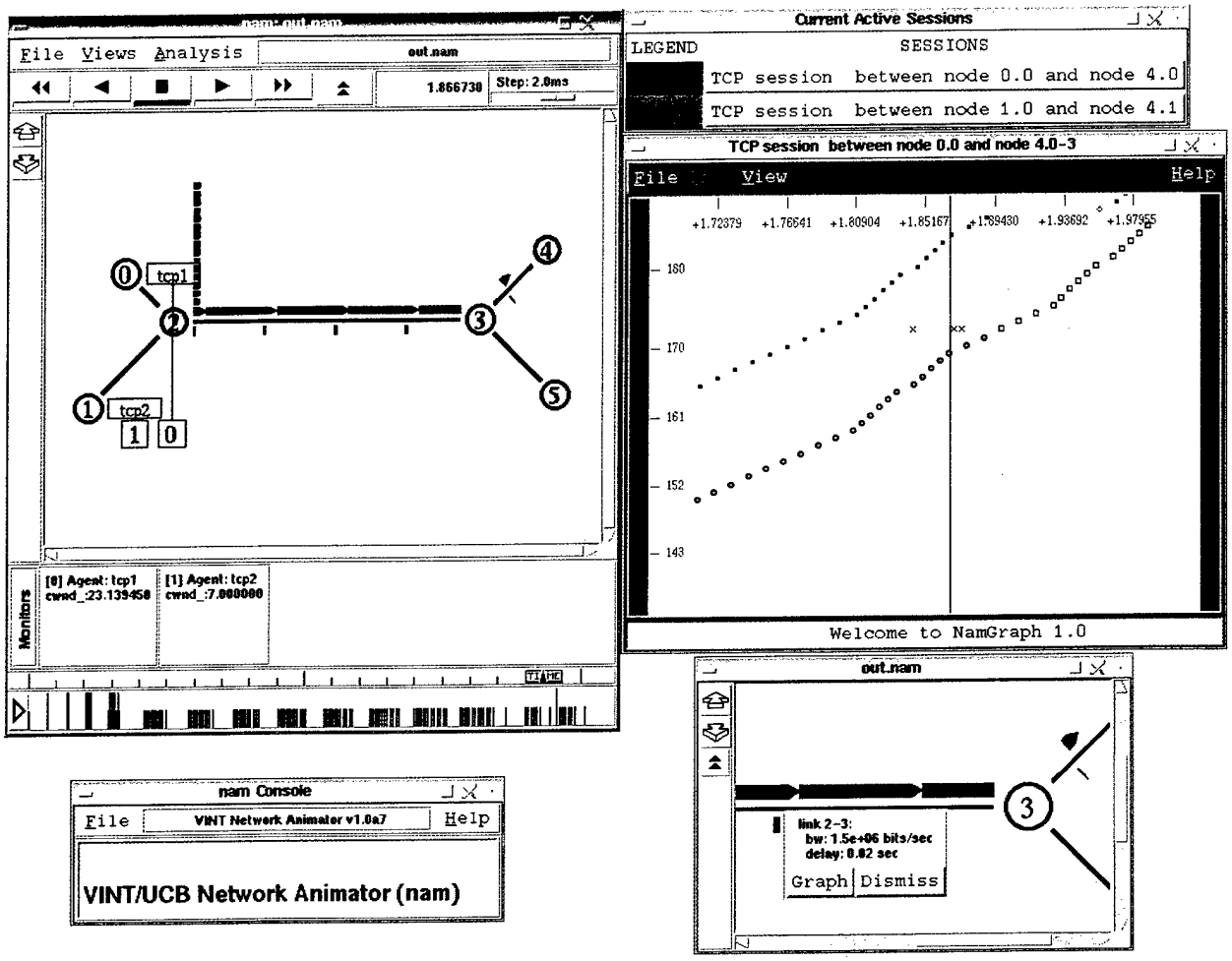


Figure 1: Basic nam operation.

quence numbers on a 2-D graph, possibly with annotations to show special events. Similar time-event graphs have proven useful in understanding reliable multicast behavior in SRM [5]. Although nam graphs are not as detailed as the most sophisticated of these graphs, they are integrated with the packet animation and time control. We plan to develop APIs to allow the end-user to annotate graphs with the details relevant to their protocol or protocol modifications.

Several groups have looked at visualization of large, static network data sets. Important questions include use of layouts based on real-world geography or network topology, how best to use animation, color, and 3-D. More generally, many researchers tackled the problem of visualization of complex data (for an overview of several approaches, see Robertson *et al.* [9]). Systems like these share the principle that multiple linked views are essential in visualizing complex data. Nam adopts this principle. It organizes visualization around the main topology view, from which a number of specialized views may be derived. These systems tend to focus on representing aggregate network data (traffic flows) to understand and monitor traffic patterns, rather than the packet-level detail necessary to design new protocols.

Several Network simulation systems include explicit support for visualization, either customized to a particular end-application or more general. Opnet includes visualization capabilities and Simphony [7] explicitly includes packet-level animation. Nam differs from this work by supporting different views of the data (packet animation and time-event graphs).

Nam is quite late in providing a GUI front-end to defining new simulations. Systems such as Opnet and Parsec [1] have provided this capability for some time. CMU's ad-hockey was designed explicitly to support node movement [11]. We believe GUI network editors are of most benefit to novice users or users running small simulations, we advocate using a scripting language to construct large or complex simulations. Nam's editing capabilities are therefore not as complete as other similar systems since nam outputs a script which can be extended by hand to access complete ns functionality.

2 Nam Basics

Nam interprets a trace file containing time-indexed network events to animate network traffic in several different ways (Figure 2). Typically this trace is generated from an ns simulation, but it can also be generated by processing data taken from a live network to

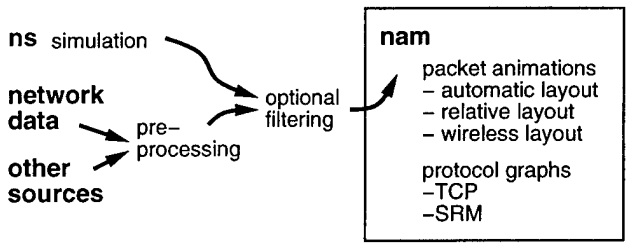


Figure 2: Block diagram of nam.

produce a nam trace. Nam usually runs off-line with the traces stored on disk, but it can also play traces from a running program through a Unix pipe.

A nam input file contains all information needed for the animation: both the static network layout and dynamic events such as packet arrivals, departures, and drops and link failures. Wireless networking simulations include node location and movement.

Figure 1 shows a typical nam session. On the top left, the main window shows packet animations. The visual size and speed of packets is proportional to packet length and the link bandwidth and delay; link 2–3 is full of TCP data moving along the top and return acknowledgement traffic along the bottom in the reverse direction. Packet color is used for different things; in this case it differentiates two different data streams (black and blue) and a red packet carrying a congestion signal. Packets move from node to node along links, and are queued up when links are full (for example, there is a large queue near node 2 corresponding to the busy link between nodes 2 and 3). Below it (in the same window) are several statistical summaries of what is happening. Boxes labeled "monitors" correspond to parameters of protocols running on particular nodes. The graph across the bottom of the window shows the utilization of a link as a function of time. The smaller bottom-right window is zoomed in on part of the same network. The window on the center-right shows a protocol-specific time-event graph of a particular flow on a given link. In this case, it plots TCP sequence numbers against time using different symbols to show data packets, acknowledgements, and acknowledgements which include explicit congestion information.

Multiple copies of nam may be executed simultaneously, in which case they may be driven in lock-step. With this synchronized, simultaneous ability to visualize the output of more than one simulation trace file, side-by-side comparisons are made possible. Such comparisons are especially useful for investigating protocol

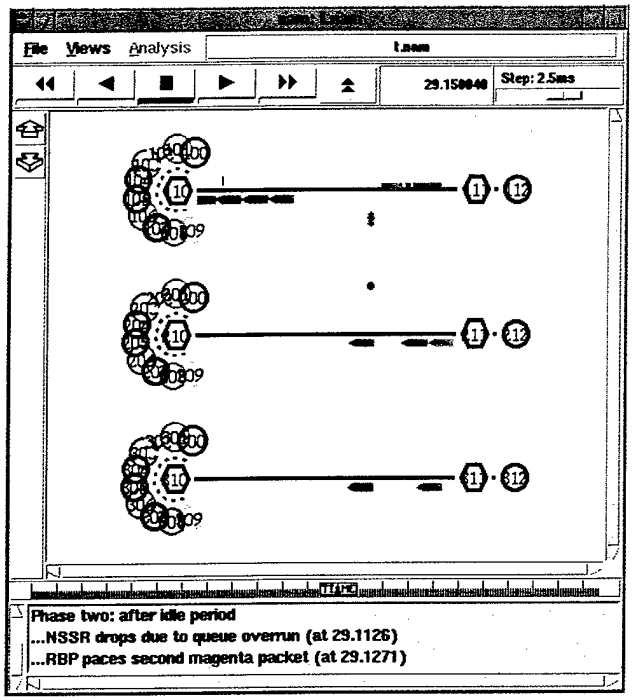


Figure 3: Packet animation in nam.

sensitivity to input parameters in the same simulation scenario (as in [5], for example).

3 Packet Animation

The core of nam is packet animation. Figure 3 shows a typical packet animation (taken from [13]). Three variants of the TCP protocol are being used to send data from web servers on the right to clients on the left. Animation here allows the viewer to quickly take in the status of each part of the network (the top link is severely congested and dropping packets, the middle link is slightly busier than bottom link), and to quickly compare the algorithms (the middle variation has one extra magenta packet while the top version sends many back-to-back packets). Nam allows the animation speed to be adjusted and played forwards or backwards, making it easy to find and examine interesting occurrences.

The first step in a new animation is displaying the network topology. Nam has three different topology layout mechanisms to accommodate different needs. The default is an *automatic layout* algorithm based on a spring-embedder model [6]; Figure 4 shows an example of this result. It assigns attractive forces on all links and repulsive forces between all nodes, and tries to achieve balance through iteration. Automatic layout can produce reasonable layouts of many networks without explicit user guidance, but it may not produce satisfactory results of complicated networks. As a remedy, nam allows the user to graphically adjust the resulting layout.

For smaller topologies, *relative layout* is possible. The user specifies the relative directions of links (left, up, down). Nam places nodes relative to each other using link directions; link length is set proportional to its bandwidth and delay. Relative layout works very well for small topologies and has the desirable property that packet movement rate is consistent with link delay and bandwidth. The network in Figure 3 uses relative layout. Disadvantages of relative layout are that the user must specify the directions of each link, that not all networks have a planer representation that satisfies delay constraints, and relative layout of a topology containing very different delays can result in very short links. For example, the 10Mb/s, 1ms delay links on the left of Figure 3 are too short to observe packet flow when shown at the same scale as the 800Kb/s, 100ms central link.

Finally, *wireless layout* assigns associates each node with a physical location in a constrained area. Each node's position is given by its 3-D coordinate (only the two dimensions are currently used for visualization) in the area and its velocity vector. Wireless visualizations typically lack explicit links.

Packet animation is straightforward once the topology is laid out. Trace events indicate when packets enter and leave links and queues. Packets are shown as rectangles with arrows at the front; queues as arrays of squares (see the left window of Figure 1). Packets can be colored based on codes set in the simulator or pre-processing to identify source and destination pairs. When queues fill, packets are literally dropped, shown as small rolling squares falling to the bottom of the display.

The only difficulty we encountered in implementing packet animation is that some events are not present in the trace file but must be generated on-the-fly. Our design philosophy was to make the trace file as explicit as possible, but some trace events are animation specific and so must be dynamically constructed. One example is identifying when a dropped packet leaves the screen. This event is not known by the simulator.

Users can control animation playback rate to focus on interesting parts of the simulation. VCR-like buttons control forwards or backwards playback, while a slider sets playback rate. Because some simulations include dead time, periods of no packet activity can

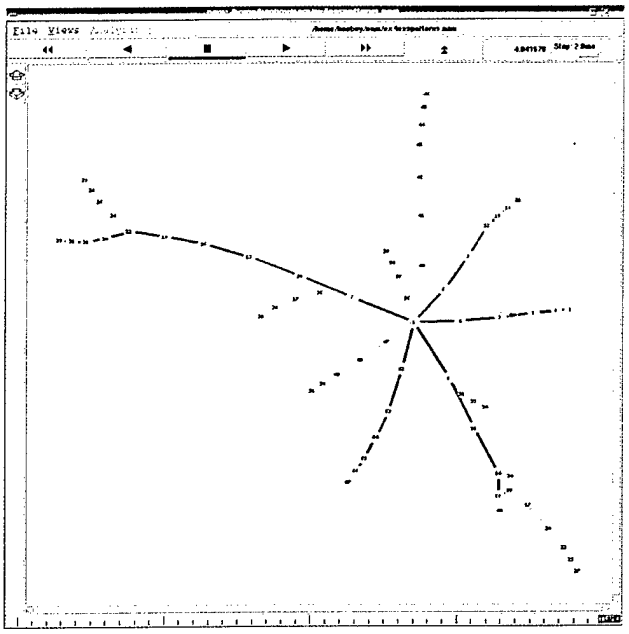


Figure 4: Link animation in a visualization of mbone loss rates.

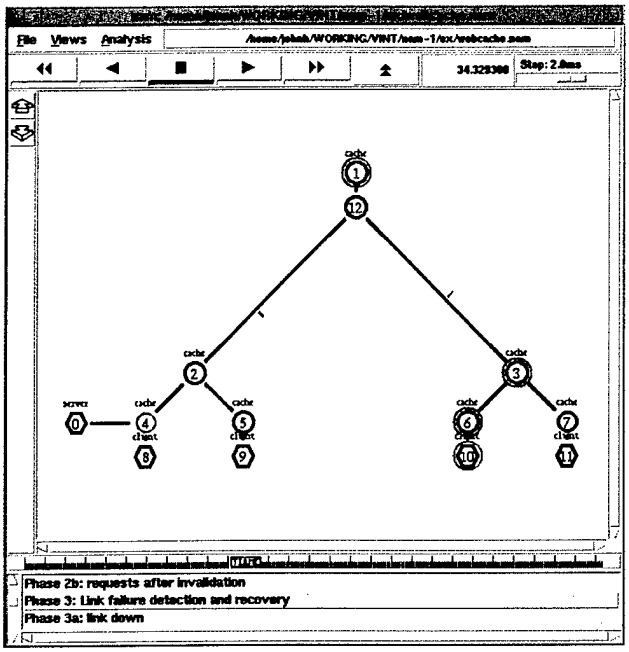


Figure 5: Visualization of applications with nam.

optionally be skipped. Interesting events in the trace can be annotated, allowing a user to jump to those events.

The animation window is interactive. Clicking on packets, links, and nodes brings up pertinent information, including statistics (described next).

In addition to packet animation, we have experimented with ways to visualize other information. Node color and shape can be specified, for example, to indicate membership in a multicast group. Protocol agents represent state of a protocol instance at an end-node. Agents can be displayed as small labeled rectangles attached to nodes.

Figure 4 shows one example of non-packet-level animation. This figure shows the topology of a portion the Internet multicast backbone (mbone) as of 1998. To determine if mbone loss was primarily in the core network or the edges we measured loss rates for various links. In the figure, different loss rates are shown with color which changes over time.

We have also found nam useful for application-level visualization. In Figure 5 we use nam to visualize cache coherence algorithms in a hierarchical web cache. Node types are shown with shapes (the clients and server are hexagons while caches are circles), Cache status (valid or out of date) is shown with node color. Algorithm status (refreshing a cache, etc.) is shown with rings around nodes.

4 Network Statistics

The animation component of nam only displays a subset of the simulation details present in the trace output. Additional information, such as packet headers or protocol state variables, are handled by other nam components. The *statistics* component provides three ways to display this additional information. First, clicking on any of the displayed objects (e.g. packets and protocol agents) will bring out a one-shot panel showing object-specific information. Second, continuous monitoring of all available object-specific information may be achieved by associating a *monitor* with entities of interest. Monitors remain associated with an object until explicitly removed by the user or until its underlying object is destroyed. These monitors are displayed in a pane in nam's main window, as illustrated in Figure 1. Third, nam uses panes (the black stripes in Figure 1) in the main window to display bandwidth utilization and packet losses on links. Clicking on a link brings out a selection panel, which allows the user to open a new pane to display bandwidth utilization or packet loss on the link.

5 Protocol-specific Graphs

In addition to detailed examination of individual simulation entities, nam supports protocol-specific representations of information with time-event graphs (where time is plotted against events such as an advancing sequence number or message transmission). These graphs have long been used to understand TCP behavior, and more recently to understand timer interaction in scalable reliable multicast [5].

Currently nam supports protocol graphs for TCP and SRM. We plan to make this facility more generic through a pluggable API for supporting other protocols. Figure 6 shows SRM (center right) and TCP (bottom center and bottom right) time-event graphs. When a graph is first brought up a nam filter scans the trace file to extract the relevant information for a specific flow or protocol.

The advantage of integrating these views with nam is that graphs and packet animation are synchronized. Moving a time slider or by clicking on an interesting event in any view updates the time in all views. Each trace event is displayed in the consistent way (i.e., color, shape, etc.) across views to help the user coordinate events.

6 Scenario Creation and Editing

We use nam in two very complementary ways to assist in scenario creation. First, we have recently extended nam to include a scenario input facility. Using a traditional drawing approach the user can add nodes, links, protocol agents. Nam then saves this scenario as an ns simulation script (in Tcl) which will be processed by the simulator.

Second, the ns scenario generator uses nam to visualize large scenario topologies. The scenario generator constructs these scenarios using tools such as Georgia Tech's ITM [4]. Nam with autolayout then presents the topology to the user for acceptance or regeneration.

Graphical scenario creation with nam is very appropriate for small scenarios with a few nodes and links. We have been happy with the design choice of using nam to produce scripts for these cases while starting with scripts directly for larger, more complex, or automated simulations. For the ns target audience of protocol designers, the effort required to learn Tcl syntax is small and this is more than offset in these scenarios by the finer control afforded and the ability to use looping constructs in place of repeated manual point-and-click operations.

7 Future Work and Conclusions

Nam development is on-going. A number of incremental improvements are desired or planned. For example, we would like to improve scenario editing capabilities, and add support for entering mobile node tracks [11]. We would also like to experiment with adding audio capabilities to the simulator. Two major focuses of future work remain. First, we would like to make nam much easier to extend, providing better internal APIs to allow users to add custom controls to the output and to control object rendering. An example application would allow users to interactively control node colors to indicate application-specific groups or characteristics. Second, we are just beginning to understand how to visualize large scale protocol actions. More work in this area is needed.

Network protocol visualization is easy to dismiss since its contributions to protocol development are indirect. Broader use of nam suggests that visualization is more than just a tool for fancy demos, but that it can substantially ease protocol debugging and help understand dynamic behavior. Because of these reasons, a growing number of researchers have used nam in their work and papers [12, 8].

Acknowledgments

Steve McCanne wrote the original version of nam in 1990 at Lawrence Berkeley National Laboratory. Marylou Orayani made substantial contributions to nam as part of her work at Berkeley in 1995 and 1996. Since 1997 nam has been maintained and enhanced by the VINT research project at USC/ISI, LBL, and Xerox PARC. Nam has also benefited from an enthusiastic VINT and ns user community. We would like to thank especially Elan Amir, Lee Breslau, Kevin Fall, Sally Floyd, Ahmed Helmy, Polly Huang, Scott Shenker, and Christos Papadopoulos. for their input to nam and this paper.

References

[1] BAGRODIA, R., MEYER, R., TAKAI, M., CHEN, Y., ZENG, X., MARTIN, J., AND SONG, H. Y. PARSEC: A parallel simulation environment for complex systems. *IEEE Computer 31*, 10 (Oct. 1998), 77–85.

[2] BAJAJ, S., BRESLAU, L., ESTRIN, D., FALL, K., FLOYD, S., HALDAR, P., HANDLEY, M.,

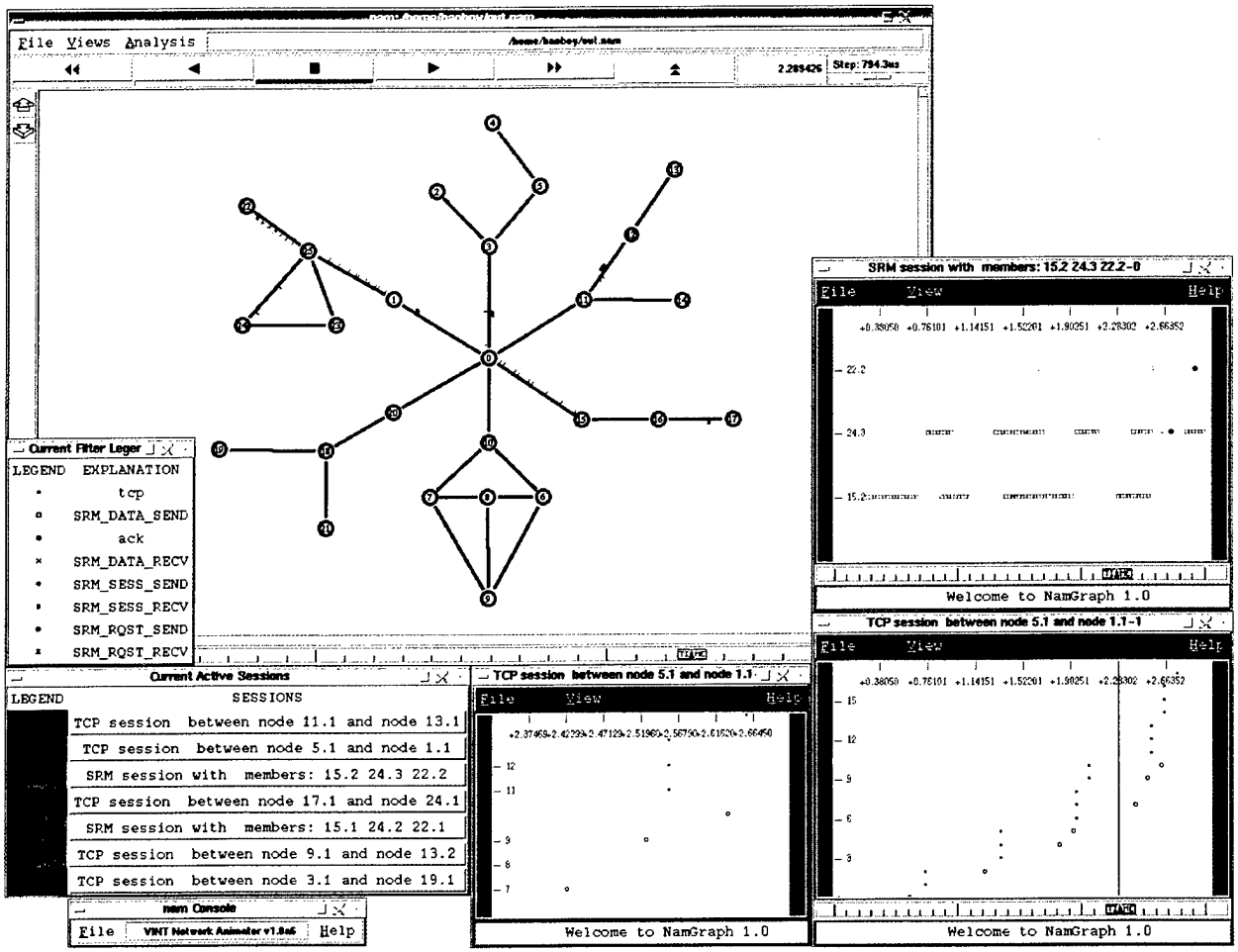


Figure 6: Nam provides protocol-specific graphs which aide in specific investigations. A TCP time/sequence-number graph is shown in the bottom center and bottom right windows. The center right window shows a plot of SRM events against time.

HELMY, A., HEIDEMANN, J., HUANG, P., KUMAR, S., MCCANNE, S., REJAIE, R., SHARMA, P., VARADHAN, K., XU, Y., YU, H., AND ZAPPALA, D. Improving simulation for network research. *IEEE Computer* (2000). to appear, a preliminary draft is currently available as USC technical report 99-702.

[3] BRAKMO, L. S., O'MALLEY, S. W., AND PETERSON, L. L. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of the ACM SIGCOMM '93* (San Francisco, CA, Sept. 1993), ACM.

[4] CALVERT, K., DOAR, M., AND ZEGURA, E. W. Modeling Internet topology. *IEEE Communications Magazine 35*, 6 (June 1997), 160–163.

[5] FLOYD, S., JACOBSON, V., LIU, C.-G., MCCANNE, S., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. *ACM/IEEE Transactions on Networking 5*, 6 (Dec. 1997).

[6] FRUCHTERMAN, T., AND REINGOLD, E. Graph drawing by force-directed placement. *Software - Practice and Experience 21*, 11 (Nov. 1991), 1129–1164.

[7] HUANG, X. W., SHARMA, R., AND KESHAV, S. The Simphony protocol development environment. submitted for publication to Infocom '99, July 1998.

[8] KERMODE, R. Scoped hybrid automatic repeat request with forward error correction (SHARQFEC). In *Proceedings of the ACM SIGCOMM* (1998), pp. 278–289.

[9] ROBERTSON, G. G., CARD, S. K., AND MACKINLAY, J. D. Information visualization using 3D interactive animation. *Communications of the ACM 36*, 4 (Apr. 1993), 56–71.

[10] SHEPARD, T. J. TCP packet trace analysis. Tech. Rep. 494, Massachusetts Institute of Technology, Feb. 1991.

[11] THE CMU MONARCH PROJECT. *The CMU Monarch Project's ad-hockey visualization Tool for ns scenario and trace files.* Carnegie-Mellon University, Aug. 1998.

[12] VARADHAN, K., ESTRIN, S., AND FLOYD, S. Impact of network dynamics on end-to-end protocols: Case studies in reliable multicast. In *Proceedings of the International Symposium on Computers and Communications* (Aug. 1998). `http://www.isi.edu/~kannan/papers/iscc98.ps.gz`.

[13] VISWESWARAIAH, V., AND HEIDEMANN, J. Improving restart of idle TCP connections. Tech. Rep. 97-661, University of Southern California, Nov. 1997.

Asymptotic Behavior of Global Recovery in SRM

Suchitra Raman, Steven McCanne
University of California, Berkeley
{suchi,mccanne}@cs.berkeley.edu
and
Scott Shenker
Xerox Palo Alto Research Center
shenker@parc.xerox.com

Abstract

The development and deployment of a large-scale, wide-area multicast infrastructure in the Internet has enabled a new family of multi-party, collaborative applications. Several of these applications, such as multimedia slide shows, shared whiteboards, and large-scale multi-player games, require *reliable* multicast transport, yet the underlying multicast infrastructure provides only a best-effort delivery service. A difficult challenge in the design of efficient protocols that provide reliable service on top of the best-effort multicast service is to maintain acceptable performance as the protocol *scales* to very large session sizes distributed across the wide area. The Scalable, Reliable Multicast (SRM) protocol [6] is a receiver-driven scheme based on negative acknowledgments (NACKs) reliable multicast protocol that uses randomized timers to limit the amount of protocol overhead in the face of large multicast groups, but the behavior of SRM at extremely large scales is not well-understood.

In this paper, we use analysis and simulation to investigate the scaling behavior of global loss recovery in SRM. We study the protocol's control-traffic overhead as a function of group size for various topologies and protocol parameters, on a set of simple, representative topologies — the cone (a variant of a clique), the linear chain, and the binary tree. We find that this overhead, as a function of group size, depends strongly on the topology: for the cone, it is always linear; for the chain, it is between constant and logarithmic; and for the tree, it is between constant and linear.

1 Introduction

The advent and deployment of IP Multicast [5] has enabled a number of new applications [17, 10, 9, 7, 22] that utilize large-scale multipoint communication over wide-area internetworks. IP Multicast extends the traditional, best-effort unicast delivery model of the Internet architecture to enable efficient multipoint packet delivery. In this model, the network delivers a packet from a source to an arbitrary number of receivers by forwarding a copy of that packet along each link of a distribution tree rooted at the source subnet (or, depending on the routing protocol, at a rendezvous point [4] or core router [1]). As with unicast, IP multicast is not *reliable* — packets might be dropped at any point along the distribution tree. However, many new multicast applications like shared whiteboards, webcast tools, and distributed simulation are not tolerant of packet losses. Whiteboard state, for example, is persistent; if a piece of a drawing update is lost, the application cannot leave the drawing in an incomplete state. Instead, that application must recover the missing packet to repair the damaged portion of the drawing. In short, this particular application, and in fact a large class of emerging applications, require a *reliable multicast* transport protocol. Although mechanisms for reliable unicast transmission are comparatively well-understood and have proven extremely successful (*e.g.*, TCP), making multicast reliable at large scales remains a formidable challenge.

A fundamental problem in the design of a reliable multicast protocol is the well-known *message implosion* [6, 19] problem. Reliable transport protocols rely on some form of feedback between or among communicating end-points to confirm the successful delivery of data. While some protocols rely on positive acknowledgments or ACKs (signalling the successful receipt of data), others rely on negative acknowledgments or NACKs (signalling the failure to receive expected or desired data). Positive acknowledgment-based schemes are successful for reliable unicast transport but scale poorly in the multicast case when there are many receivers. In this case, each delivered packet causes a flood of positive acknowledgments sent from the receivers back to the source, overwhelming either the source or the intervening routers, if not both.

A number of solutions to the ACK implosion problem have been proposed. Log-based reliable multicast [8] uses logging servers to constrain recovery traffic to localized groups of receivers. TMTP [24] and Lorax [12] construct a hierarchy in the form of a tree, in which multiple identical ACKs are fused together before they are propagated up the tree toward the root. RMTP [13] uses a similar approach based on trees that are (statically or dynamically) configured into the network rather than constructed by the application. XTP [2] takes a markedly different approach, however, and instead *multicasts* control traffic to all end-points. To limit the proliferation of this control traffic, XTP employs a "slotting and damping" algorithm: a receiver waits for a random amount of time before generating control traffic and cancels that message if some other hosts multicasts the same information first. The algorithms in SRM [6] elaborate this simple yet powerful primitive with adaptive timers that improve performance across wide-area, heterogeneous networks.

Proc. ACM SIGMETRICS '98/PERFORMANCE '98 Joint International Conference on Measurement & Modeling of Computer Systems, Madison, WI, USA, June 1998.

While TMTP, Lorax, and RMTP limit recovery traffic using unicast transmission over an artifically constructed hierarchy, XTP and SRM limit recovery traffic using multicast transmission and explicit suppression. Although this latter approach is potentially more robust because it does not require an elaborate protocol for tree construction, maintenance, and reconfiguration, it also entails potentially more overhead because recovery traffic is multicast to the entire group and not just to those members impacted by the packet loss. To address this problem, [6] proposes that their SRM reliable multicast framework be cast as two complementary pieces: a *global recovery* component that ensures the delivery of all desired data across the entire multicast session, and a *local recovery* component that constrains the reach of recovery traffic to the multicast neighborhoods where packet loss occurs. Although [6] focuses primarily on global recovery, the SRM authors argue that local recovery is an important and necessary optimization to scale their protocol to large, heterogeneous sessions. Since then, several promising approaches to local recovery have been proposed [11, 14] and the problem remains a focal point of ongoing research.

Even though a viable local recovery strategy is criticial to SRM's scalability, in certain configurations (e.g., where packet loss occurs near the root of the distribution tree), the degree to which local recovery enhances performance may be limited and the protocol's overall performance may strongly depend on that of the global recovery scheme. Hence, we claim that a thorough understanding of global recovery in SRM is not only important in and of itself, but will also be useful in predicting the performance of SRM even when coupled with local recovery.

In this paper, we use analysis and simulation to investigate the scaling behavior of global loss recovery in SRM. We study the growth control traffic (measured by NACK counts) as a function of group size for various topologies and protocol parameters, on a set of simple, representative topologies — the cone, the linear chain, and the binary tree. We find that the number of NACKs, as a function of group size, for the cone is always linear, for the linear chain is between constant and logarithmic, and for the tree is between constant and linear.

The rest of this paper is organized as follows. We start with a brief overview of the SRM protocol in Section 2. Section 3 summarizes related work. In Section 4, we describe our simulation methodology. We discuss the effects of varying the protocol parameters for the various topologies in Sections 5, 6, and 7, and conclude in section 8.

2 Overview of SRM

SRM is a NACK-based, fully-decentralized reliable multicast protocol originally described by Floyd, *et al.*, in [6]. The SRM framework builds on Clark and Tennenhouses's principle of Application Level Framing (ALF) [3], which provides an elegant solution to the problem of reliable-multicast API design because its flexibility offers applications the opportuntity to actively participate in the loss-recovery procedure.

To avoid ACK-implosion, SRM uses NACKs. Receivers detect losses from discontinuities in sequence numbers (or by other means with a generic data naming scheme [20]) and transmit NACKs as a request for retransmission of the lost data[1]. A randomized algorithm determines when a receiver transmits a NACK. These NACKs are multicast to the entire group so that any receiver, in particular the closest receiver with the requested data, may generate a repair in response to a NACK. The repair messages are also multicast to the entire group, so that all receivers that missed that packet can be repaired by a single response. The repair message traffic likewise makes use of the randomized timer algorithm.

To avoid NACK implosion, receivers that observe a NACK for data that they too have not received do not send their own NACK[2] and await the repair data. The goal of the randomized NACK transmission algorithm is to minimize the number of duplicate NACK messages sent. To accomplish this, each receiver delays the transmission of a NACK by an amount of time given by the expression

$$backoff \quad = \quad D \cdot (C_1 + C_2 r)$$

where *backoff* is the amount of delay, D is an estimate of the one-way delay from the receiver to the source that generated the lost data packet, C_1, C_2 are non-negative protocol constants, and r is a uniformly distributed random number in $[0, 1]$. This random delay provides receivers with the opportunity to *suppress* the transmission of similar pending NACKs; that is, delaying the transmission of NACKs by a random amount increases the likelihood that a NACK from one receiver is delivered to another receiver before that receiver sends its own NACK, and thus, reduces the total number of NACKs. Figure 1 illustrates the suppression mechanism in SRM.

As in [6], we call $C_1 D$ the *deterministic* delay and $C_2 Dr$ the *random* delay. The deterministic-delay component induces suppression effects across receivers situated at varying distances from the point of loss (e.g., a chain topology), while the random-delay component induces suppression effects across receivers situated at equal distances from the point of loss (e.g., a star topology). We say that a receiver's timer *fires* if no suppressing NACK has been received when its backoff period has expired.

Since NACKs are multicast to the group, any receiver that has the data can respond, not just the original source. However, we again have the potential for a control-traffic storm if all hosts respond simultaneously. Thus, to avoid repair-packet storms, SRM reuses its NACK suppression machinery to limit the number of redundant repair packets. Because both NACKs and repairs are sent to the entire multicast group, we call this the SRM *global recovery* mechanism.

A number of performance metrics have been used to characterize recovery schemes for reliable multicast, but two widely used metrics are:[3] (1) the degree of duplicate control traffic, and (2) the recovery latency. The first metric can be summarized as the average number of NACKs sent for each dropped packet, which clearly depends on the size of the group experiencing the loss. We denote this number by $N(G)$, where G is the number of members experiencing the loss. The larger this metric, the less effective the randomized timer algorithm is at suppressing duplicate NACKs and

[1] To be true to the original intentions of the SRM designers, we must admit that our use of the term "NACK" is somewhat inaccurate since it implies that the underlying protocol generates NACKs to guarantee that all data is eventually received by all receivers. In fact, SRM is receiver-reliable and does not require that all receivers obtain all data. Instead, receivers issue "repair requests" to repair only those data wanted. For this paper, we use the terms "NACK" and "repair request" interchangeably.

[2] More precisely, they scale their transmission timer awaiting a response. All receivers, if they have not received the repair data, will eventually transmit a NACK.

[3] The metrics we describe here ignore topological heterogeneity, where not all receivers are identical. More detailed performance metrics would measure the latencies on a per receiver basis.

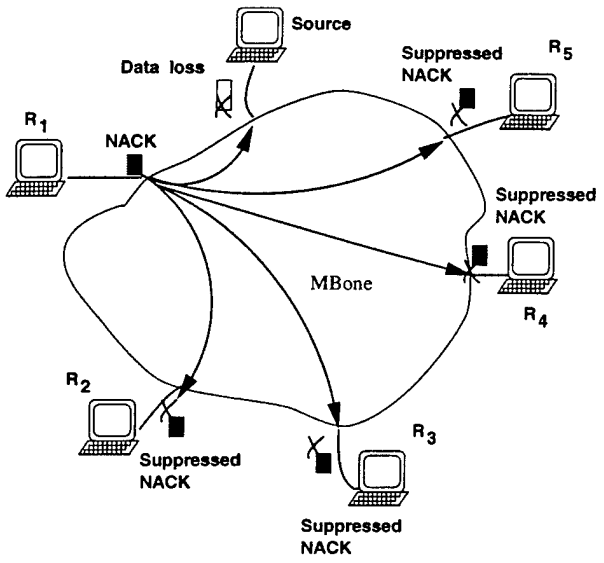


Figure 1: Suppression in SRM

avoiding NACK implosion. $N(G)$ is a non-decreasing function of G, so the suppression performance for large group sizes is a critical factor in SRM's performance.

We define the second metric, the loss-recovery latency, as the time delay between the instant a packet drop is detected to the time at which the first NACK is sent (from the perspective of a particular session member). Recovery latencies for these randomized algorithms typically decrease as group sizes increase, so the sensitivity of latency on group size is not of primary importance in the scaling behavior of SRM.

In this paper, we focus on the performance of SRM with *large* group sizes; that is, roughly speaking, the asymptotic scaling limit. Thus, we focus on the number of duplicate messages and do not address latency performance. Since the timer mechanisms for NACKs and repair messages are similar, we restrict our attention to NACKs. Therefore, our paper addresses the following question: how does the number of duplicate NACK messages increase as the group size grows? In short, what is the scaling behavior of $N(G)$ in SRM?

The scaling behavior of SRM depends both on the topology of the underlying network as well as the details of the timer algorithm. To explore the relationship between topology and scaling behavior, we experimented with three simple network topologies: the cone (a variant of a clique), line, and tree, shown in Figures 2 and 3 While these topologies are instructive because they explore the behavior of SRM under extreme toplogies, they are by no means exhaustive.

The scaling behavior also depends on several aspects of the timer algorithms. We focus on two such factors. First, we look at the dependence of the scaling behavior on the constants C_1 and C_2. There are several applications, such as large-scale multi-player games that are highly interactive, for which *low-latency* loss recovery is important, and the choices of C_1 and C_2 critically impact this. In general, the expected latency to transmit the first NACK upon detecting a loss is bounded above by $(C_1 + C_2/f)D$, where f is a function of the network topology and is always at least 2. Thus, there is a trade-off between recovery latency and the choices of C_1 and C_2. In particular, smaller values of these constants lead to better latency, but also to increased $N(G)$. The need for low latency by many applications motivates our work on investigating the (C_1, C_2) parameter space, and in particular, our consideration of $0 \leq C_1 \leq 1$ (little or no deterministic suppression).

We also briefly consider the case where C_1 and C_2 are a function of the location in the topology; this aspect of our work was inspired by the results on adaptive timers in [15]. There, the timer constants were set in response to the number of duplicates observed and the latency of the responses, and this naturally led to the parameters being different for different members — *e.g.*, members located at different depths in a tree would have different settings. We do not directly address the dynamic nature of these timer adjustments, but merely study how location dependence in C_1 and C_2 changes performance.

We then investigate how the scaling behavior depends on the accuracy of the delay D. In SRM, the i^{th} group member estimates D_{ij}, $j = 1, 2, \ldots n$, $j \neq i$, the delay from itself to each of the other members of the group. Delay estimates are calculated from round-trip time (RTT) information which is derived from timestamps in *session messages* of the SRM protocol. Since the protocol's control bandwidth is limited to a constant fraction of the total available session bandwidth, the estimated RTT does not readily track changes in actual delay for large session sizes[4]. We study how RTT estimation might affect asymptotic scaling behavior in the different topologies by comparing performance in two extreme cases: one with exact RTT estimations and one where all members have the same hardwired RTT estimate.

3 Related Work

In this section, we summarize some important prior work related to the analysis of SRM. The seminal work of Floyd *et al.* [6] simulated group sizes of up to a few hundred nodes ranging across a set of simple topologies. They showed that it was often possible to choose values of C_1 and C_2 that resulted in $N(G)$ scaling as a constant independent of G. In particular, picking $C_1 = C_2 = 2$ achieved this for the chain topology, and picking $C_2 = \sqrt{G}$ resulted in constant scaling for the star topology (a special case of the cone topology in our work). Using simulations they demonstrated that $N(G) \leq 4$ for random trees with bounded degree for session sizes of up to 100. They also proposed an adaptive algorithm to dynamically adjust C_1 and C_2 based on past information for better performance.

Our work extends their important findings in two ways. First, we investigate performance for session sizes of up to two orders of magnitude larger than in [6], thus improving our collective understanding of SRM's asymptotic behavior. Reassuringly, our results agree with [6] where the experiments overlap. More generally, we have assessed in detail the behavior of $N(G)$ as a function of C_1 and C_2. Not only do these results help us predict the performance of SRM, but they could influence the design of related sub-components of SRM, e.g., the choice of bounding values of C_1 and C_2 in the proposed adaptive algorithm. A more recent paper [15] studied scaling behavior for group sizes up to 200 members, with $C_1 = 0$ and C_2 set adaptively.

In addition, Nonnenmacher and Biersack [18] looked at the effect of timer distribution on scaling behavior and showed

[4] Even in the case of a single TCP connection, where RTT estimates are gathered on every ACK, the sender's RTT-estimator is known to often be inaccurate [21].

that exponentially distributed timers yield better scaling properties. They found that having this distribution depend on the group size could result in improved scaling. We do not address the effects of different timer distributions at any great length in this paper.

This paper is primarily concerned with global recovery in SRM with constant C_1 and C_2. Variants of SRM have been proposed that use local recovery, in which NACKs and repairs are not sent to the entire group. [6], [14] look at two methods to limit the range of these methods: hop-scoping, and local recovery groups. [15] considers methods for adaptively setting the values for C_1 and C_2. We do not consider any of the local recovery methods, nor adaptive timer setting. Thus, our work should not be seen as a statement about how SRM-like protocols should function in the future, when they may well incorporate such features, but rather as an attempt to study the current deployed version of SRM with its use of global recovery. Our hope is that understanding this basic version of the protocol may inform future design efforts to improve it.

4 Simulation Methodology

In our simulations, we studied three classes of network topologies: *cone*, *linear chain*, and *binary tree*, each with a single source. The cone is a topology where each member has the same delay δ to every other member, and a distance Δ from the source. Similarly, for the linear chain and the binary tree, δ represents the link delay between adjacent members, and Δ is the link delay from the source to the closest member(s). Figures 2 and 3 show Δ and δ for the three topologies.

We are only modelling the behavior of NACKs, so we need only consider the receivers that suffer losses. Thus, we only consider the case where the loss occurs on the link adjacent to the source [5]. This causes little loss of generality, since if the loss occurs elsewhere we need only model the topology beneath the loss point. Note, however, that then the size of the group we are considering, G, is the size of the *loss group* – the number of members experiencing a particular packet loss – and not always the size of the entire group. Session messages in SRM give members knowledge about the size of the entire group, but not about the size of the loss group. If members knew the size of the loss group they might also be able to employ various forms of local recovery (hop-scoped recovery, or local recovery groups) that would more directly address the NACK traffic problem (not just limiting the number of NACKs, but also the portion of the group they are sent to). Thus, we do not consider varying the timer constants with group size, as in [18], as this does not seem like a realistic possibility.

Furthermore, we assume that losses are detected immediately when the next packet arrives. Since a packet is delivered to different receivers at different absolute times, losses are detected at different times. This typically allows the receivers closer to the source to suppress the NACKs from receivers further away. One of the key points in our investigation is how the setting of the timer constants affects this behavior.

We used the VINT network simulator *ns* [16] for our work. In its original form, ns turned out to have prolific memory usage with heavy-weight nodes, links, and multicast routing infrastructure, and could not support more than a few hundred nodes on an ordinary workstation. However,

[5] Measurements reported in [23] show that most correlated losses occur close to the source.

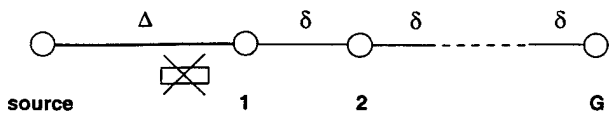


Figure 2: Linear Chain Topology: the X-ed packet marks the location of packet loss.

we took advantage of ns's extensible object-oriented architecture and made several modifications and extensions to it. Using the basic ns framework for event handling, we extended the simulator to support regular topologies with static routing without explicit routing table state. These modifications and extensions to ns enabled large-scale simulations of up to 50,000 nodes.

Losses occur on the link closest to the source, and are thus shared by all receivers in the group. We measure the average number of NACKs generated in response to a loss. The variation between different measurements is induced by the randomness in the recovery algorithm we are studying. We ran between 30 and 50 simulations of each case to compute the average value of the metrics, depending on the variance of the measured samples. Table 1 summarizes notation used in the rest of this paper.

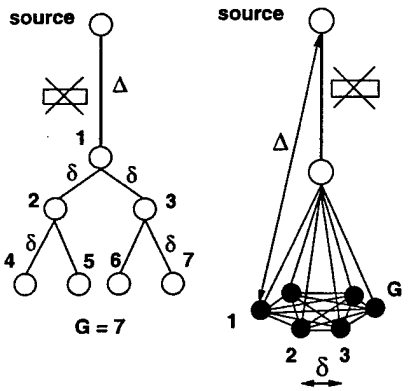


Figure 3: Binary tree and cone topologies: the X-ed packet marks the location of packet loss.

Symbol	Description
Δ	Delay from source to the closest receiver
δ	Delay of link connecting receivers
R	Δ/δ
G	Group size
N	Average number of copies of a single NACK
L	Average NACK latency caused by backoff
D_i	Estimate of one-way delay from node i to the source node
backoff$_i$, at host i	$D_i \times (C_1 + C_2 \times r_i)$ where, r_i are uniformly distributed random variables in $[0, 1]$
t_i	Absolute time at which receiver i's timer *fires*

Table 1: Summary of notation

In the following sections, we present our analytical and

simulation results for the three topologies: cone, line and binary tree.

5 Scaling in the Cone Topology

The cone topology can be used to model the case of a broadcast LAN. If the source is on the LAN then $\Delta = \delta$ but when the source is off the LAN, the delay from the LAN to the source is much greater than the LAN propagation time, yielding $\Delta >> \delta$. In general, the cone can be used to model a topology where all receivers have similar round-trip time estimates to the source. In practice, RTT estimators tend to be coarse-grained resulting in clusters of receivers with similar RTT values.

We use the following probabilistic analysis to compute the expectation of $N(G)$. Because all the receivers are at the same distance from the loss in a cone, the deterministic backoff component has no impact on the number of duplicates (all timers have the same constant offset). The average delay in transmitting the first NACK depends on the expected value of the minimum timer and is given by $\Delta(C_1 + \frac{C_2}{G+1})$. This result follows directly from noting that the expectation of the minimum of G uniformly distributed random variables in $[0,1]$ is $\frac{1}{G+1}$. The number of duplicates is equal to the expected number of timers that fire within $[t_{min}, t_{min} + \delta]$, where t_{min} is the value of the smallest timer. Since backoffs are uniformly distributed in $[C_1\Delta, (C_1 + C_2)\Delta]$, we can easily compute this expectation. Defining $\alpha = \frac{\delta}{C_2\Delta}$ we have,

$$E[N] = \begin{cases} 1 + G\alpha - \alpha^G, & \alpha < 1 \\ G & \alpha \geq 1 \end{cases}$$

Thus, the number of duplicates is roughly linear in the group size. [6] reports a similar result for the *star* topology, which is a *cone* with $\Delta = \delta$. Observe that this linear dependence applies regardless of whether the delay estimates are accurate or not. If the estimated value of the delay (assuming all members achieve the same estimate) is larger than the true estimate, then the number of duplicates is smaller, but the dependence on G is still linear. Our simulations, shown in Figure 4, confirm this result.

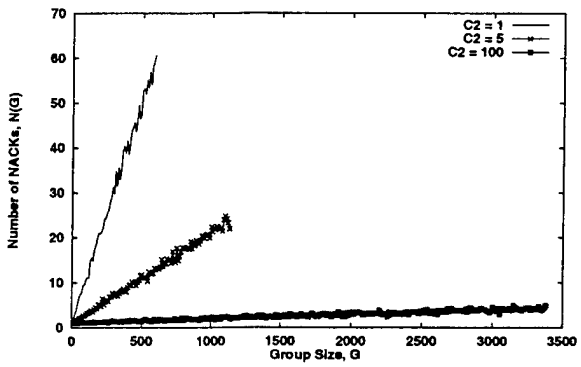


Figure 4: In the cone topology, N(G) grows linearly in G, where $\alpha = \frac{\delta}{C_2\Delta}$ and $C_1 \geq 0$, $C_2 > 0$.

$N(G)$ grows roughly linearly for any fixed timer distribution. However, as shown by Nonnenmacher and Biersack [18], if one makes the distribution dependent on the size of the loss group then one can change this linear scaling. For instance, if one takes a bimodal distribution such that with a probability $p = \frac{a}{G}$ a receiver sends a NACK immediately upon detecting a loss, and with probability $1 - p$ sends a NACK after a delay δ, then as G diverges $N(G)$ is given by $a(1-e^{-a})+Ge^{-a}$. By tuning a one can lower the slope of the linear dependence, and if one sets $a = \ln G$ the growth is logarithmic, not linear. One can remove the linear term entirely by considering the scheme where each receiver picks a number k from an exponential distribution with average $\frac{a}{G}$ and sets the backoff to $k\delta$. This is essentially a discrete version of the exponential distribution considered by Nonnenmacher and Biersack [18]. Here, the average number of NACKs is $E(N) = a$ and the average latency is $E(L) = \frac{e^a\delta}{1-e^{-a}}$. One can show that this achieves the lowest latency for a given number of NACKs (or equivalently, the smallest number of NACKs for a given latency) in the asymptotic limit. However, as we argued earlier, schemes that have the timer distribution depending on G are perhaps of little interest since the parameter G must be the size of the loss group, and once one has this information it might be better used in some local recovery approach rather than using it merely to tune the timer parameters.

6 Linear Chain

For the linear chain topology, we first consider the case where RTT estimation is exact. When $C_1 > 0$ and $C_2 \geq 0$, the data in Figure 5 suggests that $N(G)$ is constant in G.

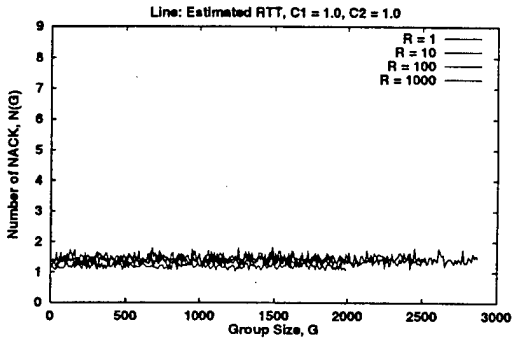


Figure 5: N(G) is a constant for $\Delta/\delta = 1, 10, 100, 1000$, with exact RTT estimation and $C_1 = C_2 = 1$. Similar results hold for other C_1 and C_2 as long as $C_1 > 0$.

We now show that in this parameter range there is a bound k on the maximal number of NACKs sent. Receiver i picks $C_1(\Delta+(i-1)\delta) \leq backoff_i \leq (C_1+C_2)(\Delta+(i-1)\delta)$. Consider some message sent at time $t = 0$, and assume that losses are detected immediately. Receiver i detects the loss at time $(\Delta+(i-1)\delta)$ and sends its NACK (if not suppressed) no later than a time $(\Delta + (i-1)\delta) + (C_1 + C_2)(\Delta + (i-1)\delta)$ and no sooner than $(\Delta + (i-1)\delta) + C_1(\Delta + (i-1)\delta)$. Therefore, receiver i and receiver j cannot both send NACKs if, assuming $j > i$,

$$(C_1+C_2)(\Delta+(i-1)\delta)+(j-i)\delta < (j-i)\delta+\delta+C_1(\Delta+(j-1)\delta)$$

This follows by recalling that it takes time $(j-i)\delta$ for i's NACK to propagate from i to j. Thus, the first member

on the line suppresses all but the next k members, where k is given by $k = \lfloor \frac{C_2 \Delta}{C_1 \delta} \rfloor$. Thus, $N(G)$ is bounded above by $k+1$. The simulations suggest that the *average number* $N(G)$ is much less than this upper bound, and in particular, is independent of R.

For $C_1 > 0$, the value of $N(G)$ appears, as shown in Figure 6, to be roughly independent of C_2. The dependence on C_1 is also shown in Figure 7, where, for a fixed G, N decreases with increasing C_1 as expected.

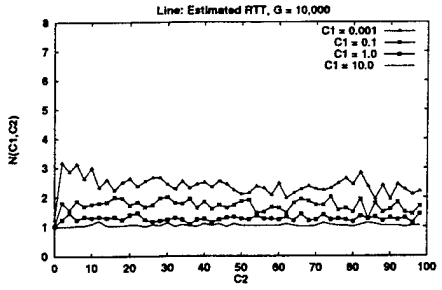


Figure 6: N as a function of C_1 and C_2.

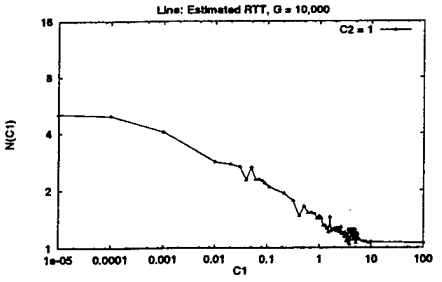


Figure 7: N as a function of C_1.

When $C_1 = 0$, there is no deterministic delay and the preceding argument fails. In fact, it appears that $N(G)$ diverges slowly with the group size G, as shown in Figure 8. We can argue that $N(G)$ does not grow faster than a certain expression derived below (but are not able to provide a lower bound). The probability that node i is not suppressed is bounded above by the probability that it is not suppressed by the members ahead of it in line. This occurs if and only if (ignoring ties) the backoff timer $t_i = min\{t_1, \ldots t_{i-1}\}$. Considering the case $\Delta = \delta$ for convenience. Using the notation $z_+ = \max[0, z]$, we have

$$Pr[t_i = min\{t_1, \ldots t_{i-1}\} | t_i = x] = \prod_{j=1}^{j=i-1} Pr[t_j \geq x]$$

$$= \prod_{j=1}^{j=i-1} (1 - x/j\delta)_+$$

and so, changing variables,

$$N(G) \leq \sum_{i=1}^{G} \int_0^1 \prod_{j=1}^{j=i-1} (1 - \frac{y}{j}) \frac{dy}{i}$$

Approximating $\prod_{j=1}^{j=i-1}(1 - \frac{y}{j})$ as $e^{-y \sum_{j=1}^{i-1} \frac{1}{j}}$ and then noting that $e^{-y \sum_{j=1}^{i-1} \frac{1}{j}} \approx e^{-y \ln i}$ and substituting into the integral, we see that this expression diverges as $\ln \ln G$.

We now consider the case where there is no RTT estimation, and all receivers use the same hardwired delay estimate D. Note that since deterministic delay is useless when round-trip times are not used (all members have the same deterministic delay), $C_2 = 0$ results in no suppression at all, and $N(G) = G$. This is true independent of topology; if there is no RTT estimation, then one needs $C_2 > 0$ or else $N(G) = G$, and $N(G)$ is independent of C_1.

Figure 9 shows $N(G)$ for the case $C_1 = 0$ and $C_2 = 1$ and fixed RTT. The growth, for all values of $R = \frac{\Delta}{\delta}$ appears to be logarithmic. Similar logarithmic-like behavior is observed in simulations with different values for C_2 and D.

The following probabilistic analysis suggests why, for $C_1 = 0$ and $C_2 = 1$, $N(G)$ grows as a logarithmic function of the group size. The backoffs are picked in the range $[0, D]$. We first compute the probability that the NACK at node i is not suppressed. The following condition must hold, for i's timer to fire:

$$d_j + r_j\delta + d_{ji} \geq d_i + r_i\delta, \ \forall j \neq i$$

where d_j is the one-way delay to receiver j from the source and d_{ij} is the one-way delay from receiver i to receiver j. r_i, r_j are uniformly distributed random numbers picked in $[0, 1]$ by the random timer mechanism. We then must have:

$$r_i < r_j, \ \forall j < i \quad (1)$$

$$r_j\delta + 2d_{ij} > r_i\delta, \ \forall j > i, \ and \quad (2)$$

$$d_{ij} \geq \delta, \forall d_{ij} \quad (3)$$

$$\Rightarrow r_i\delta < 2\delta + r_j\delta, \ \forall j > i \quad (4)$$

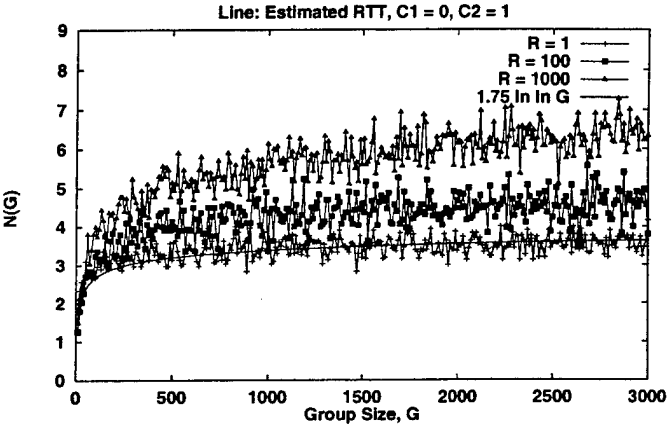


Figure 8: N(G) diverges as $\Delta/\delta = 1, 10, 100, 1000$, with RTT estimation, $C_1 = 0$, $C_2 = 1$.

From equation 4 above, we can conclude that a NACK at node i cannot be suppressed by a NACK at a later node. The condition for suppression at node i is therefore $r_i \leq min\{r_1, r_2, r_3, \ldots, r_{i-1}\}$. Thus, $P[i \ fires] = \frac{1}{i}$ and so $E[N] = \sum_{i=1}^{i=G} P[i \ fires] \approx \ln G + 0.577$. Similar logarithmic growth is seen empirically for larger C_2. The behavior of $N(G)$ for the line case is summarized in Table 6.

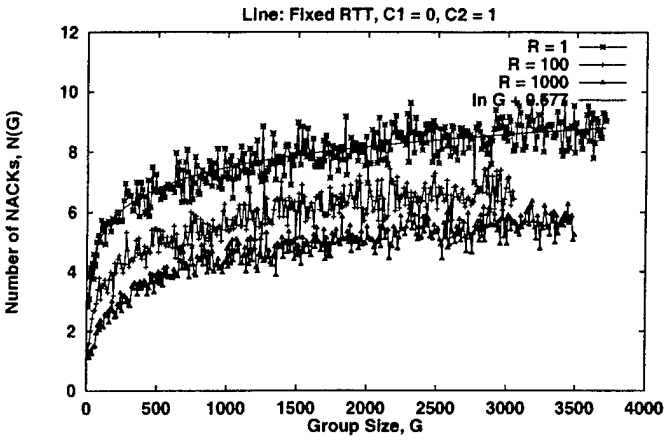


Figure 9: N(G) grows as a logarithmic function of G for $\Delta/\delta = 1, 10, 100, 1000$, fixed delay (no RTT estimation), $C_1 = 0$, $C_2 > 0$. $N(G) = \ln G + 0.577$, when $C_1 = 0, C_2 = 1$.

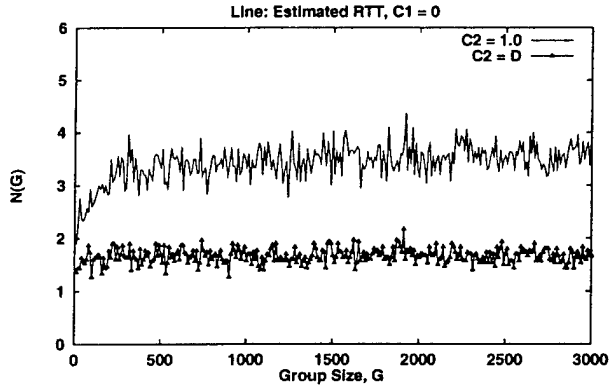


Figure 10: $N(G)$ converges to a constant when $C_2 = \sqrt{D}$ for the linear chain.

With $C_1 = 0$, $N(G)$ grows as $\ln \ln G$ for the linear chain topology. In order to reduce this growth in N(G) to a constant, while still retaining $C_1 = 0$ for the sake of low latency, we can make C_2 a function of the delay from the source. This follows the work Liu *et al.* who propose, in [15], using a new adaptive timer algorithm. Analysis similar to the previous case (equation (4)) shows that the number of duplicates is bounded by a constant when we use $C_2 = D^\epsilon$ for any $\epsilon > 0$. This is because

$$N(G) \leq \sum_{i=1}^{G} \int_0^1 \prod_{j=1}^{j=i-1} (1 - \frac{y}{j^{1+\epsilon}}) \frac{dy}{i^{1+\epsilon}}$$
$$\leq \sum_{i=1}^{G} \frac{1}{i^{1+\epsilon}} \leq \frac{1}{\epsilon}$$

The graph in Figure 10 shows that $N(G)$ converges to a constant for $\epsilon = 0.5$. We should note that because we do not have a lower bound for the case of C_2 fixed ($\epsilon = 0$). Our simulation results show that $N(G)$ diverges for $\epsilon = 0$, but our analytical proof is only for $\epsilon > 0$.

7 Binary Tree

In the binary tree topology (Figure 3), N(G) grows linearly with G when RTT is not estimated, as shown in Figures 11 and 12. The slope of this linear growth depends on C_2 and D (the fixed RTT). This linear behavior is in contrast with the logarithmic behavior observed in the line topology, but similar to the behavior in the cone topology. When RTT is known exactly, we still have linear behavior for $C_1 = 0$, as shown in Figure 12. The slope of this linear growth depends on both $\frac{\Delta}{\delta}$ and C_2.

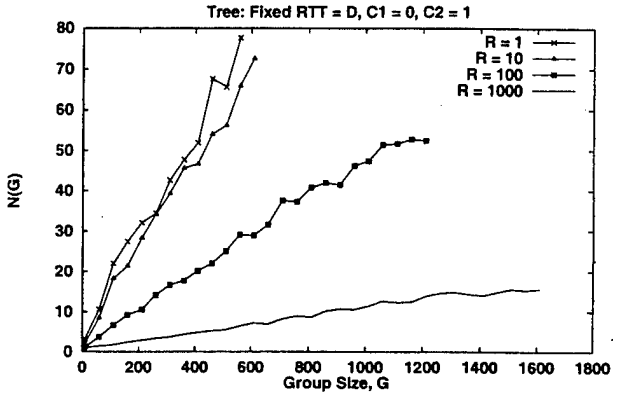


Figure 11: With $C_1 = 0$, $C_2 > 0$ and without RTT estimation, N(G) scales linearly with G for different values of $R = \Delta/\delta$.

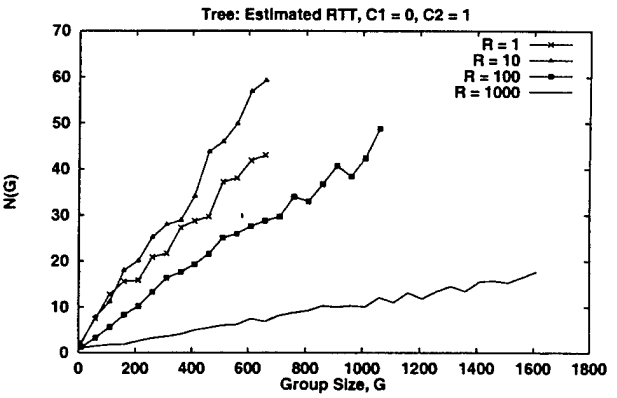


Figure 12: With $C_1 = 0$, $C_2 > 0$ and with accurate RTT estimation, N(G) scales linearly with G for different values of $R = \Delta/\delta$.

However, as soon as we have $C_1 > 0$, $D(G)$ appears to asymptotically reach a constant. Figure 13 shows the function $N(G)$ for different values of $0 \leq C_1 \leq 1$. The growth law for intermediate G is linear, and then the slope decreases as G increases. For all cases where we have been able to reach sufficiently large G, the slope continues to decrease until $N(G)$ goes to a constant.

When $C_1 > 0$, we see that the asymptotic scaling behavior depends on whether deterministic suppression or randomized suppression is dominant in reducing the number of

Δ/δ	RTT	C_1	C_2	N(G)	Figure
$1, 10, 100, 1000$	Fixed	$C_1 \geq 0$	$C_2 > 0$	Logarithmic ($\ln G + \gamma$, when $C_1 = 0, C_2 = 1$)	9
$1, 10, 100, 1000$	Fixed	$C_1 > 0$	$C_2 = 0$	Linear ($N(G) = G$)	
$1, 10, 100, 1000$	Estimated	$C_1 > 0$	$C_2 \geq 0$	Constant (≤ 4)	5
$1, 10, 100, 1000$	Estimated	$C_1 = 0$	$C_2 > 0$	Diverges	12

Table 2: Scaling behavior in the linear chain topology

NACKs. In cases where deterministic suppression is dominant, the asymptotic scaling is constant. Scaling is linear when suppression depends on the randomized suppression. In Figure 16, these two important effects are evident: as Δ/δ increases, deterministic suppression becomes weaker and randomized suppression is more effective. For large values of $\Delta/\delta > 100$, backoff timer ranges are large enough and the average separation between timers grows.

We now try to illustrate this behavior in a different form. The function $\frac{G}{N}$ plotted against G is shown in Figure 14. This ratio appears to be a linear functions of G, with the slope depending on C_1. If we label the slope of this line by m and the intercept by f, we have, for small C_1 and large G, the following form for N:

$$N = \frac{G}{mG + f}$$

The fit parameters m and f are functions of C_1 and C_2. This linear fit applies over a wide range of C_1, C_2 values. This functional form for $N(G)$ is consistent with our observation of a linear increase for small values of G, followed by this slope decreasing and the curve flattening to a constant. In particular, note that $\lim_{G\to\infty} N \to \frac{1}{m}$, a constant for a given value of C_1 and C_2. Thus, the slope of this functional fit in Figure 14 yields the asymptotic value for $N(G)$. Figure 15 shows this dependence on a log scale. $\frac{1}{m}$ decreases with increasing C_1 as expected.

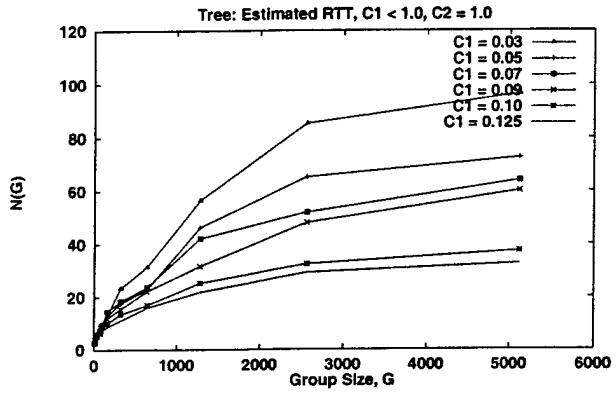


Figure 13: $N(G)$ in the binary tree for $R = \Delta/\delta = 1$, accurately estimated RTT and $0 < C_1 \leq 1$, $C_2 = 1$.

If we hold G fixed and vary R (the ratio of Δ to δ) we find that the dependence is not monotonic. Figure 17 shows this unimodal behavior. This behavior may be explained by the following reasoning. There are two kinds of suppression, deterministic and random, so-called depending on whether the possible firing times overlap or not. Deterministic suppression decreases with R, but random suppression

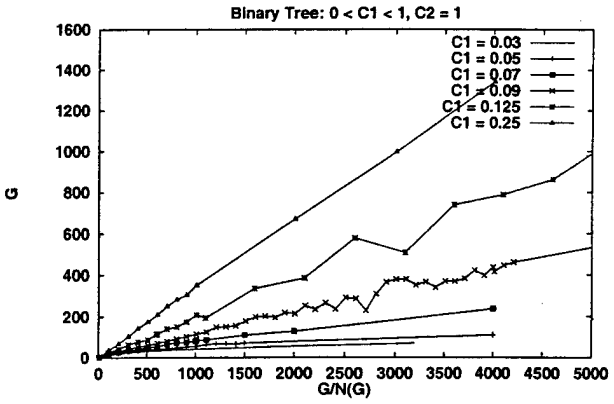


Figure 14: G/N vs. G in the binary tree for $R = \Delta/\delta = 1$, RTT estimated, $0 < C_1 \leq 1$, $C_2 = 1$.

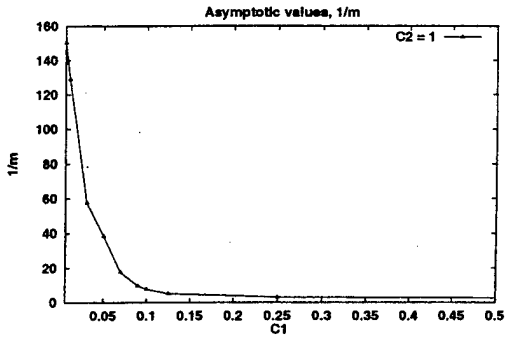


Figure 15: $\frac{1}{m} = \lim_{G\to\infty} D(G, C_1)$ as C_1 is varied. $C_2 = 1$

increases with R. Thus, as R is increased we first see an increase as the deterministic suppression becomes less effective, and then see a decrease as random suppression becomes dominant and deterministic suppression is no longer much of a factor (and so cannot decrease significantly further).

With $C_1 = 0$, and $C_2 > 0$, $N(G)$ grows linearly with G. In order to reduce this growth in $N(G)$ to a constant, while still retaining $C_1 = 0$, as we did for the linear chain topology, we make C_2 a function of the delay from the source. The adaptation algorithm described in [15] results in C_2 values that increase roughly linearly in D, the distance of a receiver from the source.

Here we do not model the dynamics of the adaptation, but instead merely insert the dependence on D directly. We consider several variants, with C_2 increasing as D, D^2, and $\sqrt{D}$. Figure 18 shows the results of these simulations. We

Δ/δ	RTT	C_1	C_2	N(G)	Figure
$1, 10, 100, 1000$	Fixed	$C_1 \geq 0$	$C_2 > 0$	Linear	12
$1, 10, 100, 1000$	Fixed	$C_1 > 0$	$C_2 = 0$	$N(G) = G$	
$1, 10, 100, 1000$	Estimated	$C_1 = 0$	$C_2 > 0$	Linear	12
$1, 10$	Estimated	$0 \leq C_1 \leq 1$	$C_2 \geq 0$	$G/(mG+f)$ $\lim_{G\to\infty} G/(mG+f) = \text{constant}$	13
$100, 1000$	Estimated	$0 \leq C_1 \leq 1$	$C_2 \geq 0$	Linear	16

Table 3: Scaling behavior in the tree topology

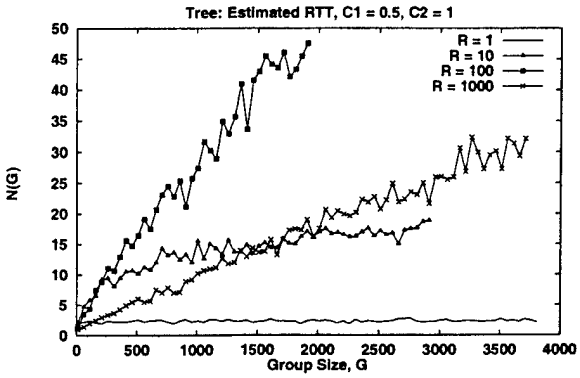


Figure 16: N(G) in the binary tree with $\Delta/\delta = 1, 10, 100, 1000$, RTT estimated, $C_1 = 0.5$, $C_2 = 1$.

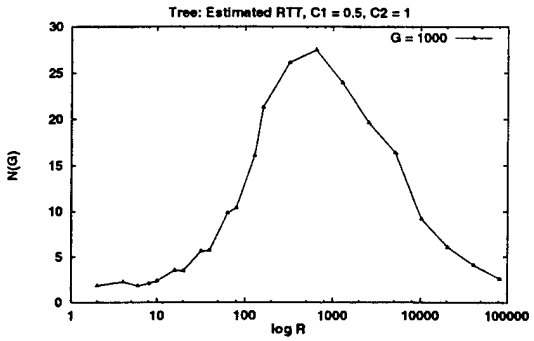


Figure 17: For small values of R, the round-trip times from the source to the receivers are distinguishable, and deterministic suppression effectively keeps the NACK count low. When Δ/δ increase, randomized suppression is the dominant cause for suppression. The "turning point" value of Δ/δ depends on the topology.

find that C_2 needs to be "super-linear" in D to make scaling constant.

8 Conclusions

In this paper, we used analysis and simulation to study the scaling behavior of global loss recovery in SRM. The SRM protocol is NACK-based and uses a randomized, timer-based decentralized algorithm to reduce NACK implosion. We use the number of NACKs $N(G)$ generated in response to a loss, as a metric for scalability. The two protocol parameters, C_1 and C_2, govern the deterministic and random delays in the firing of a NACK from a receiver. There is a trade-off between low-latency loss recovery and the number of NACKs – in general, making these parameters small leads to lower latency, but usually at the expense of poorer asymptotic scaling. We study $N(G)$ as a function of group size, G, for various protocol parameters, on a set of simple, representative topologies — the cone, the linear chain, and the binary tree.

In the cone topology, we find that random backoff is the dominant reason for suppression and scaling is linear. This linear scaling can be reduced by using a distribution that is dependent on the group size. The cone models topologies in which receivers have similar round-trip time estimates to the source. For the linear chain $N(G)$ is between constant (when $C_1 > 0, C_2 > 0$, and RTT estimation is perfect), and logarithmic, when RTT is not estimated. In the tree, scaling is between constant (when $C_1 > 0, C_2 > 0$, and RTT estimattion is perfect), and linear, when RTT is not estimated. For the linear chain we show that $C_2 = D^\epsilon$ results in constant scaling even when $C_1 = 0$, where D is the one-way delay to the source. Similarly, for the binary tree, $C_2 = D^2$ results in constant scaling.

We find that in topologies where deterministic suppression is effective in reducing the number of duplicate NACKs, asymptotic scaling tends to a constant. For topologies in which randomized suppression is mainly responsible for eliminating duplicates, asymptotic scaling is not constant, e.g., in the cone topology and in the binary tree with $\Delta >> \delta$, $N(G)$ grows linearly.

In conclusion, we have shown that there is a rich parameter space in the SRM protocol and that the best asymptotic scaling performance is sensitive to the choice of these parameters. We expect our results to be useful in obtaining a better understanding of the reasons for SRM's scaling properties in different situations, and in aiding the design and analysis of future modifications to SRM and similar protocols that use multicast transmission and suppression.

9 Acknowledgements

We would like to thank Hari Balakrishnan and Lee Breslau for extremely useful feedback on this work. At Berkeley, this research was supported by DARPA contract N66001-96-C-8508, by the State of California under the MICRO program, and by NSF Contract CDA 94-01156. At Xerox PARC, this research was supported in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contracts DABT63-94-C-0073. The views expressed here do not reflect the position or policy of the U.S. government.

References

[1] Ballardie, T., Francis, P., and Crowcroft, J.

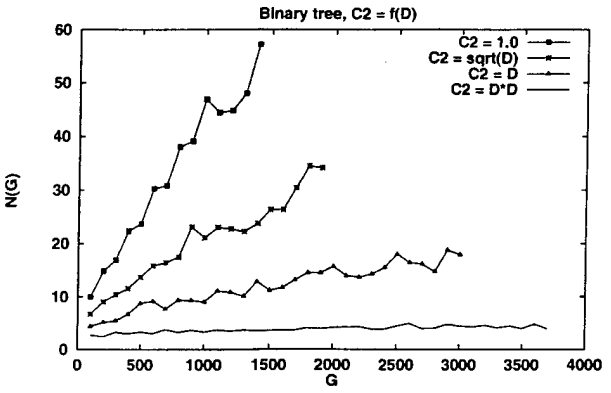


Figure 18: When $C_2 = D^{0.5}$ in the binary tree, N(G) has improved scaling.

Core Based Trees (CBT) An Architecture for Scalable Inter-Domain Multicast Routing. In *Proceedings of SIGCOMM '93* (San Francisco, CA, Sept. 1993), ACM, pp. 85–95.

[2] CHESSON, G. XTP/protocol engine design. In *Proceedings of the IFIP WG6.1/6.4 Workshop* (Rüschlikon, May 1989).

[3] CLARK, D. D., AND TENNENHOUSE, D. L. Architectural Considerations for a New Generation of Protocols. In *Proceedings of SIGCOMM '90* (Philadelphia, PA, Sept. 1990), ACM.

[4] DEERING, S., ESTRIN, D., FARINACCI, D., AND JACOBSON, V. An Architecture for Wide-Area Multicast Routing. In *Proceedings of SIGCOMM '94* (University College London, London, U.K., Sept. 1994), ACM.

[5] DEERING, S. E. *Multicast Routing in a Datagram Internetwork.* PhD thesis, Stanford University, Dec. 1991.

[6] FLOYD, S., JACOBSON, V., MCCANNE, S., LIU, C.-G., AND ZHANG, L. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of SIGCOMM '95* (Boston, MA, Sept. 1995), ACM.

[7] FREDERICK, R. *Network Video (nv).* Xerox Palo Alto Research Center. Software on-line ftp://ftp.parc.xerox.com/net-research.

[8] HOLBROOK, H., SINGHAL, S., AND CHERITON, D. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *Proceedings of SIGCOMM '95* (Boston, MA, Sept. 1995), ACM.

[9] JACOBSON, V., AND MCCANNE, S. *LBL Whiteboard.* Lawrence Berkeley Laboratory. Software on-line ftp://ftp.ee.lbl.gov/conferencing/wb.

[10] JACOBSON, V., AND MCCANNE, S. *Visual Audio Tool.* Lawrence Berkeley Laboratory. Software on-line ftp://ftp.ee.lbl.gov/conferencing/vat.

[11] KASERA, S. K., KUROSE, J. F., AND TOWSLEY, D. F. Scalable Reliable Multicast Using Multiple Multicast Groups. In *Proceedings of ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems* (June 1997).

[12] LEVINE, B. N., LAVO, D. B., AND GARCIA-LUNA-ACEVES, J. The Case For Reliable Concurrent Multicasting Using Shared Ack Trees. In *Proceedings of ACM Multimedia '96* (Boston, MA, Nov. 1996), ACM.

[13] LIN, J. C., AND PAUL, S. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings IEEE Infocom '96* (San Francisco, CA, Mar. 1996), pp. 1414–1424.

[14] LIU, C.-G., ESTRIN, D., SHENKER, S., AND ZHANG, L. Local Recovery in SRM. *Submitted to IEEE Transactions on Networking* (1998).

[15] LIU, C.-G., ESTRIN, D., SHENKER, S., AND ZHANG, L. Recovery Timer Adaptation in SRM. *Submitted to IEEE Transactions on Networking* (1998).

[16] MCCANNE, S., AND FLOYD, S. *The LBNL Network Simulator.* University of California, Berkeley. http://www-mash.cs.berkeley.edu/ns/.

[17] MCCANNE, S., AND JACOBSON, V. *vic: video conference.* Lawrence Berkeley Laboratory and University of California, Berkeley. Software on-line ftp://ftp.ee.lbl.gov/conferencing/vic.

[18] NONNENMACHER, J., AND BIERSACK, E. W. Optimal Multicast Feedback. *IEEE Infocom* (1998).

[19] PINGALI, D., TOWSLEY, D., AND KUROSE, J. F. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of SIGMETRICS '94* (Santa Clara, CA, May 1994).

[20] RAMAN, S., AND MCCANNE, S. Generalized Data Naming and Scalable State Announcements for Reliable Multicast. Tech. rep., University of California, Berkeley, CA, June 1997.

[21] STEVENS, W. R. *TCP/IP Illustrated, Volume 1 – The Protocols,* first ed. Addison-Wesley, Dec. 1994.

[22] TURLETTI, T. *INRIA Video Conferencing System (ivs).* Institut National de Recherche en Informatique et an Automatique. Software on-line http://www.inria.fr/rodeo/ivs.html.

[23] YAJNIK, M., KUROSE, J., AND TOWSLEY, D. Packet Loss Correlation in the MBone Multicast Network. *IEEE Global Internet Conference* (1996).

[24] YAVATKAR, R., GRIFFIOEN, J., AND SUDAN, M. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *Proceedings of ACM Multimedia '95* (San Francisco, CA, Nov. 1995), ACM.

Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks*

Ion Stoica
CMU
istoica@cs.cmu.edu

Scott Shenker
Xerox PARC
shenker@parc.xerox.com

Hui Zhang
CMU
hzhang@cs.cmu.edu

Abstract

Router mechanisms designed to achieve fair bandwidth allocations, like Fair Queueing, have many desirable properties for congestion control in the Internet. However, such mechanisms usually need to maintain state, manage buffers, and/or perform packet scheduling on a per flow basis, and this complexity may prevent them from being cost-effectively implemented and widely deployed. In this paper, we propose an architecture that significantly reduces this implementation complexity yet still achieves approximately fair bandwidth allocations. We apply this approach to an island of routers – that is, a contiguous region of the network – and we distinguish between edge routers and core routers. Edge routers maintain per flow state; they estimate the incoming rate of each flow and insert a label into each packet header based on this estimate. Core routers maintain *no* per flow state; they use FIFO packet scheduling augmented by a probabilistic dropping algorithm that uses the packet labels and an estimate of the aggregate traffic at the router. We call the scheme *Core*-Stateless Fair Queueing. We present simulations and analysis on the performance of this approach, and discuss an alternate approach.

1 Introduction

A central tenet of the Internet architecture is that congestion control is achieved mainly through end-host algorithms. However, starting with Nagle [16], many researchers observed that such end-to-end congestion control solutions are greatly improved when routers have mechanisms that allocate bandwidth in a fair manner. Fair bandwidth allocation protects well-behaved flows from ill-behaved ones, and allows a diverse set of end-to-end congestion control policies to co-exist in the network [7]. As we discuss in Section 4, some maintain that fair bandwidth allocation[1] plays a necessary, not just beneficial, role in congestion control [7, 19].

Until now, fair allocations were typically achieved by using per-flow queueing mechanisms – such as Fair Queueing [7, 18] and its many variants [2, 10, 20] – or per-flow dropping mechanisms such as Flow Random Early Drop (FRED) [14]. These mechanisms are more complex to implement than traditional FIFO queueing with drop-tail, which is the most widely implemented and deployed mechanism in routers today. In particular, fair allocation mechanisms inherently require the routers to maintain state and perform operations on a per flow basis. For each packet that arrives at the router, the routers needs to *classify* the packet into a flow, update per flow state variables, and perform certain operations based on the per flow state. The operations can be as simple as deciding whether to drop or queue the packet (*e.g.*, FRED), or as complex as manipulation of priority queues (*e.g.*, Fair Queueing). While a number of techniques have been proposed to reduce the complexity of the per packet operations [1, 20, 21], and commercial implementations are available in some intermediate class routers, it is still unclear whether these algorithms can be cost-effectively implemented in high-speed backbone routers because all these algorithms still require packet classification and per flow state management.

In this paper we start with the assumption that (1) fair allocation mechanisms play an important, perhaps even necessary, role in congestion control, and (2) the complexity of existing fair allocation mechanisms is a substantial hindrance to their adoption. Both of these points are debatable; developments in router technology may make such algorithms rather inexpensive to implement, and there may be solutions to congestion control that do not require fair allocation (we discuss this point more fully in Section 4). By using these two assumptions as our starting points we are not claiming that they *are* true, but rather are only looking at the implications if indeed they *were* true. If one starts with these assumptions then overcoming the complexity problem in achieving fair allocation becomes a vitally important problem.

To this end, we propose and examine an architecture and a set of algorithms that allocate bandwidth in an approximately fair manner while allowing the routers on high-speed links to use FIFO queueing and maintain no per-flow state.

*This research was sponsored by DARPA under contract numbers N66001-96-C-8528, E30602-97-2-0287, and DABT63-94-C-0073, and by a NSF Career Award under grant number NCR-9624979. Additional support was provided by Intel Corp., MCI, and Sun Microsystems. Views and conclusions contained in this document are those of the authors and should no be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, MCI, Sun, or the U.S. government.

[1] We use the max-min definition of fairness [12] which, while not the only possible candidate for fairness, is certainly a reasonable one and, moreover, can be implemented with only local information.

In this approach, we identify an *island* of routers[2] and distinguish between the edge and the core of the island. Edge routers compute per-flow rate estimates and *label* the packets passing through them by inserting these estimates into each packet header. Core routers use FIFO queueing and keep no per-flow state. They employ a probabilistic dropping algorithm that uses the information in the packet labels along with the router's own measurement of the aggregate traffic. The bandwidth allocations within this island of routers are approximately fair. Thus, if this approach were adopted within the high speed interiors of ISP's, and fair allocation mechanisms were adopted for the slower links outside of these high-speed interiors, then approximately fair allocations could be achieved everywhere. However, this approach, like Fair Queueing [7] or RED [9], still provides benefit if adopted in an incremental fashion, although the incremental adoption must be done on an island-by-island basis, not on a router-by-router basis.

We call this approach *Core*-Stateless Fair Queueing (CSFQ) since the core routers keep no per-flow state but instead use the state that is carried in the packet labels.[3] We describe the details of this approach – such as the rate estimation algorithm and the packet dropping algorithm – in Section 2.

Such a scheme cannot hope to achieve the nearly-perfect levels of fairness obtained by Fair Queueing and other sophisticated and stateful queueing algorithms. However, our interest is not in perfection, but only in obtaining reasonable approximations to the fair bandwidth allocations. We derive a worst-case bound for the performance of this algorithm in an idealized setting. This bound is presented in Section 2.

This worst-case analysis does not give an adequate guide to the typical functioning of CSFQ. In Section 3 we present results from simulation experiments to illustrate the performance of our approach and to compare it to several other schemes: DRR (a variant of Fair Queueing), FRED, RED, and FIFO. We also discuss, therein, the relative mechanistic complexities of these approaches.

The first 3 sections of the paper are narrowly focussed on the details of the mechanism and its performance (both absolute and relative), with the need for such a mechanism taken for granted. In Section 4 we return to the basic question of why fair allocations are relevant to congestion control. Allocating bandwidth fairly is one way to address what we call the *unfriendly flow problem*; we also discuss an alternate approach to addressing this problem, the *identification* approach as described in [8]. We conclude with a summary in Section 5. A longer version of this paper, containing proofs of the theoretical results as well as more complete pseudocode, can be found at `http://www.cs.cmu.edu/~istoica/csfq`.

2 Core-Stateless Fair Queueing (CSFQ)

In this section, we propose an architecture that approximates the service provided by an island of Fair Queueing routers, but has a much lower complexity in the core routers. The architecture has two key aspects. First, to avoid maintaining per flow state at each router, we use a distributed algorithm in which only edge routers maintain per flow state, while core (non-edge) routers do not maintain per flow state but instead utilize the per-flow information carried via a label in each packet's header. This label contains an estimate of the flow's rate; it is initialized by the edge router based on per-flow information, and then updated at each router along the path based only on aggregate information at that router.

Second, to avoid per flow buffering and scheduling, as required by Fair Queueing, we use FIFO queueing with probabilistic dropping on input. The probability of dropping a packet as it arrives to the queue is a function of the rate estimate carried in the label and of the fair share rate at that router, which is estimated based on measurements of the aggregate traffic.

Thus, our approach avoids both the need to maintain per-flow state and the need to use complicated packet scheduling and buffering algorithms at core routers. To give a better intuition about how this works, we first present the idealized bit-by-bit or *fluid* version of the probabilistic dropping algorithm, and then extend the algorithm to a practical packet-by-packet version.

2.1 Fluid Model Algorithm

We first consider a bufferless fluid model of a router with output link speed C, where the flows are modelled as a continuous stream of bits. We assume each flow's arrival rate $r_i(t)$ is known precisely. Max-min fair bandwidth allocations are characterized by the fact that all flows that are bottlenecked (*i.e.*, have bits dropped) by this router have the same output rate. We call this rate the *fair share rate* of the server; let $\alpha(t)$ be the fair share rate at time t. In general, if max-min bandwidth allocations are achieved, each flow i receives service at a rate given by $\min(r_i(t), \alpha(t))$. Let $A(t)$ denote the total arrival rate: $A(t) = \sum_{i=1}^{n} r_i(t)$. If $A(t) > C$ then the fair share $\alpha(t)$ is the unique solution to

$$C = \sum_{i=1}^{n} \min(r_i(t), \alpha(t)), \tag{1}$$

If $A(t) \leq C$ then no bits are dropped and we will, by convention, set $\alpha(t) = \max_i r_i(t)$.

If $r_i(t) \leq \alpha(t)$, *i.e.*, flow i sends no more than the server's fair share rate, all of its traffic will be forwarded. If $r_i(t) > \alpha(t)$, then a fraction $\frac{r_i(t)-\alpha(t)}{r_i(t)}$ of its bits will be dropped, so it will have an output rate of exactly $\alpha(t)$. This suggests a very simple probabilistic forwarding algorithm that achieves fair allocation of bandwidth: each incoming bit of flow i is dropped with the probability

$$\max\left(0, 1 - \frac{\alpha(t)}{r_i(t)}\right) \tag{2}$$

When these dropping probabilities are used, the arrival rate of flow i at the next hop is given by $\min[r_i(t), \alpha(t)]$.

2.2 Packet Algorithm

The above algorithm is defined for a bufferless fluid system in which the arrival rates are known exactly. Our task now is to extend this approach to the situation in real routers where transmission is packetized, there is substantial buffering, and the arrival rates are not known.

[2] By island we mean a contiguous portion of the network, with well-defined interior and edges.

[3] Obviously these core routers keep some state, but none of it is per-flow state, so when we say "stateless" we are referring to the absence of per-flow state.

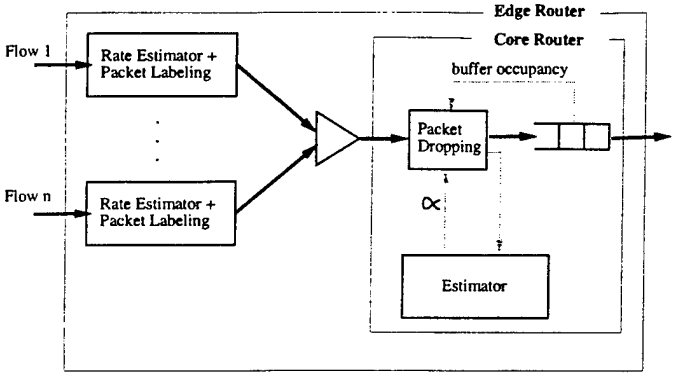


Figure 1: The architecture of the output port of an edge router, and a core router, respectively.

We still employ a drop-on-input scheme, except that now we drop packets rather than bits. Because the rate estimation (described below) incorporates the packet size, the dropping probability is independent of the packet size and depends only, as above, on the rate $r_i(t)$ and fair share rate $\alpha(t)$.

We are left with two remaining challenges: estimating the rates $r_i(t)$ and the fair share $\alpha(t)$. We address these two issues in turn in the next two subsections, and then discuss the rewriting of the labels. Pseudocode reflecting this algorithm is described in Figure 2. We should note, however, that the main point of our paper is the overall architecture and that the detailed algorithm presented below represents only an initial prototype. While it serves adequately as a proof-of-concept of our architecture, we fully expect that the details of this design will continue to evolve.

2.2.1 Computation of Flow Arrival Rate

Recall that in our architecture, the rates $r_i(t)$ are estimated at the edge routers and then these rates are inserted into the packet labels. At each edge router, we use exponential averaging to estimate the rate of a flow. Let t_i^k and l_i^k be the arrival time and length of the k^{th} packet of flow i. The estimated rate of flow i, r_i, is updated every time a new packet is received:

$$r_i^{new} = (1 - e^{-T_i^k/K})\frac{l_i^k}{T_i^k} + e^{-T_i^k/K} r_i^{old}, \qquad (3)$$

where $T_i^k = t_i^k - t_i^{k-1}$ and K is a constant. We discuss the rationale for using the form $e^{-T_i^k/K}$ for the exponential weight in Section 2.7. In the longer version of this paper [22] we show that, under a wide range of conditions, this estimation algorithm converges.

2.2.2 Link Fair Rate Estimation

In this section, we present an estimation algorithm for $\alpha(t)$. To give intuition, consider again the fluid model in Section 2.1 where the arrival rates are known exactly, and assume the system performs the probabilistic dropping algorithm according to Eq. (2). Then, the rate with which the algorithm accepts packets is a function of the current estimate of the fair share rate, which we denote by $\widehat{\alpha}(t)$. Letting $F(\widehat{\alpha}(t))$ denote this acceptance rate, we have

$$F(\widehat{\alpha}(t)) = \sum_{i=1}^{n} \min\left(r_i(t), \widehat{\alpha}(t)\right). \qquad (4)$$

Note that $F(\cdot)$ is a continuous, nondecreasing, concave, and piecewise-linear function of $\widehat{\alpha}$. If the link is congested ($A(t) > C$) we choose $\widehat{\alpha}(t)$ to be the unique solution to $F(x) = C$. If the link is not congested ($A(t) < C$) we take $\widehat{\alpha}(t)$ to be the largest rate among the flows that traverse the link, *i.e.*, $\widehat{\alpha}(t) = \max_{1 \le i \le n}(r_i(t))$. From Eq (4) note that if we knew the arrival rates $r_i(t)$ we could then compute $\alpha(t)$ directly. To avoid having to keep such per-flow state, we seek instead to implicitly compute $\widehat{\alpha}(t)$ by using only aggregate measurements of F and A.

We use the following heuristic algorithm with three aggregate state variables: $\widehat{\alpha}$, the estimate for the fair share rate; $\widehat{A}$, the estimated aggregate arrival rate; $\widehat{F}$, the estimated rate of the accepted traffic. The last two variables are updated upon the arrival of each packet. For $\widehat{A}$ we use exponential averaging with a parameter e^{-T/K_α} where T is the inter-arrival time between the current and the previous packet:

$$\widehat{A}_{new} = (1 - e^{-T/K_\alpha})\frac{l}{T} + e^{-T/K_\alpha}\widehat{A}_{old} \qquad (5)$$

where $\widehat{A}_{old}$ is the value of $\widehat{A}$ before the updating. We use an analogous formula to update $\widehat{F}$.

The updating rule for $\widehat{\alpha}$ depends on whether the link is congested or not. To filter out the estimation inaccuracies due to exponential smoothing we use a window of size K_c. A link is assumed to be *congested*, if $\widehat{A} \ge C$ at all times during an interval of length K_c. Conversely, a link is assumed to be *uncongested*, if $\widehat{A} \le C$ at all times during an interval of length K_c. The value $\widehat{\alpha}$ is updated only at the end of an interval in which the link is either congested or uncongested according to these definitions. If the link is congested then $\widehat{\alpha}$ is updated based on the equation $F(\widehat{\alpha}) = C$. We approximate $F(\cdot)$ by a linear function that intersects the origin and has slope $\widehat{F}/\widehat{\alpha}_{old}$. This yields

$$\widehat{\alpha}_{new} = \widehat{\alpha}_{old}\frac{C}{\widehat{F}} \qquad (6)$$

If the link is not congested, $\widehat{\alpha}_{new}$ is set to the largest rate of any active flow (*i.e.*, the largest label seen) during the last K_c time units. The value of $\widehat{\alpha}_{new}$ is then used to compute dropping probabilities, according to Eq. (2). For completeness, we give the pseudocode of the CSFQ algorithm in Figure 2.

We now describe two minor amendments to this algorithm related to how the buffers are managed. The goal of estimating the fair share $\widehat{\alpha}$ is to match the accepted rate to the link bandwidth. Due to estimation inaccuracies, load fluctuations between $\widehat{\alpha}$'s updates, and the probabilistic nature of our algorithm, the accepted rate may occasionally exceed the link capacity. While ideally the router's buffers can accommodate the extra packets, occasionally the router may be forced to drop the incoming packet due to lack of buffer space. Since drop-tail behavior will defeat the purpose of our algorithm, and may exhibit undesirable properties in the case of adaptive flows such as TCP [9], it is important to limit its effect. To do so, we use a simple heuristic: every

```
on receiving packet p
  if (edge router)
    i =classify(p);
    p.label = estimate_rate(r_i, p); /* use Eq. (3) */
  prob =max(0, 1 − α/p.label);
  if (prob >unif_rand(0, 1))
    α =estimate_α (p, 1);
    drop(p);
  else
    α =estimate_α (p, 0);
    enqueue(p);
  if (prob > 0)
    p.label = α; /* relabel p */

estimate_α (p, dropped)
  estimate_rate(Â, p); /* est. arrival rate (use Eq. (5)) */
  if (dropped == FALSE)
    estimate_rate(F̂, p); /* est. accepted traffic rate */
  if (Â ≥ C)
    if (congested == FALSE)
      congested = TRUE;
      start_time = crt_time;
    else
      if (crt_time > start_time + K_c)
        α̂ = α̂ × C/F̂;
        start_time = crt_time;
  else /* Â < C */
    if (congested == TRUE)
      congested = FALSE;
      start_time = crt_time;
      tmp_α = 0; /* use to compute new α */
    else
      if (crt_time < start_time + K_c)
        tmp_α =max(tmp_α, p.label);
      else
        α̂ = tmp_α;
        start_time = crt_time;
        tmp_α = 0;
  return α̂;
```

Figure 2: The pseudocode of CSFQ.

time the buffer overflows, $\hat{\alpha}$ is decreased by a small fixed percentage (taken to be 1% in our simulations). Moreover, to avoid overcorrection, we make sure that during consecutive updates $\hat{\alpha}$ does not decrease by more than 25%.

In addition, since there is little reason to consider a link congested if the buffer is almost empty, we apply the following rule. If the link becomes uncongested by the test in Figure 2, then we assume that it remains uncongested as long as the buffer occupancy is less than some predefined threshold. In this paper we use a threshold that is half of the total buffer capacity.

2.2.3 Label Rewriting

Our rate estimation algorithm in Section 2.2.1 allows us to label packets with their flow's rate as they enter the island. Our packet dropping algorithm described in Section 2.2.2 allows us to limit flows to their fair share of the bandwidth. After a flow experiences significant losses at a congested link inside the island, however, the packet labels are no longer an accurate estimate of its rate. We cannot rerun our estimation algorithm, because it involves per-flow state. Fortunately, as note in Section 2.1 the outgoing rate is merely the *minimum* between the incoming rate and the fair rate α. Therefore, we rewrite the the packet label L as

$$L_{new} = \min(L_{old}, \alpha), \tag{7}$$

By doing so, the outgoing flow rates will be properly represented by the packet labels.

2.3 Weighted CSFQ

The CSFQ algorithm can be extended to support flows with different weights. Let w_i denote the weight of flow i. Returning to our fluid model, the meaning of these weights is that we say a *fair* allocation is one in which all bottlenecked flows have the same value for $\frac{r_i}{w_i}$. Then, if $A(t) > C$, the *normalized* fair rate $\alpha(t)$ is the unique value such that $\sum_{i=1}^{n} w_i \min\left(\alpha, \frac{r_i}{w_i}\right) = C$. The expression for the dropping probabilities in the weighted case is $\max\left(0, 1 - \alpha\frac{w_i}{r_i}\right)$. The only other major change is that the label is now r_i/w_i, instead simply r_i. Finally, without going into details we note that the weighted packet-by-packet version is virtually identical to the corresponding version of the plain CSFQ algorithm.

It is important to note that with weighted CSFQ we can only approximate islands in which each flow has the same weight at all routers in an island. That is, our algorithm cannot accommodate situations where the relative weights of flows differ from router to router within an island. However, even with this limitation, weighted CSFQ may prove a valuable mechanism in implementing differential services, such as the one proposed in [24].

2.4 Performance Bounds

We now present the main theoretical result of the paper. For generality, this result is given for weighted CSFQ. The proof is given in [22].

Our algorithm is built around several estimation procedures, and thus is inherently inexact. One natural concern is whether a flow can purposely "exploit" these inaccuracies to get more than its fair share of bandwidth. We cannot answer this question in full generality, but we can analyze a simplified situation where the normalized fair share rate α is held fixed and there is no buffering, so the drop probabilities are precisely given by Eq. (2). In addition, we assume that when a packet arrives a fraction of that packet equal to the flow's forwarding probability is transmitted. Note that during any time interval $[t_1, t_2)$ a flow with weight w is entitled to receive at most $w\alpha(t_2 - t_1)$ service time; we call any amount above this the *excess service*. We can bound this excess service, and the bounds are independent of both the arrival process and the length of the time interval during which the flow is active. The bound does depend crucially on the maximal rate R at which a flows packets can arrive at a router (limited, for example, by the speed of the flow's access link); the smaller this rate R the tighter the bound.

Theorem 1 *Consider a link with a constant normalized fair rate α, and a flow with weight w. Then, the excess service received by a flow with weight w, that sends at a rate no larger than R is bounded above by*

$$r_\alpha K\left(1+ln\frac{R}{r_\alpha}\right)+l_{\max}, \quad (8)$$

where $r_\alpha = \alpha w$, and l_{max} represents the maximum length of a packet.

By bounding the excess service, we have shown that in this idealized setting the asymptotic throughput cannot exceed the fair share rate. Thus, flows can only exploit the system over short time scales; they are limited to their fair share over long time scales.

2.5 Implementation Complexity

At core routers, both the time and space complexity of our algorithm are constant with respect to the number of competing flows, and thus we think CSFQ could be implemented in very high speed core routers. At each edge router CSFQ needs to maintain per flow state. Upon each arrival of each packet, the edge router needs to (1) classify the packet to a flow, (2) update the fair share rate estimation for the corresponding outgoing link, (3) update the flow rate estimation, and (4) label the packet. All these operations with the exception of packet classification can be efficiently implemented today.

Efficient and general-purpose packet classification algorithms are still under active research. We expect to leverage these results. We also note that packet classification at ingress nodes is needed for a number of other purposes, such as in the context of Multiprotocol Label Switching (MPLS) [4] or for accounting purposes; therefore, the classification required for CSFQ may not be an extra cost. In addition, if the edge routers are typically not on the high-speed backbone links then there is no problem as classification at moderate speeds is quite practical.

2.6 Architectural Considerations

We have used the term flow without defining what we mean. This was intentional, as the CSFQ approach can be applied to varying degrees of flow granularity; that is, what constitutes a flow is arbitrary as long as all packets in the flow follow the same path within the core. In this paper, for convenience, a flow is implicitly defined as a source-destination pair, but one could easily assign fair rates to many other granularities such as source-destination-ports. Moreover, the unit of "flow" can vary from island to island as long as the rates are re-estimated when entering a new island.

Similarly, we have not been precise about the size of these CSFQ islands. In one extreme, we could take each router as an island and estimate rates at every router; this would allow us to avoid the use of complicated per-flow scheduling and dropping algorithms, but would require per-flow classification. Another possibility is that ISP's could extend their island of CSFQ routers to the very edge of their network, having their edge routers at the points where customer's packets enter the ISP's network. Building on the previous scenario, multiple ISP's could combine their islands so that classification and estimation did not have to be performed at ISP-ISP boundaries. The key obstacle here is one of trust between ISPs.

2.7 Miscellaneous Details

Having presented the basic CSFQ algorithm, we now return to discuss a few aspects in more detail.

We have used exponential averaging to estimate the arrival rate in Eq. (3). However, instead of using a constant exponential weight we used $e^{-T/K}$ where T is the interpacket arrival time and K is a constant. Our motivation was that $e^{-T/K}$ more closely reflects a fluid averaging process which is independent of the packetizing structure. More specifically, it can be shown that if a constant weight is used, the estimated rate will be sensitive to the packet length distribution and there are pathological cases where the estimated rate differs from the real arrival rate by a factor; this would allow flows to exploit the estimation process and obtain more than their fair share. In contrast, by using a parameter of $e^{-T/K}$, the estimated rate will asymptotically converge to the real rate, and this allows us to bound the excess service that can be achieved (as in Theorem 1). We used a similar averaging process in Eq. (5) to estimate the total arrival rate A.

The choice of K in the above expression $e^{-T/K}$ presents us with several tradeoffs. First, while a smaller K increases the system responsiveness to rapid rate fluctuations, a larger K better filters the noise and avoids potential system instability. Second, K should be large enough such that the estimated rate, calculated at the edge of the network, remains reasonably accurate after a packet traverses multiple links. This is because the delay-jitter changes the packets' interarrival pattern, which may result in an increased discrepancy between the estimated rate (received in the packets' labels) and the real rate. To counteract this effect, as a rule of thumb, K should be one order of magnitude larger that the delay-jitter experienced by a flow over a time interval of the same size, K. Third, K should be no larger than the average duration of a flow. Based on this constraints, an appropriate value for K would be between 100 and 500 ms.

A second issue relates to the requirement of CSFQ for a label to be carried in each packet. One possibility is to use the Type Of Service byte in the IP header. For example, by using a floating point representation with four bits for mantissa and four bits for exponent we can represents any rate between 1 Kbps and 65 Mbps with an accuracy of 6.25%. Another possibility is to define an IP option in the case of IPv4, or a hop-by-hop extension header in the case of IPv6.

3 Simulations

In this section we evaluate our algorithm by simulation. To provide some context, we compare CSFQ's performance to four additional algorithms. Two of these, FIFO and RED, represent baseline cases where routers do not attempt to achieve fair bandwidth allocations. The other two algorithms, FRED and DRR, represent different approaches to achieving fairness.

- FIFO (First In First Out) - Packets are served in a first-in first-out order, and the buffers are managed using a simple drop-tail strategy; *i.e.*, incoming packets are dropped when the buffer is full.
- RED (Random Early Detection) - Packets are served in a first-in first-out order, but the buffer management is significantly more sophisticated than drop-tail. RED [9] starts to probabilistically drop packets long

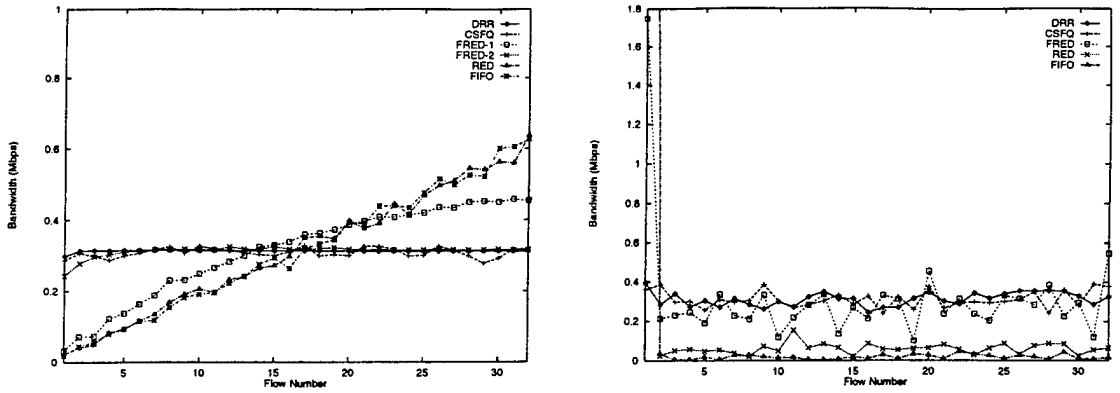


Figure 3: Simulation results for a 10 Mbps link shared by N flows. (a) The average throughput over 10 sec when N = 32, and all flows are UDPs. The arrival rate for flow i is (i + 1) times larger than its fair share. The flows are indexed from 0. (b) The throughputs of one UDP flow (indexed 0) sending at 10 Mbps, and of 31 TCP flows sharing a 10 Mbps link.

before the buffer is full, providing early congestion indication to flows which can then gracefully back-off before the buffer overflows. RED maintains two buffer thresholds. When the exponentially averaged buffer occupancy is smaller than the first threshold, no packet is dropped, and when the exponentially averaged buffer occupancy is larger than the second threshold all packets are dropped. When the exponentially averaged buffer occupancy is between the two thresholds, the packet dropping probability increases linearly with buffer occupancy.

- FRED (Flow Random Early Drop) - This algorithm extends RED to provide some degree of fair bandwidth allocation [14]. To achieve fairness, FRED maintains state for all flows that have at least one packet in the buffer. Unlike RED where the dropping decision is based only on the buffer state, in FRED dropping decisions are based on this flow state. Specifically, FRED preferentially drops a packet of a flow that has either (1) had many packets dropped in the past, or (2) a queue larger than the average queue size. FRED has two variants, which we will call FRED-1 and FRED-2. The main difference between the two is that FRED-2 guarantees to each flow a minimum number of buffers. As a general rule, FRED-2 performs better than FRED-1 only when the number of flows is large. In the following data, when we do not distinguish between the two, we are quoting the results from the version of FRED which performed better.

- DRR (Deficit Round Robin) - This algorithm [20] represents an efficient implementation of the well-known weighted fair queueing (WFQ) discipline. The buffer management scheme assumes that when the buffer is full the packet from the longest queue is dropped. DRR is the only one of the four to use a sophisticated per-flow queueing algorithm, and thus achieves the highest degree of fairness.

These four algorithms represent four different levels of complexity. DRR and FRED have to classify incoming flows, whereas FIFO and RED do not. DRR in addition has to implement its packet scheduling algorithm, whereas the rest all use first-in-first-out scheduling. CSFQ edge routers have complexity comparable to FRED, and CSFQ core routers have complexity comparable to RED.

We have examined the behavior of CSFQ under a variety of conditions. We use an assortment of traffic sources (mainly TCP sources and constant bit rate UDP sources,[4] but also some on-off sources) and topologies. Due to space limitations, we only report on a small sampling of the simulations we have run.[5] All simulations were performed in ns-2 [17], which provide accurate packet-level implementation for various network protocols, such as TCP and RLM [15] (Receiver-driven Layered Multicast), and various buffer management and scheduling algorithms, such as RED and DRR. All algorithms used in the simulation, except CSFQ and FRED, were part of the standard ns-2 distribution.

Unless otherwise specified, we use the following parameters for the simulations in this section. Each output link has a capacity of 10 Mbps, a latency of 1 ms, and a buffer of 64 KB. In the RED and FRED cases the first threshold is set to 16 KB, while the second one is set to 32 KB. The averaging constants K (used in estimating the flow rate), K_α (used in estimating the fair rate), and K_c (used in making the decision of whether a link is congested or not) are all set to 100 ms unless specified otherwise. The general rule of thumb we follow in this paper is to choose these constants to be roughly two times larger than the maximum queueing delay (*i.e.*, 64KB/10Mbps = 51.2 ms).[6] Finally, in all topologies the first router on the path of each flow is always assumed to be an edge router; all other routers are assumed without exception to be core routers.

We simulated the other four algorithms to give us benchmarks against which to assess these results. We use DRR as our model of fairness and use the baseline cases, FIFO and

[4] This source, referred to as UDP in the remainder of the paper, has fixed size packets and the packet interarrival times are uniformly distributed between $[0.5 \times avg, 1.5 \times avg)$, where avg is the average interarrival time.

[5] A fuller set of tests, and the scripts used to run them, is available at `http://www.cs.cmu.edu/~istoica/csfq`

[6] It can be shown that by using this rule an idle link that becomes suddenly congested by a set of identical UDP sources will not experience buffer overflow *before* the algorithm detects the congestion, as long as the aggregate arrival rate is less than 10 times the link capacity (see [22]).

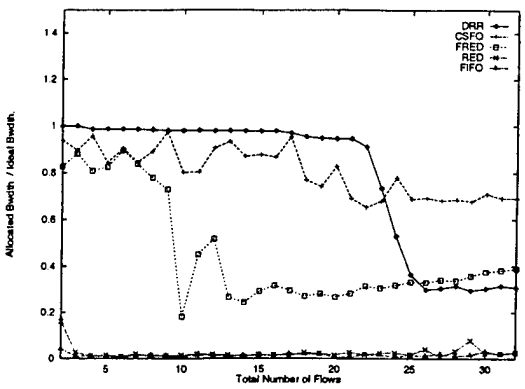


Figure 4: The normalized bandwidth of a TCP flow that competes with $N-1$ UDP flows sending at twice their allocated rates, as a function of N.

RED, as representing the (unfair) status quo. The goal of these experiments is to determine where CSFQ sits between these two extremes. FRED is a more ambiguous benchmark, being somewhat more complex than CSFQ but not as complex as DRR.

In general, we find that CSFQ achieves a reasonable degree of fairness, significantly closer to DRR than to FIFO or RED. CSFQ's performance is typically comparable to FRED's, although there are several situations where CSFQ significantly outperforms FRED. There are a large number of experiments and each experiment involves rather complex dynamics. Due to space limitations, in the sections that follow we will merely highlight a few important points and omit detailed explanations of the dynamics.

3.1 A Single Congested Link

We first consider a single 10 Mbps congested link shared by N flows. The propagation delay along the link is 1 ms. We performed three related experiments.

In the first experiment, we have 32 UDP flows, indexed from 0, where flow i sends $i+1$ times more than its fair share of 0.3125 Mbps. Thus flow 0 sends 0.3125 Mbps, flow 1 sends 0.625 Mbps, and so on. Figure 3(a) shows the average throughput of each flow over a 10 sec interval; FIFO, RED, and FRED-1 fail to ensure fairness, with each flow getting a share proportional to its incoming rate, while DRR is extremely effective in achieving a fair bandwidth distribution. CSFQ and FRED-2 achieve a less precise degree of fairness; for CSFQ the throughputs of all flows are between -11% and $+5\%$ of the ideal value.

In the second experiment we consider the impact of an ill-behaved UDP flow on a set of TCP flows. More precisely, the traffic of flow 0 comes from a UDP source that sends at 10 Mbps, while all the other flows (from 1 to 31) are TCPs. Figure 3(b) shows the throughput of each flow averaged over a 10 sec interval. The only two algorithms that can most effectively contain the UDP flow are DRR and CSFQ. Under FRED the UDP flow gets almost 1.8 Mbps – close to six times more than its fair share – while the UDP only gets 0.396 Mbps and 0.361 Mbps under DRR and CSFQ, respectively. As expected FIFO and RED perform poorly, with the UDP flow getting over 8 Mbps in both cases.

In the final experiment, we measure how well the algorithms can protect a single TCP flow against multiple

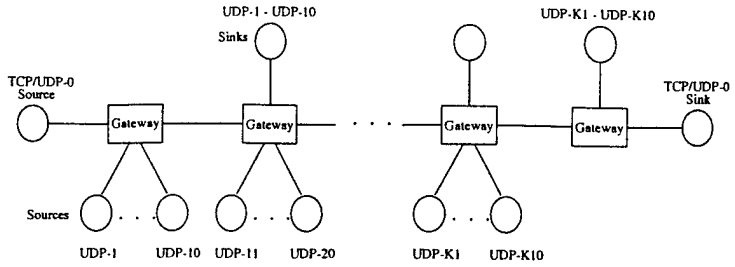


Figure 5: Topology for analyzing the effects of multiple congested links on the throughput of a flow. Each link has ten cross flows (all UDPs). All links have 10 Mbps capacities. The sending rates of all UDPs, excepting UDP-0, are 2 Mbps, which leads to all links between routers being congested.

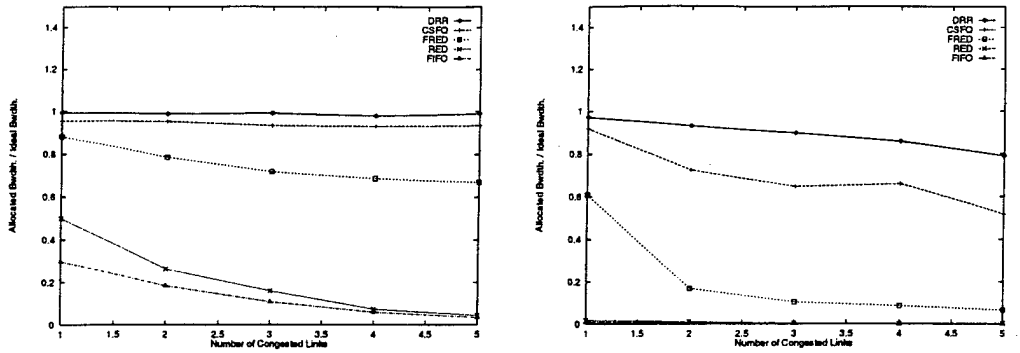
ill-behaved flows. We perform 31 simulations, each for a different value of N, $N = 2 \cdots 32$. In each simulation we take one TCP flow and $N-1$ UDP flows; each UDP sends at twice its fair share rate of $\frac{10}{N}$Mbps. Figure 4 plots the ratio between the average throughput of the TCP flow over 10 seconds and the fair share bandwidth it should receive as a function of the total number of flows in the system N. There are three points of interest. First, DRR performs very well when there are less than 22 flows, but its performances decreases afterwards. This is because the TCP flow's buffer share is less than three buffers, which significantly affects its throughput. Second, CSFQ performs better than DRR when the number of flows is large. This is because CSFQ is able to cope better with the TCP burstiness by allowing the TCP flow to have several packets buffered for short time intervals. Finally, across the entire range, CSFQ provides similar or better performance as compared to FRED.

3.2 Multiple Congested Links

We now analyze how the throughput of a well-behaved flow is affected when the flow traverses more than one congested link. We performed two experiments based on the topology shown in Figure 5. All UDPs, except UDP-0, send at 2 Mbps. Since each link in the system has 10 Mbps capacity, this will result in all links between routers being congested.

In the first experiment, we have a UDP flow (denoted UDP-0) sending at its fair share rate of 0.909 Mbps. Figure 6(a) shows the fraction of UDP-0's traffic that is forwarded versus the number of congested links. CSFQ and FRED perform reasonably well, although not quite as well as DRR.

In the second experiment we replace UDP-0 with a TCP flow. Similarly, Figure 6(b) plots the normalized TCP throughput against the number of congested links. Again, DRR and CSFQ prove to be effective. In comparison, FRED performs significantly worse though still much better than RED and FIFO. The reason is that while DRR and CSFQ try to allocate bandwidth fairly among competing flows during congestion, FRED tries to allocate buffers fairly. Flows with different end-to-end congestion control algorithms will achieve different throughputs even if routers try to fairly allocate buffer.



(a) (b)

Figure 6: (a) The normalized throughput of UDP-0 as a function of the number of congested links. (b) The same plot when UDP-0 is replaced by a TCP flow.

Algorithm	delivered	dropped
DRR	601	6157
CSFQ	1680	5078
FRED	1714	5044
RED	5322	1436
FIFO	5452	1306

Table 1: Statistics for an ON-OFF flow with 19 competing TCPs flows (all numbers are in packets).

Algorithm	mean time	std. dev
DRR	25	99
CSFQ	62	142
FRED	40	174
RED	592	1274
FIFO	840	1695

Table 2: The mean transfer times (in ms) and the corresponding standard deviations for 60 short TCPs in the presence of a UDP flow that sends at the link capacity, *i.e.*, 10 Mbps.

3.3 Coexistence of Different Adaptation Schemes

In this experiment we investigate the extent to which CSFQ can deal with flows that employ different adaptation schemes. Receiver-driven Layered Multicast (RLM) [15] is an adaptive scheme in which the source sends the information encoded into a number of layers (each to its own multicast group) and the receiver joins or leaves the groups associated with the layers based on how many packet drops it is experiencing. We consider a 4 Mbps link traversed by one TCP and three RLM flows. Each source uses a seven layer encoding, where layer i sends 2^{i+4} Kbps; each layer is modeled by a UDP traffic source. The fair share of each flow is 1Mbps. In the RLM case this will correspond to each receiver subscribing to the first five layers[7].

The receiving rates averaged over 1 second interval for each algorithm are plotted in Figure 7. Since in this experiment the link bandwidth is 4 Mbps and the router buffer size

[7]More precisely, we have $\sum_{i=1}^{5} 2^{i+4}$ Kbps = 0.992 Mbps.

Algorithm	mean	std. dev
DRR	6080	64
CSFQ	5761	220
FRED	4974	190
RED	628	80
FIFO	378	69

Table 3: The mean throughputs (in packets) and standard deviations for 19 TCPs in the presence of a UDP flow along a link with propagation delay of 100 ms. The UDP sends at the link capacity of 10 Mbps.

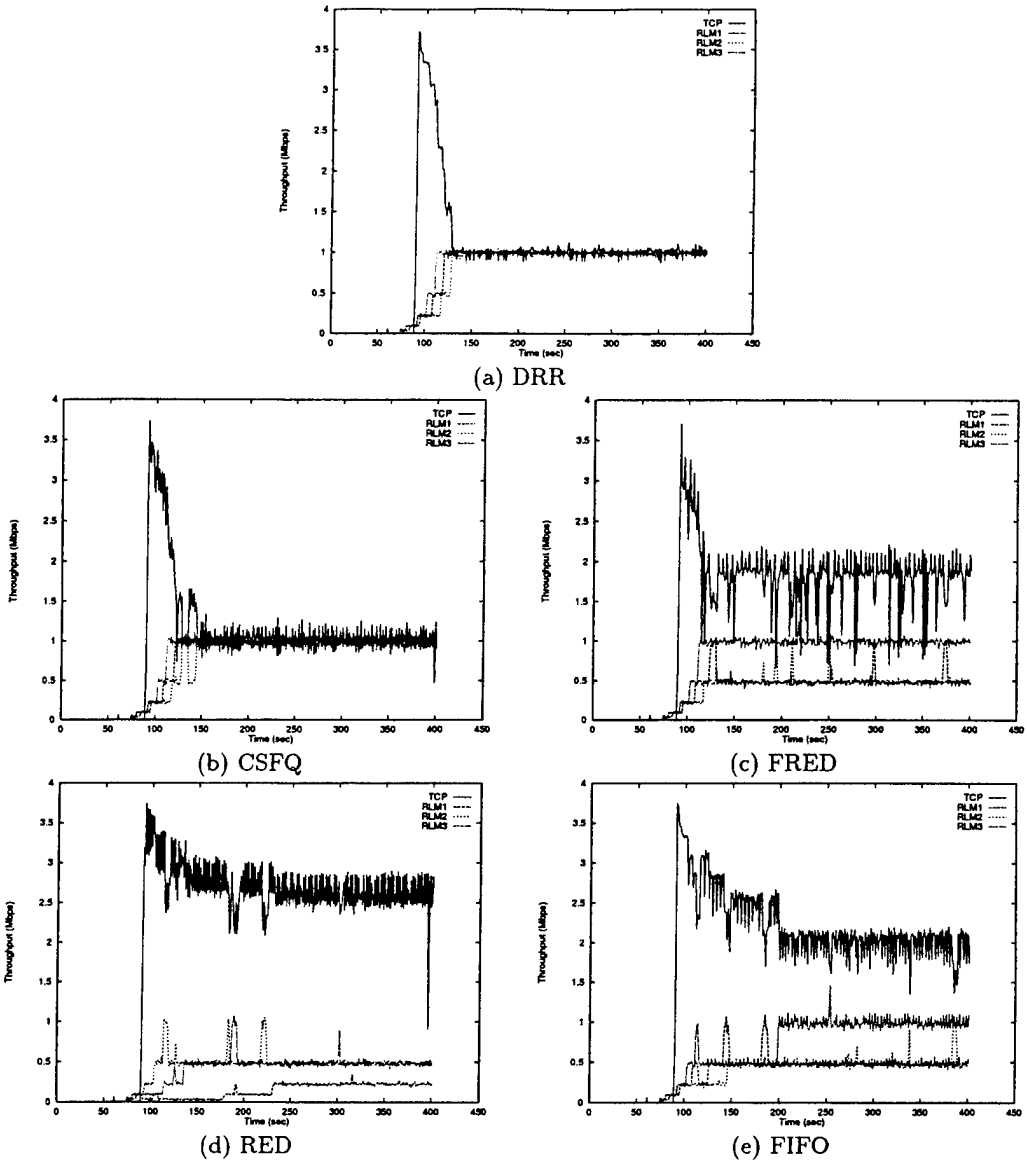
is 64 KB, we set constants K, K_α, and K_c to be 250 ms, i.e., about two times larger than the maximum queue delay. An interesting point to notice is that, unlike DRR and CSFQ, FRED does not provide fair bandwidth allocation in this scenario. Again, as discussed in Section 3.2, this is due to the fact that RLM and TCP use different end-to-end congestion control algorithms.

3.4 Different Traffic Models

So far we have only considered UDP, TCP and layered multicast traffic sources. We now look at two additional source models with greater degrees of burstiness. We again consider a single 10 Mbps congested link. In the first experiment, this link is shared by one ON-OFF source and 19 TCPs. The ON and OFF periods of the ON-OFF source are both drawn from exponential distributions with means of 100 ms and 1900 ms respectively. During the ON period the ON-OFF source sends at 10 Mbps. Note that the ON-time is on the same order as the averaging intervals K, K_α, and K_c which are all 100 ms, so this experiment is designed to test to what extent CSFQ can react over short timescales.

The ON-OFF source sent 6758 packets over the course of the experiment. Table 1 shows the number of packets from the ON-OFF source dropped at the congested link. The DRR results show what happens when the ON-OFF source is restricted to its fair share at all times. FRED and CSFQ also are able to achieve a high degree of fairness.

Our next experiment simulates Web traffic. There are 60 TCP transfers whose inter-arrival times are exponentially distributed with the mean of 0.05 ms, and the length of each



(a) DRR

(b) CSFQ

(c) FRED

(d) RED

(e) FIFO

Figure 7: The throughput of three RLM flows and one TCP flow along a 4 Mbps link .

transfer is drawn from a Pareto distribution with a mean of 20 packets (1 packet = 1 KB) and a shaping parameter of 1.06. These values are consistent with those presented in the [5]. In addition, there is a single 10 Mbps UDP flow.

Table 2 presents the mean transfer time and the corresponding standard deviations. Here, CSFQ performs worse than FRED, mainly because it has a larger average queue size, but still almost one order of magnitude better than FIFO and RED.

3.5 Large Latency

All of our experiments so far have had small link delays (1 ms). In this experiment we again consider a single 10 Mbps congested link, but now with a propagation delay of 100 ms. The load is comprised of one UDP flow that sends at the link speed and 19 TCP flows. Due to the large propagation delay, in this experiment we set the buffer size to be 256 KB, and K, K_α, and K_c to be 400 ms. Table 3 shows the average number of packets of a TCP flow during a 100 seconds interval. Both CSFQ and FRED perform reasonably well.

3.6 Packet Relabeling

Recall that when the dropping probability of a packet is non-zero we relabel it with the fair rate α so that the label of the packet will reflect the new rate of the flow. To test how well this works in practice, we consider the topology in Figure 8, where each link is 10 Mbps. Note that as long as all three flows attempt to use their full fair share, the fair shares of flows 1 and 2 are less on link 2 (3.33 Mbps) than on link 1 (5 Mbps), so there will be dropping on both links. This will test the relabelling function to make sure that the incoming rates are accurately reflected on the second link. We perform two experiments (only looking at CSFQ's performance). In the first, there are three UDPs sending data

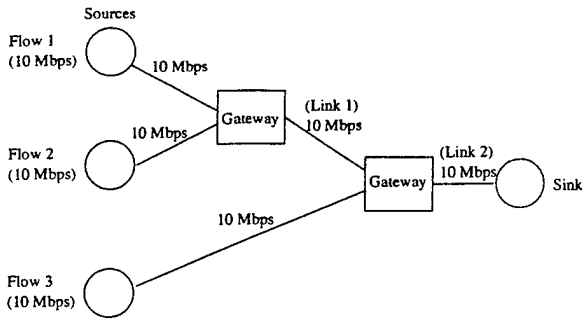


Figure 8: Simulation scenario for the packet relabeling experiment. Each link has 10 Mbps capacity, and a propagation delay of 1 ms.

Traffic	Flow 1	Flow 2	Flow 3
UDP	3.36	3.32	3.28
TCP	3.43	3.13	3.43

Table 4: The throughputs resulting from CSFQ averaged over 10 seconds for the three flows in Figure 8 along link 2.

at 10 Mbps each. Table 4 shows the average throughputs over 10 sec of the three UDP flows. As expected these rates are closed to 3.33 Mbps. In the second experiment, we replace the three UDPs by three TCPs. Again, despite the TCP burstiness which may negatively affect the rate estimation and relabeling accuracy, each TCP gets close to its fair share.

3.7 Discussion of Simulation Results

We have tested CSFQ under a wide range of conditions, conditions purposely designed to stress its ability to achieve fair allocations. These tests, and the others we have run but cannot show here because of space limitations, suggest that CSFQ achieves a reasonable approximation of fair bandwidth allocations in most conditions. Certainly CSFQ is far superior in this regard to the status quo (FIFO or RED). Moreover, in all situations CSFQ is roughly comparable with FRED, and in some cases it achieves significantly fairer allocations. Recall that FRED requires per-packet flow classification while CSFQ does not, so we are achieving these levels of fairness in a more scalable manner. However, there is clearly room for improvement in CSFQ, as there are cases where its performance is significantly below that of its benchmark, DRR. We do not yet know if these are due to our particular choices for the estimation algorithms, or are inherent properties of the CSFQ architecture.

4 Why Are Fair Allocations Important?

In the Introduction we stated that one of the underlying assumptions of this work is that fairly allocating bandwidth was beneficial, and perhaps even crucial, for congestion control. In this section we motivate the role of fair allocations in congestion control by discussing the problem of unfriendly flows, and then presenting two approaches to this problem; we end this section with a discussion of the role of punishment. In what follows we borrow heavily from [7], [3], and [8], and have benefited greatly from conversations with Steve Deering and Sally Floyd. We should note that the matters addressed in this section are rather controversial and this overview unavoidably reflects our prejudices. This section, however, is merely intended to provide some perspective on our motivation for this work, and any biases in this overview should not undercut the technical aspects of the CSFQ proposal that are the main focus of the previous sections.

4.1 The Unfriendly Flow Problem

Data networks such as the Internet, because of their reliance on statistical multiplexing, must provide some mechanism to control congestion. The current Internet, which has mostly FIFO queueing and drop-tail mechanisms in its routers, relies on end-to-end congestion control in which hosts curtail their transmission rates when they detect that the network is congested. The most widely utilized form of end-to-end congestion control is that embodied in TCP [11], which has been tremendously successful in preventing congestion collapse.

The efficacy of this approach depends on two fundamental assumptions: (1) all (or almost all) flows are *cooperative* in that they implement congestion control algorithms, and (2) these algorithms are *homogeneous* – or roughly equivalent – in that they produce similar bandwidth allocations if used in similar circumstances. In particular, assumption (2) requires, in the language of [8], that all flows are TCP-friendly.[8]

The assumption of universal cooperation can be violated in three general ways. First, some applications are *unresponsive* in that they don't implement any congestion control algorithms at all. Most of the early multimedia and multicast applications, like *vat*, *nv*, *vic*, *wb* and RealAudio fall into this category. Second, some applications use congestion control algorithms that, while responsive, are not TCP-friendly. RLM is such an algorithm.[9] Third, some users will cheat and use a non-TCP congestion control algorithm to get more bandwidth. An example of this would be using a modified form of TCP with, for instance, a larger initial window and window opening constants.

Each of these forms of noncooperation can have a significant negative impact on the performance obtained by cooperating flows. At present, we do not yet know how widespread noncooperation will be, and thus cannot assess the level of harm it will cause. However, in lieu of more solid evidence that noncooperation will not be a problem, it seems unsound to base the Internet's congestion control paradigm on the assumption of universal cooperation. We therefore started this paper with the fundamental assumption that one needs to deal with the problem of unfriendly flows.

[8] Actually, the term TCP-friendly in [8] means that "their arrival rate does not exceed that of any TCP connection in the same circumstances." Here we use it to mean that the arrival rates are roughly comparable, a property that should be more precisely called *TCP-equivalent*. We blur the distinction between TCP-friendly and TCP-equivalent to avoid an overly unwieldy set of terms in this short overview. However, we think the distinction may be rendered moot since it is unlikely that congestion control algorithms that are not TCP-equivalent but are TCP-friendly – *i.e.*, they get much less than their fare share – will be widely deployed.

[9] Although our data in Section 3.3 showed RLM receiving less than its fair share, when we change the simulation scenario so that the TCP flow starts after all the RLM flows then it receives less than half of its fair share. This hysteresis in the RLM versus TCP behavior was first pointed out to us by Steve McCanne [15].

4.2 Two Approaches

There are, in the literature, two general approaches to addressing the problem of unfriendly flows. The first is the allocation approach. Here, the router itself ensures that bandwidth is allocated fairly, isolating flows from each other so that unfriendly flows can only have a very limited impact on other flows. Thus, the allocation approach need not demand that all flows adopt some universally standard end-to-end congestion control algorithm; flows can choose to respond to the congestion in whatever manner best suits them without unduly harming other flows. Assuming that flows prefer to not have significant levels of packet drops, these allocation approaches give an incentive for flows to use end-to-end congestion control, because being unresponsive hurts their own performance. Note that the allocation approach does not provide an incentive for flows to be TCP-friendly (an example of an alternative end-to-end congestion control algorithm is described in [13]), but does provide strong incentives for drop-intolerant applications to use some form of end-to-end congestion control.[10] Of course, the canonical implementations of the allocation approach, such as Fair Queueing, all require significant complexity in routers. Our goal in this paper was to present a more scalable realization of the allocation approach.

The problem of unfriendly flows can be addressed in another manner. In the *identification* approach, as best exemplified by [8], routers use a lightweight detection algorithm to identify unfriendly flows, and then explicitly manage the bandwidth of these unfriendly flows. This bandwidth management can range from merely restricting unfriendly flows to no more than the currently highest friendly flow's share[11] to the extreme of severely punishing unfriendly flows by dropping all of their packets.

This approach relies on the ability to accurately identify unfriendly flows with relatively lightweight router mechanisms. This is a daunting task. Below we discuss the process of identifying unfriendly flows, and then present simulation results of the identification algorithm in [8]; we are not aware of other realizations of the identification approach.

One can think of the process of identifying unfriendly flows as occurring in two logically distinct stages. The first, and relatively easy, step is to estimate the arrival rate of a flow. The second, and harder, step is to use this arrival rate information (along with the dropping rate and other aggregate measurements) to decide if the flow is unfriendly. Assuming that friendly flows use a TCP-like adjustment method of increase-by-one and decrease-by-half, one can derive an expression (see [8] for details) for the bandwidth share S as a function of the dropping rate p, round-trip time R, and packet size B: $S \approx \frac{\gamma B}{R\sqrt{p}}$ for some constant γ. Routers do not know the round trip time R of flows, so must use the lower bound of double the propagation delay of the attached link; this allows flows further away from the link to behave more aggressively without being identified as being unfriendly.[12]

[10] As we discuss later, if flows can tolerate significant levels of loss, the situation changes somewhat.

[11] If identification were perfect, and this management goal achieved, all flows would get their max-min fair allocations. However, we are not aware of any algorithm that can achieve this management goal.

[12] We are not delving into some of the details of the approach layed out in [8] where flows can also be classified as very-high-bandwidth but not necessarily unfriendly, and as unresponsive (and therefore unfriendly).

Algorithm	Simulation 1			Simulation 2	
	UDP	TCP-1	TCP-2	TCP-1	TCP-2
REDI	0.906	0.280	0.278	0.565	0.891
CSFQ	0.554	0.468	0.478	0.729	0.747

Table 5: (Simulation 1) The throughputs in Mbps of one UDP and two TCP flows along a 1.5 Mbps link under REDI [8], and CSFQ, respectively. (Simulation 2) The throughputs of two TCPs (where TCP-2 opens its congestion window three times faster than TCP-1), under REDI, and CSFQ, respectively.

To see how this occurs in practice, consider the following two experiments using the identification algorithm described in [8], which we call RED with Identification (REDI).[13] In each case there are multiple flows traversing a 1.5 Mbps link with a latency of 3 ms; the output buffer size is 32 KB and all constants K, K_α, and K_c, respectively, are set to 400 ms. Table 5 shows the bandwidth allocations under REDI and CSFQ averaged over 100 sec. In the first experiment (Simulation 1), we consider a 1 Mbps UDP flow and two TCP flows; in the second experiment (Simulation 2) we have a standard TCP (TCP-1) and a modified TCP (TCP-2) that opens the congestion window three times faster. In both cases REDI fails to identify the unfriendly flow, allowing it to obtain almost two-thirds of the bandwidth. As we increase the latency of the congested link, REDI starts to identify unfriendly flows. However, for some values as high as 18 ms, it still fails to identify such flows. Thus, the identification approach still awaits a viable realization and, as of now, the allocation approach is the only demonstrated method to deal with the problem of unfriendly flows.

4.3 Punishment

Earlier in this section we argued that the allocation approach gave drop-intolerant flows an incentive to adopt end-to-end congestion control. What about drop-tolerant flows?

We consider, for illustration, *fire-hose* applications that have complete drop-tolerance: they send at some high rate ρ and get as much value out of the fraction of arriving packets, call it x, as if they originally just sent a stream of rate $x\rho$. That is, these fire-hose applications care only about the ultimate throughput rate, not the dropping rate.[14] In a completely static world where bandwidth shares were constant such "fire-hose" protocols would not provide any advantage over just sending at the fair share rate. However, if the fair shares along the path were fluctuating significantly, then fire-hose protocols might better utilize instantaneous fluctuations in the available bandwidth. Moreover, fire-hose protocols relieve applications of the burden of trying to adapt to their fair share. Thus, even when restrained to their fair share there is some incentive for flows to send at significantly more than the current fair share.[15] In addition, such

[13] We are grateful to Sally Floyd who provided us her script implementing the REDI algorithm. We used a similar script in our simulation, with the understanding that this is a preliminary design of the identification algorithm. Our contention is that the design of such an identification algorithm is fundamentally difficult due to the uncertainty of RTT.

[14] Approximations to complete drop-tolerance can be reached in video transport using certain coding schemes or file transport using selective acknowledgements.

[15] These fire-hose coding and file transfer methods also have some

fire-hoses decrease the bandwidth available to other flows because packets destined to be dropped at a congested link represent an unnecessary load on upstream links. With universal deployment of the allocation approach, every other flow would still obtain their fair share at each link, but that share may be smaller than it would have been if the fire-hose had been using responsive end-to-end congestion control. It is impossible to know now whether this will become a serious problem. Certainly, though, the problem of fire-hoses in a world with fair bandwidth allocation is far less dire than the problem of unfriendly flows in our current FIFO Internet, since the incentive to be unfriendly and the harmful impact on others are considerably greater in the latter case. As a consequence, our paper emphasizes the problem of unfriendly flows in our current FIFO Internet, and is less concerned with fire-hose flows in an Internet with fair bandwidth allocation.

Nonetheless, the fire-hose problem should not be ignored; flows should be given an incentive to adopt responsive end-to-end congestion. One possible method is to explicitly punish unresponsive flows by denying them their fair share.[16] Punishment is discussed as one possible bandwidth management approach in [8] (the approach described there is informally referred to as RED-with-a-penalty-box). Accurately identifying flows as unresponsive may be far easier than identifying them as unfriendly. However, as we saw in our simulations, doing so in the context of the identification approach is far from a solved problem; the challenge is to determine if a flow has decreased usage in response to increases in overall packet drop rates [8].

Identifying unresponsive flows is more straightforward in the allocation approach, since here one need only determine if a flow has had significantly high drop rates over a long period of time. As a proof of concept we have implemented a simple identification and punishment mechanism. First, we examine off-line the last n dropped packets and then monitor the flows with the most dropped packets. Second, we estimate the rate of each of these monitored flows; when a flow's rate is larger than $a \times \alpha$ ($a > 1$), we start dropping all of its packets. Third, we continue to monitor penalized flows, continuing punishment until their arrival rate decreases below $b \times \alpha$ ($b < 1$). Using the parameters $a = 1.2$, $b = 0.6$, and $n = 100$, we applied this algorithm to Simulation 1 in Table 5; the UDP flow was identified and penalized in less than 3 seconds. Our task was easy because the identification of unresponsive flows can be based on the result (packet drops over long periods of time) rather than on trying to examine the algorithm (detecting whether it actually decreased its rate in response to an increase in the drop rate). Note also that the allocation approach need only distinguish between responsive and unresponsive in the punishment phase, an inherently easier task than distinguishing friendly from unfriendly.

In summary, to provide incentives for drop-tolerant flows to use responsive end-to-end congestion control, it may be necessary to identify, and then punish, unresponsive flows. CSFQ with this punishment extension may be seen as a marriage of the allocation and identification approaches; the difference between [8] and our approach is largely one of the relative importance of identification and allocation. We start with allocation as fundamental, and then do identification only when necessary; [8] starts with identification, and then considers allocation only in the context of managing the bandwidth of identified flows.

overhead associated with them, and it isn't clear whether, in practice, the overheads are greater or less than the advantages gained. However, one can certainly not claim, as we did above for drop-intolerant applications, that the allocation approach gives drop-tolerant applications a strong incentive to use responsive end-to-end congestion control algorithms.

[16] Another possible method, used in ATM ABR, is to have network provide explicit per flow feedback to ingress nodes and have edge nodes police the traffic on a per flow basis. We assume this is a too heavyweight a mechanism for the Internet.

5 Summary

This paper presents an architecture for achieving reasonably fair bandwidth allocations while not requiring per-flow state in core routers. Edge routers estimate flow rates and insert them into the packet labels. Core routers merely perform probabilistic dropping on input based on these labels and an estimate of the fair share rate, the computation of which requires only aggregate measurements. Packet labels are rewritten by the core routers to reflect output rates, so this approach can handle multihop situations.

We tested CSFQ, and several other algorithms, on a wide variety of conditions. We find that CSFQ achieve a significant degree of fairness in all of these circumstances. While not matching the fairness benchmark of DRR, it is comparable or superior to FRED, and vastly better than the baseline cases of RED and FIFO. We know of no other approach that can achieve comparable levels of fairness without any per-flow operations in the core routers.

The main thrust of CSFQ is to use rate estimation at the edge routers and packet labels to carry rate estimates to core routers. The details of our proposal, such as the estimation algorithms, are still very much the subject of active research. However, the results of our initial experiments with a rather untuned algorithm are quite encouraging.

One open question is the effect of large latencies. The logical extreme of the CSFQ approach would be to do rate estimation at the entrance to the network (at the customer/ISP boundary), and then consider everything else the core. This introduces significant latencies between the point of estimation and the points of congestion; while our initial simulations with large latencies did not reveal any significant problems, we do not yet understand CSFQ well enough to be confident in the viability of this "all-core" design. However, if viable, this "all-core" design would allow all interior routers to have only very simple forwarding and dropping mechanisms, without any need to classify packets into flows.

In addition, we should note that it is possible to use a CSFQ-like architecture to provide service guarantees. A possible approach would be to use the route pinning mechanisms described in [23], and to shape the aggregate guaranteed traffic at each output link of core routers [6].

One of the initial assumptions of this paper was that the more traditional mechanisms used to achieve fair allocations, such as Fair Queueing or FRED, were too complex to implement cost-effectively at sufficiently high speeds. If this is the case, then a more scalable approach like CSFQ is necessary to achieve fair allocations. The CSFQ islands would be comprised of high-speed backbones, and the edge routers would be at lower speeds where classification and other per-flow operations were not a problem. However, CSFQ may still play a role even if router technology advances to the stage where the more traditional mechanisms can reach sufficiently high speeds. Because the core-version of CSFQ could presumably be retrofit on a sizable fraction

of the installed router base (since its complexity is roughly comparable to RED and can be implemented in software), it may be that CSFQ islands are not high-speed backbones but rather are comprised of legacy routers.

Lastly, we should note that the CSFQ approach requires some configuration, with edge routers distinguished from core routers. Moreover, CSFQ must be adopted an island at a time rather than router-by-router. We do not know if this presents a serious impediment to CSFQ's adoption.

References

[1] J.C.R. Bennett, D.C. Stephens, and H. Zhang. High speed, scalable, and accurate implementation of packet fair queueing algorithms in ATM networks. In *Proceedings of IEEE ICNP '97*, pages 7–14, Atlanta, GA, October 1997.

[2] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM'96*, pages 120–128, San Francisco, CA, March 1996.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. K. Ramakrishnan, S. Shenker, and J. Wroclawski. Recommendations on queue management and congestion avoidance in the internet, January 1998. Internet Draft.

[4] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. A Framework for Multiprotocol Label Switching, November 1997. Internet Draft.

[5] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic evidence and possible causes. In *Proceedings of the ACM SIGMETRICS 96*, pages 160–169, Philadelphia, PA, May 1996.

[6] R. L. Cruz. SCED+: Efficient Management of Quality of Service Guarantees. In *Proceedings of INFOCOM'98*, pages 625–642, San Francisco, CA, 1998.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[8] S. Floyd and K. Fall. Router mechanisms to support end-to-end congestion control, February 1997. LBL Technical Report.

[9] S. Floyd and V. Jacobson. Random early detection for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, July 1993.

[10] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, CA, June 1994.

[11] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM'88*, pages 314–329, August 1988.

[12] J. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 7(29):954–962, July 1980.

[13] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of ACM SIGCOMM'91*, pages 3–15, Zurich, Switzerland, September 1991.

[14] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM '97*, pages 127–137, Cannes, France, October 1997.

[15] S. McCanne. *Scalable Compression and Transmission of Internet Multicast Video*. PhD dissertation, University of California Berkeley, December 1996.

[16] J Nagle. On packet switches with infinite storage. *IEEE Trans. On Communications*, 35(4):435–438, April 1987.

[17] Ucb/lbnl/vint network simulator - ns (version 2).

[18] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proceedings of the INFOCOM'92*, 1992.

[19] S. Shenker. Making greed work in networks: A game theoretical analysis of switch service disciplines. In *Proceedings of ACM SIGCOMM'94*, pages 47–57, London, UK, August 1994.

[20] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of SIGCOMM'95*, pages 231–243, Boston, MA, September 1995.

[21] D. Stilliadis and A. Varma. Efficient fair queueing algorithms for packet-switched networks. *IEEE/ACM Transactions on Networking*, 6(2):175–185, April 1998.

[22] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair banwidth allocations in high speed nteworks, June 1998. Technical Report CMU-CS-98-136, Carnegie Mellon University.

[23] I. Stoica and H. Zhang. LIRA: A model for service differentiation in the internet. In *Proceedings of NOSSDAV'98*, London, UK, July 1998.

[24] Z. Wang. User-share differentiation (USD) scalable bandwidth allocation for differentiated services, May 1998. Internet Draft.

A Scalable Web Cache Consistency Architecture*

Haobo Yu
USC/Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina del Rey, CA 90034
haoboy@isi.edu

Lee Breslau
AT&T Labs-Research
75 Willow Road
Menlo Park, CA 94025
breslau@research.att.com

Scott Shenker
International Computer Science Institute
1947 Center Street
Berkeley, CA 94704
shenker@icsi.berkeley.edu

Abstract

The rapid increase in web usage has led to dramatically increased loads on the network infrastructure and on individual web servers. To ameliorate these mounting burdens, there has been much recent interest in web caching architectures and algorithms. Web caching reduces network load, server load, and the latency of responses. However, web caching has the disadvantage that the pages returned to clients by caches may be *stale*, in that they may not be consistent with the version currently on the server. In this paper we describe a scalable web cache consistency architecture that provides fairly tight bounds on the staleness of pages. Our architecture borrows heavily from the literature, and can best be described as an *invalidation* approach made scalable by using a caching hierarchy and application-level multicast routing to convey the invalidations. We evaluate this design with calculations and simulations, and compare it to several other approaches.

1 Introduction

The world-wide-web has become an important component of the global information infrastructure. The rapid increase of web usage has imposed a heavy load on the network and server infrastructure, and significant delays are not uncommon. To mitigate the effects of this increased usage, there has been much recent interest in developing and deploying techniques for web caching (see, for example, [8, 29, 33] and references therein). Web caching has several beneficial effects: it lowers the load on servers, reduces the overall network bandwidth required, and lowers the latency of responses.

However, web caching does have (at least) one serious disadvantage. If a page has been modified after being stored in a cache, the version of the page delivered to the requesting client[1] may be inconsistent with the server's version of that page. We call such inconsistent pages *stale*, and call consistent pages *fresh*; the degree of staleness is the delay between when the page was changed on the server and when the previous version was delivered. To make this precise, consider the version of the page that was delivered to the client. Let $t = M$ be the time the delivered version was first rendered invalid by being modified at the server. Let $t = R$ be the time the cache responds to the client's request for that page. We then define the staleness[2] to be $\max(0, R - M)$.

For many pages, being significantly stale is not a serious problem. For some pages, however, clients may care a great deal if the pages are substantially stale. For instance, it is clear that pages devoted to current news stories (*e.g.*, CNN) should be as fresh as possible. Other examples of pages that are sensitive to being stale – we will call such pages *perishable* – are catalogs, product information, and code distribution pages. Perishable pages need not have zero staleness (*i.e.*, a news page could be a minute or so out of date without serious harm), but they should not be significantly stale.

One could most easily meet the freshness needs of perishable pages by circumventing caching; this can be accomplished by marking pages as uncacheable, or by merely expecting users to manually hit the "reload" button. However, since some perishable pages are likely to be quite popular – news sites in particular – one would like to ensure the relative freshness for these pages while retaining the advantages of caching. Because there is a finite latency between the server and the cache, it is impossible to guarantee absolute freshness (*i.e.*, true consistency between what the cache

*We would like to thank Mike Spreitzer and Marvin Theimer for their collaboration in the early stages of this work. They are responsible for many of the key ideas that inspired the design described in this paper. We would also like to thank Pei Cao for several helpful conversations. This research was supported by the Defense Advanced Research Projects Agency (DARPA) through the VINT project at USC/ISI under DARPA grant DABT63-96-C-0054 and at Xerox PARC under DARPA grant DABT63-96-C-0105. Lee Breslau and Scott Shenker were at Xerox PARC while this work was carried out.

[1] We use the term *client* to refer to a browser or other user process at the end host that generates requests for pages.

[2] Note that even if the staleness is zero by this definition, the page may be out of date when it actually arrives at the client due to changes made at the server while the data was in transit to the client; this, however, is not a problem with the caching infrastructure – since this source of inconsistency occurs even if the request was sent directly from the client to the server rather than being handled by a cache – and so we do not consider it part of being stale.

delivers and the current version at the server) without instituting write-locking on servers.[3] While write-locking is sensible for keeping file systems consistent, it makes less sense for web pages,[4] since write-locking merely masks the underlying reality that the content delivered is different than the content the server thinks is most current. Thus, the most practical goal is to merely limit the degree of staleness – *i.e.*, to achieve *loose consistency* – rather than trying to achieve strict consistency. We believe such loose-consistency guarantees should be sufficient for the vast majority of perishable pages.

In this paper we focus on the design of a scalable web cache consistency architecture that meets this goal. Our design retains the benefits of web caching (as listed above), while providing fairly tight limits on the degree of staleness of delivered pages. Of course, as we review in Section 2, there has been much previous work on techniques to achieve various degrees of consistency for web pages; the architecture we propose combines many of the features of these previous proposals, melding them together in a scalable fashion. Moreover, our proposal can easily be extended to support the *pushing* of data, in which modified pages are sent to caches even before clients have requested them.

Since we envision, at least initially, that a small fraction of pages are perishable, our design can be restricted to those pages that are deemed by the server to be perishable; that is, our proposal does not change how caches handle nonperishable pages and only modifies how caches handle perishable ones. Our design does make use of a caching hierarchy. However, this hierarchy can be replaced by a cache mesh, as we describe in Section 3.2.

We evaluate this design in two ways. We first investigate its behavior analytically in a very simplified setting, and then present simulation results in a somewhat more realistic setting. In both cases we compare our proposed design against several other schemes.

This paper is addressing the question of design, not of deployment. That is, we are asking: can one design such a scalable web consistency architecture? We are most definitely not addressing the question of whether such an architecture, once designed, should be deployed (although we discuss this question briefly in Section 7) since the question of deployment is a complicated cost/benefit tradeoff involving many nontechnical factors, such as the future usage of the web and the economics of the ISP business. However, deployment can only occur if a scalable web consistency architecture exists, and our contribution here is to demonstrate that such a design is indeed possible.

This paper has 7 sections. We begin in Section 2 by reviewing several of the previous approaches to ensuring consistency. We present our approach in Section 3, starting with our basic scheme and then adding in the ability to push pages. We then evaluate this design analytically in Section 4 and through simulations in Section 5. We discuss additional design issues in Section 6, and conclude with a brief discussion of our results in Section 7. We include estimates of cache state and network bandwidth requirements in an appendix.

[3] If the cache receives a request for a page, obtains a fresh version of the page from the server, and then delivers the page to the client, the page would still be stale when delivered if the page was modified on the server between the time the server sent the page to the cache and when it arrived at the cache. The only way to avoid this would be to write-lock the page during the interval while the page was being delivered to the cache.

[4] The crucial distinction between file systems and web pages, in terms of the role of write-locking, is that web pages have a single logical writer (the hosting server) whereas files have many logical writers (they can be written from many hosts). Merging multiple writers requires strict consistency, whereas handling multiple readers does not.

2 Previous Approaches

All web caching proposals attempt to achieve some degree of consistency, but the approach taken to achieve consistency depends greatly on the degree of consistency desired. In this section we briefly review three basic approaches to consistency. These approaches function both as inspirations for our proposed architecture and also as benchmarks against which we evaluate our design in Sections 4 and 5.

2.1 Time-To-Live

The simplest way to achieve some limited form of consistency is to associate a time-to-live with each page. When a request arrives at a cache after the TTL for the requested page has expired, the cache sends an If-Modified-Since (IMS) message to the server (or parent cache) to determine if the version held by the cache is still valid. If the TTL is fixed then the staleness is bounded by this TTL (plus the latency between the server and the cache). Setting small values of the TTL provides fairly tight consistency guarantees, but also mitigates against some of the benefits of web caching, since many IMS requests will be forwarded to the server even though the page is still valid. The limit of TTL=0 generates an IMS for every request, thereby guaranteeing no staleness; we call this scheme *poll-always*.

It has long been known that files exhibit the property that the longer they have gone unmodified, the longer they are likely to go unmodified [3, 4]. In [7] this insight was used to develop an adaptive TTL scheme in which the TTL is set, at the first request after each TTL expiration, to be proportional to the page's age (current time minus the last modification time); the algorithm takes, as a parameter, the constant of proportionality (called the update threshold in [15]) used to update the TTL. However, adaptive TTL schemes do not give an upper bound on the staleness of a page, since the TTL can grow without bound.

2.2 Invalidation

In the TTL approach, the cache can only guess as to whether a page is still valid. A very different approach to consistency requires servers to send explicit *invalidation* signals to caches when pages are modified. The invalidation approach is most easily explained, as we do below, when considering only the interaction between a server and a client without caches as intermediaries; later, when presenting our design, we will discuss the role of invalidations in the presence of proxy caches.

In its simplest incarnation, an invalidation scheme works as follows: each server keeps track of all clients who have requested a particular page and then, whenever that page changes, notifies those clients. We say that servers have an *invalidation contract* with the clients so that clients are assured that they will be informed of any changes to pages they have read.

While invalidation schemes are effective in limiting staleness, they incur the cost of requiring the server to keep state on every client of each page. Thus, this approach does not scale well in the limit of many readers per page; both the state required to store the list of readers, and the OS and

network burden of having to contact every reader of a page when it changes, grow linearly in the number of readers.[5]

This scaling problem can be overcome by using multicast to transmit the invalidations. By assigning a multicast group to each page, and having clients join the groups associated with the pages they have accessed, the burden on the server is greatly reduced; the server need not keep any readership state, and need only send a single invalidation message to inform the group of any page modifications. Such an approach is described in [28], and the somewhat related idea of *pushing* content (rather than sending invalidations) via multicast is described in [23, 27, 28]. However, while multicast solves the scaling problems at the server, it creates (following the law of conservation of difficulty) another one at the routers. The state required by such schemes in routers is substantial, easily on the order of hundreds of thousands of addresses (judging by the proxy traces in [19]); this is certainly too much for many currently deployed routers. Moreover, the rate at which clients would be joining and leaving multicast groups, as they read and discard pages, will likely create an unscalable overhead on the routing infrastructure [17].

A recent proposal [9] includes information about related pages in responses to page requests; this information may include invalidations and delta-encoded page updates. It can be used to greatly improve consistency on average but it does not provide staleness assurances.

2.3 Lease

The lease approach to consistency combines features of the TTL and invalidation approaches; see [13] for the basic reference on leases in file systems, and see [31] for applications of these ideas to web caching. In the simplest version of this approach, whenever a cache stores a page, it requests a *lease* from the server. Whenever a page changes, the server notifies all caches who hold a valid lease of the page; the invalidation contract applies only while the lease is valid. If a cache receives a request for a page with an expired lease, it renews the page's lease by sending an IMS to the server before responding to the request. While the lease is valid, the approach is exactly like invalidation, but the expiration of leases resembles the TTL approach. One wants to choose the length of the lease so that the number of readers holding valid leases remains reasonably small when writes are made, but most reads occur while the lease is still valid. In distributed file systems, leases are usually short (seconds or minutes) [4], but in the Web context using overly short leases makes the scheme roughly equivalent to TTL.

Yin *et al.* [31] presented two volume lease algorithms aimed at reducing validation traffic of short leases. They assign a long lease to every page, and a short lease to sets of pages called *volumes*. A cache must renew a lease whenever either the page lease or the volume lease expires. The advantage of this approach is that the overhead of renewing the short leases is amortized over the many pages in a volume.

3 Our Approach

Our approach borrows quite freely from these previous approaches. It is based primarily on multicast-based invalidations, but avoids the scalability problem by using a hierarchy of caches.[6] The multicast groups are associated with caches, not pages, and the caches send heartbeats to each other that are the equivalent of cache-to-cache volume leases. In contrast to a previous use of volume leases [31], the unit of our lease is all pages in a cache, instead of a single page or page group. Caches maintain a server table in order to locate where servers are attached to the hierarchy. Invalidation messages for a page, which may be sent both up and down the hierarchy, are filtered so as to limit the scope of distribution. Client requests are forwarded through the caching hierarchy to the server or to the first cache containing a valid copy of the requested page.[7] We first describe the basic protocol and then describe how to add pushing to the architecture.

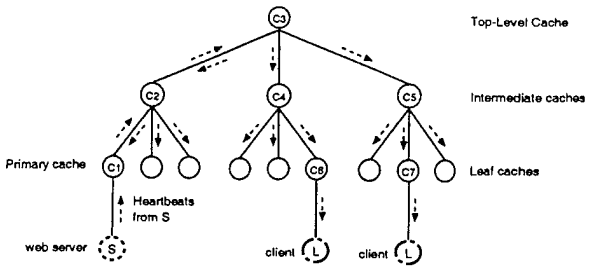


Figure 1: Example of a single multicast caching hierarchy. The arrows indicate the propagation directions of heartbeats.

3.1 Simple Description of Protocol

To describe the algorithm most compactly we first consider the special case where all caches are infinite, all pages are part of this consistency architecture, there is a single stable caching hierarchy with all caches having synchronized clocks, and no caches fail (although we make no assumption about the reliability of communication between caches). Aspects of the design associated with more realistic settings are addressed in Section 6. The descriptions given here (and in Section 6) are rather cursory and informal; a more complete and detailed description of the entire protocol can be found in [32].

Hierarchy The caching hierarchy (Figure 1) is glued together by multicast. Each parent cache *owns* a unique multicast group, in the sense that it is responsible for allocating the group address, and it is the only sender in the group. Each child cache joins the group owned by its parent. Thus, parents need not know who their children are, and children can choose their parents freely as long as cycles are prevented, and that is easily accomplished with a convention on assigning each cache to one of a few levels – *e.g.*, leaf caches, intermediate caches, and top-level caches – and requiring that parents always outrank their children. We do not address the issue of hierarchy establishment and maintenance; see [25] for one approach to these issues. We discuss alternatives to the use of a hierarchy later in this section.

Heartbeats The hierarchy is kept alive by *heartbeats*. Each group owner sends out a periodic heartbeat message to its

[5] Also, in the oversimplified version just described, there are robustness problems when servers lose their state or when network partitions occur. These robustness issues can be addressed, as we shall see in Section 3.

[6] We discuss alternatives to a hierarchy in Section 3.2.

[7] An extension that allows requests to bypass the caching hierarchy, thus reducing response latency, is described in Section 6.

associated multicast group; let τ be the time period between heartbeats. The heartbeat functions as a volume lease of length T to its children; this lease applies to all pages sent by the cache to its children. The time period of the lease starts when the message was generated (reflected in its timestamp), not when it was received. Typically τ will be significantly less than T ($\frac{T}{\tau} = 5$ in our simulations) so that if one or a few consecutive heartbeat are lost – which is a possibility since we are not sending them reliably – the lease won't expire unnecessarily. Each child cache compares the current time to the last heartbeat's timestamp (or, more precisely, the highest timestamp among all received heartbeats). If this time gap ever reaches T then the lease on all pages from that server expires and all such pages are marked as invalid.

Invalidations On top of these heartbeats we piggyback explicit invalidations. We need only invalidate pages that have been requested (by a client or another cache) after they were last rendered invalid; we call these *read pages*. Each heartbeat message contains a list of all read pages that have been rendered invalid at the parent cache within the last time period T. Thus, if a read page is rendered invalid at the parent cache at time $t = 0$ then by time $t = T$ each child cache has either received a heartbeat with an invalidation for that page, or has expired the lease from that parent cache (and thereby rendered the page invalid). A child cache that had a previously valid copy of the page will mark it invalid and propagate the invalidation if and only if the page was previously read; otherwise it ignores the invalidation.

Attaching Servers In addition to heartbeats going down the hierarchy, we also have a set of heartbeats traveling up the hierarchy from servers towards the top-level cache. To describe this, we first define how servers attach to the hierarchy. Each web server is attached to a cache (not necessarily a leaf cache) in the hierarchy, which we call the server's *primary cache*. Upon attaching, each server must reliably unicast a JOIN message to its primary cache. This message is forwarded upwards (by each cache to its parent cache) via reliable unicast until it reaches the top-level cache. We say that the parent cache *sources* a server from a child cache if it receives that server's JOIN message from a child cache (and has not received a LEAVE message for that server; we define LEAVE messages below). Each cache has a listing of those servers it sources (*i.e.*, those servers attached below it); we call this list the *server routing table* (Figure 2). If a cache does not source a server, we say that its server routing table entry for the server points to the parent cache. Note that the top-level cache knows about all servers attached in the hierarchy.

Servers send (via unreliable unicast) periodic heartbeats to their primary cache, also piggybacking invalidations of any read pages as we described above. Similarly, every child cache who sources at least one server must unicast heartbeats to its parent, along with piggybacked invalidations. A cache can ignore invalidations for unread pages (pages that are not in residence in the cache are automatically considered unread pages). Invalidations are thus propagated from the server to every cache from which the page has been read. If a cache C1 is closer to the server than cache C2 along this propagation path, we call C1 an *upstream* cache (compared with C2); otherwise, we call it a *downstream* cache. Each upstream cache is said to maintain an invalidation contract with its immediate downstream cache(s) for any page that has been read by a downstream cache.

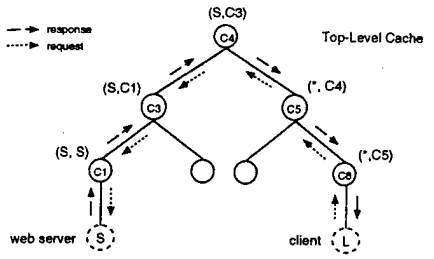


Figure 2: An example of server routing table setup. Routing table entries are shown in parentheses next to each cache. Each entry is in the form (S,C), where C is the next hop cache towards server S. A "*" indicates a default entry. The arrows show how requests flow from a client to the server, and how responses flow in the reverse direction.

When a time period T has passed without cache C1 hearing from cache C2 from whom it sources a server, cache C2 and all the servers sourced from cache C2 are removed from cache C1's server routing table. Cache C1 then sends a LEAVE message to its parents and children, notifying them that those servers are no longer sourced from cache C1, and therefore all of the pages from those servers should be considered invalid. (More details are described in Section 6.) LEAVE messages are a form of invalidation, and are included in the heartbeats (rather than being sent reliably).

Handling Requests We now describe how client requests are handled, as illustrated in Figure 2. Clients can attach to any cache in the hierarchy; we call this the client's primary cache. In particular, a client can attach to its own local cache (*i.e.*, the browser's cache) and then use a nearby proxy cache as a parent cache (as they typically do now). When a client requests a page, it sends the request to its primary cache. The primary cache, and recursively all caches the request visits, first checks to see if the page is resident in the cache. If it is not, then the cache forwards the request to the next cache designated by the server routing table. When the request is fulfilled, by either the originating server or by some intermediary cache, the response takes the reverse path through the caching hierarchy towards the client. The reverse path is automatically set up because every cache has an open HTTP connection to the the requester before it responds.

3.2 Discussion

We list below three important properties of this scheme (proofs can be found in [32]). As stated above we assume a stable hierarchy, synchronized clocks, and the proper functioning of caches, but make no assumptions about the reliability of communication.

Property 1 *If there are no invalidations in transit or waiting to be sent, then if a cache C in the hierarchy has a page P marked as invalid, then no downstream cache considers P valid (i.e., it is either invalid or not in residence).*

Property 2 *When a cache C receives an invalidation for a page P marked as invalid, it may safely discard the invalidation without affecting the resulting state of all downstream caches.*

Property 3 *Assume that each cache uses the same timeout period T. Consider a server S1, a client attached to cache C2 requesting the page, and assume that there are H cache-hops between S1 and C2. Then the maximal staleness of a page hosted on S1 delivered to the client is HT.*

Property 2 follows directly from Property 1. Together they allow us to reduce redundant invalidation traffic. Property 3 sets an upper bound of page staleness for every cache in the hierarchy.

We believe this scheme is a scalable approach to web cache consistency, and is essentially an application-level version of multicast distribution of invalidations. To clarify this analogy, consider a design which has a multicast group per version of a page and in which requesting the page is equivalent to joining the group for that version of the page; when a version of the page is rendered invalid, invalidations are sent to the group associated with that version, and multicast routing makes sure the invalidation ends up at every client and cache that has that version of the page. This is exactly what happens in our design, except that our design has no explicit notion of groups, and all "routing" of invalidations is done by the caches keeping track of the read pages and forwarding invalidations for those read pages.[8] The use of heartbeats facilitates robustness and failure detection.

Before proceeding, we elaborate on the use of caching hierarchies in this design. Our protocol requires application level routing to route messages among clients, servers and caches. Cache hierarchies provide a simple way to do this, but there are other possible cache organizations. The only requirement is that the cache organization provides source-independent and acyclic application-level routing of messages between servers and caches. That is, there must be a single (application-level) path between a cache and a server, and when superimposed, the set of paths to a server from all caches is loop-free. A cache mesh, in addition to a cache hierarchy, can also accomplish this goal.

We see the tradeoff between a mesh and a hierarchy as follows. The hierarchy provides a simple mechanism to reduce the (application-level) routing state in caches. This is particularly true at the leaves of the hierarchy, since a cache only needs explicit information about servers below it in a hierarchy. A mesh, on the other hand, eliminates the bottleneck of a root cache at the expense of increased state at other caches. Since a mesh organization has neither implicit information about cache location, as is provided by the default parent entry in a hierarchy, nor aggregable cache address allocation as is available in IP routing [12], reducing the routing state at caches is difficult. In addition, the lack of aggregation implies increased processing and communication overhead to establish and maintain the routing state. For example, information about changes in server state must be propagated to all caches in the mesh.

Given this tradeoff, we see the choice of a hierarchy as reasonable for the following reasons. First, it places the largest burden on a smaller number of caches (root or other high level caches) that are most easily engineered to meet this load. Engineering all caches to meet the state requirements of a mesh is likely a more difficult problem. Second, estimates of the load on root caches, provided in Appendix A, indicate that the load on the root caches is manageable. Therefore, in this paper we describe our design in the context of a cache hierarchy, nevertheless, it works for both meshes and hierarchies. Moreover, hybrid approaches are possible; for instance, leaf caches could be attached to a general mesh topology, reducing the state requirements on leaves and reducing traffic in the core.

Above we assumed an ideal environment for the sake of discussion, however, our design is capable of handling various issues related to more realistic contexts: *e.g.*, clock skew, finite cache, failure recovery, incremental deployment, etc. We address these issues briefly in Section 6, and refer the interested reader to [32] for additional details.

3.3 Adding Push to the Architecture

There is one aspect of performance that caching cannot improve: the latency suffered by the first request to an unread page. The concept of *pushing* data from the server to caches is of some interest, precisely because it reduces this first access latency so dramatically. While pushing is not directly related to caching, it fits within our architecture and addresses an important web performance issue, so we have included it in our design. We now briefly present a simple proposal for pushing. One only wants to push popular pages that are likely to be read before they are modified again. Servers could identify pages that are sufficiently popular that they should be pushed, or clients could request certain pages be pushed (see [32] for designs of that flavor). Here we present a more adaptive algorithm that chooses which pages to push based on the request and writing pattern. We call this scheme *selective push*.

Rather than pushing the entire page, we push only the delta's from the previous version of the page, which are typically rather small [19]. On the way up the caching hierarchy the updates are sent via reliable unicast. On the way down, we use a single unreliable multicast sent to a cache's multicast group.[9] Pushing the page does not remove the need for sending invalidations for the previous version, since the data could be lost in transit.

We use a heuristic to decide if a page is sufficiently popular to be pushed. We do not make a single global decision about whether or not to push a page; instead, each cache, and the originating server, make their own independent decision about whether or not to push the page. Every cache (and the server) keeps a counter A_P (initialized to 0) and a *push bit* for each of its pages. If the bit is 1, the cache will forward all pushed updates of the page to all of its downstream caches. The heuristic uses three positive constants: θ, γ, and β. Whenever a cache receives an invalidate of page P, it sets $A_P = A_P - \gamma$; whenever it receives a request for P, it sets $A_P = A_P + \beta$. If $A_P > \theta$ for some threshold θ, the cache (or the server) sets the push bit of the page to 1; otherwise the push bit is set to 0. In addition, we let each downstream cache notify its immediate upstream cache when a pushed page is first read; these read notifications are forwarded recursively until they hit a read page. This allows caches who have pushed the page to still get accurate readings on whether the pushed page was read downstream before the page was invalidated.

Recent work has addressed the issue of pushing web pages. Continuous Multimedia Push (CMP) [24] assigns a unique multicast group to every popular page and continuously multicasts pages to their groups. They found that multicast push is preferable to caching only when pages are

[8] Note that our analogy to application-level multicast is completely unrelated to our use of real multicast to communicate between parent and child caches.

[9] Unreliable distribution is sufficient, since pushing affects performance and not correctness of the protocol. However, one could use SRM [11] or other reliable multicast protocol for this distribution; we have not done so in our simulations to reduce complexity, but it is a very natural design choice.

very popular and change very frequently. LSAM [27] assigns one multicast group per "topic"; popular pages of similar topic (*e.g.*, SuperBowl) are multicast to a unique group when they are created or modified. Our scheme is similar in spirit to these approaches, but quite different in implementation. We use application-level "routing" of pushes that is equivalent to multicast, and we adaptively decide which pages are sufficiently popular to push.

4 Analytical Performance Evaluation

If we assume, as we will throughout this paper, that caches are effectively infinite,[10] then the behavior of our web caching consistency protocol can be analyzed on a per-page basis; if no meta-state or page data is deleted from a cache due to space considerations, then the message generation behavior (*i.e.*, invalidations, etc.) for a given page is independent of what happens for all other pages.[11] We now analytically evaluate the performance of our proposed protocol in a very simple setting. We consider a single client, a single cache, and a single server. The client sends out requests (reads) for a particular page, and the server modifies (writes) that page.

We compare several different web consistency approaches. The first, *omniscient* TTL (OTTL), is not a realistic scheme, but it provides a useful benchmark; in this scheme caches magically know when a page has been modified and only send the IMS request in those cases. The second is *poll-always* (PA) which, as we discussed in Section 2, is just a TTL approach with TTL=0. The other two are variants of our invalidation scheme: our basic invalidation scheme with no page pushing (BINV) and our invalidation scheme with pages always pushed (PINV).[12] To make the modeling easier, we assume there is no delay between when invalidations are generated and their being sent out (*i.e.*, invalidations don't wait for the next heartbeat). Thus, all of the protocols described here provide the same level of strong consistency; if we ignore page modifications made after the server has responded to a request and before the response arrives at the cache, then there are no stale pages delivered by any of these protocols. We do not study the looser policies of adaptive TTL or fixed TTL here because their finite timeout periods makes the analysis intractable; we evaluate them using simulation in Section 5.

Since none of these algorithms depends on absolute time, we care only about the patterns of reads and writes arriving at a cache. We can characterize the behavior of these algorithms by describing which messages get sent upon one of these four events: a read following a write (WR), a read following a read (RR), a write following a write (WW), and a write following a read (RW). Let F_{RR}, F_{RW}, F_{WR}, F_{WW} denote the average rate at which the patterns RR, RW, WR, and WW occur, respectively. We model the reading and writing as Poisson processes of rate r and w, respectively, and so the frequencies of events can be computed as follows: $F_{RR} = \frac{r^2}{r+w}$, $F_{WW} = \frac{w^2}{r+w}$, $F_{RW} = F_{WR} = \frac{rw}{(r+w)}$.

Table 1 summarizes the bandwidth usage, server hit count, and cache response delay of each protocol for these four events. The relative performance in terms of server hit counts and response time holds regardless of the read and write rates. PINV completely eliminates server hits,[13] and BINV and OTTL have the same server hit count, which is less than PA. The same ordering applies to response time: PINV has no delays, OTTL and BINV have an intermediate level of delay, and PA has the most delay. The bandwidth comparison of these algorithms is less clear and, in some cases, depends on the values of the various parameters.

	OTTL	PA	BINV	PINV
RR	delay: 0 bw: 0 hc: 0	delay: $2\ d_1$ bw: $2b_{IMS}$ hc: 1	delay: 0 bw: 0 hc: 0	delay: 0 bw: 0 hc: 0
RW	bw: 0	bw: 0	bw: b_{inv}	bw: b_P+b_{inv}
WR	delay: d_1+d_2 bw: b_P+b_{IMS} hc: 1	delay: d_1+d_2 bw: b_P+b_{IMS} hc: 1	delay: d_1+d_2 bw: b_P+b_{GET} hc: 1	delay: 0 bw: 0 hc: 0
WW	bw: 0	bw: 0	bw: 0	bw: b_P+b_{inv}

Table 1: Table of bandwidth, server hit count, and delays for each of the four events: RR, RW, WR, WW. b_{inv} is the cumulative size of a repeated set of invalidation messages. b_P is the average size of a page. b_{GET} is the size of an HTTP GET request. b_{IMS} is the size of an IMS request. b_{ntf} is the size of a read notification message. d_1 is one way delay of IMS, GET, invalidation and responses. d_2 is the one way delay of transmitting a page from server to cache.

For convenience, we assume $b_{IMS} = b_{inv} = b_{ntf} = b_{GET}$, and let b_{ctl} denote this size. Since these are all small packets, we do not introduce significant errors by ignoring the size differences. Notice that OTTL uses less bandwidth than any other scheme. PA uses less bandwidth than BINV if and only if $2r < w$; the tradeoff is between PA sending an IMS and response on reads following reads versus BINV sending an invalidate message on writes following reads. PA uses less bandwidth than PINV if and only if $(\frac{r}{w})^2 < \frac{1+\frac{b_P}{b_{ctl}}}{2}$. Lastly, PINV uses less bandwidth than BINV if and only if $\frac{r}{w} > 1 + \frac{b_P}{b_{ctl}}$.

If one assumes the size of pages dominates the size of the control messages then the limit of $b_{ctl} = 0$ may provide some insight. When $b_{ctl} = 0$ then all the protocols except PINV require the same bandwidth (pages are transmitted whenever a modified page is first read). BINV has the same performance, in terms of server hit counts and response times, as the OTTL, our idealized benchmark. BINV has lower response time and server hit count than PA. This performance gap grows as the reading rate increases, since BINV's advantage is that it need not contact the server (thereby incurring server hit counts and delay) when a valid page is read; when the read rate is much lower than the write rate, few of the requests find a valid page at the cache, but as the read rate increases more of these requests find a valid page at the cache. Thus, if the bandwidth of control messages can be ignored, then the main performance criteria separating BINV from PA are server hit counts and response times, not bandwidth, and these performance gaps become more significant as the reading rate increases. PINV eliminates hit counts and delays but at the cost of increased bandwidth.

In order to make our analysis in this section tractable, we assumed a very idealized environment and did not consider every protocol. In the next section we will use simulations to evaluate all of the consistency protocols in a somewhat

[10] See Appendix A for further discussion of this assumption.

[11] The only degree of interaction is the number of pages over which the overhead of heartbeats is shared.

[12] PINV can be seen as a version of *mirroring* in which updated pages are automatically mirrored at remote sites.

[13] Of course, this reduction in server hits comes at the cost of the server pushing the data; however, we believe that the cost of answering a request may be higher than that of pushing a page update.

more realistic setting.

5 Simulations

In this section we use simulations, performed using the *ns* [2] simulator, to evaluate the performance of our proposal, and to compare it to several other approaches. In particular, we investigate the performance of our basic invalidation protocol (BINV), along with the variants selective push (SINV) and push-always (PINV), and compare them to poll-always (PA), adaptive TTL (ATTL), fixed TTL (FTTL) and omniscient TTL (OTTL).

We evaluate these various web cache consistency protocols using two categories of metrics: user-centric metrics and infrastructure-centric metrics. The user-centric metrics, which quantify the user's level of satisfaction with the service provided, are client response time[14] and staleness. We measure staleness in three ways: the maximum and average staleness taken over all pages, and the percentage of pages which are delivered stale (stale hit rate). Most previous papers on web consistency used stale hit rate as the only metric for staleness; we prefer to emphasize the average staleness, since staleness is not a binary property. That is, how out-of-date a page is, not just whether or not the page is stale, may be important. The infrastructure-centric metrics quantify (aspects of) the burden placed on the network infrastructure by these various protocols; we measure the total network bandwidth (in byte-hops), the bandwidth at the server, and the rate of (GET and IMS) requests at the server.

Recall that several of these algorithms have adjustable parameters that control their performance: the heartbeat rate h for the invalidation-based algorithms, the TTL value for FTTL, and the threshold for ATTL. We are not interested in measuring the tradeoff between staleness and bandwidth achievable by each of these protocols. Rather, we assume low average staleness is a performance requirement and ask how much bandwidth and delay are incurred by the protocols to achieve a particular level of staleness. Therefore, we set the heartbeat rate for BINV to be 10 per minute and then vary the parameters for FTTL and ATTL so that they all have roughly equivalent average staleness.[15] The additional parameters required in SINV are set as follows: $\gamma = 1$ (invalidation constant), $\beta = 2$ (request constant), $\theta = 8$ (push threshold).

We begin our simulations with a very basic scenario, and then later describe several additional scenarios. The results show that our invalidation scheme can achieve the same staleness as the TTL approaches with lower response time and overhead. The advantages are most pronounced for popular pages which do not change often.

5.1 Basic Scenario

In this scenario we consider a single two-level caching hierarchy (5 leaf caches and a top-level cache) embedded in a simple network topology, as shown in Figure 3. As we discussed in Section 4, if we treat the caches as infinite then the behavior attributed to each page is independent of other pages. Consequently, we choose the workload in our basic scenario to have only a single page so that we can focus more narrowly on how the performance of these consistency

[14]The latency between sending a request and complete receipt of the response.

[15]We are not able to accomplish this in all cases. We elaborate on this below.

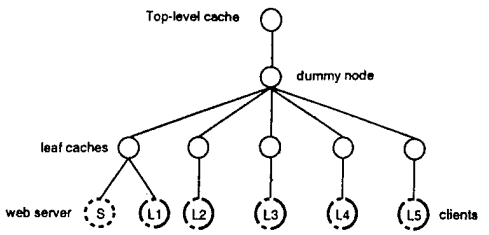


Figure 3: Network topology in the basic scenario. All links between server/clients and leaf caches have 10Mb bandwidth and 2ms delay. All links among caches and the dummy node have 1.5Mb bandwidth and 50ms delay.

protocols depends on the reading and writing patterns of a page. This single page, chosen to be 1KB in size, is read and written according to Poisson processes with average rates r (per-client) and w, respectively. We consider two cases: a *write-dominated* (WD) page, where the read rate (per-client) is one per 2.5 hours and the write rate is 1 per 15 minutes ($\frac{w}{r} = 10$); and a *read-dominated* (RD) page, where the read rate (per-client) is 1 per 2 minutes and the write rate is 1 per 10 minutes ($\frac{r}{w} = 5$).

We now describe some of the simulation details. The IMS and GET messages are 43 bytes, and each invalidation record adds an additional 32 bytes to a heartbeat. Because in reality the header of a heartbeat is amortized over many pages, we ignore it in these single-page simulations. The RD and WD simulations were run for approximately one day and five days (simulation time), respectively, with the initial 7 and 15 minutes taken to be a warmup period (for the RD and WD simulations, respectively).

Tables 2 and 3 show the results for RD and WD pages, respectively. Because of the sensitivity of the results to the tuning parameters, exactly matching the average staleness across protocols is difficult. When confronted with this, we chose parameter values for ATTL and FTTL that yielded slightly *higher* average staleness than our BINV benchmark (e.g., 8.37 and 11.9 msec versus 8.06 msec in Table 2). This gives us a lower bound on the overhead and delay incurred for the ATTL and FTTL to match the staleness of BINV. We first discuss the RD case, and begin by comparing BINV to the TTL-style protocols. Compared with PA, BINV uses 26% less bandwidth, has 27 times less server hit count and 10 times faster response time. Because FTTL and ATTL are required to maintain the same low staleness as BINV, they both have small TTL values (ATTL threshold equals 0.0105 and FTTL time-to-live equals 9.5 seconds) and therefore behave like PA. Their bandwidth is slightly higher and their response time and server hit count are much higher than those of BINV. BINV's performance is similar to that of OTTL, but it has slightly higher bandwidth consumption due to its invalidation overhead. Comparing BINV to PINV, we find, as expected, that pushing data reduces response time and eliminates server hits while increasing bandwidth by only about 6%. Because the read rate is so much higher than the write rate, updated pages are eventually fetched from the server, so pushing them out immediately for this read-dominated workload does not incur additional bandwidth overhead. SINV's performance is very close to that of PINV.

Turning to the WD case, we see that the problem of matching the average staleness across the tunable protocols is exacerbated. This is due to the fewer number of stale hits (in absolute terms) in a write-dominated work-

	BINV	ATTL (0.0105)	FTTL (9.5)	OTTL	PA	SINV	PINV
AS	8.06	8.37	11.9	0.00	0.00	0.34	0.09
MS	4.95	17.00	8.43	0.00	0.00	1.08	0.21
SR	0.38	0.15	0.27	0.00	0.00	0.06	0.05
TB	6.90	8.73	8.42	5.75	9.33	7.35	7.38
CR	0.06	0.53	0.48	0.05	0.61	0.04	0.04
SH	123	2684	2324	124	3300	8	0
SB	158.4	694.1	617.4	147.9	824.2	153.9	153.6

Table 2: Statistics of a read-dominated page in the basic scenario. AS: average staleness (millisecond); MS: maximum staleness (second); SR: stale hit rate (%). TB: total bandwidth (MB-Hop). CR: client response time (second). SH: server hits. SB: server bandwidth (KB).

	BINV	ATTL (0.1)	FTTL (80)	OTTL	PA	SINV	PINV
AS	1.22	544.4	13.1	0.00	0.00	1.22	0.00
MS	0.29	76.19	3.11	0.00	0.00	0.29	0.00
SR	0.42	0.84	0.42	0.00	0.00	0.42	0.00
TB	1.40	1.41	1.41	1.35	1.42	1.40	6.16
CR	0.41	0.60	0.61	0.46	0.62	0.41	0.26
SH	155	224	230	151	236	149	1
SB	184.7	193.7	194.9	179.1	197.2	184.8	275.4

Table 3: Statistics of a write-dominated page in the basic scenario.

load. Nonetheless, the data show that BINV's performance advantage is now reduced. For the time-to-live value shown in the table, FTTL has worse staleness than BINV, nearly the same bandwidth but only about 50% longer response time and 50% higher server hit count. ATTL has worse average staleness while the other metrics are comparable to FTTL. PA has performance very similar to FTTL (reflecting the very small TTL used in FTTL). Again, PINV achieves very low response time and server hit count, but this time at the cost of a factor of 4 in bandwidth consumption. Note that SINV behaves like BINV in this WD case, but behaved more like PINV in the RD case; this was the goal of the adaptive algorithm in SINV, to actively push pages only when they are read-dominated.

These results are completely consistent with the theoretical analysis of Section 4. The major benefits of invalidation schemes (over TTL-based schemes) are savings of response time and server hit count, and these benefits are much more pronounced in the read-dominated case. Adding push increases these advantages further, but at the cost of significantly more bandwidth in the WD case.

In this basic scenario, and in each of the following scenarios, we assume that the heartbeat rate h is greater than the write rate w times the number of cache-hops H. This will likely be true for the vast majority of pages, however there are some pages, such as those containing stock quotes, that will change faster than $\frac{h}{H}$. If such pages are also popular, our invalidation approach will deliver a significant fraction of pages stale (since the invalidations are still in transit from server to leaf cache); see [32] for more details. Such pages are better delivered using multicast techniques, such as Continuous Multicast Push [24].

5.2 More Complex Topology

In the second scenario, to test the effect of having a more complicated network topology, we took a 3-level caching hierarchy (leaf, intermediate, and top-level), with a branching ratio of 2 at each level, and embedded it into a 300 node random transit-stub network topology created by the GT-ITM [5] topology generator. The top-level cache and all intermediate caches are on transit nodes. All leaf caches are in the stub network associated with the transit node where the parent intermediate cache resides, and each intermediate cache is in a different stub network.

	BINV	ATTL (0.01)	FTTL (12)	OTTL	PA	SINV	PINV
AS	15.6	16.2	17.5	0.00	0.00	2.47	0.88
MS	6.44	26.46	10.47	0.00	0.00	1.25	0.94
SR	0.54	0.12	0.34	0.00	0.00	0.26	0.16
TB	18.16	23.04	22.67	15.55	23.83	19.45	19.53
CR	0.18	0.49	0.44	0.12	0.53	0.12	0.12
SH	124	2290	1880	126	2583	8	1
SB	158.6	607.2	560.9	150.3	694.5	154.0	153.6

Table 4: Statistics of a read-dominated page in a more complex topology.

	BINV	ATTL (0.01)	FTTL (95)	OTTL	PA	SINV	PINV
AS	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MS	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SR	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TB	6.38	6.47	6.45	6.24	6.48	6.38	26.68
CR	0.76	0.91	0.89	0.76	0.91	0.76	0.68
SH	131	192	186	130	193	131	1
SB	156.1	156.8	156.5	154.2	161.0	156.1	274.8

Table 5: Statistics of a write-dominated page in a more complex topology.

Tables 4 and 5 present the results from simulations on this topology with RD and WD pages, respectively. The basic relative trends in the data appear unaffected by introducing a more complicated topology. For the RD page, the TTL approaches have worse response time and server hit counts than BINV, and the push approaches offer reduced response time, server hit counts, and staleness without incurring any additional bandwidth. Compared with the RD case, BINV's advantages are greatly reduced in the WD case.

5.3 More Complex Workload

The Poisson workload used so far is not intended to be an accurate model of reality; rather, it is merely a simple test case. We have augmented the simulations presented here with simulations on a wide variety of other workloads. We have considered compound pages, where the page contains multiple objects (such as embedded graphics). We have also considered reading and writing processes that are heavy-tailed and processes that are uniformly distributed. The results from these simulations are presented in [32]. Those results were qualitatively similar to these presented here, and space limitations prevent us from including them. However, we do want to present data from one additional workload.

Our previous data was generated using artificial read and write processes. To get a sense of a more realistic scenario, we now consider a trace-driven workload consisting of the read sequence of a single page extracted from a real trace. We pick two pages, one popular and one unpopular, from a 5-day segment of the UCB Home-IP trace [14], and apply the consistency algorithms to the two pages. The popular page has 62,582 requests, and the unpopular page has 21. No page modification data is available for these traces, so we used a Poisson model with an average of one modification

	BINV	ATTL (0.0015)	FTTL (8)	OTTL	PA	SINV	PINV
AS	1.32	1.36	1.65	0.00	0.00	0.05	0.01
MS	4.69	10.90	8.76	0.00	0.00	2.11	0.18
SR	0.07	0.04	0.06	0.00	0.00	0.01	0.01
TB	27.16	75.15	62.07	22.03	91.75	27.07	27.07
CR	0.01	0.45	0.33	0.01	0.60	0.01	0.01
SH	119	39087	25182	119	58124	2	1
SB	72.6	8342	5380	41.8	12381	64.8	64.1

Table 6: Statistics of a popular page in the UCB Home-IP trace.

	BINV	ATTL (0.2)	FTTL (2800)	OTTL	PA	SINV	PINV
AS	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MS	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SR	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TB	279.8	281.8	279.2	279.2	283.0	279.8	2781.5
CR	0.61	0.80	0.80	0.77	0.83	0.61	0.55
SH	18	19	21	17	21	18	0
SB	42.5	43.1	43.5	42.6	43.5	42.5	217.5

Table 7: Statistics of an unpopular page in the UCB Home-IP trace.

per hour (based on data in [10]). With this modification rate, the popular page is read-dominated, and the unpopular page is write-dominated. Tables 6 and 7 present the results from simulations for these two pages.

These results are consistent with our previous results. The only novelty here is the fact that for the popular page the IMS overhead of the TTL approaches is more evident. In order to maintain the same staleness as BINV, ATTL required 3 times as much bandwidth as BINV, and FTTL more than doubled bandwidth.

5.4 The Effect of Packet Losses

Up to this point, our simulations do not include any packet losses. We now return to our basic scenario and introduce per-link packet loss rates in order to evaluate the effect of packet losses on the consistency protocols.

In our protocol, both invalidations and pushed updates are sent out via unreliable multicast. When packet loss is present, we expect that performance will degrade. Because invalidations are piggybacked in several consecutive heartbeats, but pushes are sent only once, we expect that invalidations are less vulnerable to packet loss than pushes. In order to test these expectations, we introduced 3% per-link losses into our basic scenario. For the network shown in Figure 3, 3% per-link loss rate corresponds to end-to-end loss rates between 3% and 6% (which is intended to match the loss rates of between 2.65% and 5.28% found in [22]). Results are shown in Tables 8 and 9; the data presented are averages over 9 runs.

Packet loss increases the bandwidth and response time for all the protocols. BINV's stale hit rate and average staleness increase slightly, and the maximum staleness increases significantly, because the lost invalidations need at least another heartbeat interval to reach leaf caches. SINV behaves similarly to BINV but, as expected, PINV, is more significantly affected by packet loss; its average staleness and maximum staleness are increased substantially.

When loss rate grows even bigger, some caches will time out due to consecutively lost heartbeats, and our failure recovery mechanism will be triggered (see Section 6). This will impose a transient increase in response latency (because all affected cached pages are invalidated, and an IMS will be generated by the next request).

	BINV	ATTL (0.013)	FTTL (12)	OTTL	PA	SINV	PINV
AS	9.88	12.5	12.2	0.00	0.00	0.73	0.43
MS	7.15	20.77	11.42	0.00	0.00	1.81	1.38
SR	0.41	0.27	0.30	0.00	0.00	0.09	0.06
TB	7.16	9.19	8.88	6.01	9.92	7.56	7.58
CR	0.09	0.74	0.67	0.09	0.93	0.05	0.05
SH	128	2543	2154	128	3260	8	1
SB	198.7	754.5	669.8	180.5	936.3	194.8	192.3

Table 8: Statistics of a read-dominated page in the basic scenario with 3% per-link loss rate.

	BINV	ATTL (0.03)	FTTL (135)	OTTL	PA	SINV	PINV
AS	34.9	544.4	333.1	0.00	0.00	34.9	1.22
MS	4.40	76.19	76.19	0.00	0.00	4.40	0.29
SR	0.84	0.84	0.84	0.00	0.00	0.84	0.42
TB	1.59	1.57	1.54	1.50	1.59	1.59	6.27
CR	0.53	0.93	0.95	0.75	0.89	0.53	0.36
SH	155	239	222	151	239	149	1
SB	224.3	236.5	219.1	223.0	238.8	212.1	316.4

Table 9: Statistics of a write-dominated page in the basic scenario with 3% per-link loss rate.

5.5 Related Work

There have been several recent papers comparing the effectiveness of TTL and invalidation approaches: Worrell [30], Gwertzman and Seltzer [15], and Cao and Liu [6]. Worrell claimed that when FTTL has similar bandwidth consumption as unicast invalidation, it has 20% stale hits, and therefore concluded that unicast invalidation is preferable for strong consistency. Gwertzman and Seltzer argued that bimodal lifetime of web pages makes ATTL the preferred choice; their trace-driven simulation showed that ATTL had few stale hits (<5%) and took much less bandwidth than unicast invalidation. Using real systems in trace-driven experiments, Cao and Liu confirmed that ATTL had few stale hits, but they found that ATTL and unicast invalidation had similar bandwidth usage. Moreover, they found that unicast invalidation at times led to increased latency because of the message processing overhead at the server.

Our results differ from those in previous work for a couple of reasons. Compared to simple unicast invalidation, our invalidation protocol can avoid much of the redundant invalidation traffic. Thus, in most cases, it takes less or the same bandwidth as ATTL while achieving the same level of page staleness and resulting in much less server load and client response time. At the same time, our work is somewhat complementary to the previous investigations. Because we focus on single-page workloads when evaluating this protocol, we are able to identify more precisely the effect of different reading and writing processes on the results. In addition, we focus on average staleness, rather than the stale hit rate, as the crucial staleness metric. Finally, because we assume that perishable pages require very low staleness, we focus our simulations on operating regimes with much lower staleness measures than previous studies.

6 Additional Design Issues

We have presented the basic design of our protocol in an ideal environment with infinite caches that never fail, a single stable hierarchy with synchronized clocks, and with all pages included in the architecture. In this appendix we discuss additional aspects of the design to cope with more realistic settings.

Clock Skew In Section 3 we assumed that the clocks in the caching hierarchy were perfectly synchronized. However, if the maximal clock skew between a cache and its upstream and downstream neighbors is bounded by ϵ then the cache timeout period should be $T-\epsilon$ instead of T. We assume that in typical cases $T \gg \epsilon$ so this modification in the protocol will have little impact.

Finite Cache Caches are, in reality, finite. While we argue in Appendix A that our design does not require unrealistically large amounts of state in caches, it is important that the design can cope with situations where the cache has exceeded its capacity. First, to keep the invalidation contract in force, a cache need only remember the *meta-data* (the URL and the last-modification time) about the page, and can freely discard the actual contents of the page. Second, if the cache is forced to discard the meta-data itself, then it must send an invalidation for that page to its children and/or its parent depending on whether the page has been read from those directions. While this may impact performance, the correctness of the protocol is unaffected.

Failure Recovery The algorithm as described deals with the case where a cache fail-stops. However, it does not describe how a cache can recover from a failure. We require that caches recover in a *naive* state; that is, they invalidate all pages in the cache and send a LEAVE message to their parent and child caches. This allows all affected invalidation contracts to be broken before the cache reattaches. We have the following property:

Property 4 *As long as caches that have failed recover in a* naive *state then the three properties in Section 3.1 hold even in the presence of failures and recoveries.*

One remaining problem is how to recover the server routing entries that were evicted during a partition or lost during a failure. There are two cases. First, if a parent cache C1 times out a child cache C2 from whom it sourced servers, it needs to send a JOIN_QUERY after hearing from C2 again. C1 can piggyback the JOIN_QUERY in a heartbeat, just as it does with invalidations. Second, if C2 times out C1, C2 needs to send C1 a JOIN which contains its server routing table, *i.e.*, all of the servers from which it has heard JOINs. In both cases, when C1 recovers its routing table, it needs to notify its parent of its current routing table.

Direct Request Using a hierarchy (or cache mesh) to forward requests to servers can introduce significant delay [1]. Because requests in our hierarchy might travel both up and down the hierarchy, this risk of delay is higher. However, we can extend our design so that the client's primary cache can, upon a cache miss, go directly to the server to get the data. When the cache receives the data, it then, after handing the data to the client, establishes the invalidation contract by sending a *pro forma* request up the hierarchy. The pro-forma request is used merely to establish the required correct state in the hierarchy, and does not elicit a reply of data from the caches or the server. The pro-forma carries with it the Last-Modified time of the page returned by the server. It stops being forwarded when it hits a cache which has that version of the page, or meta-data for it, in residence. If the pro-forma hits a cache (or server) that has a more recent version of the page in residence, an invalidate is generated and sent back down the path. If the pro-forma hits a cache with a valid older version of the page, no action need be taken since an invalidate is on the way. In this manner, the caching hierarchy provides invalidations while the delivery of actual web pages bypasses this hierarchy. This alleviates some of the disadvantages of a web caching hierarchy, such as parent cache overloading and increased response time [26].

Multiple Hierarchies and Multi-Homing There will obviously be multiple caching hierarchies in the Internet, although we expect the number to be relatively limited (less than, say, 100). Our design can easily be extended to handle these multiple hierarchies by having the Top-level cache of one hierarchy contact caches in other hierarchies. This can be accomplished using a single multicast group comprised of the members of all Top-level caches. Each top-level cache multicasts its heartbeats to this group, as well as to its multicast group in its own hierarchy. Whenever a top-level cache, call it TLC1, gets a request for an unknown web server, it queries the server about its top-level cache, call it TLC2, and then forwards the request to TLC2 as if TLC2 were a parent cache.

While our design requires that a server only attaches to a single cache in a given hierarchy, we allow it to attach to multiple hierarchies; we call this a *multi-homed* server. The design works without significant modification.

Supplying Service to a Subset of Pages We do not expect that all pages will need the level of consistency provided by our architecture. In order to provide invalidations on a subset of all web pages, we propose a new HTTP header field that describes whether or not the page should be subject to this consistency architecture. The simplest approach is to have the server set this field. There are some situations where it might be appropriate to allow a client to set this field, thereby requesting invalidation service for the page. Of course, the server must be willing to support this service by participating in the sending out of heartbeats and invalidations. There are some subtle issues in both of these approaches which are too detailed to discuss here but are covered in [32].

Deploying in Existing Cache Hierarchies In order to implement our protocol in existing cache hierarchies, we can enhance ICP [29], the *de facto* inter-cache communication protocol, to support our consistency protocol. Four new types of ICP messages are needed: *heartbeat*, *JOIN*, *LEAVE*, *request notification*, and *PUSH*. If direct request is desired, another message type, *pro forma* is needed. Because these messages do not interact with existing ICP messages, adding them to ICP is straightforward.

7 Conclusion

In this paper we have presented and evaluated a web cache consistency protocol based on invalidation. Our proposal

builds on previous work in the literature, combining the ideas of multicast invalidations with volume leases and incorporating them within a caching hierarchy to make the design more scalable. Our performance evaluation suggests that when the heartbeat rate h is larger than the writing rate times the number of hops (wH), then the invalidation approach is very effective in keeping pages relatively fresh. When pages are write-dominated, then the invalidation approach offers few advantages since all the protocols, if they are to ensure freshness, must go back to the server to get a valid page. However, when pages are read-dominated, which we think will be the common case for perishable pages (*e.g.*, CNN and other news pages), then the invalidation approach offers significant reductions in server hit counts and client response time. In both cases, our invalidation scheme requires similar or less bandwidth than the TTL-style protocols.

Our analysis focused exclusively on the technical aspects of the protocol. However, the remaining questions, and the barriers to deployment, may be more economic and institutional in nature. Our design uses a set of relatively stable and well-managed caching hierarchies (though it can work with other cache organizations). Currently this does not describe the current state of web caching, and so assuming the existence of caching hierarchies may seem like a dubious foundation on which to build our architecture. However, the institutional trends in ISPs appear to be one of consolidation, and in the future these large ISPs may very well provide such a caching hierarchy as part of their service (and the mirroring service provided by @Home is some evidence in this direction). Moreover, the hierarchy we envision does not require central management (since parents need not know the list of their children explicitly) nor must it be deployed ubiquitously to be useful, so the barriers to its realization are somewhat reduced.

In addition, the deployment of any such a web cache consistency protocol would only be undertaken if ISPs determine that there is sufficient demand for relatively fresh versions of perishable pages. It seems clear that perishable pages comprise only a small fraction of current web usage. On this basis one might be tempted to dismiss the consistency problem as unimportant. However, if the web is to serve as the foundation on which much of the information infrastructure is built, then perhaps it should be augmented to meet the needs of this class of pages.

Clearly the whole issue of deployment, depending as it does on such unknowables as the future usage and economics of the web, and the nature of the ISP business, is far beyond our ken. We only caution that the growth path of the web caught many of us by surprise, and we should be humble in our confidence to predict, based on its current usage and existing institutional arrangements (where we expect the case for its deployment is weak) whether the future of the web would be significantly aided by deploying such a consistency architecture, and whether it is organizationally feasible. Our goal here was merely to demonstrate that it is indeed technically feasible.

References

[1] BAENTSCH, M., BAUM, L., MOLTER, G., ROTHKUGEL, S., AND STURM, P. World-Wide Web caching – the application level view of the Internet. *IEEE Communications Magazine 35*, 6 (June 1997). http://www.uni-kl.de/AG-Nehmer/Projekte/GeneSys/Papers/communic.ps.

[2] BAJAJ, S., BRESLAU, L., ESTRIN, D., FALL, K., FLOYD, S., HALDAR, P., HANDLEY, M., HELMY, A., HEIDEMANN, J., HUANG, P., KUMAR, S., MCCANNE, S., REJAIE, R., SHARMA, P., SHENKER, S., VARADHAN, K., YU, H., XU, Y., AND ZAPPALA, D. Virtual Internetwork Testbed: Status and research agenda. Tech. Rep. 98-678, University of Southern California, July 1998. *ns* web site: http://mash.cs.berkeley.edu/ns.

[3] BAKER, M., HARTMAN, J. H., KUPFER, M. D., SHIRRIFF, K. W., AND OUSTERHOUT, J. Measurements of a distributed file system. In *Proceedings of the ACM Symposium on Operating Systems Principles* (Oct. 1991), pp. 198–221.

[4] BLAZE, M. A. *Caching in Large-Scale Distributed File Systems.* PhD thesis, Princeton University, Jan. 1993.

[5] CALVERT, K., DOAR, M., AND ZEGURA, E. Modelling Internet topology. *IEEE Communications Magazine* (June 1997).

[6] CAO, P., AND LIU, C. Maintaining strong cache consistency in the World-Wide Web. In *Proceedings of the International Conference on Distributed Computing Systems* (May 1997), pp. 12–21.

[7] CATE, V. Alex - a global filesystem. In *Proceedings of the 1992 USENIX File System Workshop* (Ann Arbor, MI, May 1992).

[8] CHANKHUNTHOD, A., DANZIG, P., NEERDAELS, C., SCHWARTZ, M., AND WORRELL, K. A hierarchical Internet object cache. In *USENIX Conference Proceedings* (1996), pp. 153–63.

[9] COHEN, E., KRISHNAMURTHY, B., AND REXFORD, J. Improving end-to-end performance of the Web using server volumes and proxy filters. In *Proceedings of the ACM SIGCOMM* (1998).

[10] DOUGLIS, F., FELDMANN, A., KRISHNAMURTHY, B., AND MOGUL, J. Rate of change and other metrics: a live study of the world wide web. Tech. Rep. 97.24.2, AT&T Labs, Dec. 1997. A shorter version appeared in Proc. of the 1st USENIX Symposium on Internet Technologies and Systems.

[11] FLOYD, S., JACOBSON, V., LIU, C., MCCANNE, S., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. *ACM/IEEE Transactions on Networking 5*, 6 (Dec. 1997), 784–843. ftp://ftp.ee.lbl.gov/papers/srm_ton.ps.Z.

[12] FORD, P. S., REKHTER, Y., AND BRAUN, H.-W. Improving the routing and addressing of IP. *IEEE Network Magazine 7*, 3 (May 1993), 10–15.

[13] GRAY, C., AND CHERITON, D. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the ACM Symposium on Operating Systems Principles* (1989), pp. 202–210.

[14] GRIBBLE, S. D., AND BREWER, E. A. System design issues for Internet middleware services: Deductions from a large client trace. In *Proceedings of The USENIX Symposium on Internet Technologies and Systems* (Dec. 1997).

[15] GWERTZMAN, J., AND SELTZER, M. World-Wide Web cache consistency. In *Proceedings of the USENIX Conference Proceedings* (Copper Mountain Resort, CO, USA, Dec. 1996), pp. 141–51.

[16] INKTOMI INC. Inktomi Traffic Server, 1998. http://www.inktomi.com/products/traffic/product.html.

[17] KUMAR, K., RADOSLAVOV, P., THALER, D., ALAETTINOGLU, C., ESTRIN, D., AND HANDLEY, M. The MASC/BGMP architecture for inter-domain multicast routing". In *Proceedings of the ACM SIGCOMM* (Vancouver, Canada, Sept. 1998). http://catarina.usc.edu/estrin/papers/masc-bgmp-arch.ps.

[18] LAWRENCE, S., AND GILES, C. L. Searching the Web: General and scientific information access. *IEEE Communications Magazine 37*, 1 (Jan. 1999), 116–121.

[19] MOGUL, J., DOUGLIS, F., AND FELDMANN, A. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of the ACM SIGCOMM* (Sept. 1997), pp. 181–194. http://www.acm.org/sigcomm/sigcomm97/papers/p156.ps.

[20] NETCRAFT. The Netcraft Web server survey. http://www.netcraft.com/Survey/.

[21] NUA INC. Nua Internet surveys. http://www.nua.ie/surveys/how_many_online/index.html.

[22] PAXSON, V. End-to-end Internet packet dynamics. In *Proceedings of the ACM SIGCOMM* (1997).

[23] RODRIGUEZ, P., AND BIERSACK, E. W. Continuous multicast distribution of Web documents over the Internet. *IEEE Network Magazine* (March-April 1998).

[24] RODRIGUEZ, P., BIERSACK, E. W., AND ROSS, K. W. Improving the WWW: Caching or multicast. In *Proceedings of The Third International WWW Caching Workshop* (June 1998).

[25] ROSENSTEIN, A., LI, J., AND TONG, S. Y. MASH: The multicasting archie server hierarchy. *SIGCOMM Computer Communication Review 27*, 3 (July 1997).

[26] TEWARI, R., DAHLIN, M., VIN, H., AND KAY, J. Beyond hierarchies: Design considerations for distributed caching on the internet. Tech. Rep. CS98-04, Department of Computer Sciences, UT Austin, May 1998.

[27] TOUCH, J. The LSAM proxy cache - a multicast distributed virtual cache. In *Proceedings of The Third International WWW Caching Workshop* (June 1998).

[28] VAHDAT, A., EASTHAM, P., AND ANDERSON, T. WebFS: A global cache coherent filesystem. Tech. rep., Dept of EECS, UC Berkeley, Dec. 1996. http://www.cs.berkeley.edu/~vahdat/webfs/webfs.html.

[29] WESSELS, D., AND CLAFFY, K. ICP and the Squid web cache. *IEEE Journal of Selected Areas in Communication 16*, 3 (Apr. 1998).

[30] WORRELL, K. J. Invalidation in large scale network object caches. Master's thesis, Department of Computer Science, University of Colorado, 1994.

[31] YIN, J., ALVISI, L., DAHLIN, M., AND LIN, C. Using leases to support server-driven consistency in large-scale systems. In *Proceedings of the 18th International Conference on Distributed Computing System* (May 1998).

[32] YU, H., BRESLAU, L., AND SHENKER, S. A scalable web cache consistency architecture. Tech. Rep. 99-708, Dept. of Comp. Sci., Univ. of Southern Calif., June 1999.

[33] ZHANG, L., FLOYD, S., AND JACOBSON, V. Adaptive web caching. Project proposal, Feb. 1997. http://irl.cs.ucla.edu/AWC/proposal.ps.

A Estimation of State and Bandwidth Requirements

Our architecture requires cache state and inter-cache communication in order to provide loose consistency. In this section we provide some very crude estimates on the cache state and inter-cache bandwidth required by our scheme. These estimates, which should not be taken as a definitive quantitative statement about the overhead of the protocol, indicate that the scheme is indeed feasible.

A.1 State Requirements

Our protocol introduces two additional items into the cache state: page metadata and the server routing table. We first estimate the amount of metadata that might be stored in a cache. If we assume that that a top-level cache holds no more than 320 million pages (the estimate of all publicly indexable web pages [18]), and one meta-data record contains 80 bytes (which is enough for a URL, last-modification time, push counter and several flags), this results in about 25.6GB of metadata. This is quite small compared to modern large caches [16], and is dwarfed by the storage requirements needed to store the actual pages.

Next, we estimate the size of the server routing table. The top-level cache, if there is only a single hierarchy, has a list of every server. We assume there are roughly 4 million web servers (Netcraft's web server survey [20]). The resulting size of the server routing table is on the order of 32MB, assuming 4 bytes to store each server address and 4 bytes for each child cache address. This again poses no challenge to well-equipped caches. Thus, for the purposes of analyzing our design, we can reasonably assume that caches are effectively infinite (at least as far as meta-data is concerned).

A.2 Invalidation traffic

Our design generates an invalidation every time a read page is written, and we now seek to estimate how much traffic this produces. Let's characterize every page P by a reading rate r_P and a writing rate w_P. The number of invalidations generated by a page is bounded above by $\max[r_P, w_P]$; we will call a page *write-dominated* if $r_P < w_P$ and *read-dominated* if $r_P \geq w_P$. A bound on the invalidation rate for a given cache is $\sum_{P\ in\ cache} \max[r_P, w_P]$ where the sum is over all valid pages in the cache.

We first estimate the traffic seen at a top-level cache. If there is significant logical locality to requests, so that pages tend to be more frequently requested by clients close to them in the hierarchy, then there will be many pages that are never cached at the top-level cache. However, we have no way of estimating the extent of this effect, and so will assume the worst case that all pages are indeed cached at the top-level cache. We estimate that the entire Web has 1 billion pages, which is three times the size of publicly indexable pages [18]. To estimate r_P and w_P, we use numbers from the DEC proxy traces cited in [10]. This trace covers a large population (7400 distinct clients), and contains 505,000 requests of 204,000 distinct pages over a period of 2 days. Most pages, roughly 80%, have only one access in the trace, and we consider these to be write-dominated pages. It is difficult to estimate r_P from the trace due to its limited duration. Instead, we use the average number of such pages read by each user, then extend that rate to the web population. In the DEC trace, about 50% of the requests went to these write-dominated pages. We can compute the read rate of such pages by each user: $0.5 * \frac{505000}{7400} * \frac{1}{2*24*3600} = 0.0002$ (request/user/second). We now extend this to the entire web user population. It is estimated that the web has 151 million users as of December, 1998 [21], and we assume that 1% of these users are as active as those in the DEC trace, and the rest are 100 times less active. This yields a total sum of r_P over all write-dominated pages of $0.0002 * (1.51 + 149.49 * 0.01) * 10^6 = 601$ (invalidation/second).

We consider the other 20% of pages read-dominated. From figures in [10], we conservatively estimate their average change rate as once every 1 hour ($w_P = 0.00028$ per second). Without any evidence on which to base a more educated guess, we conservatively assume that 0.1% of all Web pages are sufficiently popular to be read-dominated. Recall we estimate there are 1 billion web pages, so the sum of w_P over all read-dominated pages is $0.00028 * 0.001 * 10^9 = 280$ (invalidations/second).

If we assume that each invalidation is repeated 5 times, and 32 bytes per invalidation, this yields a total traffic level of $(280 + 601) * 1280 = 1.1$Mbps. Repeating this calculation for the AT&T trace in [10] yields an estimate of 1.7Mbps.

We next estimate the traffic at an intermediate-level cache. We assume that the DEC and AT&T traces are reasonable representatives of intermediate-level caches; using their estimates of the number of readers and the number of pages in residence (rather than the global numbers used in the top-level estimates), we arrive at estimates of 75Kbps and 90Kbps for the DEC and AT&T traces, respectively.

The above estimates assume all pages are included in the consistency architecture; we do not expect that most pages will be considered perishable, and so the consistency architecture will be carrying all web pages only a small fraction of the total web traffic. Moreover, we completely neglected any locality of reference, and made rather generous assumptions about the number of popular pages (.1% of the web!). Nonetheless, in spite of these rather pessimistic assumptions, the overall bandwidth levels are rather reasonable.