The Impact of False Sharing on Shared Congestion Management[*]

Aditya Akella, Srinivasan Seshan
Carnegie Mellon University

Hari Balakrishnan
Massachusetts Institute of Technology

Abstract

Several recent proposals for sharing congestion information across concurrent flows between end-systems overlook an important problem: two or more flows sharing congestion state may in fact not share the same bottleneck. In this paper, we categorize the origins of this false sharing *into two distinct cases: (i) networks with QoS enhancements such as differentiated services, where a flow classifier segregates flows into different queues, and (ii) networks with path diversity where different flows to the same destination address are routed differently. We evaluate the impact of false sharing on flow performance and investigate how false sharing can be detected by a sender. We discuss how a sender must respond upon detecting false sharing. Our results show that persistent overload can be avoided with window-based congestion control even for extreme false sharing, but higher bandwidth flows run at a slower rate. We find that delay and reordering statistics can be used to develop robust detectors of false sharing and are superior to those based on loss patterns. We also find that it is markedly easier to detect and react to false sharing than it is to start by isolating flows and merge their congestion state afterward.*

1. Introduction

The predominant model for congestion control on the Internet has been based on TCP connections implementing the slow-start and additive-increase/multiplicative-decrease (AIMD) algorithms [10]. While this model has been quite successful at preventing congestion collapse, it is not optimal when multiple concurrent flows from a sender to a receiver share a bottleneck. These flows compete with each other rather than learn from each other about the properties of the network path (especially visible in the context of the Web), and, as an ensemble, display more aggressive behavior compared to single TCP connections.

A number of studies of this problem have been conducted over the past few years [3, 4, 15, 17, 23], and several interesting proposals have emerged to correct the shortcomings caused by TCP connections implementing congestion control in isolation. These studies all propose *sharing* congestion information across concurrent flows, where a group of flows form a *macroflow* and use common congestion control. These proposals include schemes where the granularity of sharing is a common destination host (more precisely, a host network interface) [3, 4, 5, 15, 23], and those where there is more aggressive sharing of information across all destination hosts on the same IP subnetwork [16, 21, 23]. All the above proposals for shared congestion management assume that the bottleneck links traversed by the flows forming a macroflow are exactly identical. However, this assumption does not always hold true. This is due to *false sharing:* two or more flows sharing congestion state may in fact not share the same bottleneck(s).

There are at least two important classes of situations where false sharing might occur without the sender's knowledge. First, in networks with *service differentiation*, where quality of service (QoS) enhancements are applied to network hosts, and second, in networks with *path diversity*, where multiple flows in the same macroflow are routed along different paths. With the increasing deployment of Differentiated Services and increased interest in multi-path routing techniques, a tacit formation of a macroflow on a per-destination basis is not likely to be correct.

The first potential problem with false sharing is that there is a danger of the slower bottleneck being overwhelmed because the sender infers an incorrect bottleneck rate based on information from a faster flow in the macroflow. The second potential problem is that the faster flow suffers a significant performance penalty, sending much slower than the rate that even a single TCP connection on that path would otherwise achieve. These problems arise because the sender observes different bottleneck bandwidths, round-trip times (RTTs), and packet loss rates for the flows that are sharing congestion control information. As a result, it is possible that false sharing might cause the techniques for shared congestion management to, in fact, be detrimental both to the network and to the applications using them.

This paper analyzes the consequences of false sharing in an attempt to quantify the effects of sharing congestion in-

[*] This research was supported in part by NSF under Grant No.1120094-ANI-0085920 and by an NSF CAREER Award and by DARPA under Grant No. MDA972-99-1-0014.

formation across an ensemble of flows. In particular, we address the following specific questions:

1. Impact. What impact does false sharing have on performance and correctness? Does false sharing compromise congestion control, by causing a bottleneck link to become persistently overloaded? Or does it degrade performance by causing flows to perform significantly worse than if they were not sharing congestion information?

2. Detection. When can a sender detect false sharing between two flows? How do delay differences, reordering rates, and packet loss patterns compare in this regard?

3. Response. How should congestion sharing systems be modified to deal with false sharing? Should the default behavior be to optimistically share information and separate flows if false sharing is detected, or vice versa?

We answer these questions using extensive simulation and analysis. We focus on TCP-like congestion avoidance and control algorithms [2] since they are the dominant set of techniques in use today. (Different congestion control algorithms (e.g., TCP-friendly algorithms [11]) may yield somewhat different results.) For TCP-style window-based congestion control, we find, from analysis and simulation, that persistent overload can be avoided even for extreme false sharing, but that higher bandwidth flows run slower. This throughput reduction can be a severe performance penalty, but one that can be avoided by the sender analyzing packet delay and reordering statistics. We develop and evaluate a comprehensive set of tests for a variety of false sharing cases, building on previous work by Rubenstein *et al.* who describe methods for detecting if two connections share at least one common bottleneck link [20]. We find that delay-based statistics are more robust indicators of false sharing than loss patterns. In the context of implementing these tests in practical congestion sharing systems, we find that it is easier for the sender to detect and react to false sharing than it is to start by isolating flows and enforce sharing afterward. This suggests that the default macroflow construction should be to aggregate flows together, and separate them later if deemed necessary by our techniques.

2. Related Work

False sharing can be a problem with the currently popular Differentiated Services (DiffServ) architecture [7, 9, 14]. In DiffServ, elements in the network can, based on a network-internal policy and the result of packet classification, allocate different bandwidths, impose different loss rates, or provide different latencies to packets belonging to different flows of the same macroflow. In contrast, the Integrated Services architecture involves the active participation of end-systems [8, 24], which eliminates false sharing.

When flows within a macroflow traverse different routes, the potential for false sharing arises as the two flows may not share a bottleneck. Unless *all* their bottlenecks are shared, it is incorrect to share congestion state between them. In practice, there are two important scenarios where path diversity causes false sharing: (i) dispersity routing, and (ii) network address translation (NAT).

The original idea of dispersity routing [12, 13], advocates using different paths between a sender and receiver to improve both performance and reliability. More currently, this idea has been advocated for load balancing within network clouds [22]. There are several granularities over which such techniques can be used: per-packet, per-flow, or per source-destination pairs. If done on a per-packet basis, it interacts badly with TCP's current lack of robustness in the face of persistent re-ordering and is, therefore, discouraged today. When done on a per-flow basis, false sharing is likely, especially if there are bottlenecks among the unshared paths. When done on a per source-destination basis, false sharing can still occur when a sender shares congestion information across multiple destinations.

NATs and firewalls are common in today's Internet infrastructure. When there are multiple possible physical destinations for a single visible destination IP address, false sharing may arise. Although the flows will usually share a common path up to the location where the translation is done, the paths from that point to different eventual destinations may have different performance characteristics.

Our approach to detecting false sharing builds on previous work by Rubenstein *et al.*, who provide an analytic framework for a closely related problem [20]. However, there is one important difference between our focus and theirs. We are interested in whether *all* bottlenecks are shared between two flows on a macroflow, whereas Rubenstein's work is concerned with determining if two flows share *any* bottleneck at all.

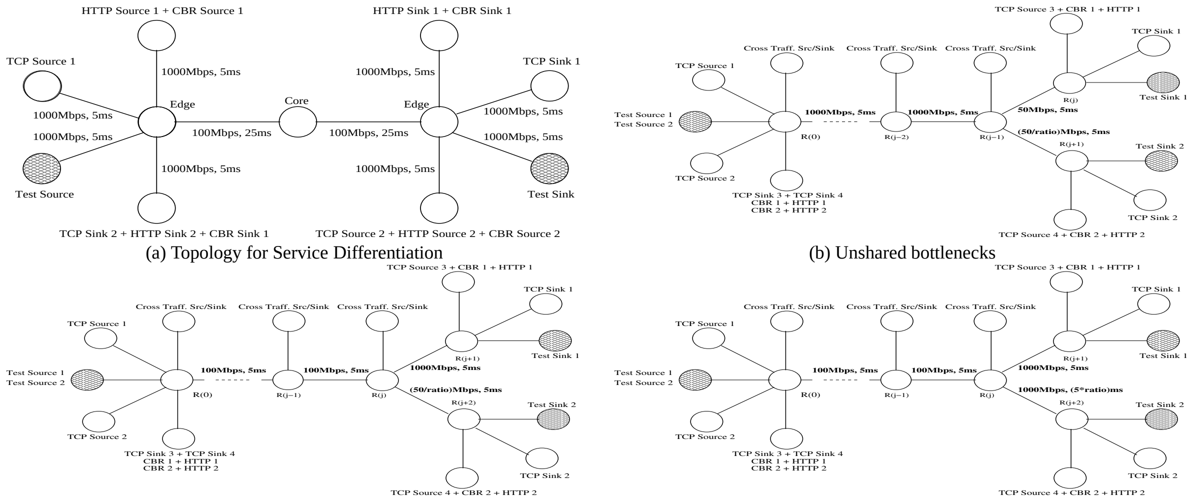
3. The Impact of False Sharing

This section analyzes the performance impact of false congestion sharing. First, we present a brief sketch of our analytical results from a simple model of the observed throughput for two flows that wrongly share congestion information (Section 3.1). We then discuss results from our simulations (Section 3.2).

3.1. Analytical Model

We derive an analytic formula for the observed flow throughput when macroflow-based congestion management is done. The key result we show is the following (this result is easy to generalize):

Consider two paths P_1 and P_2 with high degrees of statistical multiplexing, such that the packet loss rate on each path



(a) Topology for Service Differentiation

(b) Unshared bottlenecks

(c) Semi-shared bottlenecks

(d) Fully-shared bottlenecks

Figure 1. Simulation topologies: In (a), we vary the fraction of bandwidth a allocated to AF flows. In (b) and (c), we vary the ratio of bottleneck bandwidths. In Figure (d) we vary the relative RTTs of the two test paths.

changes negligibly if one flow is added or removed. Consider two flows F_1 and F_2 with propagation RTTs of R_1 and R_2 respectively. Suppose that the flows, in isolation, achieve throughputs of λ_1 and λ_2, on paths P_1 and P_2 respectively. Then, *when they share congestion information and share bandwidth equally, the throughput λ_{share} obtained by each flows F_1 and F_2, in packets/s, is given by*:

$$\lambda_{share} \approx \frac{3R_{min}}{2(R_1+R_2)^2} + \frac{\lambda_1\lambda_2}{\sqrt{\left(\frac{1}{R_1}+\frac{1}{R_2}\right)(\lambda_1^2R_1+\lambda_2^2R_2)}} \tag{1}$$

where $R_{min} = min(R_1, R_2)$.

The complete derivation and discussion of the drawbacks of our analysis can be found in [1]. The following observations about the impact on performance can be made from Equation 1 above:

(1) $\lambda_{share} < \min(\lambda_1, \lambda_2)$. The slower connection is never forced to send at a rate higher than its bottleneck link can sustain. Therefore, for TCP-style congestion control, false sharing does not compromise correct congestion control.

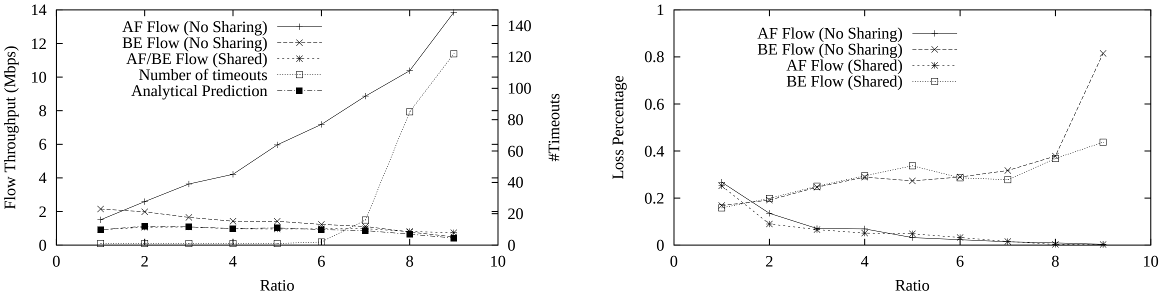
(2) However, false sharing does have an impact on performance. From Equation 1, if $\lambda_1 = k\lambda_2$ and $R_1 = R_2$, then $\lambda_{share} = \lambda_1/\sqrt{2(k^2+1)}$. This shows that the flow that achieved the higher throughput suffers badly under false sharing; for large k, this falls off as $\sim 1/k$.

In the next section, we experimentally show the impact of false sharing on long-lived flows, and validate our experimental results against our analysis. We also show experimental results for the impact on short lived flows.

3.2. Experimental Validation

The first set of our simulation experiments are aimed at verifying the impact of false sharing in networks with service differentiation. For these simulations, we use Nortel's public implementation of DiffServ in ns-2 [6]. We model two traffic classes: an Assured Forwarding (AF) class, and a Best-Effort (BE) class. AF flows are policed and packets exceeding an allocated profile are marked at a higher drop-precedence level by the policer. All downstream routers honor these marks using RIO [7]. Bandwidth is shared using a weighted round-robin (WRR) scheduler between the AF and BE classes, while buffer management uses RED (for BE) and RIO (for AF). (Due to lack of space, queue configuration details are omitted here. See [1] for these details.) Excess bandwidth is shared in proportion to the bandwidth allocated to each traffic class. We assume that all buffers are shared between the two classes and not pre-partitioned.

We use the topology in Figure 1(a). All TCP flows use TCP Newreno. There are $N_a = 10$ AF long TCP connections and $N_b = 40$ BE long TCP connections as background traffic between `TCP Source 1` and `TCP Sink 1`. There are 40 BE long flows and 10 AF long flows in the reverse direction between `TCP Source 2` and `TCP Sink 2`. The traffic in either direction also contains a mix of HTTP traffic and constant rate CBR flows between the `HTTP Source`s, `CBR source`s and their respective sinks. The maximum number of HTTP connections for each HTTP source was 5. There were 10 CBR flows in either direction. Other parameters for the HTTP sources and the CBR sources were set to the default values in ns-2. We set up a `Test Source` with two TCP connections, one



(a) Per-flow throughput

(b) Per-flow loss rate

Figure 2. Impact of False Sharing: In (a), we plot the per-connection throughput against the fraction of bandwidth allocated to AF traffic. In (b), we plot the per-flow loss rates with and without sharing.

in the AF class (to `Test Sink 1`) and one in the BE class (to `Test Sink 2`), and measure their performance in two cases: (i) when they are in the same macroflow and share congestion state, and (ii) when they are in separate macroflows. The fraction a of bandwidth allocated to the AF class is varied from 10% to 90%. At $a \approx 20\%$, the input traffic mix is well-matched to the bandwidth allocation.

Figure 2(a) shows the resulting throughput for the AF and BE connections, plotted as a function of a, for the unshared and shared congestion state cases. In the shared case, the curves for the AF and BE flows are identical, since the total macroflow bandwidth is allocated equally to the two TCP connections. In the absence of sharing, as a increases, the throughput of each AF connection also increases, while the throughput of each BE connection decreases. In contrast, the per-connection throughput in the shared case first increases and then decreases as a increases. Formally: Suppose N_a AF and N_b BE TCP connections operate in a network with a fraction a of the bottleneck allocated to AF flows, and compete with a two-flow macroflow with one BE and one AF flow. If λ denotes the bottleneck bandwidth, *the per-flow throughput on the macroflow is given by*:

$$\lambda_{shared}(a) = \frac{3}{8R} + \frac{a(1-a)\lambda}{2(a^2N_b^2 + (1-a)^2N_a^2)} \qquad (2)$$

assuming that the RTTs of the two classes of flows are the same, R, and are independent of a. This is easy to see from Equation 1. The predicted throughput in Equation 2 is also shown in Figure 2(a). The shape of this curve is similar to the experimental one right above it, and the match is a close one (but not precise). For our values of N_a and N_b, the theoretical maximum of the per-connection throughput occurs at $a = \frac{1}{1+(N_b/N_a)^{\frac{2}{3}}} \approx 28.4\%$, which is close to the experimental observation.

Towards the tail of the curve in Figure 2(a), our analytical prediction does not match the experimental observation. This is due to the excessive number of time-outs in this regime, which are not considered in our analysis. Indeed, from Figure 2(a) (the $y2$ axis), time-outs are excessively common in this region. Also, close to the tail, the throughput due to sharing is slightly higher than the throughput of the lower-bandwidth flow in the unshared case implying that false-sharing might compromise end-to-end congestion control. However, this is not true, as we argue next. Figure 2(b) shows the loss rates for the two connections. Notice that the loss rate for the slower connection (AF when $a < 0.2$ and BE when $a > 0.2$) does not increase appreciably when congestion state is shared, despite the other flow on the macroflow having a lower loss rate (even if the extent of false sharing is very high). This shows that the slower of the two flows never overloads its bottleneck link (Notice that this observation holds true even towards the tail of the curve). These results confirm the fact that *TCP congestion control is not compromised due to false sharing.*

We also ran simulations for various other situations in which flows within a macroflow traverse distinct bottlenecks. The topologies for these simulations are shown in Figures 1(b), (c) and (d). In all cases, there are two test flows between the `Test Source`s and the corresponding `Test Sink`s amidst cross traffic and reverse-path traffic composed of long-lived TCP flow, HTTP traffic and constant rate-CBR flows, just as described for the topology in Figure 1(a). The number of shared bottlenecks can be tuned in the case of these topologies (Figures 1(b), (c) and (d)). The base number of topologies are none for Figure 1(b), and 1 each for Figure 1(c) and (d). Across each shared link in the path, we also have cross traffic composed of HTTP and constant rate CBR flows in either direction. We measure the throughput of the two flows when they are together in the same macroflow sharing congestion state, and again when they are not sharing any congestion information for the base number of shared bottleneck links. In all cases, we found that the faster connection was slowed down by the slower sender exactly as predicted by Equation 1. Also, the loss rate of the slower sender was never higher upon false-sharing, as in the service differentiation case. The results for higher numbers of shared links were similar.

3.3. Impact on Short-Lived Flows

So far, we have examined the impact of false-sharing on long-lived flows. In this section, we provide insight into the impact on much shorter, web-like flows. To study the impact of short-lived flows, we use the topology in Figure 1(b) with the base number of shared bottleneck links (zero). We pick a size L for the length of the short-lived flows. At regular intervals of 200ms, from 15 seconds into the start of the simulations, we start pairs of short TCP flows of size L packets, one each between the two `Test Source`-`Test Sink` pairs in Figure 1(b). The rest of the cross traffic remains unaltered. We test with various values of $L = 10, 15, \ldots, 50$ and compare the average loss rate at the two bottleneck links when the pairs of flows are each forced to shared congestion state with the respective loss rates when the pairs of flows are in different macroflows. Figure 3 plots the loss rates on the two bottleneck links for both the shared and unshared cases. For brevity, we only show results for $L = 50$. The results for other values of L are similar. As the figure shows, short-lived flows undergoing false-sharing tend to cause an increase in the ambient loss-rate. As the extent of false-sharing increases, the impact on the ambient loss rate also become more pronounced. However, this increase in loss rate is not an outcome of aggressive or undesirable behavior due to false-sharing. One of the benefits of shared congestion management systems is that short flows can avoid performing slow start by using congestion information from other ongoing flows. As a result, short flows perform significantly better. However, this increase in the transmission rate of short flows also increases the loss rate of the network although no flow behaves in a TCP unfriendly manner. This increase in loss rate occurs even under genuine sharing.

3.4. The Nature of Shared Bottlenecks

First, we discuss the salient features of the topologies in Figure 1. In Figures 1(b) and (c), we vary the ratio of bottleneck bandwidths on the two paths taken by the test flows across experiments. In Figure 1(d), we vary the round trip times of the two paths while keeping the bottleneck bandwidths fixed. These three topologies reflect the varying extents to which the test flows could share the common bottlenecks they encounter (if any).

The test flows shown in Figure 1(b) have no bottleneck in common at all. We refer to this as the *unshared bottlenecks* scenario. Packet losses as a function of time on either flow for this case are shown in Figure 4(a). The top and the bottom ‘rows’ show *loss events* on individual flows while the middle row shows *actual losses* on the flows plotted together, with back to back losses (which comprise a loss event) shown as a ‘pile’. Notice that the losses on any one of the flows are completely uncorrelated with occurrence of losses on the other flow. In fact, the losses within any single flow are much more correlated than across the flows. The delays experienced by the packets within the two flows, shown in Figure 4(b) are similarly uncorrelated. This suggests that, one way of detecting false sharing in such situations, is to leverage the uncorrelated losses and delays experienced by the participating flows.

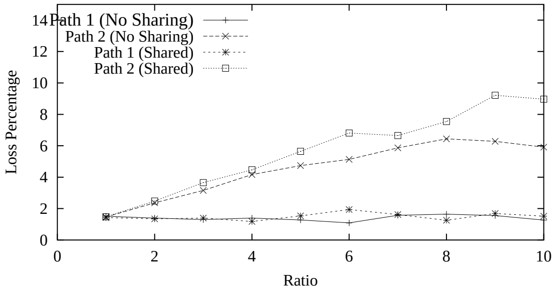
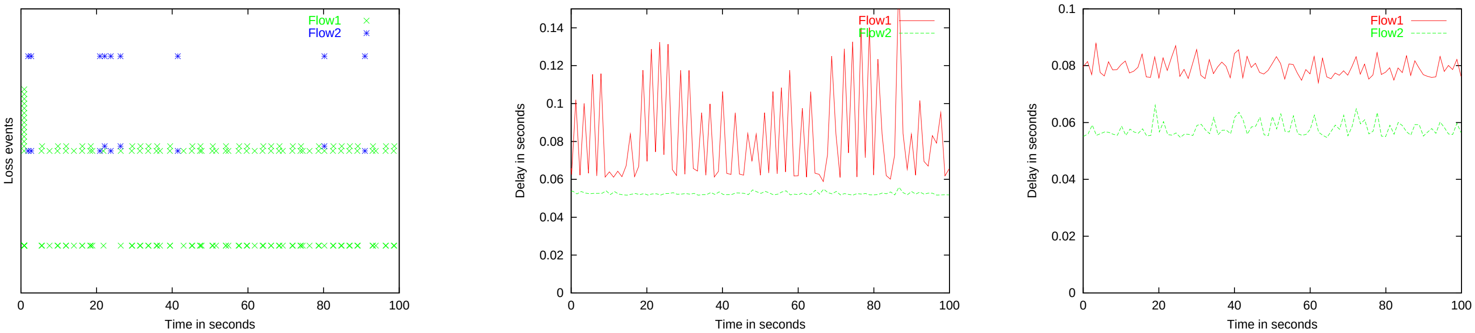


Figure 3. Impact on short-lived flows: The flows are 50KB in size. X-axis plots the value of *ratio* in Figure 1(b). “Path 1” is the faster of the two paths.

However, flows on the Internet may share bottlenecks to varying degrees. The previous case presents a situation in which flows share *no* intermediate bottlenecks. At the other extreme, flows might have *all* bottlenecks in common, as shown, in Figure 1(d). We refer to this as the *fully-shared* bottlenecks case. There could be other situations in between between, where flows can have only some bottlenecks in common and might face other distinct bottlenecks downstream, as shown in Figure 1(c) (*semi-shared bottlenecks*).

In situations like Figure 1(d), false sharing might occur when the participating flows take paths that have very different round-trip delays. However, this is the trickiest to detect of the three cases since we cannot rely on the variations in packet loss and delay statistics. To help further appreciate the difficulty of detecting false sharing in the fully shared case, the delays of the two test flows undergoing false sharing in the experiment shown in Figure 1(d) is shown in Figure 4(c). Notice that the delays of the two flows are highly correlated in contrast to the case presented in Figure 4(b). Thus, relying naively on the packet loss and delay statistics will likely yield the incorrect conclusion that the flows should share congestion information. Therefore, we need a technique different from the ones based on loss and delay statistics to detect false sharing in such situations.



(a) Uncorrelated losses

(b) Uncorrelated delays

(c) Correlated delays

Figure 4. Loss and delay correlations: In (a) and (b), we show the losses and delays for the two test flows of Figure 1(b). Figure (c) shows the delay statistics for the two test flows in Figure 1(d).

4. Detecting False Sharing

In this section, we explore the effectiveness of detection techniques based on loss and delay statistics in various scenarios. Our goal is to devise tests that can detect false sharing with reasonably good accuracy and as quickly as possible. We call the time it takes to detect false sharing the *detection time*. For a given set of traffic, topology, and network conditions, the detection time is a function of the statistical confidence in the reported result.

4.1. Test Description

Rubenstein *et al.* [20] developed two sets of techniques to identify whether a pair of flows shared *some* common bottleneck. Our objective is to use a similar set of tests as part of a comprehensive end-to-end congestion sharing system. To be used as part of such an architecture, these tests must be enhanced to handle the following important issues:

1. Two flows undergo false sharing if even one of their bottlenecks is not common to them. In both the service differentiation and diverse routing cases, we observe that flows can share some bottlenecks but not others.

2. Rubenstein *et al.*'s tests work especially well when unshared flows traverse entirely different bottlenecks that introduce independent delays and losses. In networks where two flows are served differently at the same bottleneck, there may be a non-negligible statistical dependence between the delays experienced by the two flows, which needs to be handled by the detection scheme.

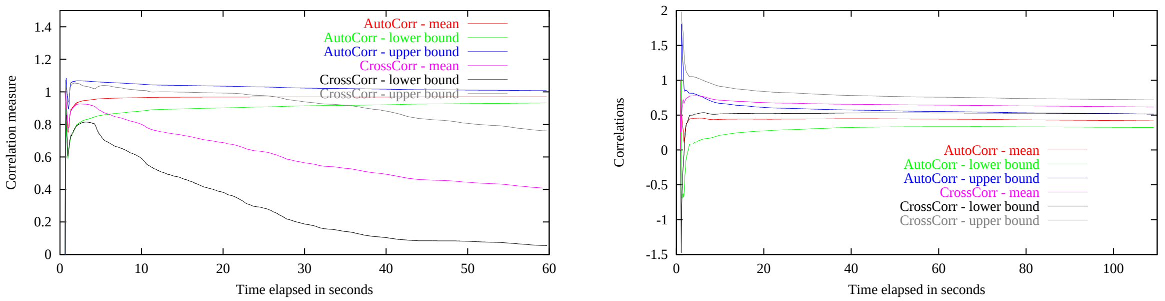
3. It is often the case that a scheduler at the sender apportions bandwidth in non-uniform ways to the different flows on the same macroflow. For example, it might cause one flow to send two packets for every packet sent by another.

4. Many congestion control algorithms depend upon the round trip time to a destination. Flows with all bottlenecks in common, may still have different propagation delays. Aggregating them would therefore cause false sharing to occur.

The rest of this section describes how the basic ideas of loss- and delay-correlations and packet reordering can be further developed to handle the above issues.

4.1.1. Loss-correlation Tests When a loss occurs on a flow, we typically assume that it is because some router has decided to drop the packet. Since this router is low on buffer space, either because the instantaneous or time-averaged buffer occupancy (in RED) is high, it is more likely to drop more packets in the near future compared to other times. Therefore, the conditional loss probability for packets arriving at the router soon after a loss is expected to be higher than the overall loss rate for all packets at the router. In addition, the more recent the previous packet loss, the higher this conditional loss probability.

For an endpoint, this observation implies that losses are likely to come in bursts on a single flow. If two separate flows from a source share a common bottleneck then the likelihood of a flow observing a loss should be higher as well, if the other flow has recently observed a loss. In fact, if packets from the different flows are transmitted closer together in time than packets from the same flow, the conditional loss probability should be higher across the flows than within the same flow. However, if the separate flows do not share a bottleneck, it is very unlikely that the arrival of losses should have such a temporal-correlation across flows. Based on this observation, Rubenstein *et al.* designed a test [20] to determine if a pair of flows (Flow A and Flow B) share a bottleneck. This test, which we call the *asymmetric loss-correlation test*, picks one of the flows (Flow A) as the comparison flow and computes the following: (1) **Asymmetric loss auto-correlation:** Conditional loss probability for packet on Flow A following a loss on Flow A. (2) **Asymmetric loss cross-correlation:** Conditional loss probability for packet on Flow B following a loss on Flow A.



(a) False Sharing

(b) Genuine Sharing

Figure 5. Correlation-based detection: The plots show the mean of the two delay-correlation metrics bound by their respective 90% confidence intervals.

The test compares the value of these two metrics. If the cross-correlation is higher than the auto-correlation then the conclusion is that a shared bottleneck is present. This test helps detect if the bottlenecks that Flow A encounters are encountered by Flow B as well. The test will not detect that Flow B traverses additional bottlenecks not faced by Flow A which is key to detection of false sharing. Therefore, we define a new test, the *symmetric loss-correlation test*, which uses the following metrics: (1) **Symmetric loss auto-correlation metric:** The number of back-to-back losses on both flows divided by the number of loss events on both flows. (2)**Symmetric loss cross-correlation metric:** The conditional loss probability for packet on the alternate flow following a loss on either flow.

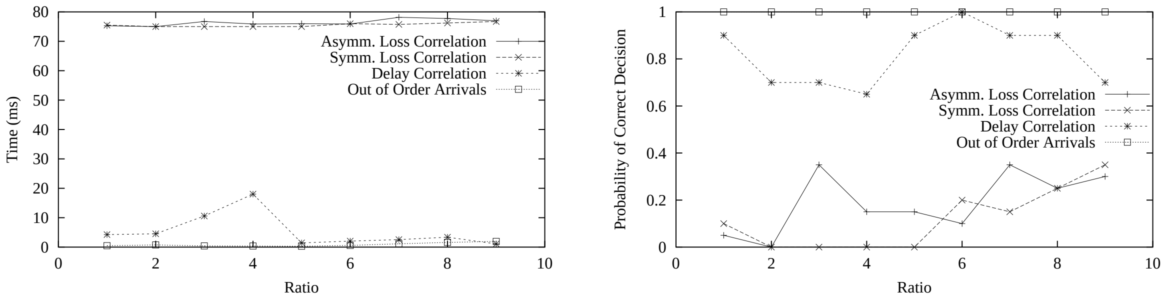
As in Rubenstein *et al.*'s tests, if the cross-correlation is less than or equal to the auto-correlation, then it is likely that the two flows do not share a bottleneck link. In our test, the metrics are the same regardless of which flow is A and which is B and the test is designed to detect if Flows A and B share an identical set of bottlenecks.

A key assumption made by this loss-correlation test, as well as the delay-correlation test described in the next section, is that the transmission spacing between packets from different flows is smaller than within the same flow. This is generally not true because many applications and transport protocols transmit packets in bursts, causing multiple packets from the same flow to be sent back-to-back. This will cause the auto-correlation metric to be higher than the cross-correlation metric (since the samples are taken closer together in time). Rubenstein *et al.* suggest sending additional "Poisson Probes" to generate packet samples in such a way that the packet inter-arrival time at a shared point of congestion is higher across flows than within the same flow. In contrast, we rely on the congestion sharing system to schedule existing data traffic such that there is a reasonable degree of interleaving of packets from different flows within a macroflow. Therefore, we do not need to send any additional traffic to perform our tests. However, with a scheduler, when one of the flows has been given a higher share of the bandwidth, it may send many packets in a row. In such situations, we only consider conditional loss probability samples in which the required interleaving of packets occurs. This same sub-sampling technique, also applied to the delay-correlation tests, ensures that the comparison between cross-correlation and auto-correlation can be used even when bandwidth is apportioned unequally.

4.1.2. Delay-correlation Test The delay experienced by a packet is composed mainly of the propagation time and queuing delay. Like the loss behavior of a path, current queuing delay on a path is variable and strongly related to recent values of queuing delay. Therefore, we can again compare the delays of consecutive packets within a flow and those across flows to detect shared paths.

Delay measurement presents some difficult challenges. For example, clocks on end hosts cannot easily be synchronized. Therefore, any tests must rely on either the change of delay over time or the relative delay of packets between a single source and destination. We use the delay-correlation test as defined by Rubenstein *et al.* to compare these changes over time. We describe the test briefly here.

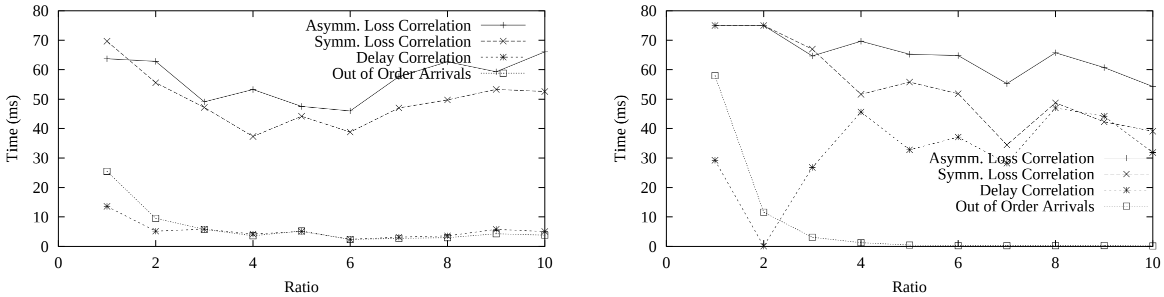
In the delay-correlation test, each packet transmitted is marked with a timestamp at the source. At the receiver, the difference between the source timestamp and current time is recorded. Assuming that clock drift is minimal, the lack of clock synchronization may result in all measurements for a single flow may being off by a fixed constant. The end hosts compute two metrics using this data: (i) the correlation of a packet's delay to the previous packet on the same flow (delay auto-correlation), and (ii) the correlation of a packet's delay to the previous packet on the other flow (delay cross-correlation). Since the computation of correlation eliminates any constant differences between the flows, the lack of synchronization has no affect on the values of these metrics. As in the case for loss, if the cross-correlation met-



(a) Speed (detection time)

(b) Accuracy (Probability of arriving at correct decision)

Figure 6. Detection of False Sharing: Results for the Service Differentiation case plotted as a function of the fraction of bandwidth apportioned to AF. The correct decision is *no share*.



(a) Unshared bottlenecks

(b) Fully-shared bottlenecks

Figure 7. Detection speed, unshared (a) and full-shared cases (b): In the fully-shared case, we plot the time taken to reach a decision of *no share* for out-of-order test and to reach a decision of *share* for other tests.

ric is equal to or less than the auto-correlation metric, then it is likely that the two flows do not share a bottleneck link.

Figure 5(a) shows how the delay correlation metrics and the 90% confidence intervals for these metrics evolve over the duration of a transfer. The metrics are computed between a DiffServ AF flow and a DiffServ BE flow from a single source. These flows are treated differently by the network. The cross-correlation metric quickly drops below the auto-correlation metric and by 40 seconds the 90% confidence intervals no longer overlap where we can clearly identify that the flows do not share any bottlenecks. The loss-correlation metrics exhibit similar behavior. Figure 5(b) shows the delay-correlation metrics and their 90% confidence intervals for two DiffServ BE flows from a single source. Here, the cross correlation metric increases beyond the auto-correlation metric, albeit slowly, and by 100 seconds the confidence intervals no longer overlap. In general, for both the loss and delay correlation tests, we use the 90% confidence intervals around the cross and auto-correlation metrics to decide which one is greater.

4.1.3. Out-of-order Test The delay-correlation test does nothing to identify that the different flows have fundamentally different magnitudes of delay. The topology in Figure 1(d) illustrates a situation with this problem. The bottleneck link is shared by both the test flows, because of which both the loss-correlation and the delay-correlation tests will identify the flows as sharing a bottleneck. However, the two flows have very different end-to-end delays and many congestion control algorithms, including TCP's AIMD, would need to treat them differently. Measuring the magnitude of this delay is difficult due to the lack of synchronized clocks. Also, we would like not to rely on round-trip measurements to avoid noise that the return path may introduce.

The out-of-order test handles this situation by looking for packet reordering from a source. Each packet is marked with an increasing sequence number at the source. Since recent studies [19, 18] have shown that reordering by more than three packets is relatively rare on most of today's Internet, we assume that significant degrees of reordering are caused by false sharing and check to see if packets belonging to flows in the same macroflow *consistently* arrive out of order to detect false sharing (when reordering occurs on 10 different occasions or more). However, for this test to work, packets must be delivered to the same physical destination. Therefore, this test (as described) cannot be applied to detect scenarios such as NATs en route. In such situations, we rely on RTTs as a fall-back strategy. Also, the out-of-order test, produces either a decision that the flows do not share a bottleneck or an inconclusive result. However, the previous tests answer that either a pair of flows share a link, a pair of flows do not share a link, or remain inconclusive.

4.2. Detection Time and Accuracy

We now evaluate the performance of these detection schemes in a variety of situations. We consider two metrics for each test: (i) the time until the test returns the correct answer and (ii) how often the run returns a correct. We evaluate the tests in four different scenarios of Figure 1. The results shown here are averages over 20 simulation runs.

Figure 6 shows the performance of these tests on the topology in Figure 1(a) (DiffServ). The source in this topology transmits two flows through the network: a BE flow and an AF flow. We vary the allocation of bottleneck bandwidth to the different DiffServ classes. Recall that a 20% allocation of bandwidth to AF flows corresponds to an equal per flow bandwidth share in this case. Figure 6(a) shows the average time before a test returns the decision of “no share”. Runs in which a test never returns the correct answer are not reported on the graph. Figure 6(b) shows how often the various tests succeed at detecting the false sharing. From these graphs, we see that the out-of-order tests provide the best results, the loss-based tests do not produce accurate or timely results, and the delay-correlation test produces results quickly, but not accurately. Figure 7(a) shows the results for detection as a function of extent of false sharing for the unshared bottlenecks scenario (Figure 1(b)). As in the case with DiffServ, the delay based tests perform best.

The results for fully-shared scenario (Figure 1(d)) are shown in Figure 7(b) which shows the detection speed of the various tests. Since all packet drops and queuing delays are caused by a single router, we expect the loss-correlation and delay-correlation tests to identify that the two flows share all bottlenecks. These tests do indeed produce the result of *shared* bottleneck or remain inconclusive (accuracy results are not shown here). However, we need a test that indicates that false sharing is occurring. Figure 7(b) shows that the out-of-order test does quickly report that different RTTs are present. (The out-of-order test also produces the correct decision of *no share* almost always. Again the accuracy results for this test are not shown here.) However, various congestion control schemes may treat this situation differently. Algorithms that care about bottleneck sharing but not about RTT differences can treat this situation as an opportunity for sharing. If RTT affects the congestion control (e.g. TCP, TFRC) then false sharing should be detected.

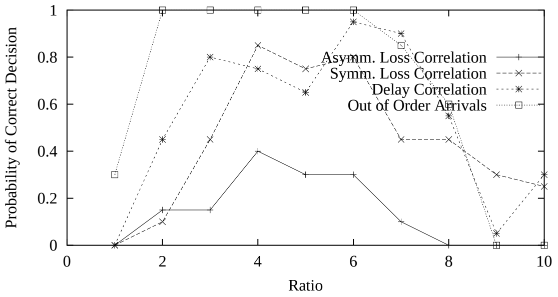


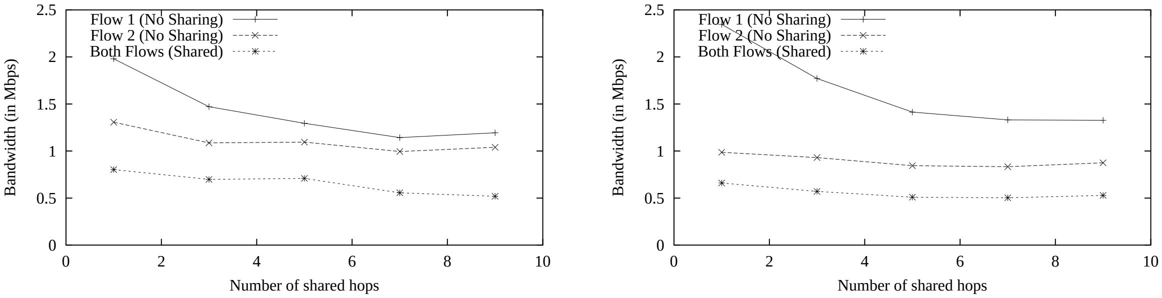
Figure 8. Semi-shared case: Detection accuracy.

The accuracy of the tests in the partially shared bottlenecks scenario (Figure 1(c)) is shown in Figure 8. The basic performance trade-offs of the three test are similar to those in the other scenarios. Notice that unlike the symmetric loss-correlation test, which correctly identifies the false sharing, the asymmetric test can only identify the single shared bottleneck and frequently produces an incorrect or inconclusive result. While the performance difference is especially evident in this scenario, the symmetric test performed significantly better in the other scenarios as well.

4.3. The Effect of Multiple Bottlenecks

The results for the accuracy and speed of our loss-, delay- and order-based detection tests presented in the previous section are for the case when the two test flows share the bare minimum number of bottlenecks – 0 for the topology in Figure 1(b), and 1 for the Figures 1(c) and (d). In this section, we present results for the effect of increased number of shared hops between the flows undergoing false sharing. Conceivably, the detection tests become less effective in this regime. Intuitively, there are two reasons for this. Firstly, the cross traffic at each additional causes the two flows to see increasingly similar loss and delay patterns at higher number of shared hops. Detection tests now take longer to observe a significant mismatch in delay or loss correlations. Secondly, at higher number of shared hops, the bandwidth performance seen by flows undergoing false-sharing becomes increasingly similar, reducing the penalty due to false sharing. Next, we quantify these effects.

Figure 9 plots detection speed using the out-of-order test for the fully-shared bottleneck topology of Figure 1(b). As the number of shared hops between the two flows increases the time taken to detect false-sharing shows a corresponding increase (the curves shift upward) as expected.



(a) Ratio = 3

(b) Ratio = 7

Figure 10. Multiple bottlenecks: Penalty due to false sharing for the topology in Figure 1(d) for ratio $= 3, 7$.

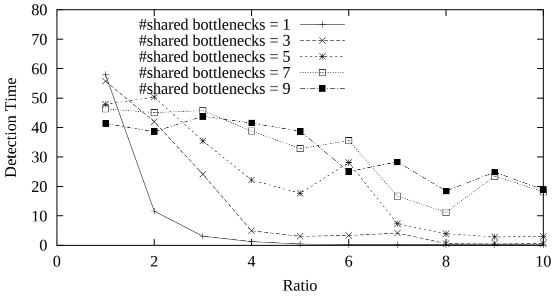


Figure 9. Multiple bottlenecks: Detection time for the out-of-order test in the fully-shared scenario.

Figure 10(a) shows the penalty due to false sharing as a function of the number of shared hops when the value of *ratio* in Figure 1(d) (the ratio of the delays of the last hops in the two test paths) is fixed at 3. Figure 10(b) shows the corresponding results when the ratio is fixed at 7. In either figure, as the number of shared hops increases, the difference in the bandwidths of the two flows when they are not in the same macroflow diminishes as a result of a drop in the "extent" of false-sharing. However, when compared to Figure 10(b) the corresponding difference in the bandwidth performance is smaller in Figure 10(a), where the *ratio* is lower. Figure 9 shows that the performance of the detection test is accordingly inferior at lower values of *ratio*. This suggests that the detection tests perform reasonably well, as long as the penalty due to false-sharing is reasonably high, irrespective of the number of shared hops.

5. Response to False Sharing

We examine how false sharing tests can be integrated into shared congestion management systems. We answer two questions: (i) Do congestion sharing systems need a simple mechanism that turns information sharing on and off or should they be modified otherwise? (ii) How quickly do such systems recover after detecting false sharing?

5.1. Design Issues

Key characteristics of the detection tests encourage a default behavior of sharing information across flows and detecting false sharing instead of separating flows and detecting when they share all bottlenecks. One of the reasons for this is the scheduling issue discussed in Section 4. These tests work best when packets are transmitted in a nice interleaved fashion. Such scheduling is only possible when the flows are placed in the same macroflow and share congestion information. Secondly, delay and loss-correlation tests detect false sharing much more quickly than genuine sharing (see Figures 5(a) and (b)). Essentially, in order to detect sharing, the tests must measure a higher cross-correlation than auto-correlation. Since it is likely that the cross-correlation is only marginally higher than auto-correlation, it takes quite some time before the tests can confidently state that the flows share a bottleneck. In the case of false sharing, it is very likely that the cross and auto-correlations are significantly different. Therefore, the tests quickly detect false sharing. Thirdly, it does not make sense to apply the out-of-order test with a default of no-sharing because it never produces an output of "share" for test flows. It may also seem that a default of sharing congestion state can cause in a host that is trying to detect false sharing to transmit data too quickly and flood the network. However, as we showed in Section 3, the bandwidth achieved during false sharing is limited by the slower of the flows. Also, the loss rate on the affected links is not increased by the incorrect congestion control actions. hence, the right behavior for a congestion control system is to share congestion information whenever possible and detect incorrect sharing.

Upon detecting false sharing, the congestion control system should stop sharing information across the affected flows. In the Congestion Manager this is done by associating the flows with separate macroflows. This process is reversible. If at some later time, the end host identifies that a pair of flows did not have false sharing, it should share congestion information across the flows by re-assigning them to the same macroflow. The lack of significant penalty associated with making an incorrect decision encourages the use of relatively small confidence intervals on the different tests. As shown in Figure 11, this significantly reduces the detection time. Also, once false-sharing has been detected to a particular destination, the default behavior for immediate future connections, short or long, to the destination should be to *not* share congestion state.

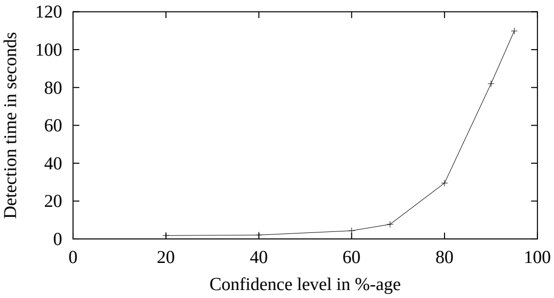


Figure 11. Detection time vs. level of confidence.

5.2. Performance

Next, we show that using the detection tests, the performance of flows undergoing false sharing can be restored within a reasonable amount of time. To this end, we run simulations as follows: (1) Run a simulation with two test flows in the same macroflow to completion. (2) Detect false sharing, if any, using the out-of-order test, at time T_{detect}, say. (3) Re-run the simulation under the exact same conditions as the previous run using the same random seed for the simulation with flows sharing congestion state (The receivers in ns-2 are yet to be changed to automatically detect and respond to false sharing.). After time T_{detect}, put the flows into distinct macroflows and run to completion. We apply this methodology to the scenario in Figure 1(a).

Figure 12 shows the fraction of the fair-share bandwidth attained by the AF flow over time, starting from the detection of false sharing, for different values of the fraction of the underlying bandwidth apportioned to AF traffic for the service differentiation topology in Figure 1(a). The typical time of detection using reordering tests is between 5-10s from the start of the connection. It can be seen that in less than a factor of 3 of the time taken to detect false sharing the performance is restored to reasonable levels.

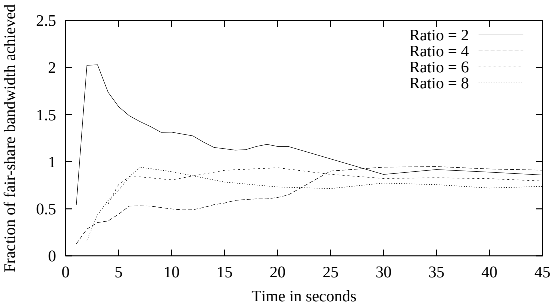


Figure 12. End-system response to detection: Service differentiation scenario.

6. Summary

False sharing may arise in shared congestion management systems, when such systems unknowingly force flows facing distinct bottlenecks to share congestion state. We study this problem in detail, investigating sources of false sharing and analyzing its negative impact. Using simulation and analysis, we show that for shared congestion control using TCP-style algorithms, the correctness of congestion control is not compromised even when two or more flows of very different bandwidths end up sharing information. In particular, the slower sender does not overload its bottleneck link, but faster senders are heavily penalized due to false sharing. We also present tests based on packet loss patterns, packet delay, and packet reordering statistics to accurately detect false sharing and find that reordering statistics are the most robust and fast detectors (5-10 secs) of false sharing. Finally, for integrating these tests into a shared congestion management system, our simulation results show that it is easier to start with the default set to share congestion information than with the default set to “don’t share”.

In conclusion, we believe that our results provide the first, and definitive insights that address and eliminate a key problem that has stalled the deployment of shared congestion management systems in the Internet.

References

[1] A. Akella, S. Seshan, and H. Balakrishnan. The Impact of False Sharing on Shared Congestion Management. *CMU Technical Report CMU-CS-01-135*, June 2001.

[2] M. Allman and V. Paxson. *TCP Congestion Control*. Internet Engineering Task Force, April 1999. RFC 2581.

[3] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. In *Proc. IEEE INFOCOM*, 1998.

[4] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. ACM SIGCOMM*, pages 175–187, September 1999.

[5] H. Balakrishnan and S. Seshan. The congestion manager. RFC 3124, IETF, June 2001.

[6] N. Biswajit. DiffServ Model for the NS2 simulator. `http://www7.nortel.com:8080/CTL/#software`.

[7] D. Clark and W. Fang. Explicit Allocation of Best-Effort Packet Delivery Service. *IEEE/ACM Trans. on Networking*, Aug. 1998.

[8] D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms. In *Proc. ACM SIGCOMM*, August 1992.

[9] IETF Differentiated Services (diffserv) working group. `http://www.ietf.org/html.charters/diffserv-charter.html`.

[10] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM*, pages 314–329, August 1988.

[11] J. Mahdavi and S. Floyd. The TCP-Friendly Website. http://www.psc.edu/networking/tcp_friendly.html, 1998.

[12] N. Maxemchuk. Dispersity Routing. In *Proceedings of ICC*, pages 41–10—41–13, San Francisco, CA, June 1975.

[13] N. Maxemchuk. Dispersity Routing in High-Speed Networks. *Computer Networks and ISDN Systems*, 25:645–661, January 1993.

[14] K. Nichols, V. Jacobson, and L. Zhang. *A Two-bit Differentiated Services Architecture for the Internet*. Internet Engineering Task Force, Apr 1999. Internet Draft draft-nichols-diff-svc-arch-02.txt; work in progress.

[15] V. Padmanabhan. *Addressing the Challenges of Web Data Transport*. PhD thesis, UC Berkeley, September 1998.

[16] V. Padmanabhan. Coordinating Congestion Management and Bandwidth Sharing for Heterogeneous Data Streams. In *Proc. NOSSDAV Conf.*, Basking Ridge, NJ, June 1999.

[17] V. Padmanabhan and J. Mogul. Improving HTTP Latency. In *Proc. Second International WWW Conference*, Oct. 1994.

[18] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. ACM SIGCOMM '97*, Sept. 1997.

[19] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, U. C. Berkeley, May 1997.

[20] D. Rubenstein, J. Kurose, and D. Towsley. Detecting Shared Congestion of Flows via End-to-end Measurement. In *Proc. ACM SIGMETRICS*, June 2000.

[21] S. Savage, N. Cardwell, and T. Anderson. The Case for Informed Transport Protocols. In *Proc. 7th Workshop on Hot Topics in Operating Systems (HotOS VII)*, March 1999.

[22] D. Thaler and C. Hopps. *Multipath Issues in Unicast and Multicast Next-Hop Selection*. Internet Engineering Task Force, November 2000. RFC 2991.

[23] J. Touch. *TCP Control Block Interdependence*. Internet Engineering Task Force, April 1997. RFC 2140.

[24] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource reSerVation Protocol. *IEEE Network*, 7:8–18, September 1993.