UNIVERSITY OF CALIFORNIA

SANTA CRUZ

DISSERTATION PROPOSAL: EFFICIENT PERFORMANCE GUARANTEES ON STORAGE NETWORKS

A dissertation proposal submitted in partial satisfaction of the requirements for the degree of

Doctor of Philosophy

in

COMPUTER SCIENCE

by

Andrew G. Shewmaker

June 2012

The Thesis of Andrew G. Shewmaker is approved:

Table of Contents

List of Figures

List of Tables

Abstract

Dissertation Proposal: Efficient Performance Guarantees on Storage Networks

by

Andrew G. Shewmaker

Current storage networks do not provide end-to-end Quality of Service (QoS). The intention of this research is to produce both a formal theory and real-world implementation of flexible, general, and fine-grained performance guarantees on standard commodity network hardware. These guarantees will be composable with guarantees made for CPU, disk, and memory resources such that end-to-end QoS on storage networks can be provided. Systems built using these methods will avoid over-provisioning, virtualize the performance of remote storage access, and experience improved reliability and administration over more traditional solutions.

“Those who do not understand TCP are destined to reimplement it.”

- Jon Postel

“It’s not a big truck. It’s a series of tubes.”

- from Senator Ted Stevens famous explanation of congestion on the Internet

Acknowledgments

I would like to thank Scott Brandt, Carlos Maltzahn, Richard Golding, and Theodore Wong for allowing me to join their research in Storage Quality of Service. Thanks also to Tim Kaldewey, Anna Povzner, Roberto Pineiro and other students in the Systems Research Laboratory.

This research integrates and extends work from two independent efforts at the University of California Santa Cruz in the Systems Research Laboratory and the Computer Communication Research Group led by J.J. Garcia-Luna-Aceves.

Thanks to Gary Grider and Carolyn Connor at Los Alamos National Laboratory for encouraging me to pursue graduate work and for their work establishing the Institute for Scalable Scientific Data Management collaboration between LANL and UCSC. All of my management at LANL have been extremely supportive of my continuing education, and I am grateful they enabled me to come to Santa Cruz for an entire school year while remaining an employee. Josip Loncaric, in addition to being both my team leader and later group leader, provided me with valuable insight into clock synchronization protocols.

Chapter 1

Introduction

This paper proposes avenues of exploration leading to improved theory and solutions to the problems involved in providing efficient real-time performance guarantees on storage networks. The primary problem in standard commodity local area networks is congestion, exhibited by increases in transmission delay and packet loss. In particular, throughput collapse due to large numbers of simultaneous bursts is a common scenario that has proven difficult to prevent in storage networks that require extreme throughput rates.

The problem is complicated by the need to minimize overheads both on hosts and the network itself. Radon, the name of the family of algorithms presented later, has shown some promise in allowing multiple hosts to cooperatively respond to congestion and enforce their collective guarantees using local knowledge.

However, the initial Radon work only dealt with Gigabit Ethernet and simple network topologies. It requires additional theoretical work to prove that desired bounds are not violated and that buffer requirements are minimized. Finally, the previous work did not result in an implementation that could be used in real applications. This follow-on work will remedy those deficiencies with additional theoretical work, simulation, performance tests on both Ethernet and Infiniband networks, and the development of Linux Kernel module implementations of Radon algorithms suitable for use in the real world.

The rest of this proposal is organized as follows: Chapter 2 provides background on storage networks and networking technology in general. Chapter 3 describes what can be measured on a network, since you cannot control what you cannot measure. That is followed in chapters 4 and 5 by a discussion of the previous work performed by the author in developing both new rate-based and window-based congestion control algorithms, referred to collectively as Radon. The final chapter describes the new research to be done in order to take the existing Radon algorithms from promising ideas to a formally proven theory with a rigorously tested implementation.

Chapter 2

Background

Most storage networks are composed of best-effort devices that strive for good performance while offering no guarantees. Network hardware with built-in Quality of Service (QoS) features exists, but is relatively expensive and is usually limited to static, priority-based configurations that distinguish between classes of traffic rather than individual streams. Furthermore, network QoS has been treated independently from other resources when there are, in fact, explicit relationships between the network and other resources. As a part of a larger end-to-end storage QoS project shown in Figure 2.1, this research focuses on a more general cooperative network protocol, Radon, that does not rely on expensive specialized network hardware.

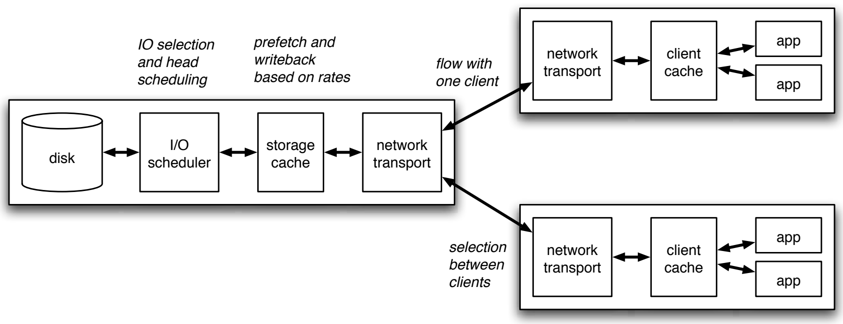


Figure 2.1: Radon is a component of a larger Storage QoS project

2.1 Classes of Storage Networks

Current storage networks vary in capability, from the cheap and unreliable to the expensive and robust. All lack end-to-end QoS. Enterprise storage networks are converging to technologies that provide reliable transport, but it is a slow process that leaves some room for compromise in the definition of 'reliable'.

The three major classes of storage networks are Network Attached Storage (NAS), the Storage Area Network (SAN), and the distributed file system. NAS is the most common and least expensive storage network, where one or more servers individually provide a file system interface over a standard Ethernet network. More expensive SANs are composed of storage arrays connected with a high performance network such as Fibre Channel and appear as a local device to a host. Distributed file systems come in Wide Area Network and Local Area Network (LAN) variants. Wide area systems often serve large numbers of users, operate over a large variety of technologies, and are generally grown rather than designed. In contrast, local area systems are designed to provide a high performance parallel file system for a set of well-defined users.

At the same time that the market is pressuring vendors to unify storage network technology (e.g. Fibre Channel Over Ethernet and Converged Enhanced Ethernet), none provide hard latency or throughput guarantees. Infiniband is an advanced network architecture that provides a reliable transport using credit-based flow control and levels of QoS. However, priority-based QoS is more appropriate for static network flows than the dynamic flows seen in storage networks. This research focuses on providing flexible, general, and fine-grained performance guarantees on a local area distributed file system.

2.2 Unreliable Transport

The most common storage area network implementations include some form of Ethernet, with clients and servers communicating through TCP/IP sockets. Reliable transport is provided by TCP/IP, an upper layer protocol implemented on the hosts of the network rather than the network's internal hardware (e.g. switches). Because the congestion control is implemented at end points which have little or no knowledge of the state of the shared resource between them, the harmful effects of congestion occur first in the shared resource. On the positive side, Ethernet storage networks are inexpensive and easy to implement, but they begin failing to perform at higher loads, suffering from packet loss and high variation in packet delivery delays.

Figure 2.2 demonstrates congestion in a simple switch model. Packets con-

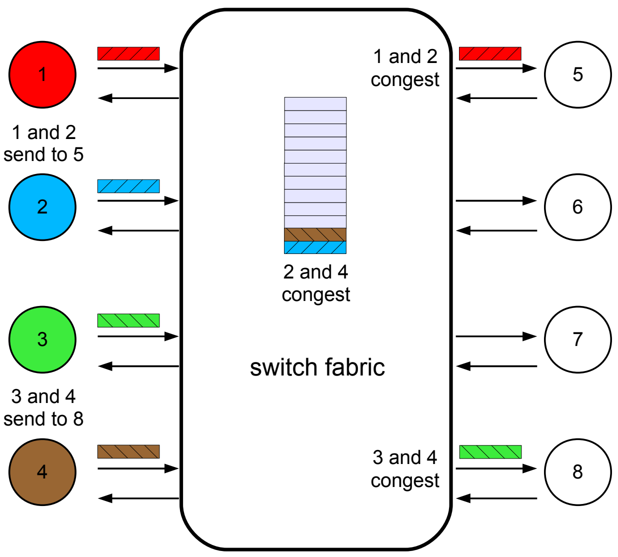
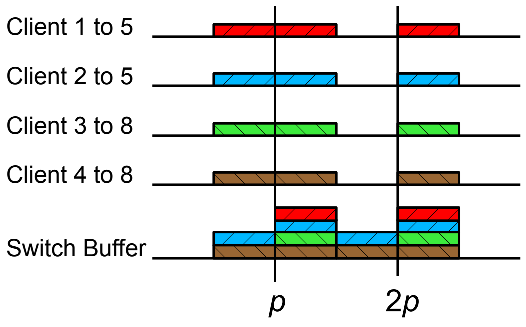


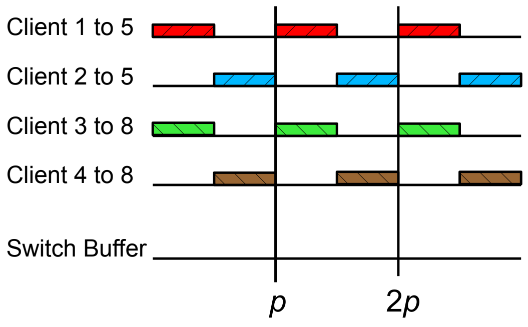
Figure 2.2: Congestion in a simple switch model

tending for the same destination port are queued. Continuous contention may cause previously isolated streams to interfere with each other. In the worst case, the queue will overflow and packets will be lost. Distributed file systems experience a particular case of congestion called incast [31, 52] where a file spread among many servers is sent in simultaneous bursts to a client, which can overflow a switch buffer with little or no warning signs.

Figure 2.3a shows the worst-case behavior described by Figure 2.2 over three periods. Assuming that each client is using half of the period to transmit, the switch will never exceed four units of buffer space, and each packet will always be served within the period of its arrival. On the other hand, Figure 2.3b shows the best-case behavior, where the transmissions from each client interleave perfectly, the switch uses no buffer



(a) Worst case



(b) Best case

Figure 2.3: Time-series plot of a switch buffer with and without congestion

space at all, and the latency for individual packets is minimal.

Le Boudec's and Thiran's Network Calculus [34] thoroughly describes this behavior using Min-plus algebra to reason about the equations representing arrival and service curves. The analysis reveals that for feasible flows, a switch's buffer requirement is the sum of the burst sizes, regardless of any other parameter. They prove that bursts must be paid only once in a switch fabric, which is obvious when one considers that the traffic becomes serialized after the initial incast. Furthermore, greedy shapers keep arrival constraints. In other words, they do not increase delay or buffer requirements, and they conserve arrival constraints. Le Boudec's Network Calculus puts traditional traffic shaping as provided by Linux queueing disciplines or by an Internet Service Provider's DiffServ QoS on a firm theoretical foundation.

However, there are tradeoffs inherent in using these sorts of traffic shapers. Fine-grained shaping reduces the amount of performance you can expect from a single

NIC and is expensive in CPU time. Enforcing a burst size of one means packets are being sent one at a time, which would halve achieved throughput for individual flows. Coarse-grained shaping increases switch buffer requirements. A 288-port switch would require 912 MB of RAM if every connection is expected to handle 4 MB bursts. In reality, large switches are created using many crossbar chips with a couple dozen ports. So a 288-port switch made up of 36 24-port crossbar would only require 96 MB of RAM if it was shared amongst all crossbars, or up to 3456 MB otherwise. This amount of memory is feasible, but could get undesirable with more and larger bursts. That is one reason why large-scale, reliable networks, such as Infiniband, implement robust message-level and byte-level flow control using credits.

Ethernet switches may support a limited form of link-level flow control by sending a PAUSE command to transmitting devices to indicate they should slow down for a specified amount of time. This can help in some situations, but the current standard is implemented using multicast packets and does not differentiate between senders. It also ignores Ethernet priorities. Emerging Ethernet standards, collectively known as either Data Center Ethernet or Converged Enhanced Ethernet, include Priority-based Flow Control – addressing the flaws in the existing Ethernet PAUSE command, Enhanced Transmission Selection – allowing different priorities to share each others spare bandwidth, and Congestion Notification – providing upper layer protocols such as TCP with information to help them set their transmission rates. Even with its upcoming enhancements, Ethernet will probably not match the performance and reliability of the technology described in the following section, though it will likely remain less expensive.

2.3 Reliable Transport

Enterprise storage networks are generally built with something like Fiber Channel or Infiniband. Reliable transport is provided by lower layer protocols implemented in the host adapters as well as the switches. Because the congestion control is implemented throughout the network, the harmful effects of congestion are pushed back to the senders. On the positive side, enterprise storage networks perform well at high loads, do not suffer from packet loss, and experience lower packet delivery delays, but they are expensive and require more expert knowledge to implement.

Just as we see pressure on Ethernet to gain reliable transport features, we see Infiniband hardware and software stacks possessing features that make it compatible to one degree or another with Ethernet and IP protocols. The Linux Kernel developers have added more pressure to converge by stating that they will only accept one RDMA API. With Ethernet enhancements, it might appear that Infiniband will die out, but it possesses a more aggressive roadmap than Ethernet for higher speed networks, is able to use much larger packet sizes while achieving lower latencies with smaller packets, and is generally implemented better feature-for-feature. Infiniband will probably always have more reliable transport and faster speeds than Ethernet, as long as it continues to market to High Performance Computing.

2.4 Real-world Considerations

Regardless of whether or not a transport is reliable, there are other real-world considerations. The maximum transmit unit (MTU) and buffer sizes should increase with speed of the network, otherwise the number of interrupts the hosts' CPUs must handle exceed its capabilities. The network adapter may offload some of the protocol processing overhead from the CPU, but this does not always result in a positive result for all workloads and it can complicate debugging problems in the network. In addition to the capabilities of hardware, the topology of the network also has implications with regard to making performance guarantees.

A Network Interface Controller (NIC) affects the performance that can be achieved and the amount of overhead it imposes on a host computer. A large amount of effort has gone into interrupt moderation techniques, where one interrupt is delivered for a group of packets in order to reduce the amount of work required by the CPU. While this can successfully increase throughput, it can also cause an undesirable increase in latency variation. In Linux, drivers can be written to conform to either the softnet or the "New API" [56]. NAPI solved several problems with softnet, including avoiding interrupt livelock [41] and packet re-ordering, by switching between interrupt driven and polling modes depending on the number of packets being received. The Linux kernel is continuing to evolve, traditional interrupt handling with top and bottom halves will likely be replaced by the real-time Linux tree's threaded interrupt handlers. Also, significant work has already been merged to allow hardware with multiple queues to scale its processing over multiple cores.

While small packets can be transferred much faster than large packets, they inefficiently use system resources. Overall performance can be maximized if the Maximum Transmission Unit (MTU) of the NIC is a multiple of the operating system's memory page size. This allows the PCI bus to use its maximum transfer size and not requiring an extra transfer to handle the last part of a packet.

NICs have become able to offload various pieces of the work generally handled by the operating system. A TCP Offload Engine (TOE) implements an entirely separate TCP stack in hardware. Generic Segmentation Offload (GSO), refers to a NIC with the ability to take a large buffer and split the data into packets on behalf of the operating system. The complement of GSO is Generic Receive Offload (GRO), which merges received packets. One of the primary benefits of both GRO and GSO is a reduction of the load on the PCI bus. When segmenting or merging packets the hardware must also support offloading the checksum operation, so the main CPU also does less work. There are security and performance pitfalls with any of these advanced hardware features, so many drivers allow them to be disabled. What may be beneficial on a multi-user system may impede performance on a router or hinder packet filtering functionality. One of the more useful features from a resource management standpoint is the availability of hardware generated timestamps for each packet.

Network performance is largely determined by a protocol's flow control, which manages the rate at which a stream injects data into the network when there is no congestion, and congestion control, which adapts the rate, burstiness, and timing of packet transmissions when congestion is detected. TCP/IP is the most widely deployed end-to-end network protocol, but its congestion control algorithms do not provide any performance guarantees. It continuously tries to increase throughput at the sender by increasing the window (burst) size and uses packet loss as a congestion signal to throttle the sender drastically. Even for a single connection, this results in a sawtooth pattern for throughput over time and a large variance in packet delays, as a switch's queue continually overflows and drains. Feng, et. al [9] show that the flaws in TCP's congestion control dramatically worsen as the number of streams scale up in local area distributed system because the streams tend to respond to congestion in lock step.

Many researchers have sought to improve or replace TCP, but change has proven to be difficult because TCP actually does a decent job in many situations, it is fairly robust, the Internet community desires new protocols to be TCP-friendly, and also because it is difficult to get buy-in for new ideas from a significant portion of the Internet community. The current default variant used by the Linux kernel is called CUBIC [17, 16] because of the function it uses to modify its window size. It is intended mostly to enhance behavior on high bandwidth networks with large delays. A storage network with high bandwidth and low delay may benefit more from tuning variables in TCP stacks that are best left untouched for normal Internet usage. For instance, researchers from CMU are investigating if it can recover from incast faster if it had a much smaller Retry Time Out (RTO).

Table 2.4 summarizes the scope of the resource management problem for various topologies of both wired and wireless networks. The simplest case is one in which

Network Description	Complexity
two linked hosts	$2 \times \mathit{host\ port}$
n chained hosts	$2(n-1) \times \mathit{host\ port}$
d dimensional P2P fabric	$2dn \times \mathit{host\ port}$
switched	$2 \times \mathit{host\ port} + \mathit{switch\ port}$
hub or wireless	$2 \times \mathit{host\ port} + \mathit{collision\ domain}$
wireless P2P fabric	$dn \times (2 \times \mathit{host\ port} + \mathit{collision\ domain})$

Table 2.1: Scope of resource management problem for various types of networks.

two hosts are directly connected to each other, and even then each host must schedule use of its network interface between competing processes and threads. The resource management problem quickly increases in difficulty as the topology of a network becomes more complex, but the distinguishing feature of network resource management is its distributed nature. Scheduling decisions cannot rely on global knowledge without incurring prohibitive communication overhead, making statistical multiplexing generally preferred over other strategies. Radon is initially intended for use on switched networks, but will later be adapted for wired peer-to-peer networks such as a multidimensional tori. As Radon is applied to more complex networks, admission control will need to be combined with the routing algorithm in order to maximize utilization of the network.

Figure 2.4 depicts a canonical storage network–a closed, full bisection bandwidth, Fat-Tree [35] network composed of standard Gigabit Ethernet switches. If the network is segmented into equal parts, then the links connecting the switches allow all pairs of hosts to communicate at their full link bandwidth.

Real-world enterprise storage networks may provide less than full bisection bandwidth or redundant routes, but a basic Fat-Tree is a reasonable starting point for

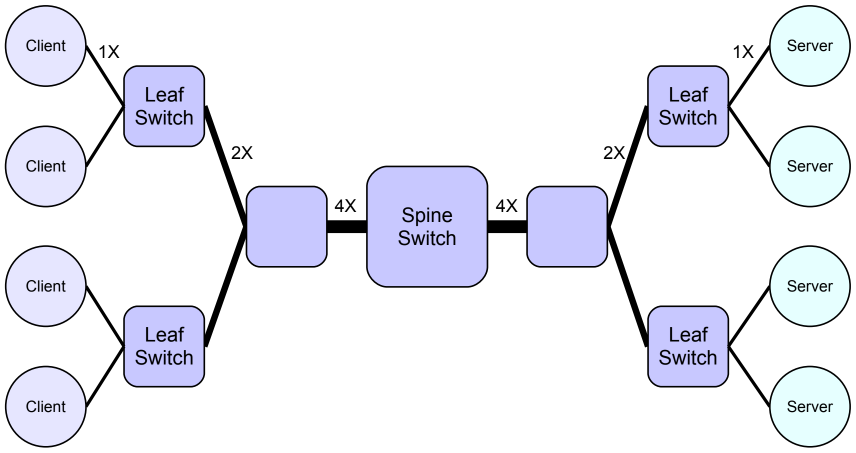


Figure 2.4: Canonical Fat-tree storage network

QoS research. For now, it is assumed that the storage network is tightly controlled, that the only significant source of packet loss is due to a buffer overflow. After achieving guaranteed performance at the single switch level, further research must explore the affects of more complex fabrics. A good algorithm should be able to guarantee up to the bisection bandwidth of a Fat-Tree network.

Chapter 3

Network Resource Measurements

In order to know what can be guaranteed on a network, it must be clearly understood what can be measured given that there is no global clock. Time is the fundamental characteristic of network measurements–barring packet loss, which must be prevented. This chapter examines how various synchronization methods, congestion control protocols, and tools measure different types of delay and how they handle noise and measurement errors. Figure 3.1 depicts two packets and their corresponding ACKs between two hosts with independent clocks, and is referenced throughout the chapter.

3.1 Measurements in Synchronization Protocols

Early work on logical clocks [33] gave a notion of a global ordering of events, and later research eventually led to the creation of the widely used Network Time Protocol (NTP) [40]. NTP calculates the RTT and clock offset between two hosts with one pair of data and ACK packets (see Figure 3.1), such that $RTT = (C_i - S_i) - (A_i - R_i)$

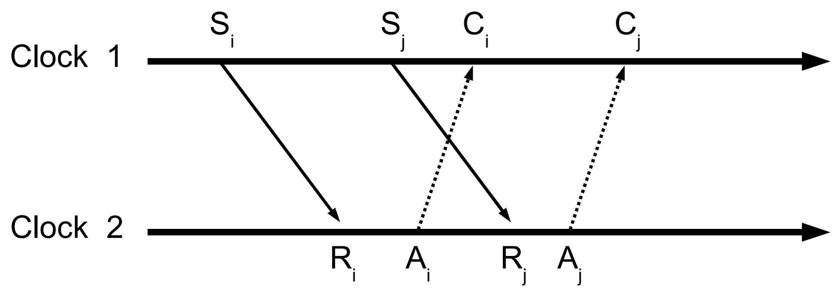


Figure 3.1: Measuring delay using timestamps corresponding to **S**end, **R**eceive, **A**CK, and **C**ompletion events for (data, ack) packet pairs i and j.

and $\mathit{offset} = \frac{(R_i - S_i) + (A_i - C_i)}{2}$. Clients filter multiple independent master clock sources, select the best sources, combine them, and remove noisy measurements using a hybrid Phase Locked Loop (PLL) and Frequency Locked Loop (FLL). While the methods of filtering measurements used in clock synchronization vary, the common idea is to ignore network affects by only using the minima. This reduces the number of good measurements, especially when a network is under load.

While NTP is commonly used throughout the Internet, both embedded and high performance computing communities have found its approximately 1 ms synchronization under optimum circumstances to be insufficient. IEEE 1588 defines the Precision Time Protocol (PTP) [1], and is primarily intended for LANs containing measurement devices requiring tight coordination. It supports synchronization in the nanosecond range when supported by hardware, or within 10 μs in a software implementation. At a high level PTP is similar to NTP, measuring RTT and clock offsets the same way and using the PLL/FLL implemented in most operating systems for use with NTP.

PTP [12] performs simpler filtering than NTP and the initial version is based on multicast rather than unicast/broadcast–both implementation decisions appropriate for embedded systems on a LAN. Also, because of NTP's venerable age and broad portability, it has used standard `gettimeofday()` style calls for send and receive timestamps. By the time PTP came on the scene, many operating systems had added a standard socket option for receive timestamps–though the matching send timestamp option is still missing. The most well know software implementation of PTP initially used a kernel patch to get accurate send timestamps, but now uses a multicast loop back device in combination with the receive timestamp socket option in order to ease installation and aid in portability. If NTP were to take advantage of the "new" timestamp interfaces, then it would likely achieve synchronization as well as software PTP. However, the second version of IEEE 1588 specifies how switches can add timing information to PTP packets or even synchronize directly with hosts. At the same time, Linux kernel developers in the employ of Intel are working on adding support for hardware [45] and software send timestamps. These features have been demonstrated to provide clock synchronization with $\pm 20\,ns$ accuracy.

BTime [39] has been developed for use with computational Linux clusters, and is designed to keep tightly interconnected nodes synchronized with each other rather than with national time standards as NTP does. It employs a more sophisticated Kalman filter [28] than either NTP or PTP and uses heuristics that take advantage of assumptions appropriate for high performance local networks. Computational clusters are not entirely divorced from the outside world, so the master clock of a cluster is still usually adjusted by NTP. These adjustments are nonlinear and can be large, from BTime's perspective, so it uses raw clock information from the Linux kernel to account for NTP's behavior and make its own smoother time adjustments.

It may be appropriate to synchronize some systems with widely available GPS technology. While it can provide clock pulses with 100 *ns* accuracy, system noise generally limit the clock synchronization between hosts to around 10 μs. Additionally, this style of synchronization operates with less administrator intervention. Unfortunately, this technology requires roof access and is cost prohibitive.

Paxson [50] has shown that the adjustments made by NTP do not help systems make accurate measurements of packet transfer times, and can in fact be detrimental. In fact, [49, 51] find that clock rate stability is more important than synchronization, and prefer using the CPU's more stable Time Stamp Counter (TSC) to the system's real-time clock to measure time differences. Later research [59] focused on creating an integrated solution that meets the need for absolute time provided by NTP and accurately measuring time differences on small scales.

3.2 Measurement Tools and Standards

System administrators traditionally use ping [43] to measure RTT, and descendants of ttcp [44] (nuttcp [15], netperf [27], iperf [2], etc.) to measure throughput and jitter (delay variation). Tcpdump and libpcap are used when detailed packet capture and tracing is required, though packet capture at Gigabit and higher speeds starts becoming impractical [57].

UDPmon [18] is well suited for measuring one way delays (OWDs) due to its use of the CPU's stable TSC, rather than the real-time system clock that most other tools use. Its sister tool, ethmon, provides identical functionality implemented on raw sockets. The benefit of using the ethmon tool is that the send and receive timestamps should be more accurate since there is no system level queueing effects between the driver and the application. In either case, the effects of interrupt throttling in driver code is clearly seen as increased variation in OWDs.

Standards for network metrics have slowly taken shape. The Internet Engineering Task Force (IETF) produced a definition for IP Packet Delay Variation (IPDV) [13] in order to reduce the confusion caused by multiple meanings for the concept of jitter. More recently, the IETF put out RFC4656, A One-way Active Measurement Protocol (OWAMP) [58]. OWAMP assumes that clocks are synchronized with GPS or NTP, and uses a schedule that both the sender and the receiver know to gather comprehensive measurements. The development is driven partly by abuse of the ping utility by malicious entities, and the observed need for a more secure alternative.

Network researchers have developed many other methods and tools [21, 6, 30, 32, 14] focused on measuring capacity or available bandwidth that have not yet become widespread. Many of these tools use deterministic models with filtering on measurements taken with one, two, or more packets. Some of the tools are rendered ineffective because they make assumptions concerning the behavior of queues or ACKs that do not always hold, while others lose effectiveness in the presence of cross traffic.

The current state of the art, embodied by Pathload [24], is to use packet trains to measure one way delay (OWD). This allows constantly changing available bandwidth to be tracked and accurate knowledge of its variance to be developed [23, 25]. With regard to Figure 3.1, $OWD_i = R_i - S_i$. Pathload is unconcerned about clock synchronization because it only cares about the differences between OWDs, so a constant clock offset does not matter. Clock skew during a stream measurement is on the order of nanoseconds during Pathload's short streams, and should be negligible when compared to the effects of queueing in the network. Pathload first partitions and then finds medians for each partition. It then performs trend analysis on its OWD measurements using both the Pairwise Comparison Test and the Pairwise Difference Test. Each test detects different cases of increasing network queueing.

3.3 Measurements in Congestion Control Protocols

TCP uses an estimate of the round-trip time (RTT) between two hosts to decide if an acknowledgement (ACK) packet has been lost. Given a new measurement $RTT = C_i - S_i$ (see Figure 3.1), a new RTT estimate is produced using an alpha beta filter. This notion of RTT is less rigorous than that used by clock synchronization protocols, since it does not take into account the delay on the receive side between R_i and A_i, but TCP's timestamps are more accurate than NTP's because they occur in kernel space. The TCP Extensions for High Performance [20] later added timestamps to the header of data and ACK packets so that TCP gathers more RTT measurements as its window size increases. This extension is critical for high bandwidth networks.

New extensions and complete replacements for TCP's congestion control algorithms are continually being created. Packet loss remains the primary signal of congestion, partly due to difficulty experienced when attempting to correlate number of packets in flight with round-trip time [5]. Proponents of delay-based congestion control [48, 26, 4] must still work to address concerns about its effectiveness [54, 55]. It is likely that any new variant will only become successful if it uses a more sophisticated Kalman filter [22] so that its congestion response is not thrown off by noisy measurements.

TCP Santa Cruz [48] stands out from other delay based congestion control protocols because it uses relative forward delay (RFD), the difference between the receive and send times of two packets, rather than RTT. Referring to Figure 3.1, forward delay is defined as $D_{i,j} = (R_j - R_i) - (S_j - S_i)$, and it clearly indicates whether congestion is increasing, decreasing, or static along the forward path. In fact, RFD is exactly the same concept as Pathload's use of the difference between OWDs, just defined several years earlier. Whereas Pathload is rate based and uses its measurements to determine the available bandwidth, TCP Santa Cruz is window based uses its measurements to model the bottleneck switch's queue depth.

The Probe Control Protocol (PCP) [3] uses short probes similar to Pathload to detect the available network bandwidth. PCP uses a least squares fit to determine whether a series of noisy delay measurements are increasing or decreasing. The probability distribution associated with the fit line is used when PCP randomly accept the result of a probe. Send time stamps are improved (compared to Pathload's) by using the same packet capture library that is used by tcpdump.

Linux can switch between multiple TCP variants, and the latest default is CUBIC [17], which focuses on addressing issues TCP has with effectively utilizing links with large bandwidth-delay products. It also employs a hybrid slow start algorithm which uses heuristics inspired by Pathload's techniques for measuring available bandwidth to find a safe exit point from slow start's doubling of the congestion window. Instead of using packet train probes, they use hints from ACK spacing and round-trip delays. They do this because Pathload's probing does not lend itself to incremental deployment, and also because Linux does not provide high-resolution system clocks or real-time interrupt handling.

3.4 Measurement Summary

Table 3.1 summarizes the different types of delay discussed in this chapter. Round-trip times do not require clocks to be synchronized; but they neither separate time due to network or system effects, nor do they differentiate between asymmetric forward and reverse paths. Accurate one way delay measurements require clocks that are synchronized with a stable rate, which are generally not available. RFD measurements can be made with unsynchronized clocks, and are also best made using clocks with stable rates. In all cases, it is best if timestamps can be generated as close to the actual send and receive times as possible in order to differentiate between queueing effects on the host and on the network. Measurements appear noisy because other effects, like interrupt throttling are included. Packet trains are preferable to packet pairs because they allow those effects to be mostly ignored using a filter and trend analysis.

Name	Description
NTP Round-trip Time	$RTT_i = (C_i - S_i) - (A_i - R_i)$
TCP Round-trip Time	$RTT_i = C_i - S_i$
One Way Delay	$OWD_i = R_i - S_i$
Relative Forward Delay	$RFD_{i,j} = (R_j - R_i) - (S_j - S_i)$

Table 3.1: Summary of network resource measurements.

The most useful metric discovered for reasoning about network effects is RFD between two packets. Changes in this delay indicate whether congestion is increasing or decreasing and can be used to determine available bandwidth. Another possible use of RFD would be to increase the number of measurements useful for time synchronization.

RFD measurements isolate network information from clock information. Subtracting positive RFDs from OWDs should remove network queueing affects, at least between the two OWDs. The absolute network queueing effect could possibly be eliminated using a queue model as in TCP Santa Cruz. Improving clock synchronization is beyond the scope of this Storage Network QoS project, but this line of reasoning does warrant further investigation.

Chapter 4

RAD on Networks (Radon)

The Resource Allocation and Dispatching (RAD) scheduling model [7] has proven to be an effective way to provide a range of performance guarantees [36], first for CPU and later for disk [53] resources. Its success is due to the separation of *Resource Allocation*, which answers the question "how much?", from *Dispatching*, which answers the question "when?" The *Resource Allocation* for a given task is specified using a reservation of some fraction of the resource at some granularity period, or more concisely, *Rate*. *Dispatching* schedules the work defined by the *Rate* such that it is finished by the *Deadline* at the end of each period.

The intention of adapting RAD to the network resource is to provide flexible, general, and fine-grained performance guarantees on standard commodity network hardware similar to what CPU and disk resources enjoy under the same general model. One important goal of applying the same scheduling model to every level of the operating system is the desire to compose guarantees system-wide.

4.1 Multiple Dispatchers in RAD

RAD was originally developed for a single resource and single dispatcher. In the case of switched networks, the RAD model must now take into account the existence of multiple dispatchers per resource, where the resource is a transmit port on a switch. The admission process on each receiving host ensures that the aggregate utilization of each switch port is not greater than one. By using the information provided by the resource allocation, dispatchers are able to cooperatively and precisely manage flow control and congestion control for streams of packets in a network and minimize the use of the queue on a switch. The standard definitions for the RAD model apply:

Resource Allocation A task T_i's reservation (u_i, p_i), where u_i is network time utilization and p_i is the length of the period for which u_i is guaranteed.

Dispatching A task T_i has a budget $e_i = u_i \cdot p_i$, and is made up of a sequence of jobs $J_{i,j}$, each possessing a release time $r_{i,j}$ and a deadline $d_{i,j} = r_{i,j} + p_i$.

The major challenge of guaranteeing network resources is to avoid dispatching synchronized bursts of packets while minimizing communication and synchronization overhead. Ideally this means that a host uses only its own information to determine when to dispatch requests. Several variations of Radon will be discussed, but all will focus on using relative forward delay (RFD) in some way to detect congestion, and reservation information to determine the response to that signal.

4.2 Flexibility of RAD

One of the ways the RAD model is flexible is that it allows the reservation to be specified in different types of units. Usually people prefer to specify I/O reservations as the amount of data per period, but that may not be the best way to implement the resource management. A data reservation is not appropriate if the usage pattern of the reservation has a large effect on the rate achieved. For example, disk I/O is greatly effected by sequentiality while network I/O is greatly effected by the size of the individual jobs. If the achieved rates are dependent on the way in which the resource is used, then it is better to implement resource management with time reservations.

Choosing RAD as a general model does not specify whether flow control is implemented in terms of the space between packets or space between, and size of, bursts of packets. From the perspective of the operating system, it would be nice if there were fewer interrupts so that work could be done in batches. However, bursts require larger buffers, introduce large amounts of variation in delay, and make accurate timestamping difficult. Monopolizing an entire CPU's time to handle network traffic has not been considered practical, but many newer systems possess more processor cores than they have work for. Also, new network interfaces are beginning to appear on the market that will ease that load by including support for hardware timestamping. Ideas for managing flow control in terms of windows will be discussed, but the initial implementation of Radon will focus on manipulating the wait time between packets rather than bursts.

4.3 Detection of Congestion and its Severity

TCP Santa Cruz [48] modeled queueing in a switch by summing the RFDs over an interval and dividing that by the average packet service time during that same interval. TCP Santa Cruz was implemented in the ns-2 network simulator, and so did not have to contend with noisy delay measurements. Therefore Radon will detect congestion and determine its severity by combining Pathload's median filter and trend analysis with the TCP Santa Cruz queue model.

Pathload was designed to perform short, active probes of the network, while Radon is intended to measure delays continuously and passively. Where Pathload took the median of several 100 packet trains and then performed its analysis, Radon will take the median of every five delay measurements and perform a continual trend analysis. This should allow rapid reactions to signs of congestion and only require short bursts of computational overhead. In the future, it could be interesting to explore whether inflection points in delay measurements could be determined and whether or not that knowledge could eliminate some work for the trend analysis.

4.4 Response to Congestion

The response to congestion in TCP is a multiplicative decrease in window size, but hard real-time performance guarantees require strictly bounded responses. A congestion control algorithm should also explicitly deal with the incast problem, which particularly afflicts storage networks. Therefore, Radon's response is designed using the

RAD model and traditional real-time scheduling theory.

Scheduling algorithms like Earliest Deadline First (EDF) [37] would require all dispatchers contending for the same resource to know the release times of all jobs so that they can agree on the earliest deadline. Furthermore, the clocks of the dispatchers must be synchronized at a granularity corresponding to the differences between deadlines, and not just at the granularity of the periods. A different scheduling algorithm is clearly needed when a resource is scheduled by multiple dispatchers.

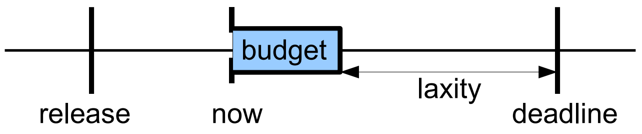


Figure 4.1: Laxity

The Least Laxity First (LLF) [42] scheduling algorithm uses the notion of laxity, depicted in Figure 4.1. The laxity of a job $l_{i,j}$ is defined as the time remaining before the job must be scheduled in order to meet its deadline, $l_{i,j} = d_{i,j} - t - e'_i$, where t is the current time and e'_i is the budget remaining in the period. In contrast with EDF, which schedules based on the deadline a job must be finished, LLF schedules based on the deadline a job must be started. LLF is optimal for scheduling a single resource in the same sense that EDF is, if a feasible schedule exists, then both will find one. Implementing LLF across multiple dispatchers would require just as much communication and synchronization as EDF, but it lends itself to an approximation suitable for distributed dispatchers because the measure of laxity is relative while deadlines are absolute.

Two distributed approximations to LLF, referred to as Percent Budget and Less Laxity More (LLM), are now presented. They were developed with window based flow control in mind. While no congestion is detected, streams of packets are transmitted as fast as possible up to the amount allowed by the budget. When congestion is detected, each sender will use a normalized notion of laxity to determine its change in window size. Let the *%laxity* of a job be defined as $\frac{l_{i,j}}{d_{i,j} - t}$, or the time until the job must start divided by the time remaining until the deadline. Note that another way to think of the relative urgency of a job is according to its *%budget*, which is in fact $(1 - \%laxity) = \frac{e_i}{d - t}$.

Flow Control Budget (in packets) $m_i = e_i / pktS$, where $pktS$ (s/packet) is the worst case packet service time

Congestion Control Windows adjusted in size and dispatch time

Percent Budget Window Target $w_{op} = (1 - \%laxity) \cdot w_{max}$

Less Laxity More Window Target $w_{op} = \min\left(m_i, \max\left(\frac{w_{max}}{l_{i,j} + 1}\right), 2\right)$

Size Change $w_{change} = \frac{-|w_i - w_{op}|}{2}$

Dispatch Offset $w_{offset} = \frac{N_{obs}}{pktS} \cdot \text{rand}$

Where w_i is the current window size and N_{obs} is the depth of the bottleneck switch's queue modeled using observations of relative forward delay. The resulting window size is also obviously bound by the minimum window size and the remaining budget.

Even if individual hosts do not know who among them has the least laxity, they can cooperatively control congestion using the relative measure of their own laxity or budget. Nothing has been formally proven concerning the guarantees made by LLM, Percent Laxity, or Percent Budget.

Due to limitations of the available hardware and the decision to implement a Radon prototype in userspace, the quality of transmission timestamps are problematic, especially for window-based algorithms. On the other hand, a rate-based algorithm provides a userspace program the opportunity to generate and use timestamps more easily. For this research to become practical, either an entire processor must be free to handle interrupts or the NIC must provide support for timestamping. Also, Radon should be moved down in the software stack, most likely into the network subsystem's queueing discipline code. Here follows the above congestion control ideas adapted to a rate-based approach.

Flow Control Budget (in packets) $m_i = e_i/pktS$, where $pktS$ (s/packet) is the worst case packet service time

Congestion Control Rate adjusted by changing wait time between packets

Percent Budget Packet Wait Time $w_{op} = \dfrac{wait_{min}}{\%budget}$

Size Change $w_{change} = \dfrac{-|w_i - w_{op}|}{2}$

New Packet Wait Time $wait_{new} = \min(wait_{max}, \max(wait_{min}, w_{change}))$

Where w_i is the current packet wait time.

4.5 Related Work

Radon's approach belongs to the family of cooperative end-to-end protocols rather than methods that require router involvement [11, 29]. TCP/IP is the most widely deployed end-to-end network protocol, but its congestion control algorithms [19] cannot provide any (real-time) guarantees. Its use of packet loss as a congestion signal results in oscillating throughput and a large variance in packet delay. Instead, real-time audio and video network applications have built their protocols on top of the basic datagrams provided by IP or UDP [10], but their protocols are unaware of congestion and unsuitable for storage.

A simple cooperative traffic shaping [38] approach limits the size of the queue used in an Ethernet switch, and subsequently provides real-time guarantees. However, this technique fails to utilize more than half of the theoretical capacity of Gigabit Ethernet and requires hosts to know the size of the switch buffer.

Some variants of TCP [48, 26] use delay measurements to control congestion and can successfully limit the number of packets in a bottleneck queue. TCP Santa Cruz [48] can differentiate between congestion on the forward and reverse paths, while FAST TCP [26] can achieve weighted proportional fairness.

PCP [3] uses probe packets to detect if the network can currently support a specific load and converges to a desired throughput using short, paced, high-rate bursts. PCP is shown to outperform traditional TCP in various ways including response time and loss rate, and recovers from incast after some packet loss. Despite all of those positive features, PCP is still a best-effort protocol.

In addition to scheduling a switch's transmit ports, it is clear that the both the sending and receiving hosts must schedule the use of their NICs by various processes on the system. VRE-NET [8] and Netnice [46] deal with the local resource management problem. Eventually, a hierarchical scheduler must solve the local and distributed network resource management problems together.

Chapter 5

Evaluation of Radon

The following experiments evaluate a prototype of a rate-based Radon algorithm on older, less capable hardware than is now available. While these results have been superseded by experiments on later hardware, they do illustrate the promise of the Radon algorithms better than experiments done on newer hardware. This is because the older hardware was more resource constrained, so Radon had more opportunity to show improvement. Later experiments done on systems with 64-bit CPUs, more capable NICs, faster and wider pathways to the NIC and memory, and more mature switching technology easily handled the experimental workloads both with and without congestion control. On reflection, it seems appropriate that Gigabit Ethernet technology would finally become reliable when 10 Gigabit products begin to enter the mainstream.

5.1 Setup

The experiments were performed on a cluster of seven nodes, each node possessing an Intel ® Celeron ® CPU 2.53 GHz processor and an 82541PI Gigabit adapter. The cluster's primary network used an Extreme ® Summit 400-24t switch, with all management traffic confined to a separate switch. The Gigabit adapter's interrupt throttling was disabled, and the number of descriptors used for receiving and transmitting packets was set to 4096 and 80 entries, respectively. The number of descriptor entries were set this way in order to minimize the amount of queueing of packets on the card while enabling full link utilization. The Summit 400-24t supports a 9216 Byte maximum packet size (Jumbo Frame), policing or rate limiting on ingress, 802.1q tagging and DiffServ marking, and shaping on egress with eight QoS queues per port. None of the QoS features of the switch were used.

UDPmon [18] was chosen as a basis for the following experiments partly because its code is easy to understand and modify compared to other well known network measurement tools, partly because it uses the stable time stamp counter (TSC) rather than the system's real-time clock, and partly because its sister tool, ethmon, duplicates UDPmon's functionality over raw ethernet sockets. While UDP sockets are easier to use, raw Ethernet sockets bypass abstraction layers and operating system buffers, providing the best opportunity possible for accurate timestamps in a userspace application. UDPmon was modified to call `sched_yield()` immediately after `sendto()` in order to make its send timestamps as accurate as possible. It was also modified to use the timestamp data to model the switch queue depth, and then to increase or decrease the wait time between packets according to the apparent congestion. The packet buffers were kept as small as possible. Unfortunately, 1000 Mbps requires a large receive buffer. This affects TSC timestamps, but not the system's socket timestamps. The socket timestamp is close to actual arrival time, but is based on the system's unstable real-time clock rather than the TSC. In an attempt to get a good comparison of TSC and system timestamps, NTP was turned off in order to increase system clock stability. The system clocks were synchronized at the beginning of each experiment.

Modern operating systems generally throttle the rate at which the network generates interrupts in order to preserve CPU resources. Fortunately, some network drivers allow interrupt throttling to be explicitly set. In this case, the Intel E1000 driver was forced to generate an interrupt for every packet. Jumbo packets were used in order to match filesystem block size. An individual 4KB packet can be sent faster than an 8KB packet, but two cannot, and the system can only approach gigabit speeds by using the larger packet size. It requires half as many interrupts. Although packets can be made as large as 9KB, that can turn out to be a poor choice due to memory page size and limitations of the experiment system's older 66 MHz PCI bus.

5.2 Single Stream

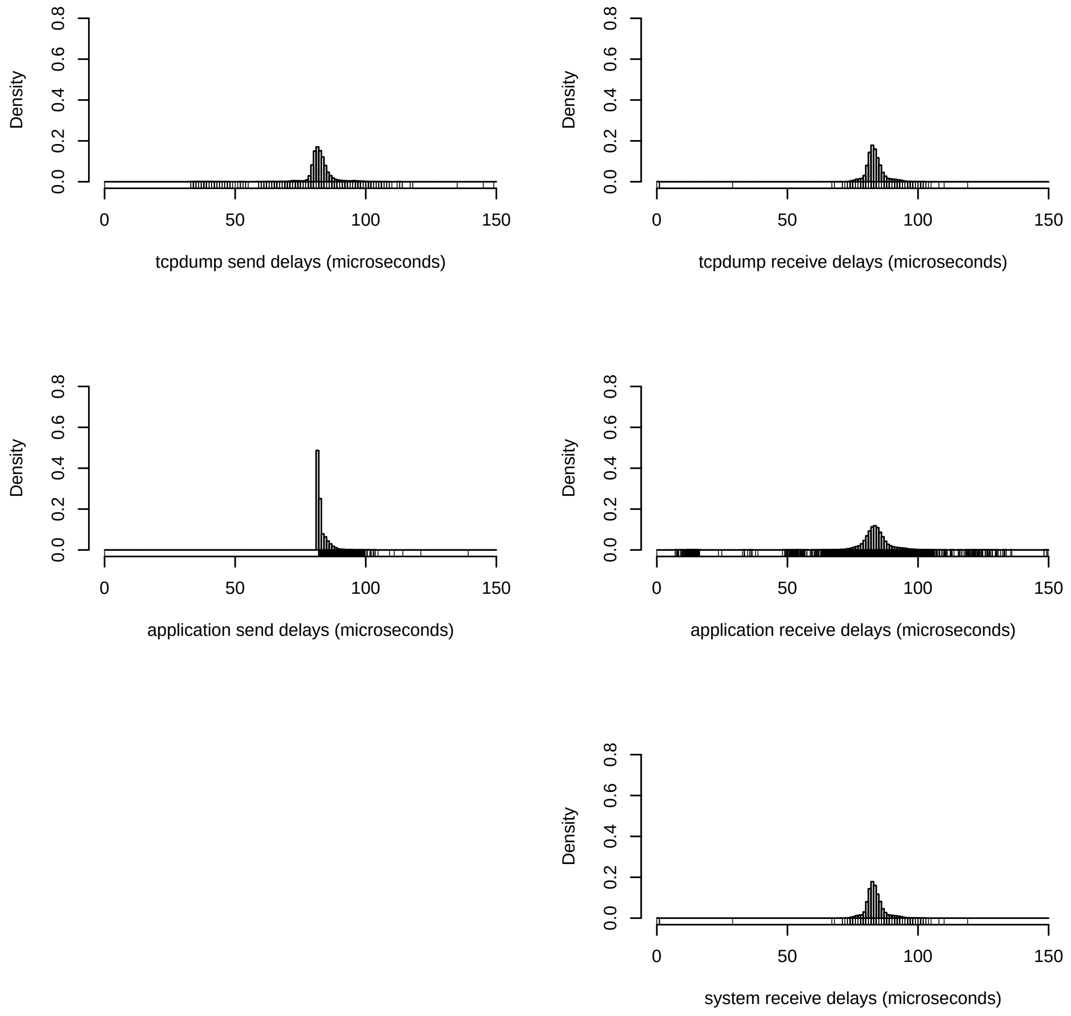


Figure 5.1: Histograms comparing the quality of timestamp measurements

Figure 5.1 shows the difference in the quality of timestamps taken by the `tcpdump` program, by the application itself, and by the operating system's socket time-stamping mechanism. The distributions of the measured delays between two packets is most similar between `tcpdump` and the socket timestamp because the packets are time stamped at a similar point close to the hardware. The application's send delay distribution is one-sided because the application never sends a packet until it has waited the amount of time specified on its command line. Its receive delay distribution is broad because it is measuring operating system queuing effects on both the client and the server.

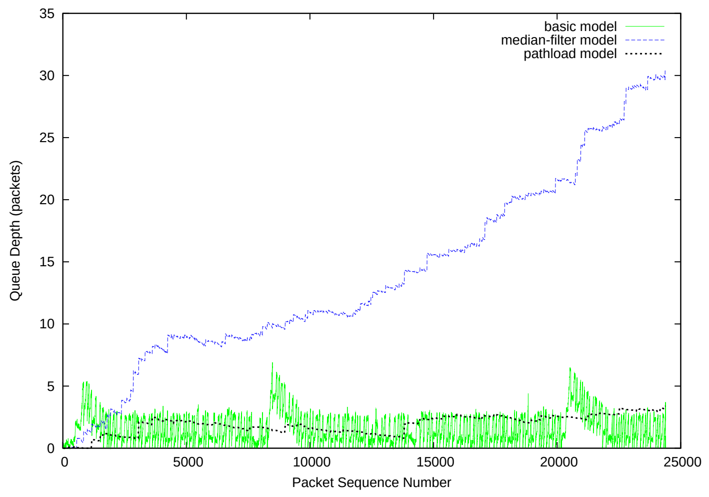


Figure 5.2: Comparison of queue modeling for a single network stream

Figure 5.2 shows a best case stream between a client and a server with a rate of 765 Mbps, or 84 μs between packets. A single stream cannot cause collisions, so the switch queue does not fill, and the models reflect the the variation in inter-packet delay due to the operating systems of the client and server. Both the basic and Pathload models show a consistently small queue depth, while the median-filter model has an unreasonable, ever-increasing trend.

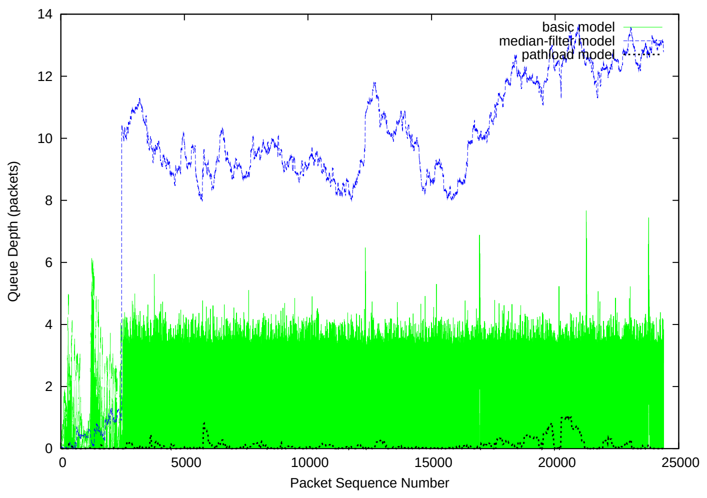


Figure 5.3: Comparison of queue modeling for a single adaptive network stream

Figure 5.3 shows the same best case stream as above, but it changes the wait time between packets according to the modeled depth of the median-filter queue model. The result was an average rate of 739 Mbps and 87 μs between packets. All of the models show a smaller queue depth, and the median-filter model no longer has an unreasonable increasing trend.

5.3 Single Stream Punctuated

In Figures 5.4 and 5.5, there is one longer running stream punctuated by five other short streams throughout its lifetime. The first stream attempts to send packets with a 86 μs inter-packet wait time, or 749 Mbps, and the others attempt to use the theoretically remaining bandwidth at different times with a packet wait time of 257 μs. A user might expect this load to show some queue buildup on the switch and rarely drop packets since the total offered load never exceeds the capacity.

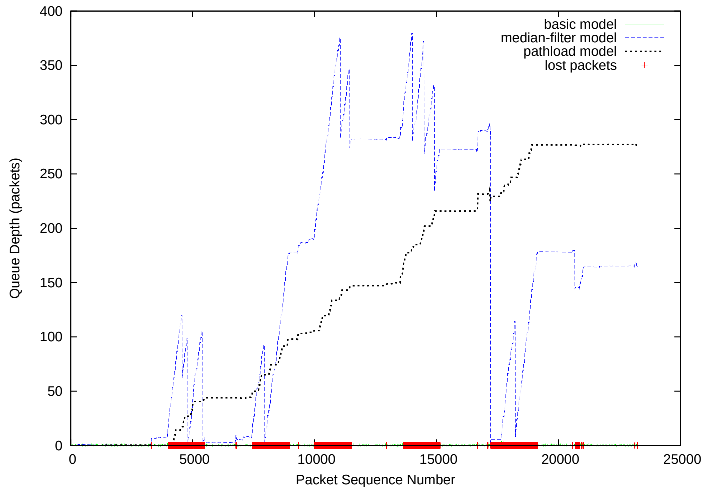


Figure 5.4: Comparison of queue modeling for a punctuated stream

The lost packets clearly mark the duration of each short stream in Figure 5.4. The primary stream lost 24% of its packets and the each of the others lost approximately 4% of theirs. The streams in Figure 5.5 adapt their inter-packet wait time according to the median-filter queue depth model, achieving good results with regard to preventing most packet loss. Tables 5.3 and 5.3 show that primary stream lost only 2.5% of packets and the others each lost one out of 500. On one hand, it is unfortunate that the congestion control favored the primary stream over the streams with less demanding reservations. On the other hand, this indicates that a reservation with a very short inter-packet latency requires a significant amount of overhead, about 20% of the link capacity in this case. This observation fits well with the experience of the other RAD-based schedulers.

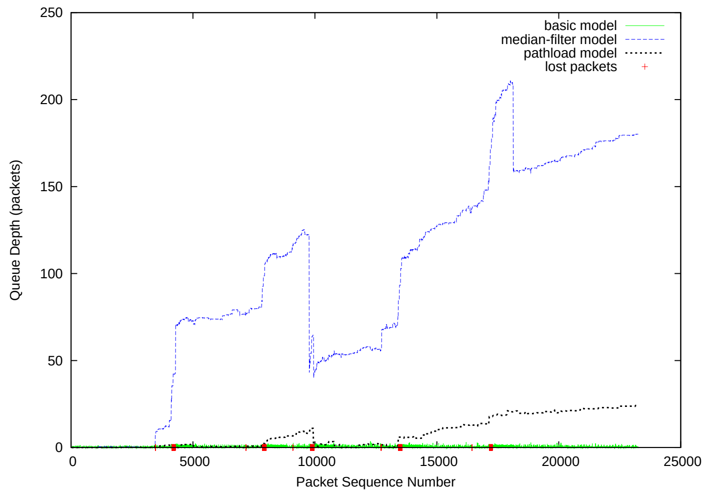


Figure 5.5: Comparison of queue modeling for a punctuated adaptive stream

It is interesting to note that the Pathload queue model shows an always increasing trend in figure 5.4 due to its need to analyze timestamps in groups of 100. The median-filter model only looks at 5 timestamps at a time, so it reflects the filling and

Node	send rate (Mbps)	num recv	num lost	%lost	recv rate (Mbps)	recv wire rate (Mbps)
1	747.117	17571	5652	24.0	565.489	570.131
2	250.080	481	19	3.8	245.459	247.474
3	250.107	478	22	4.4	244.248	246.253
4	250.128	477	23	4.6	244.240	246.245
5	250.109	478	22	4.4	240.838	242.816
6	250.159	481	19	3.8	238.908	240.870

Table 5.1: Summary of rates and losses for a punctuated stream.

Node	send rate (Mbps)	num recv	num lost	%lost	recv rate (Mbps)	recv wire rate (Mbps)
1	743.848	22666	589	2.5	725.206	731.159
2	1.476	499	1	0.2	1.478	1.490
3	1.476	499	1	0.2	1.478	1.490
4	1.476	499	1	0.2	1.478	1.490
5	1.476	499	1	0.2	1.478	1.490
6	1.476	499	1	0.2	1.478	1.490

Table 5.2: Summary of rates and losses for a punctuated adaptive stream.

draining of the queue more quickly. The basic model stays close to zero. It does not filter out any noisy measurements, and does not appear to give strong indications of congestion.

The statistics available on the switch were useless with regard to verifying the accuracy of the different models. It did not appear to count the number of packets dropped for each transmit port, since the tool always reported zero even when a port was deliberately offered a load that exceeded its capability.

Node	send rate (Mbps)	num recv	num lost	%lost	recv rate (Mbps)	recv wire rate (Mbps)
1	163.197	4925	151	3.000	158.403	159.703
2	163.153	5056	20	0.390	162.574	163.909
3	163.181	5071	5	0.099	163.077	164.416
4	163.189	5073	3	0.059	163.159	164.498
5	163.115	5072	4	0.079	163.063	164.401
6	163.167	5074	2	0.039	163.133	164.473

Table 5.3: Summary of rates and losses for six fairshare clients.

Node	send rate (Mbps)	num recv	num lost	%lost	recv rate (Mbps)	recv wire rate (Mbps)
1	163.127	5035	41	0.810	161.836	163.165
2	163.216	5058	18	0.350	162.705	164.041
3	163.017	5074	2	0.039	163.019	164.357
4	163.130	5072	4	0.079	163.063	164.402
5	163.135	5073	3	0.059	163.136	164.475
6	163.078	5073	3	0.059	163.017	164.355

Table 5.4: Summary of rates and losses for six adaptive fairshare clients.

5.4 Six Fairshare Streams

Figures 5.6 and 5.7 show six clients sending data to the same server at a rate of 166 Mbps and 394 μs wait times between packet transmissions. Tables 5.4 and 5.4 show that the adaptive steams experience a reduction in the amount of packet loss as in the previous experiments. After an initial loss of packets in primarily the first and second streams, both the adaptive and non-adaptive streams rarely lose packets. With packets spaced apart as far as they are, it is obvious there is simply little chance of congestion causing packet loss. In fact, almost all clients detect a stabilized queue depth after approximately 4000 packets.

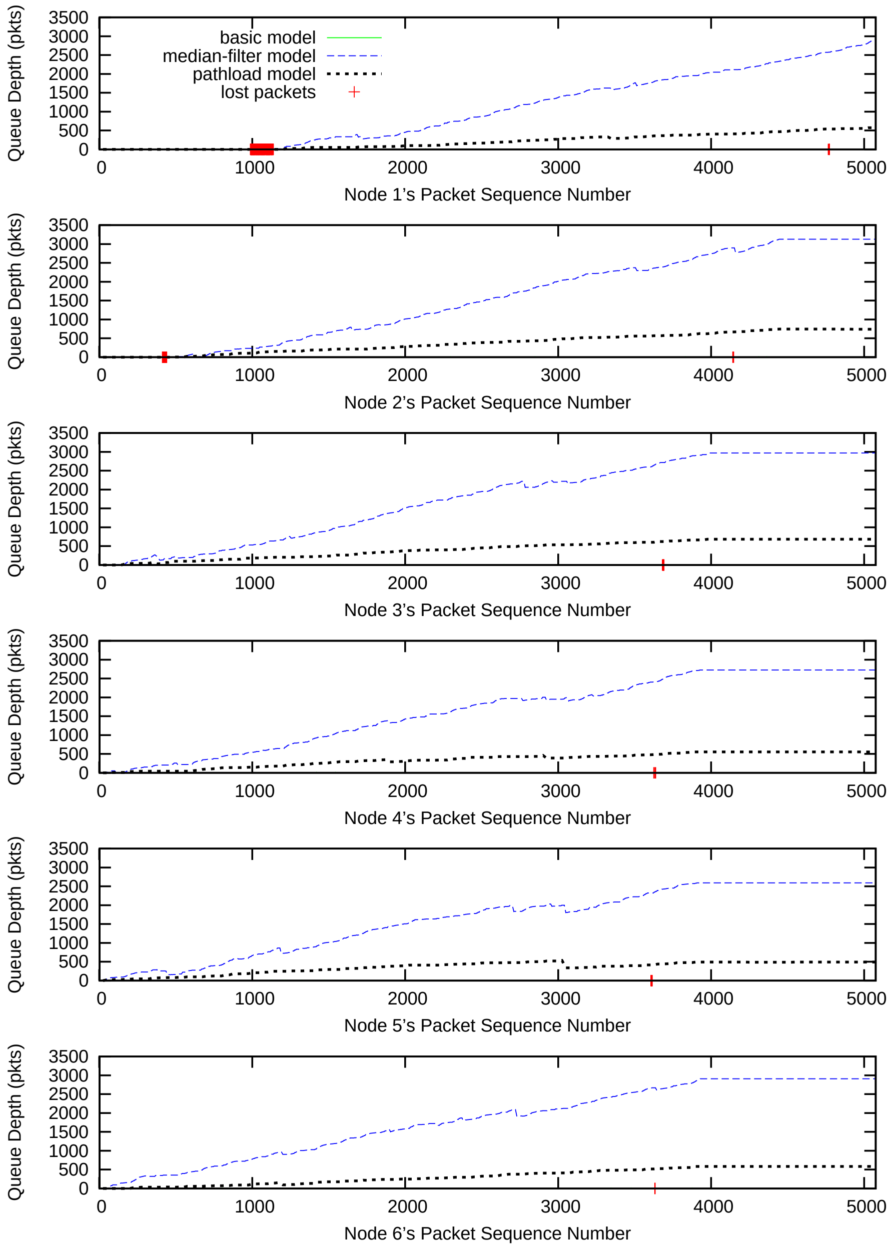


Figure 5.6: Comparison of queue modeling for six fairshare streams

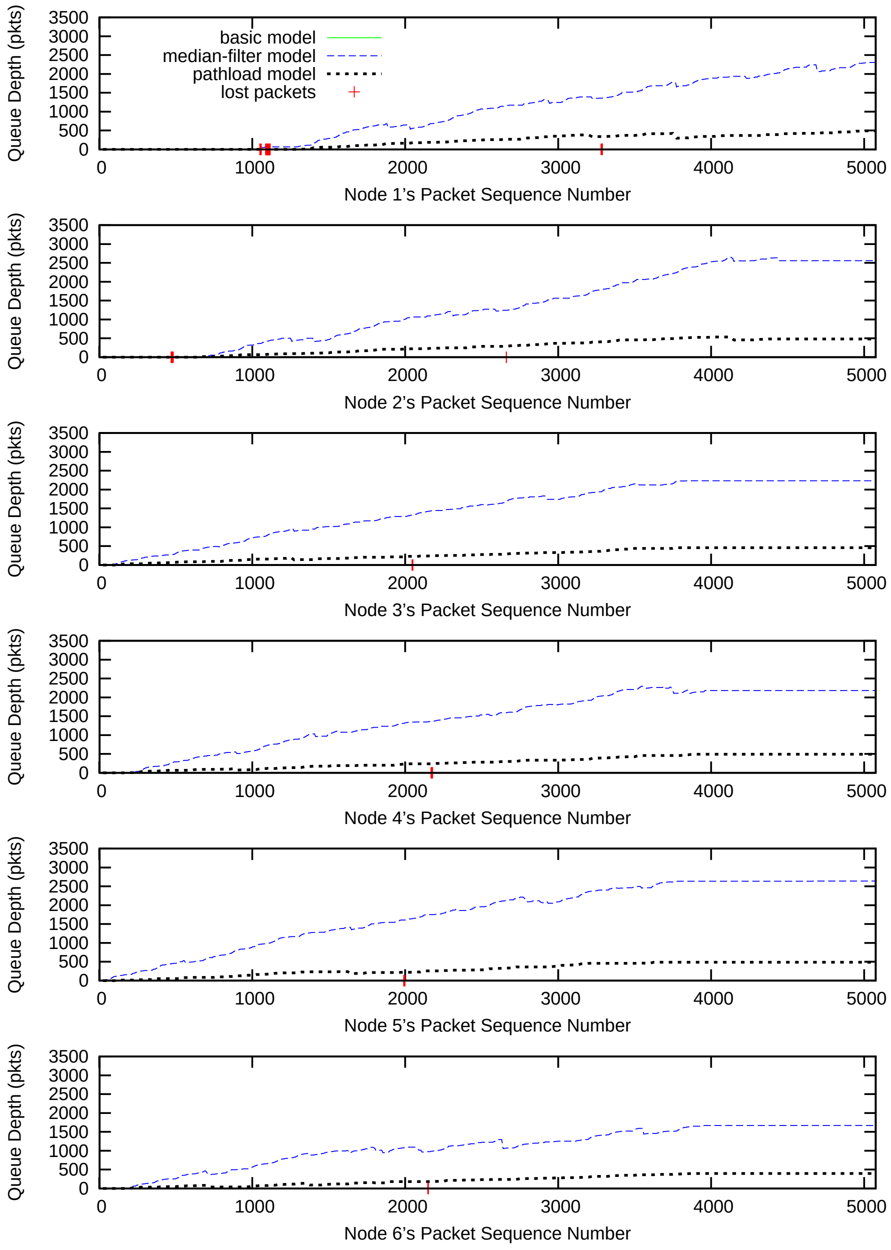


Figure 5.7: Comparison of queue modeling for six fairshare adaptive streams

5.5 Six Unfair Streams

The previous experiments explore the effectiveness of Radon when there is one primary stream, or where all streams are of equal importance. This final experiment, whose streams are described in Table 5.5, tests Radon with a mixed workload. Together, the sum of the lesser streams' reservations equal the primary stream's reservation. Also, only the two smallest streams share the same rate.

Node	Inter-packet Wait Time (μs)	Send Rate (Mbps)
1	131	500.00
2	262	250.00
3	524	125.00
4	1048	62.5
5	2097	31.25
6	2097	31.25

Table 5.5: Inter-packet wait times and send rates in the unfair experiment.

In this experiment, the adaptive streams did not fair much better than the non-adaptive streams. Tables 5.5 and 5.5 show a slight reduction in packet loss was observed, but Radon was unable to avoid 35% packet loss in the primary stream. This could be due to the fact that none of the link capacity was set aside for overhead. Another possible reason for the poor results could be related to incast since the inter-packet wait times were multiples of each other.

Other than slightly fewer lost packets, Figures 5.8 and 5.9 are remarkably similar. Note that although the X axes show a different number of packets for each stream, those packets were all transmitted over approximately the same two second

Node	send rate (Mbps)	num recv	num lost	%lost	recv rate (Mbps)	recv wire rate (Mbps)
1	490.617	9507	5759	38.0	305.618	308.127
2	245.311	7454	179	2.3	239.648	241.615
3	122.704	3813	3	0.1	122.675	123.682
4	61.358	1906	2	0.1	61.358	61.861
5	30.672	953	0	0.0	30.728	30.980
6	30.663	951	2	0.2	30.637	30.888

Table 5.6: Summary of rates and losses for six unfair streams.

Node	send rate (Mbps)	num recv	num lost	%lost	recv rate (Mbps)	recv wire rate (Mbps)
1	489.946	9928	5333	35	318.661	321.277
2	245.215	7477	156	2	240.278	242.250
3	122.693	3816	0	0	122.823	123.831
4	61.352	1908	0	0	61.398	61.902
5	30.661	953	0	0	30.734	30.986
6	30.672	953	0	0	30.707	30.959

Table 5.7: Summary of rates and losses for six adaptive unfair streams.

interval. The timelines are proportionally correct, but should not be interpreted to be precise in relation to each other.

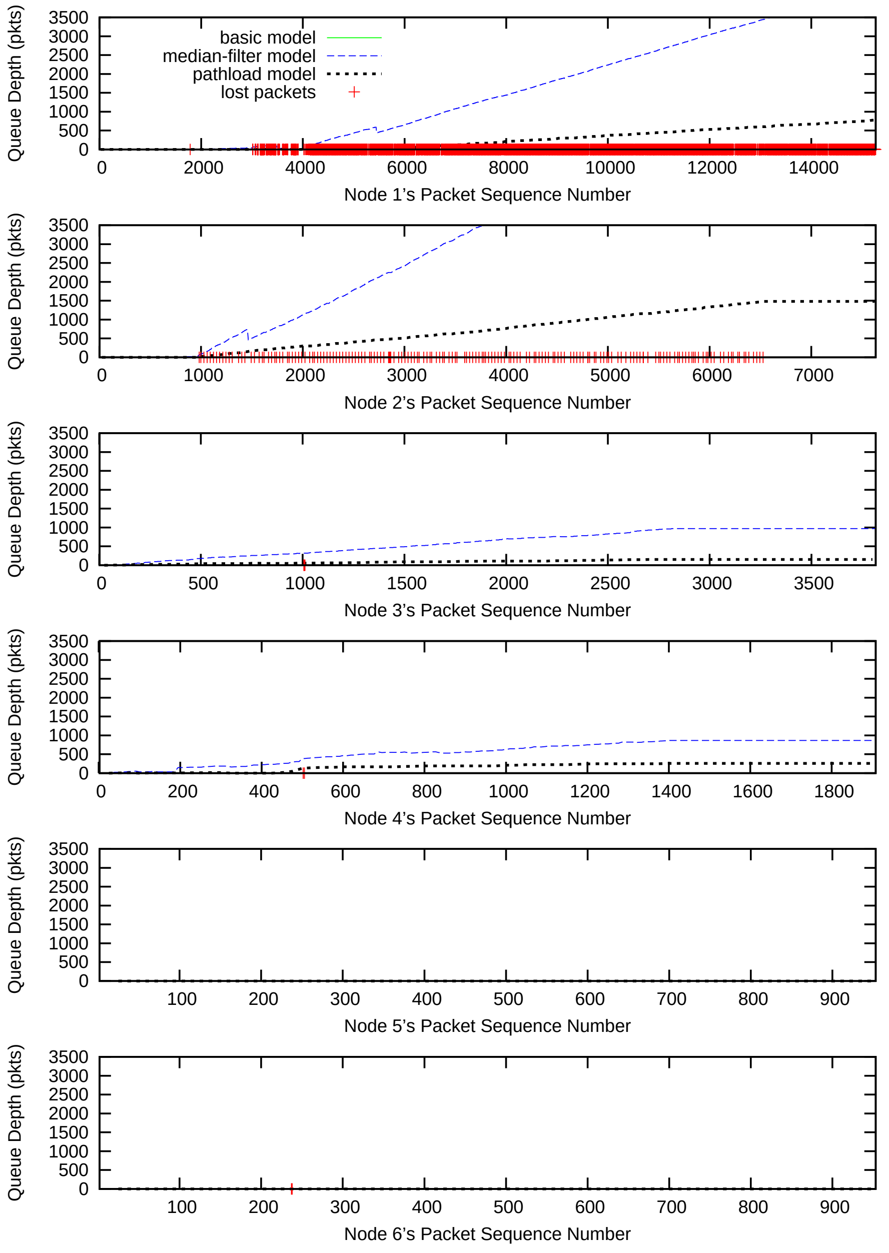


Figure 5.8: Comparison of queue modeling for six unfair streams

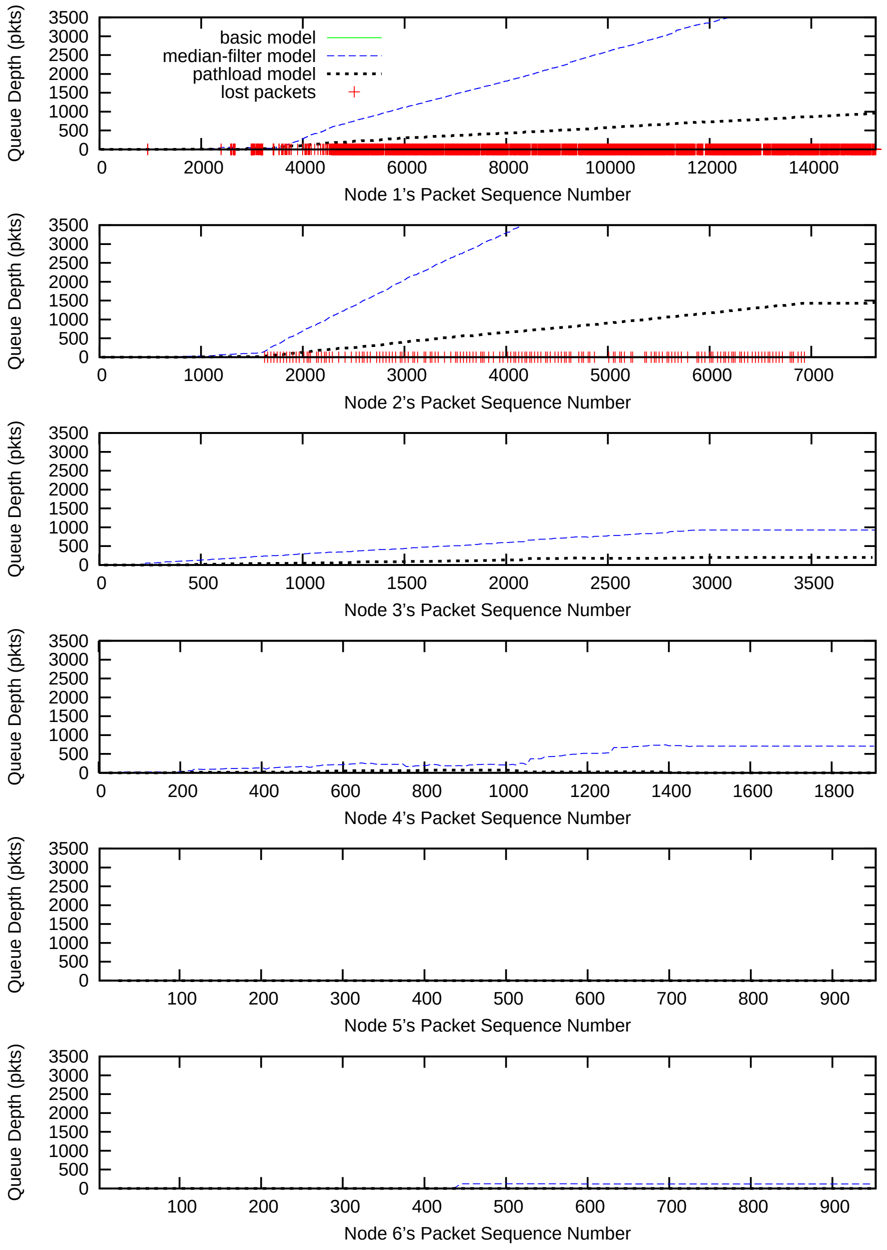


Figure 5.9: Comparison of queue modeling for six unfair adaptive streams

5.6 Conclusion

The initial prototype of a rate-based Radon algorithm demonstrates that a host can detect congestion by measuring changes in relative forward delay and respond in such a way to avoid a large amount of packet loss. While TCP responds to congestion with a multiplicative decrease in window size, Radon responds according to a definition of urgency derived from real-time scheduling theory. Radon's window-based variant is yet to be prototyped successfully due to the difficulty of attaining accurate transmission timestamps from userspace. When it is implemented, it will be an improvement over the rate-based variant because it should make more efficient use of the hardware and it explicitly deals with incast by offsetting the dispatch time of the next window.

Chapter 6

Proposed Research

This paper proposes that RAD real-time scheduling theory can be applied to network resource management in a way that will be a significant improvement over the state of the art.

Current storage networks do not provide end-to-end Quality of Service (QoS). The intention of this research is to produce both a formal theory and real-world implementation of flexible, general, and fine-grained performance guarantees on standard commodity network hardware. These guarantees will be composable with guarantees made for CPU, disk, and memory resources such that end-to-end QoS on storage networks can be provided.

6.1 Research Questions

There are two main questions that need to be answered about Radon, and they can expanded into many individual questions.

1. What can be proven about Radon?
    (a) Can it be proven that buffer bounds are not violated?
    (b) Can it be proven that buffer requirements on bottleneck links are minimized?
    (c) Can it be proven that Radon is able to be used to provide guarantees on top of credit-based flow control?
2. Is Radon practical in the real world?
    (a) Can flexible RAD-based scheduling algorithms improve upon conventional priority-based scheduling on networks?
    (b) What are the CPU requirements of Radon?
    (c) Do RAD reservations lead to any promising new routing algorithms in complex network topologies?
    (d) Can Radon be composed with RAD-based schedulers for other resources to provide end-to-end QoS?

6.2 Contributions

Expected contributions from this work are as follows:

1. A proven theory of efficient, real-time network performance management for both reliable and unreliable transports
2. An accurate network simulation of the theory

3. An implementation of the theory that is useable by real-world applications

Furthermore, systems built using these methods will avoid over-provisioning, virtualize the performance of remote storage access, and experience improved reliability and administration over more traditional solutions.

6.3 Research Plan

Network Calculus combined with real-time scheduling theory and simulation will be used to address the first set of research questions asking what can be proven about Radon. Then questions about the practicality of Radon will be answered with a real-world implementation and thorough experimentation. Theory and practice will be tied together by running the same experiments in both simulation and on actual hardware. An estimated six months of effort in theory and simulation should yield answers to the first set of research questions, with another nine months required to both develop a real-world implementation and compare it to the simulation results.

6.3.1 Theory

Le Boudec's Network Calculus [34] can be used to formally prove the buffer bounds of the Radon algorithms. Informally, Radon limits the amount of data it transmits during a period so the worst case switch buffer requirement is the sum of the burst sizes from all streams. It will be more difficult to prove whether or not Radon minimizes the buffer size. It would also be interesting to attempt use model checking and automated theorem proving tools.

6.3.2 Simulation

Accurate switch models will be developed for use with the NS-3 simulator.

Earlier research into Radon attempted to create multiple implementations for both simulators and real world systems. On the one hand, a simulator promises the ability to inspect all levels of the system and allows complex systems to be investigated inexpensively. On the other hand, simulators may not be realistic enough. The simulators investigated so far possess extremely limited switch models, so accurate modules for switches and reliable transports must be developed first. Granted, TCP Santa Cruz utilized the NS-2 simulator to simulate the depth of the switch's queue depth, however it is not currently well regarded, and is being superseded by NS-3.

6.3.3 Implementation

It is critical to replicate simulation results with a real-world implementation. Previous work attempted to produce a userspace Radon and achieve limited success. More work was done in attempt to create a Linux Kernel queueing discipline (qdisc). However, the qdisc infrastructure is intended for self-contained traffic shaping strategies whereas Radon requires external feedback. So, the next attempt to create a real-world implementation should be as a congestion control plugin, either for the Datagram Congestion Control Protocol (DCCP), TCP, or possibly something implemented in the Infiniband software stack.

6.3.4 Experiments

The simulated and real-world experiments comparing Radon to kernel-based IP traffic shaping and Infiniband QoS will require additional thought. Certainly, experiments similar to those performed earlier will be repeated. However, a more portable and powerful testing framework is necessary in order to make the comparisons valid. Pakin's coNCePTuaL [47] appears to be a framework that will effectively allow different software and hardware stacks to be compared, once additional plugins are developed.

Bibliography

[1] Precision time protocol, IEEE standard 1588, 2002.

[2] Iperf the tcp/udp bandwidth measurement tool, 2003.

[3] Thomas Anderson, Andrew Collins, Arvind Krishnamurthy, and John Zahorjan. PCP: efficient endpoint congestion control. In *nsdi06*, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[4] Sumitha Bhandarkar, A. L. Narasimha Reddy, Yueping Zhang, and Dimitri Loguinov. Emulating aqm from end hosts. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 349–360, New York, NY, USA, 2007. ACM.

[5] Saad Biaz and Nitin H. Vaidya. Is the round-trip time correlated with the number of packets in flight? In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 273–278, New York, NY, USA, 2003. ACM.

[6] J. Bolot. Characterizing end-to-end packet delay and loss in the internet, 1993.

[7] Scott A. Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, pages 396–407, December 2003.

[8] Hui Cheng and Steve Goddard. Vre-net: A qos-supported network subsystem for multimedia applications. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*, page 113, Washington, DC, USA, 2006. IEEE Computer Society.

[9] Wu chun Feng and Peerapol Tinnakornsrisuphap. The adverse impact of the TCP congestion-control mechanism in heterogeneous computing systems. In *International Conference on Parallel Processing*, pages 299–306, 2000.

[10] David D. Clark. The design philosophy of the darpa internet protocols. pages 54–62, 1992.

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

[12] Kendall Correll, Nick Barendt, and Michael Branicky. Servo design considerations for software-only implementations of the ieee 1588 precision time protocol. In *Proceedings of Conference on IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2005.

[13] C. Demichelis and P. Chimento. Ip packet delay variation metric for ip performance metrics (ippm), 2002.

[14] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Trans. Netw.*, 12(6):963–977, 2004.

[15] Bill Fink. Nuttcp tcp/udp measurement tool, 2007.

[16] Sangtae Ha and Injong Rhee. Hybrid slow start for high-bandwidth and long-distance networks. *PFLDnet*, 2008.

[17] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

[18] Richard Hughes-Jones, Peter Clarke, and Steven Dallison. Performance of 1 and 10 gigabit ethernet cards with server quality motherboards. *Future Gener. Comput. Syst.*, 21(4):469–488, 2005.

[19] V. Jacobson. Congestion avoidance and control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 314–329, New York, NY, USA, 1988. ACM.

[20] V. Jacobson, R. Braden, and D. Borman. Tcp extensions for high performance, 1992.

[21] Van Jacobson. pathchar—a tool to infer characteristics of internet paths, 1997.

[22] Krister Jacobsson, Hkan Hjalmarsson, Niels Mller, and Karl Henrik Johansson. Estimation of rtt and bandwidth for congestion control applications in communication networks. In *43rd IEEE Conference on Decision and Control (CDC04)*, 2004.

[23] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput, 2002.

[24] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth, 2002.

[25] Manish Jain and Constantinos Dovrolis. Ten fallacies and pitfalls on end-to-end available bandwidth estimation. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 272–277, New York, NY, USA, 2004. ACM.

[26] C. Jin, D. Wei, and S. Low. Fast tcp: Motivation, architecture, algorithms, performance, 2004.

[27] Rick Jones. Netperf tcp/udp measurement tool, 2005.

[28] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[29] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications,technologies, architectures, and protocols for computer communications*, pages 89–102, New York, NY, USA, 2002. ACM.

[30] Srinivasan Keshav. Packet-pair flow control. Technical report, Murray Hill, New Jersey, 1994.

[31] Elie Krevat, Vijay Vasudevan, Amar Phanishayee, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems. In *Proc. Petascale Data Storage Workshop at Supercomputing'07*, November 2007.

[32] Kevin Lai and Mary Baker. Measuring link bandwidths using a deterministic model of packet delay. *SIGCOMM Comput. Commun. Rev.*, 30(4):283–294, 2000.

[33] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[34] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet.* Springer-Verlag, Berlin, Heidelberg, 2001.

[35] Charles E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, 1985.

[36] Caixue Lin, Tim Kaldewey, Anna Povzner, and Scott A. Brandt. Diverse soft real-time processing in an integrated system. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS 2006)*, pages 369–378, Rio de Janeiro, Brazil, December 2006.

[37] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[38] J. Loeser and H. Haertig. Low-latency hard real-time communication over switched ethernet, 2004.

[39] Josip Loncaric. Btime cluster-wide clock synchronization, 2005.

[40] Dave L. Mills. Network time protocol (version 3) specification and implementation. Network Working Group Request for Comments: 1305, March 1992.

[41] Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15:217–252, 1997.

[42] Aloysius K. Mok. *Fundamental Design Problems of Distributed Systems for Hard Real-time Environment*. PhD thesis, Massachusetts Institute of Technology, May 1986.

[43] Mike Muus. Ping, 1983.

[44] Mike Muus. Ttcp: Tcp test, 1984.

[45] P. Ohly, D. Lombard, and K. Stanton. Hardware assisted precision time protocol. design and case study., 2008.

[46] T. Okumura, M. Moir, and D. Mosse. netnice: nice is not only for cpus-a simple subnetwork bandwidth management scheme. *Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on*, pages 388–395, 2000.

[47] Scott Pakin. The design and implementation of a domain-specific language for network performance testing. *IEEE Trans. Parallel Distrib. Syst.*, 18:1436–1449, October 2007.

[48] Christina Parsa and J. J. Garcia-Luna-Aceves. Improving TCP congestion control over internets with heterogeneous transmission media. In *Proceedings of the 7th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 1999.

[49] Attila Pásztor and Darryl Veitch. Pc based precision timing without gps. *SIGMETRICS Perform. Eval. Rev.*, 30(1):1–10, 2002.

[50] Vern Paxson. On calibrating measurements of packet transit times. *SIGMETRICS Perform. Eval. Rev.*, 26(1):11–21, 1998.

[51] Vern Paxson. Strategies for sound internet measurement. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 263–271, New York, NY, USA, 2004. ACM.

[52] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.

[53] Anna Povzner, Tim Kaldewey, Scott Brandt, Richard Golding, Theodore M. Wong, and Carlos Maltzahn. Efficient guaranteed disk request scheduling with fahrrad. In *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 13–25, New York, NY, USA, 2008. ACM.

[54] R S Prasad, M Jain, and C Dovrolis. On the effectiveness of delay-based congestion avoidance. In *In Proceedings of Second International Workshop on Protocols for Fast Long-Distance Networks*, pages 3–4, 2004.

[55] S Rewaskar, J Kaur, and D Smith. Why dont delay-based congestion estimators work in the real-world. Technical report, Department of Computer Science, UNC Chapel Hill, 2005.

[56] Jamal Hadi Salim, Robert Olsson, and Alexey Kuznetsov. Beyond softnet. In *ALS '01: Proceedings of the 5th annual Linux Showcase & Conference*, pages 18–18, Berkeley, CA, USA, 2001. USENIX Association.

[57] Fabian Schneider, Jrg Wallerich, and Anja Feldmann. Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware. In *In Proceedings of the 8th International Conference on Passive and Active Network Measurement*, pages 207–217, New York, NY, USA, April 2007. Springer-Verlag Berlin Heidelberg.

[58] S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, and M. Zekauskas. A one-way active measurement protocol (OWAMP), 2006.

[59] Darryl Veitch, Julien Ridoux, and Satish Babu. Robust synchronization of absolute and difference clocks over networks. *Accepted for publication, IEEE/ACM Trans. on Networking, to appear June*, 2009.