The Impact of False Sharing on Shared Congestion Management[1]

Srinivasa Aditya Akella, Srinivasan Seshan
Hari Balakrishnan[2]

June 2001
CMU-CS-01-135

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]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
[2]LCS, MIT

Keywords: Congestion Control, Shared Congestion Management, False Sharing

Abstract

Several recent proposals have been made for sharing congestion information across concurrent flows between end-systems, where the proposed granularity for sharing has ranged from all flows to a common host, to all hosts on a shared LAN. This paper addresses the problem of *false sharing* caused by these proposals: two or more flows sharing congestion state may in fact not share the same bottleneck. We characterize the origins of 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, a situation that occurs in dispersity routing, load-balancing, and with network address translators (NATs). We evaluate the impact of false sharing on flow performance and consider whether it might cause a bottleneck link to become persistently overloaded. We then consider how false sharing can be detected by a sender and how different metrics (loss rate, delay distribution, and reordering) compare for this purpose. Finally, we consider the issue of how a sender must respond when it detects false sharing.

Our simulation 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, somewhat surprisingly, that it is markedly easier to detect and react to false sharing than it is to start by isolating flows and merging their congestion state together afterwards.

1 Introduction

The predominant model for congestion control in the Internet has until recently been based on TCP connections implementing the slow-start and additive-increase / multiplicative-decrease (AIMD) algorithms [11]. 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, and, as an ensemble, display more aggressive behavior in terms of their effective AIMD parameters compared to single TCP connections. These problems are especially visible in the context of the Web, where clients routinely open several concurrent connections to download data from a server.

A number of studies of this effect have been conducted over the past few years [2, 3, 20, 22, 28], and several interesting proposals have emerged to correct the shortcoming caused by TCP connections implementing congestion control in isolation. The essence of these proposals lies in *sharing* congestion information across concurrent flows. These include schemes where the proposed granularity of sharing is a common destination host (more precisely, a host network interface) [2, 3, 4, 20, 28], and where there is more aggressive sharing of information across all destination hosts on the same IP subnetwork [21, 26, 28]. In either case, we call the set of flows that share congestion information a *macroflow*, following the terminology used in the IETF's Endpoint Congestion Management (ECM) working group for such an aggregation.

While the potential benefits of sharing information at granularities larger than a single flow are large and well-studied in the literature, much less is known about the potential drawbacks of shared congestion control. It is possible that these techniques may in fact be potentially detrimental both to the network, and to the applications using them. The reason for this is *false sharing:* two or more flows sharing congestion state may in fact not share the same bottleneck. Obviously, the hard (and interesting) problem with false sharing arises only when it happens without the sender's knowledge. Unfortunately, there are two important broad situations where such false sharing might occur on the Internet: 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.

QoS-enhanced networks that use Integrated Services [7, 29] involve the active participation of end-systems, which eliminates the false sharing problem. On the other hand, false sharing is a significant problem with the currently popular Differentiated Services (DiffServ) architecture [6, 8, 17] where elements in the network can, based on a network-internal policy, allocate different bandwidths, impose different loss rates, or provide different latencies to packets belonging to different flows of the same macroflow.

Dispersity routing for load-balanced packet forwarding [15, 16] and network address translators (NATs) [9] cause packets on the same macroflow to be routed along diverse paths, potentially via different bottlenecks. Dispersity routing is used to take advantage of multiple possible paths between a pair of points, usually routers, in a network [27]. Of course, the bottleneck bandwidth, loss behavior, and delays along these paths can be quite different, especially under dynamic traffic conditions. NATs are a common technique to share an IP address among a set of different machines in a private network. To the sender, all transmissions to these machines appear to be to a single IP destination. However, once the address translation is done, the packets may take paths with different characteristics to their respective destinations "behind" the NAT, which can confound a sender's notion of the shared congestion state.

In each of the above scenarios, the sender observes different bottleneck bandwidths, round-trip times (RTTs) and packet loss rates for the flows that are sharing congestion control information. Since these are the critical parameters for congestion control algorithms, the sender may take improper actions in response. This paper analyzes the consequences of false sharing in an attempt to precisely quantify the effects of sharing congestion information 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?

1. Does false sharing compromise end-to-end congestion control, perhaps by causing a bottleneck link to become persistently overloaded?
2. Does false sharing degrade the performance of individual flows, causing them to perform significantly worse than if they were not sharing congestion information?

2. Detection: Under what conditions can false sharing be detected by a sender, which can then separate the conflicting flows into separate groups? 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?

1. Do these modifications maintain the benefits of sharing (when it is appropriate), while alleviating the negative impact of false sharing?
2. Should the default behavior of such systems be to optimistically share information and then separate flows if false sharing is detected, or vice versa?

We answer these questions using a combination of simulation and analysis. We focus on TCP-like congestion avoidance and control algorithms [1], since they are the dominant set of techniques in use today. It is possible that different congestion control algorithms (e.g., other "TCP-friendly" algorithms [14]) will yield somewhat different results.

For TCP-style window-based congestion control, we find that persistent overload can be avoided even for extreme false sharing, but that higher bandwidth flows run at the rate of the slower bottleneck. We derive an analytic formula for the expected performance under a few assumptions; our formula is validated by experiments. This throughput reduction can be a severe performance penalty, but one that can be avoided by the sender analyzing packet delay and reordering statistics, which we find are more robust indicators of false sharing than loss patterns. 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 described methods for detecting if two connections share at least one common bottleneck link [25]. 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 merging their congestion state together afterwards. This suggests that the default macroflow construction should be to aggregate flows together, and separate them later if deemed necessary by our techniques.

The rest of the paper is organized as follows. The next section describes the problems that arise from false sharing. Section 3 describes how tests for detecting false sharing perform. Section 4 describes how the sender should respond when it detects false sharing. We survey related work in Section **??** and conclude in Section 5.

2 Impact of False Sharing

This section analyzes the performance impact of false congestion sharing. We start with a simple analytic model of the observed throughput and loss rates for two flows that wrongly share congestion information (Section 2.1). We then discuss results of our simulations of networks with differentiated services (Section 2.2) and path diversity (Section 2.3).

2.1 Analysis

In this section, we derive analytic formulas for the observed flow throughput and loss rate when macroflow-based congestion management is done. Although we consider only the case when there are two flows on the macroflow, F_1 and F_2, which share the bandwidth of the macroflow equally, these results are easy to generalize.

Our network model considers two flows F_1 and F_2 traversing paths P_1 and P_2, respectively. In the absence of sharing, each flow independently implements a window-based congestion management algorithm (e.g., TCP). When congestion state is shared, there is only one congestion management algorithm running on the macroflow. It implements the same window-based strategy as in the unshared case, but this time, treats acknowledgments from F_1 and F_2 identically. Each time an acknowledgment arrives on either flow, the congestion window for the macroflow increases by the same amount as in the unshared case. Each time a packet loss (or congestion notification) is detected on either path, the congestion window reduces by the same amount as in the unshared case. This macroflow congestion window is allocated to the two flows so as to ensure that each flow gets the same throughput, by (for example) a round-robin scheduler. We analyze two properties of this network model: per-flow throughput, which is half the total macroflow throughput, and macroflow packet loss rate.

2.1.1 False Sharing Reduces Observed Flow Throughput

We derive the throughput of each flow while sharing congestion information, in terms of the throughput obtained by each flow in the absence of sharing. This result is experimentally validated in Sections 2.2 and 2.3 under different conditions.

Before we set out to derive the effective throughput, we lay out the assumptions we make and simultaneously define the notation used in our derivation. We assume that the congestion management system at the end host reacts to at most one loss per propagation RTT. The Congestion Manager [3], for example, works in this manner. The RTT estimate used by the Congestion Manager is the minimum of RTTs seen on all the flows in a macroflow. In our case, the congestion management system reacts to atmost one loss event (if any) every $min(R_1, R_2)$ seconds.

We assume that the individual flows each use a TCP-like reliable transport protocol that reacts to losses events (atmost one per RTT), rather than to losses themselves. Widely used versions of TCP, like Newreno, behave in a similar manner. We assume that loss events on a particular flow are independent of each other.

We also assume that the advertised windows of receivers on either of the flows are large enough to not constrain the respective senders in any manner. Besides, our derivations work under the assumption that individual flows see identical throughput when aggregated into a single macroflow. This is because, we assume that the congestion management system uses a fair scheduler (the scheduler assigns equal weights to the constituent flows) and that the flows always have backlogged queues.

Finally, we assume that all packets within a flow are sized uniformly and that the size of a packet is the same across all the individual flows.

Let f_i denote the loss event rate on flow i. Let f denote the loss event rate on the macroflow, M, comprising F_1 and F_2. It is a well known fact (see, for example, [10, 13, 18, 19]) that the throughput λ seen by a flow that reacts to loss events at the rate of once every f packets is given by,

$$\lambda \approx \frac{1}{R\sqrt{\frac{2f}{3}}} \tag{1}$$

where R is the propagation RTT of the flow and λ is in *packets/s*.

Claim 1 *Consider two paths P_1 and P_2 with high degrees of statistical multiplexing, such that the packet loss rate on each path 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 obtained by each flows F_1 and F_2, in packets/s, is given by:*

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

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

Let W denote the average size of the congestion window for the macroflow M. We first draw the following inferences:

(i) Since the two flows observe equal throughput on sharing, their shares of W must be proportional to their respective RTTs.

(ii) The above fact implies that the effective throughput of either flow on sharing is $\frac{W}{R_1+R_2}$.

(ii) It follows from the previous fact that the throughput of the macroflow M, $\lambda_M = \frac{2W}{R_1+R_2}$.

Now, to prove the claim we first derive an expression for f in terms of the throughputs seen by the individual flows, λ_1 and λ_2, and their RTTs, R_1 and R_2.

Suppose that M reacts to a congestion event at time t. M will wait for at least R_{min} seconds from t, on the average, before it can react to the next loss event. It follows from inference (iii) that M would wait for at least $W' = \lambda_M R_{min} = \frac{2WR_{min}}{R_1+R_2}$ packets before reacting to the next loss event.

Note that at any particular instant, the probability that a loss event would occur, $\rho = f_1\omega_1 + f_2\omega_2$, where ω_i is the probability that, in a congestion window full of packets, a particular packet belongs to flow i. It follows from inference (ii) that for $i = 1$ and 2, $\omega_i = \frac{R_i}{R_1+R_2}$ and hence $\rho = \frac{f_1R_1+f_2R_2}{R_1+R_2}$. Using Equation 1 to express the f_i's in the previous expression for ρ in terms of λ_i's and R_i's, we obtain,

$$\frac{2\rho}{3} = \frac{1}{R_1 + R_2}\left(\frac{1}{\lambda_1^2 R_1} + \frac{1}{\lambda_2^2 R_2}\right) \tag{3}$$

It can be shown that the expected number of packets before M reacts to the next congestion event after t is $W' + \frac{1-\rho}{\rho}$. From the definition of f, we have $f = \frac{1}{W' + \frac{1-\rho}{\rho} + 1} = \frac{1}{\frac{2WR_{min}}{R_1+R_2} + \frac{1}{\rho}}$. We also have $W = \frac{1}{\sqrt{\frac{2f}{3}}}$. Solving for W, after a few approximations based on the fact that $\rho \ll 1$, we get,

$$W = \frac{3R_{min}}{2(R_1 + R_2)} + \frac{1}{\sqrt{\frac{2\rho}{3}}} \tag{4}$$

Equations 3 and 4 along with observation (ii) yield,

$$\lambda = \frac{3R_{min}}{2(R_1 + R_2)^2} + \frac{\lambda_1 \lambda_2}{\sqrt{\left(\frac{1}{R_1} + \frac{1}{R_2}\right)(\lambda_1^2 R_1 + \lambda_2^2 R_2)}}$$

This demonstrates the claim.

□

In Section 2.2 we show the validity of Equation 2 for the scenario with individual flows of different loss rates and identical RTTs. In a later section (Section 2.3.3), we show that the equation holds for the more general case with flows of different RTTs.

2.1.2 False Sharing Increases Observed Flow Loss Rate

We compare the net loss rate observed by two flows F_1 and F_2 under shared and unshared TCP-like congestion management. A fundamental property of TCP's window-based congestion control is the dependence of the connection throughput on the packet loss rate (p) and round-trip time (RTT). As mentioned in the previous section the throughput λ of a TCP connection with packet size S is, to first order, given by:

$$\lambda \approx \frac{K \times S}{RTT\sqrt{p}}$$

This equation suggests that the impact of false sharing on loss rate with shared congestion management can be formalized. We assume a simple randomized loss model, where packets are dropped on each flow F_i independently with fixed probability p_i. We assume that p_i is small enough that multiple losses do not occur in the same RTT, and timeouts are rare.

Claim 2 *Suppose flows F_1 and F_2 traverse paths with equal RTTs and independent loss probabilities p_1 and p_2, and implement TCP-like congestion management. Then, the observed packet loss rate when they share congestion state on a macroflow is $(p_1 + p_2)/2$, while the loss rate with separate congestion management is approximately $\sqrt{p_1 p_2}$.*

The shared case is easy to see: since the throughputs are equal for both flows, over any long-enough duration of time, the same number of packets are sent on both paths. The average loss rate is therefore simply the arithmetic mean of p_1 and p_2.

If each flow implements separate congestion management, then $\lambda_i \propto 1/\sqrt{p_i}$ for each flow. If a total of N packets are sent across both flows, then the number of packets sent on connection i, N_i is proportional to λ_i. Thus, $N_i \propto 1/\sqrt{p_i}$. This implies that $N_1 = \frac{N\sqrt{p_2}}{(\sqrt{p_1}+\sqrt{p_2})}$ and likewise for N_2. Also, the *total* number of packets sent on flow i at loss rate p_i required to successfully transfer N_i distinct packets (including retransmissions of lost packets) is equal to the sum $N_i(1+p_i+p_i^2+\ldots) = \frac{N_i}{1-p_i} \approx N_i(1+p_i)$ for small p_i. The overall loss rate without sharing congestion information is therefore:

$$p_{noshare} \approx 1 - \frac{N}{N_1(1+p_1)+N_2(1+p_2)}$$

After some algebra, this simplifies to $p_{noshare} = \sqrt{p_1 p_2}$, the geometric mean of the two loss rates.

Thus, $p_{shared} \geq p_{noshare}, \forall p_1, p_2$, which implies that the net loss rate imposed by shared congestion control is higher than the individual ones, which implies that the aggregate throughput in the presence of sharing is smaller than without sharing. This final observation is intuitive: in the absence of sharing, the connection with lower loss rate ends up sending *more* packets compared to the connection with higher loss rate, whereas sharing forces an equal number of packets on both connections. Our simple analysis quantifies the loss in throughput at any pair of loss rates p_1, p_2. This demonstrates the claim 2.

$\square$

We now turn to discussing our simulation results for two scenarios: networks with DiffServ-based QoS, and networks with path diversity.

2.2 Service Differentiation

In a common scenario that results in false sharing, a single host transmits multiple flows through the network to a single destination. Unknown to the source host, the network may classify these different flows into different QoS classes, treating these flows differently in terms of allocation bandwidth and buffer resources. These different flows now may no longer display similar performance upon encountering a bottleneck.

This type of behavior is typical of the IETF Differentiated Service (DiffServ) architecture, which defines three kinds of Per-Hop Behaviors (PHBs): Expedited Forwarding (EF), Assured Forwarding (AF) and Best Effort (BE) [8, 12]. Here, we only describe our experimental setup for investigating the effects of false sharing under a simple DiffServ model, rather than the spectrum of DiffServ implementation possibilities.

Our experiments use Nortel's public implementation of DiffServ in ns-2 [5], which explicitly models edge routers with per-source-to-destination policers and core routers without per-flow state. We model two traffic classes: an AF class, and a 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 [6].

Bandwidth sharing is done using a weighted round-robin (WRR) scheduler between the AF and BE classes, while buffer management uses RED (for BE) and RIO (for AF) (see Table 1).

	min_{thresh}	max_{thresh}	max_p	w_q
AF-lowdrop	$0.2Q_{max}$	$0.8Q_{max}a$	0.05	0.003
AF-highdrop	$0.1Q_{max}$	$0.6Q_{max}a$	0.15	0.003
BE	$0.2Q_{max}$	$0.8Q_{max}(1-a)$	0.05	0.003

Table 1: RIO (for AF) and RED (for BE) parameters for the DiffServ simulations. Q_{max}, the amount of available buffering, is roughly equal to the bandwidth-delay product of the network (delay assumed to be $\approx$ 100ms). a is the fraction of bandwidth allocated to AF traffic class.

Excess bandwidth is shared in proportion to the bandwidth allocated to each traffic class. We experimented with two styles of buffer management: (i) where all packet buffers are shared between the two classes and not pre-partitioned, and (ii) where all packet buffers are pre-partitioned in proportion to the WRR bandwidth allocations. When both traffic classes are active, the two schemes should perform nearly equally, but when traffic arrives in bursts between the two classes, the first scheme, being dynamic, ought to do better. All results reported in this paper are for the first scheme of dynamic buffer sharing. The results for the second scheme were not significantly different, because all our experiments had active flows on both traffic classes (this is the scenario when the false sharing problems are harder to solve).

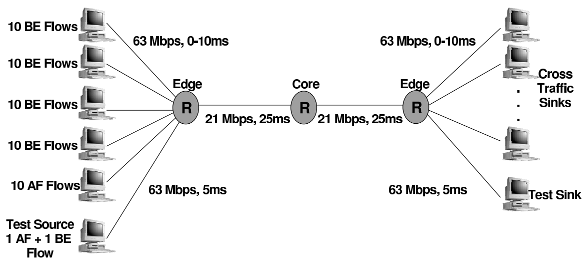

Figure 1: Topology for the service differentiation case.

Our experiments use the simple dumbbell topology shown in Figure 1. There are a total of $N_a = 10$ AF long-running TCP connections and $N_b = 40$ BE long-running TCP connections as competing background traffic. We set up a "test source" with two TCP connections, one in the AF class and one in the BE class, and measure their performance in two cases: when they are in the same macroflow and share congestion state, and when they are separate. We do this as a function of the fraction a of bandwidth allocated to the AF class, varying a from 10% to 90%. Observe that when $a \approx 20\%$, the input traffic mix is well-matched to the bandwidth allocation. Since all traffic sources are elastic (TCP), any excess capacity will tend to be used by active connections. These traffic sources share a bottleneck link of 21 Mbps and have round-trip (propagation) times (RTTs) uniformly distributed in the interval $[80, 120]$ ms. The test connections have a propagation RTT of 100 ms.

Figure 2 shows the resulting throughput for the AF and BE control connections, plotted as a function of a, for the unshared and shared cases. In the shared case, the two curves are essentially identical, since the total macroflow bandwidth is allocated equally to the two (AF and BE) TCP connections.

As expected in the absence of sharing, as a increases, the throughput of each AF connection also increases, while the throughput of each BE connection decreases. Because the amount of AF

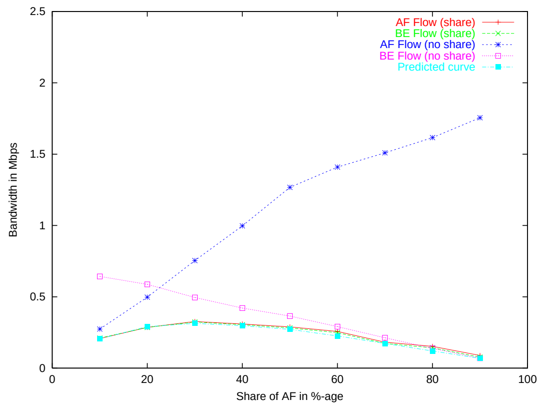

Figure 2: Per-connection throughput as a function of a, the percentage of bandwidth allocated to the AF traffic class. The graph also shows the theoretical prediction for per-connection throughput in the shared case.

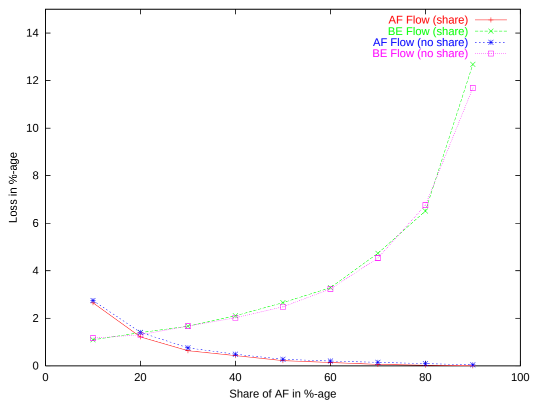

Figure 3: Packet loss rates as a function of a, the percentage of bandwidth allocated to the AF traffic class.

bandwidth is shared equally among the AF connections, and likewise for BE, the two curves are linear. The two lines cross at about $a = 0.2$, which corresponds to $a = N_a/(N_a + N_b)$, the expected cross-over point.

In contrast, the per-connection throughput in the shared case first increases and then decreases as a increases. We can quantify this formally.

Property 1 *Suppose that N_a AF and N_b BE TCP connections are operating in a network with a fraction a of the bottleneck allocated to AF flows, and are competing 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:*

$$\lambda_{shared}(a) = \frac{3}{8R} + \frac{a(1-a)\lambda}{2(a^2N_b^2 + (1-a)^2N_a^2)} \tag{5}$$

assuming that the RTTs of the two classes of flows are the same, R, and are independent of a.

The intuition behind this result is that shared congestion control with equal allocations to each flow *forces* an equal number of packets to be sent on each TCP connection over any reasonable time duration. If a total of S packets are transmitted on the macroflow, $S/2$ are sent on each connection. If the bottleneck bandwidth is λ, and the AF fraction is a, then the per-connection AF bandwidth is roughly $\lambda_a = a\lambda/N_a$ and the BE bandwidth is roughly $\lambda_b = (1-a)\lambda/N_b$.

Equation 2 in Section 2 expresses the per-connection throughput when traversing paths with different available bandwidths. Substituting λ_a and λ_b into that equation and setting $R_1 = R_2 = R$ yields the required result.

□

The predicted throughput in Equation 5 is shown along with the experimental values in Figure 2, alongside the experimental values (the lowest curve). The shape of this curve is similar to the experimental one right above it, and the match is a close one (but not precise). We also note that for our values of N_a and N_b, the theoretical maximum of the per-connection throughput occurs at $a = \frac{1}{1+\left(\frac{N_b}{N_a}\right)^{\frac{2}{3}}} \approx 28.43\%$, which is close to the experimental observation.

Figure 3 shows the connection loss rates for each path. The main observation is 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. However, from Claim 2, the per-connection loss rate across both connections is higher than without sharing, because in the former case, more packets would have been sent on the faster (lower loss-rate) path.

We conducted several other simulations of false sharing with such service differentiation, finding in each case that for the TCP-style window-based congestion management algorithm, the result was that the faster connection was slowed down by the slower one, according to the derived prediction. We did not find any situations where the slower link was persistently overloaded as a consequence of the sender being confused by the faster connection into sending packets quicker on the slower one. Once again, this is explained by Equation 5 for per-connection throughput, which shows the throughput becoming smaller as $a \to 1$.

2.3 Path Diversity

When a macroflow contains two flows that traverse different routes, the potential for false sharing arises, since the two flows may not share a bottleneck. In fact, false sharing may occur even when the two flows share some bottlenecks; unless *all* their bottlenecks are shared, it would be incorrect to share congestion state between the two flows.

In practice, there are at least two important scenarios where path diversity causes false sharing: (i) dispersity routing and load-balancing, and (ii) network address translation (NAT).

The idea of dispersity routing, explored by Maxemchuk [15, 16], 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 [27]. In general, 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. The result is that individual flows by and large run over the same path through their lifetime. 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, the false sharing problem does not occur for macroflows that are constructed per-destination. However, the problem still persists when a sender shares congestion information across multiple destinations.

NATs (with or without firewall functionality) are common in today's Internet infrastructure. In this scenario, there are multiple possible physical destinations for a single visible destination IP address, with traffic to the different destinations generally using different transport-layer ports addressed to the NAT box. Although the flows may share a common path up to the location where the address translation is done, the path from that point to the destination can vary greatly and have different bottlenecks. If the sender shares congestion information across all connections to a destination, a NAT (or proxy) destination may therefore cause false sharing.

We classify the false sharing caused by path diversity into three categories according to the nature of shared bottlenecks: *unshared bottlenecks*, *semi-shared bottlenecks*, and *fully-shared bottlenecks*. We describe and investigate the impact of false sharing in each of these cases in turn.

2.3.1 Unshared bottlenecks

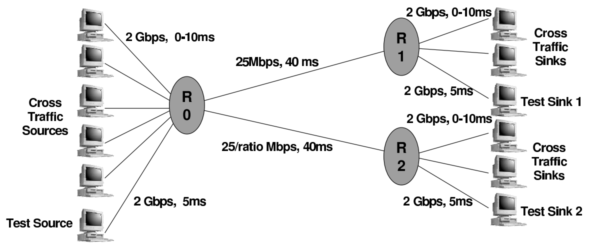

Figure 4: Topology for the unshared bottleneck case.

In this scenario, the two flows do not share a bottleneck link; i.e., there is no persistent queueing or packet loss on any shared router or link. This situation is shown in Figure 4, which is our simulation topology for experiments. We vary the ratio of bottleneck bandwidths on the different paths in different experiments.

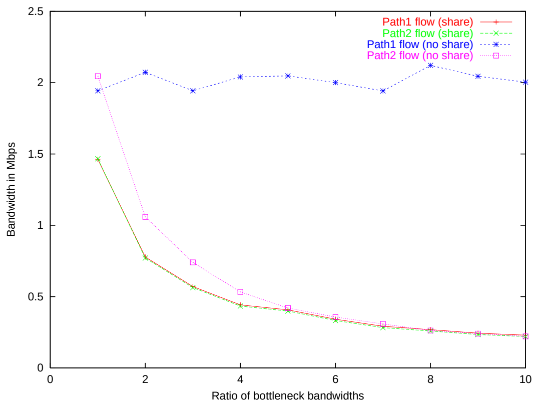

Figure 5: Per-connection throughput as a function of the ratio of bottleneck bandwidths for the unshared-bottleneck case.

Figure 5 shows the measured throughput of the two connections when they do not share congestion information (the two curves on top) and when they do share congestion information (the two overlapping curves at the bottom), as a function of the ratio of bottleneck bandwidths. At each ratio, the measured per-connection throughput in the shared case is close to the prediction of Equation 2 from Section 2. As the bandwidth ratio gets large, the per-connection throughput in the shared case tends toward the throughput of the slower connection in the unshared case.

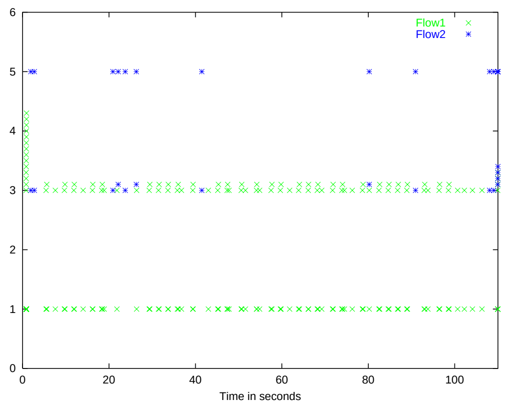

Figure 6: Losses on the two test flows plotted as a function of time elapsed. The top and the bottom ‘rows’ show losses on the individual flows. The points in the central portion show the losses across the two flows plotted together, with back to back losses shown as a ‘pile’ of losses. Notice how more back-to-back losses occur on one of the two flows (flow1), than back-to-back losses across the two flows.

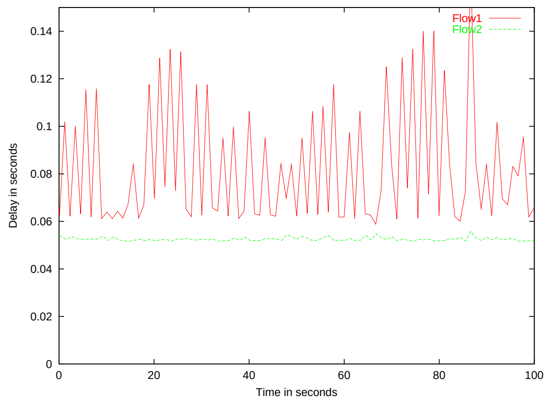

Figure 7: Delay experienced by the two test flows plotted as a function of time elapsed. Notice that the delays observed by the two flows vary independently of each other.

As with the service differentiation case, the loss rate for a flow does not change appreciably when the two flows are aggregated to share congestion state. However, interesting conclusions can be drawn by taking a closer look at the time-series plots of the loss events and delays experienced by the two flows. Figure 6 shows a plot of the instances at which losses occur in each of the two flows with unshared bottlenecks. The two flows here belong to the same macroflow. The losses seem uncorrelated across the two flows—the losses on one of the flows occur at times that are, for the most part, unrelated to the loss events on the other flow. Figure 7 shows the delays that the packets on each of the two test flows experience as a function of time elapsed. The variation in delay in one of the flows is essentially independent of the variation in delay of the other flow.

These observations suggest that one way of detecting false sharing is to take advantage of the fact that loss and variations in delay within a flow are more correlated than loss and variations in delay across flows.

2.3.2 Semi-shared bottlenecks

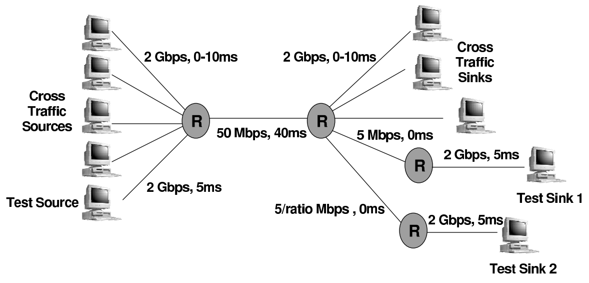

Figure 8: Topology for the semi-shared bottleneck case.

For situations falling into this case, one of the unshared paths contains a bottleneck link for its flow. As a result the flows may sometimes share observations of congestion events and at other times observe information about their respective paths. The topology used for modeling this situation is depicted in Figure 8. We vary the ratio of the bottleneck bandwidths across the different experiments.

Figure 9 shows the measured throughput of the two connections when they do not share congestion information (the two curves on top) and when they do share congestion information (the two overlapping curves at the bottom), as a function of the potential bandwidths of the bottleneck links to the control destinations. Again, notice that the two connections consistently see a bandwidth that is at-most the bandwidth seen the slower of the two connections when the two do not share congestion information. Also, as the extent of false sharing increases (higher ratios of potential bandwidths), the bandwidth of each of the two flows converges to the minimum of the bandwidths in the case where the connections do not share congestion state. The loss rates for the two flows do not change significantly due to false sharing.

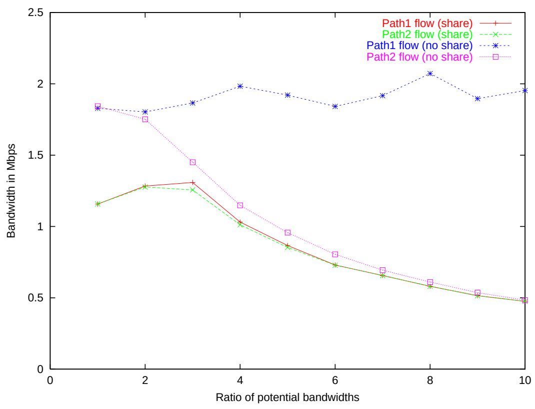

Figure 9: Per-connection throughput as a function of the ratio of potential bandwidths for the semi-shared-bottleneck case.

In this case too, it was observed that there were more back to back losses on a single flow than there were across the two flows. Similarly, the plots for delays for the two flows (not shown here) seemed to evolve independent of each other with time.

2.3.3 Fully-shared bottlenecks

The final set of experiments for the path diversity class are those involving fully-shared bottlenecks. In this scenario, neither of the unshared paths contain a bottleneck link for the flows. As a result, the flows may have different round trip times but should observe similar loss behavior. This model topology used for this situation is depicted in Figure 10. We vary the propagation RTTs across the different paths for different experiments.

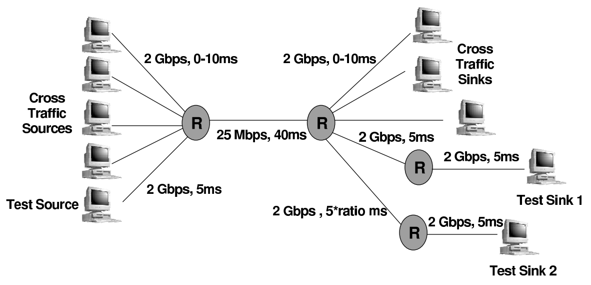

Figure 10: Topology for the fully-shared bottleneck case.

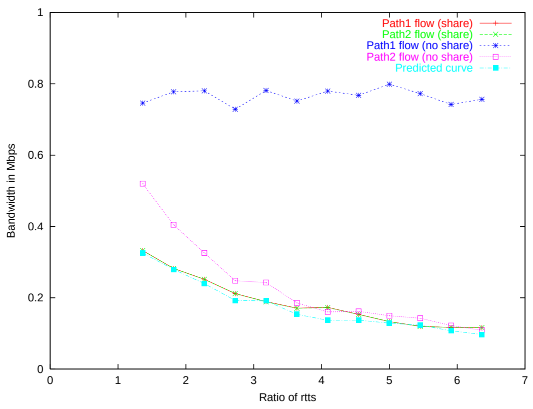

Figure 11: Per-connection throughput as a function of the ratio of round trip times of the two connections for the fully-shared-bottleneck case.

Figure 11 shows the observed throughput of the two connections plotted as a function of the ratio of the round trip times experienced by them. The predicted values of the observed bandwidth are also plotted on the same graph. The bandwidth follows the same trend as in the previous two cases. As can be seen, the observed macroflow bandwidth in this case (the most general case where the two connections see different round trip delays) follows the predicted value with reasonable accuracy.

Figure 12 is a graph of the propagation delays observed by packets on the two flows for a typical run of the experiment. Note that the variations in delay over time are correlated to a non-trivial extent. This should be the case, because the two flows in fact do share a common point of congestion, the queue at the bottleneck link.

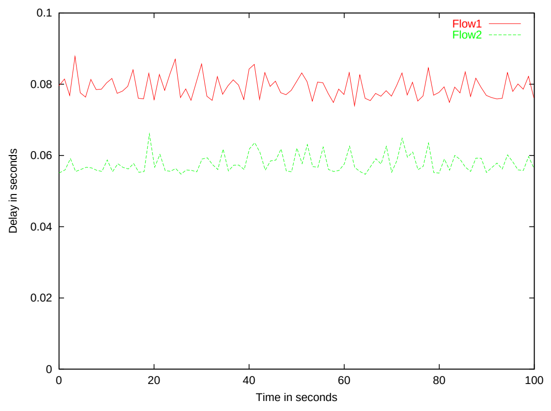

Figure 12: Delay experienced by the two test flows plotted as a function of time elapsed for the fully shared bottleneck case. Notice how the delays observed by the two flows vary in a coupled manner.

3 Detecting False Sharing

The previous sections showed that in the presence of false sharing, the different flows involved typically experience different delays and/or loss rates. This leads to the possibility that by monitoring delays and loss rates over time, end-systems can identify that false sharing is occurring and take the appropriate corrective measures. In this section, we explore the effectiveness of detection techniques in various scenarios. Our goal is to devise tests that can detect false sharing with reasonably good accuracy, 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, it is a function of the statistical confidence that can be had in the reported result.

3.1 Test Description

Rubenstein *et al.* [25] 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 differentially served 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. Even if two flows have all their bottlenecks in common, they may have different propagation delays. Aggregating them together would therefore create false sharing.

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

3.1.1 Loss-correlation Tests

When a loss occurs on a flow, we assume, as is typical of most Internet paths, that it is because some router has decided to drop the packet in order to make more buffer space available.[1] Since this router is low on buffer space, either because the instantaneous or time-averaged (in RED) buffer occupancy is high, it is more likely to drop more packets in the near future compared to other times. As a result, 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 sent through this router. In addition, the more recent the previous loss event, the higher this conditional loss probability.

From the perspective of an endpoint, if these assumptions are true, then the implication of this observation is that losses are likely to come in bursts on a single flow. If two separate flows from a single source share a same bottleneck, this observation should be true across the combination of the flows as well. The likelihood of a flow observing a loss is higher 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 loss-based test [25] to determine if a pair of flows (flow A and flow B) share a common bottleneck. This test, which we call the the *asymmetric loss-correlation test*, chooses one of the flows (flow A) as the comparison flow and computes the following metrics:

1. **Loss auto-correlation metric**. The conditional loss probability for the packet on flow A following a loss on flow A.
2. **Loss cross-correlation metric**. The conditional loss probability for packet on flow B following a loss on flow A.

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 relationship is based on the assumption that packets on different flows have less time between transmissions than packets on the same flow.

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. To prevent false sharing, we must determine if the two flows face an identical set of bottlenecks. Therefore, we define a new test, the *symmetric loss-correlation test*, which uses the follow metrics:

[1] If uncorrected losses occur en route for other reasons, such as packet corruption, this assumption is obviously invalid.

1. **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. **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 metric is less than or equal to the auto-correlation metric, then it is likely that the two flows do not share a bottleneck link. In our test definition, the metrics are the same regardless of which flow is A and which is B and the test is designed to detect if flow A and B share an identical set of bottlenecks.

A big 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 on average smaller than within the same flow. The reason that this is not true in our system is that many applications and transport protocols transmit packets in bursts, causing multiple packets from the same flow to be sent back-to-back. Such behavior will cause the auto-correlation metric to certainly be higher than the cross-correlation metric (since the samples are take closer together in time). One technique we use to prevent such behavior is to require the congestion sharing system to schedule transmissions such that there is a reasonable degree of interleaving of packets. When flows are placed in a single macroflow, they are scheduled to get their allocated shares on small time scales. This ensures reasonable behavior when the shares are equal. However, when one of the flows has been given a higher share of the bandwidth, it may still 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 is also applied to the delay-correlation tests, and ensures that the comparison between cross-correlation and auto-correlation can be used even when bandwidth is apportioned unequally.

3.1.2 Delay-correlation Test

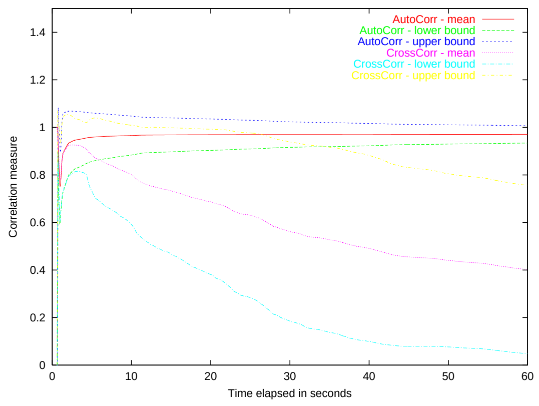

Figure 13: Estimates for correlation coefficients in the delay-correlation test for a pair of flows as a function of time. The plot shows the mean of the two delay-correlation metrics bound by their respective 90% confidence intervals.

The delay that a packet experiences is caused mainly by propagation time and queueing delays. While the propagation time is fixed, the other contributing factors can vary significantly over time. However, like the loss behavior of a path, current queuing delay on a path is strongly related to recent values of queuing delay. As a result, similar techniques of comparing the behavior between back-to-back packets within a flow and back-to-back packets across flows can be used 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 the 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 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 metric 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 13 shows how the delay-correlation metrics and the 90% confidence intervals for this metric evolve over the duration of a transfer. In this illustrated test, the metric is computed between a DiffServ AF flow and a DiffServ BE flow from a single source. Since 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. At that point the host can clearly identify that the flows do not share a bottleneck and that false sharing has occurred. The loss-correlation metrics exhibit a similar behavior. Figure 14, similarly, shows the delay correlation metrics and their 90% confidence intervals for two DiffServ BE flows from a single source. The cross correlation metric in this case, increases beyond the auto-correlation metric, albeit slowly, and by 100 seconds the confidence intervals no longer overlap.

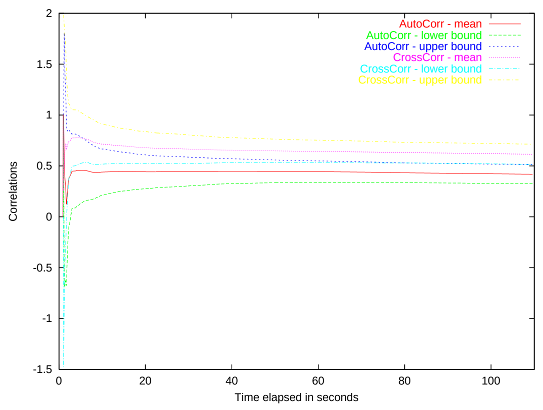

Figure 14: Evolution of the correlation measures and their respective confidence intervals with time for the case where the two flows actually share a common point of congestion. Notice how the time until the two confidence intervals separate is much higher than in the case where there is no shared point of congestion.

3.1.3 Out-of-order Test

While the delay-correlation test does an effective job at monitoring the changes to delay that a shared bottleneck introduces, it does nothing to identify that the different flows have fundamentally different magnitudes of delay. The multipath routing topology in Figure 15 illustrates a topology with this problem. The bottleneck link is shared by both flows, because of which both the loss-based test and the delay-correlation test 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 this magnitude of this delay is difficult due to the lack of synchronized clocks. In addition, we would like not to rely on round-trip measurements to avoid measurement 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 [24, 23] have shown that re-ordering by more than three packets is relatively rare on most of today's Internet; we therefore assume that significant degrees of reordering are caused by false sharing. One limitation of this test is that packets must be delivered to the same physical destination in order to detect the re-ordering. Therefore, this test (as described) cannot be applied to detect situations such as NATs en route. In such situations, we rely on round trip times as a fall-back strategy.

We note that this test does not produce the same format of results as the previous tests. The previous tests answer that either a pair of flows share a link, a pair of flows do not share a link, or that the measurements are inconclusive. The out-of-order, on the other hand, produces either a decision that the flows do not share a bottleneck or an inconclusive result.

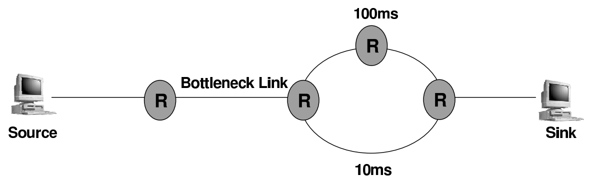

Figure 15: Topology for Multipath routing

3.2 Detection Time

How effectively do these techniques identify false sharing situations? In this section, we evaluate the performance of these detection schemes in a variety of situations. We consider two different metrics for each test: 1) the time until the test returns the correct answer and 2) how often the run returns a correct, incorrect or inconclusive answer. We evaluate the tests in four different scenarios: 1) DiffServ with AF-BE false sharing, 2) path diversity with no shared bottleneck, 3) path diversity with shared bottlenecks and 4) path diversity with partially shared bottlenecks. The tests were applied to 10 simulation runs and the average of the results are presented here.

Figures 16, 17 and 18 show the performance of these tests on topologies in which DiffServ causes false sharing. The topology used in these tests is shown in Figure 1. The source in this topology transmits two flows through the network. One of these flows is marked as a BE flow and the other an AF flow. Since it is likely that the difference in bandwidth allocated to the two flows may make detection easier or more difficult, we vary the allocation of bottleneck bandwidth to the different classes. There are 4 times as many BE flows than AF flows, therefore, a BE/AF bandwidth ratio of 4 corresponds to an equal per flow bandwidth share.

Figure 16 shows the average time before a test returns the decision to not share the link. Runs in which a test never returned the correct answer are not reported on the graph. The following pair of graphs, Figures 17 and 18, show how often the delay-correlation and the symmetric loss-correlation tests succeeded or failed at detecting the false sharing. No equivalent graph is presented for the out-of-order test since it correctly reported the false sharing in all runs. From these graphs, we see that the out-of-order tests provides 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. One interesting point to notes is that for the loss-correlation tests have significant trouble detection congestion when the fair share bandwidth for the two flows are similar. In such situations (e.g. the AF share is 20%), the loss-correlation test produces the incorrect result 70% of the time!

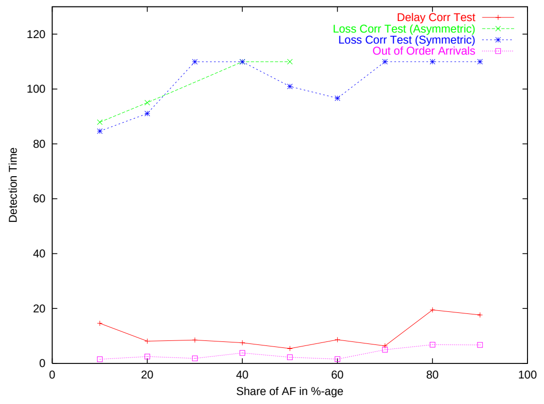

Figure 16: Time until detection of false sharing for the various detection tests plotted as a function of a, the fraction of bandwidth apportioned to AF.

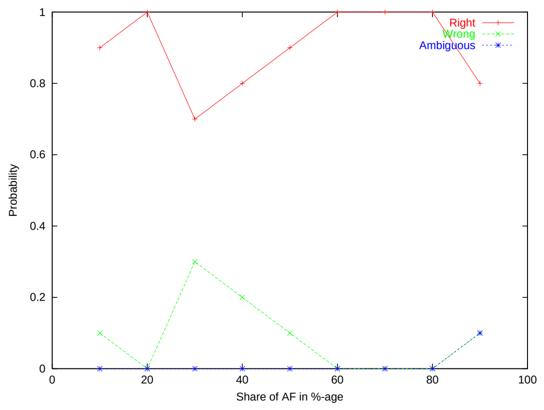

Figure 17: Probability of arriving at the correct decision for the delay-correlation test plotted as a function of a, the fraction of bandwidth apportioned to AF.

Figure 19, shows the detection time plotted as a function of extent of false sharing for the path diversity scenario with no shared bottlenecks. As in the case with DiffServ, the delay based tests perform best. In addition, the loss-correlation test has similar difficulties when the available bandwidth for the flows are similar.

The results for the path diversity with fully shared bottlenecks are displayed in Figures 20,

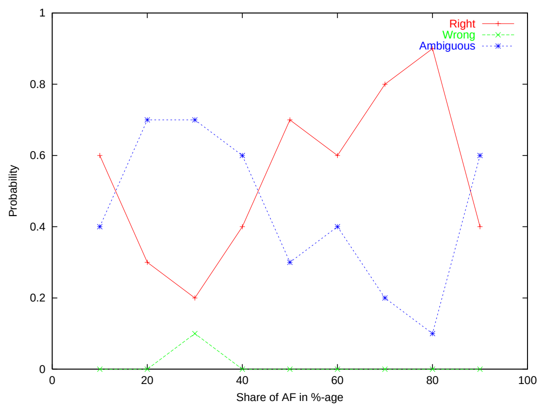

Figure 18: Probability of arriving at the correct decision for the loss-correlation test plotted as a function of a, the fraction of bandwidth apportioned to AF.

21 and 22. Since all packet drops and queueing 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. Figures 21 and 22 confirm that these tests typically produce the result of shared bottleneck in this scenario. However, what we desire is a test that indicates that false sharing is occurring. Figure 20, shows that the out-of-order test does quickly report that different RTTs are present. Different congestion control algorithms may treat this situation differently. For those that only care about bottleneck sharing and not about RTT differences, this can be treated as an congestion information sharing opportunity. In systems where RTT affects the congestion control (e.g. TCP and TFRC) false sharing could be identified.

The performance of the tests in the partially shared bottlenecks scenario is summarized by Figure 23. The basic performance tradeoffs of the three test are similar to those in the other scenarios. For this scenario, we have also presented the results from Rubenstein's original asymmetric loss-correlation test (figure 24). Unlike the symmetric loss-correlation test, which correctly identifies the false sharing (figure 25), 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.

We also need to ensure that these tests do not incorrectly detect false sharing in situations where none exists. We can infer this from the behavior of the tests on the path diversity with full bottleneck sharing. In this test, the loss and delay-correlation tests output only a decision that a shared bottleneck exists or an inconclusive result. This shows that the tests do not report an incorrect response. We perform no similar study of our re-ordering test. The only way such a test can produce an incorrect response is if reordering of more than 3 packets occurs for some other reason. Unfortunately, the ns simulator does not reproduce any of the reordering behavior that the real Internet does.

4 Response to False Sharing

In this section, we examine how false sharing tests can be integrated into congestion information sharing systems. Do congestion sharing systems simply need a mechanism to turn information

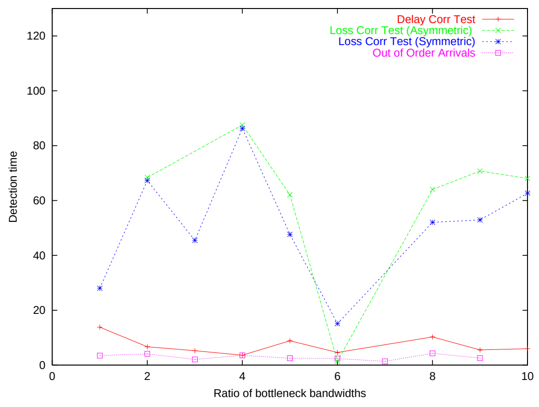

Figure 19: Time until detection of false sharing for the various detection tests plotted as a function of the ratio of bottleneck bandwidths in the unshared bottleneck case.

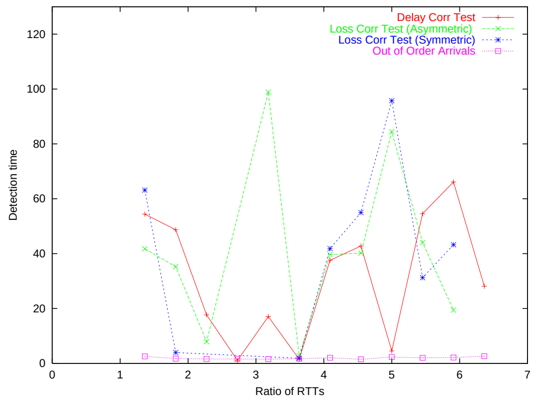

Figure 20: Time until detection of sharing for the various detection tests plotted as a function of the ratio of round trip times of the two control flows in the fully shared bottleneck case.

sharing on and off between flows or do they need modified in other ways as well? In addition, how quickly do such systems recover after false sharing has been detected?

4.1 Design Issues

The performance characteristics of detection tests has significant implications on how they should be incorporated into other systems.

For example, 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 that this design is preferable is the scheduling issue discussed in Section 3. These detection tests work best when packets are transmitted in a nicely interleaved fashion. However, such scheduling is only possible when the flows are placed in the same macroflow and share congestion information. Another reason to prefer a default of sharing is that the delay and loss-correlation tests detect false sharing much more quickly than they detect shared bottlenecks. This issue can be understood by examining Figures 13 and 14.

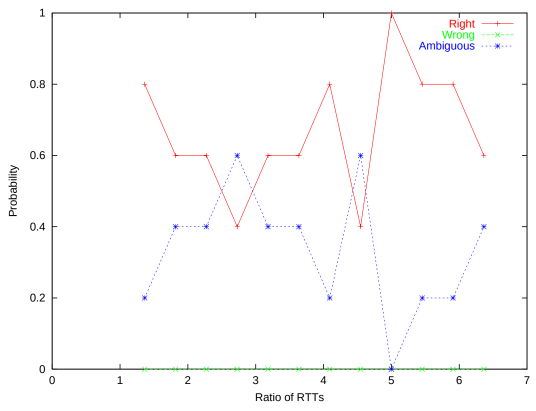

Figure 21: Probability of arriving at the correct decision for the delay-correlation test plotted as a function of the ratio of round trip times, in the fully shared bottleneck case. Notice that the correct conclusion in this case is *share*.

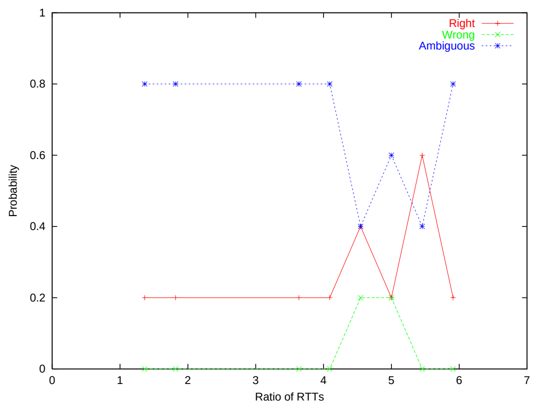

Figure 22: Probability of arriving at the correct decision for the symmetric loss-correlation test plotted as a function of the ratio of round trip times, in the fully shared bottleneck case. Again the correct conclusion is *share*.

The basic issue is that in order to detect a shared bottleneck, the test must measure a higher cross-correlation than auto-correlation. However, it is likely that the cross-correlation will be only marginally higher than the auto-correlation. Therefore, it takes quite some time before the test can confidently state that the flows share a bottleneck. In the case of false sharing, it is quite likely that the cross and auto-correlation should be significantly different. Therefore, the test quickly comes to the conclusion that false sharing is occurring. Finally, it does not make much sense to apply the out-of-oder test with a default of no-sharing because it never outputs that the test flows share a bottleneck link.

One concern with using a default of sharing congestion information is that it may result in dangerous congestion behavior. While a host is determining that false sharing is occurring, it may transmit data too quickly and flood the network. However, as we showed in Section 2, the bandwidth achieved during false sharing is limited by the slower of the flows. In addition, the loss rate on the affected links is not increased by the incorrect congestion control actions. Therefore, the appropriate behavior for a congestion control system is to share congestion information whenever possible and detect incorrect sharing.

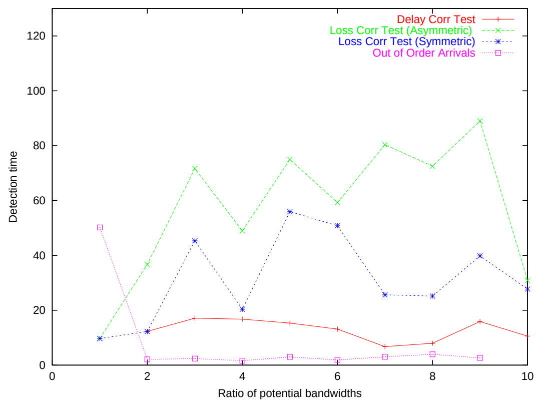

Figure 23: Time until detection of false sharing for the various correlation metrics plotted as a function of the ratio of potential bandwidths of the two control flows in the semi-shared bottleneck case.

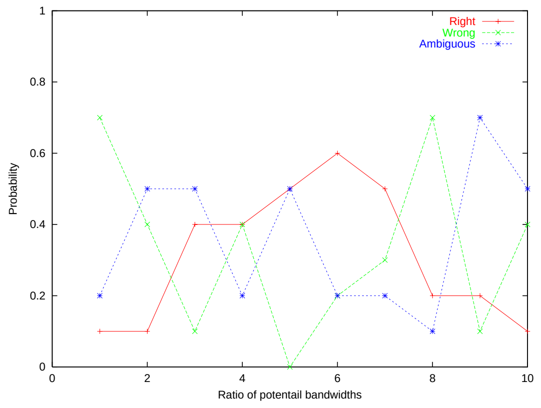

Figure 24: Probability of arriving at the correct decision for asymmetric loss-correlation test plotted as a function of the ratio of potential bandwidths, in the semi-shared bottleneck case.

Once false-sharing is detected, the congestion control system should stop sharing information across the affected flows. In the Congestion Manager system this is done by associating the different 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 26, this helps reduce the detection time significantly.

4.2 Performance

While false sharing is occurring, flows from a source may face a significant performance penalty. In addition, once the false sharing is detected, it may take some time before the appropriate performance levels are restored. However, in false sharing situations where the penalties are the greatest, the detection time is likely to be the smallest. In this section we show that given sufficient

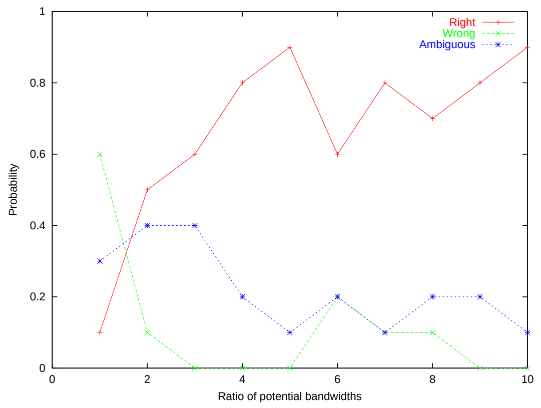

Figure 25: Probability of arriving at the correct decision for the symmetric loss-correlation test plotted as a function of the ratio of potential bandwidths, in the semi-shared bottleneck case.

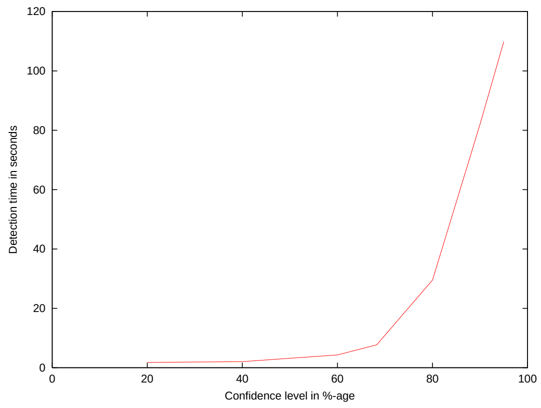

Figure 26: Detection time as a function of the level of confidence.

longevity of the connections, their performance will eventually be restored to reasonable levels.

For testing the effectiveness of the proposed techniques, we ran a set of experiments in the following manner:

1. Run the experiment, after eliminating all possible sources of randomization, until completion. Start by putting the two test flows into the same macroflow MF_1.
2. Use the appropriate detection test (reordering tests) to determine the point in time, T_{detect} at which false sharing can be detected.
3. Run the experiment, again, with all sources of randomization turned off. During the run, at time T_{detect}, switch one of the two test flows from macroflow MF_1 to a different macroflow MF_2. Run the experiment to completion.

Figures 27 and 28 are representative examples of how effective this technique can be in restoring the performance seen by the flow that suffered undue penalty due to false sharing. Figure 27 shows the fraction of the fair-share bandwidth that the AF flow observes as a function of the time elapsed since detection of false sharing for various values of a, the fraction of the underlying bandwidth

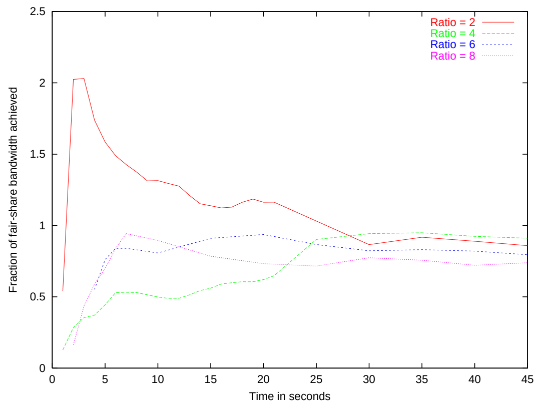

Figure 27: Fraction of fair-share bandwidth achieved by the AF flow plotted as a function of time, starting from the instant false sharing was detected and the switch to a different macroflow was made.

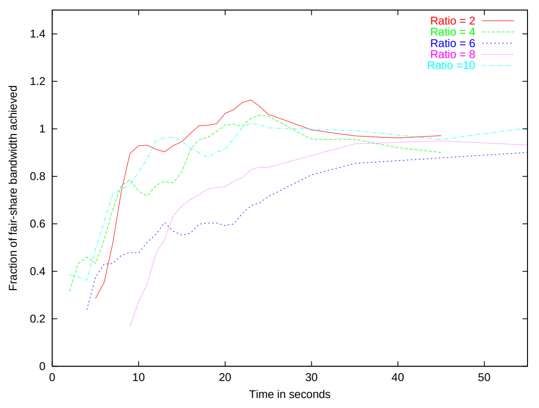

Figure 28: Fraction of fair-share bandwidth achieved by the faster of the two flows (in the path diversity scenario) plotted as a function of time, starting from the instant false sharing was detected and the switch to a different macroflow was made.

apportioned to the AF traffic. Figure 28 is a similar plot for the faster of the test flows for the unshared bottleneck scenario in the path diversity case. 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. Thus the connections, given sufficiently long lifetime, converge to their fair share of bandwidth.

5 Conclusion

In this paper we investigated the impact false sharing on shared congestion management. We studied the effect of false sharing under the two scenarios, service differentiation and path diversity, each representing a possible origin of false sharing. We showed that the penalty due to false sharing falls entirely on the faster of the two senders. We did not encounter any situation in which the slower of the two senders was persistently overloaded due to false sharing. We also analytically derived the throughput of flows undergoing shared congestion control under some assumptions.

We proposed schemes to detect false sharing based on the fact that when false sharing occurs, each flow would see a different bottleneck. Hence the losses and delays it experiences are uncorrelated to those of the other flows. We observe that delay correlation and out of order arrivals are, in general, far more robust at detection of false sharing than loss correlation.

We investigated how false sharing detection tests can be incorporated into congestion management systems. We argued in favor of starting with a default of sharing information across flows. Finally, we quantified the effectiveness of our proposals for detection and correction of false sharing and show that the connections, given sufficient longevity, would converge to their fair share of bandwidth.

References

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

[2] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. In *Proc. IEEE INFOCOM*, volume 1, pages 252–262, San Francisco, CA, March 1998.

[3] 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.

[4] H. Balakrishnan and S. Seshan. *The Congestion Manager*. Internet Engineering Task Force, Nov 2000. Internet Draft draft-balakrishnan-cm-03.txt (http://www.ietf.org/internet-drafts/draft-balakrishnan-cm-03.txt). Work in progress, expires May 2001.

[5] N. Biswajit. DiffServ Model for the NS2 simulator (Nortel Networks). http://www7.nortel.com:8080/CTL/software, 2000.

[6] D. Clark and W. Fang. Explicit Allocation of Best-Effort Packet Delivery Service. *IEEE/ACM Trans. on Networking*, 6(4), August 1998.

[7] 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.

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

[9] K. Egevang and P. Francis. *The IP Network Address Translator (NAT)*. Internet Engineering Task Force, May 1994. RFC 1631.

[10] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Trans. on Networking*, 7(4):458–472, August 1999.

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

[12] K. Kilkki. *Differentiated Services for the Internet*. Macmillan Tehcnology Series, June 1999.

[13] T. V. Lakshman, U. Madhow, and B. Suter. Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A study of TCP/IP Performance. In *Proc. Infocom 97*, April 1997.

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

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

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

[17] 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.

[18] T. Ott, J. Kemperman, and M. Mathis. The Stationary Distribution of Ideal TCP Congestion Avoidance. ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps, 1996.

[19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A Simple Model and its Empirical Validation. In *Proc. ACM SIGCOMM*, September 1998.

[20] V. Padmanabhan. *Addressing the Challenges of Web Data Transport*. PhD thesis, Univ. of California, Berkeley, September 1998.

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

[22] V. N. Padmanabhan and J. C. Mogul. Improving HTTP Latency. In *Proc. Second International WWW Conference*, October 1994.

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

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

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

[26] 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)*, pages 58–63, March 1999.

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

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

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