TCP Fairness in Multipath Transport Protocols

by

Ronald Henry Tse

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

Bachelor of Arts (Honors) in Computer Science

at

BROWN UNIVERSITY

May 2006

Author . . . .
Department of Computer Science
May 5, 2006

Certified by . . . .
John H. Jannotti
Assistant Professor of Computer Science
Thesis Supervisor

Accepted by . . . .
Thomas W. Doeppner
Director of Undergraduate Studies,
Associate Professor (Research) and Vice Chair of Computer Science

TCP Fairness in Multipath Transport Protocols

by
Ronald Henry Tse

Submitted to the Department of Computer Science
on May 5, 2006, in partial fulfillment of the
requirements for the degree of
Bachelor of Arts (Honors) in Computer Science

Abstract

Multipath transfers are a promising technique for enhancing the reliability of Internet connections, making better use of multihoming, and using spare network capacity. However, naïve designs and implementations of multipath transfer protocols risk substantial unfairness to well-behaved TCP flows.

In this thesis, I propose two novel definitions of *multipath TCP-fairness*, and will show that multipath transport can compete fairly with traditional TCP flows even on a single bottleneck. I describe three possible approaches that achieves multipath TCP-fair transfers, with congestion control performed independently on each of many subflows over which data are striped. In addition, two promising, novel approaches were implemented and evaluated in NS-2: a multi-priority based scheme that rapidly backs off on congested paths, and a bi-level scheme that performs an additional level of congestion control on top of its multiple subflows.

Examination of each schemes ability to adhere to the proposed definitions of multipath TCP-fairness was performed, and evaluation shows that both approaches meet the definition of multipath TCP-fairness while readily consuming spare bandwidth. Comparisons to previous multipath approaches demonstrate scenarios in which they exhibit unfairness to traditional TCP flows.

Thesis Supervisor: John H. Jannotti
Title: Assistant Professor of Computer Science

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multipath transfers are a promising technique for enhancing the reliability of Internet connections, making better use of multihoming, and using spare network capacity. However, naïve designs and implementations of multipath transfer protocols risk substantial unfairness to well-behaved TCP flows.

The current chapter describes the background, motivation and contributions of the thesis, as well as the proposal and reasoning for the two novel definitions of *multipath TCP-fairness*.

Chapter 2 explores three different techniques towards achieving multipath TCP-fairness, each motivated by different concerns and goals.

Chapter 3 describes the design and implementation of two novel and promising approaches in *ns-2*[1]: a multi-priority based scheme that rapidly backs off on congested paths, and a bilevel scheme that performs an additional level of congestion control on top of its multiple subflows.

Chapter 4 evaluates each schemes ability to adhere to the proposed definitions of multipath TCP-fairness and performance benefits. Results show that both of these techniques provide desirable fairness and performance properties compared to single path transport and previous multipath approaches, though Multi-Priority TCP appears to provide more desirable utilization.

Chapter 5 reviews related work, and concludes the thesis with thoughts about the prospects of multipath transport on the Internet, including the need for future work.

1.1 The Problem

Very often, multiple paths exist between pairs of Internet hosts [2]. Although current Internet routing mechanisms generally forces packets between two distinct hosts to follow a single path, a number of techniques have arisen to allow end hosts to use multiple paths of their own choosing. Overlay networks such as RON [3] allow hosts to route packets through intermediate waypoints. Source routing schemes such as Platypus [4] and NIRA [5] permit users to select from several paths and ISPs. Multihoming, for example, is a practical way to empower edge networks to route through any of several ISPs with flexibility [6, 7], and has been increasingly deployed in recent years.

With the ability to choose among multiple paths, comes the prospect of using multiple paths *simultaneously* to increase bandwidth utilization, decrease transfer times and increase reliability [8, 9, 10, 11, 12]. However, today's multipath routing techniques are not guaranteed to provide truly independent paths. A set of paths obtained by such methods may be partially distinct, but can share bottleneck links. Congestion control and the basic definition TCP-fairness in the face of such uncertainty are difficult.

Since the Internet congestion collaspe [13] of the 1980's, end-to-end congestion control has become the "social-contract" agreed upon by Internet users. Especially for TCP, which has since become the dominant transport protocol of reliable traffic. The current stability of the Internet largely relies on TCP's end-to-end congestion control mechanism and the issues it implies.

Fairness among TCP streams is already extensively studied in academia [13, 14, 15]. TCP-fairness is important: in a campus situation (multiple-users with a single outgoing link), it is undesirable to allow a single user to consume the entire campus' outgoing bandwidth. A similar argument also applies to multihomed networks and ISPs, where the most desired situation is to grant a fair share of bandwidth to each active user.

However, in the context of multipath TCP, TCP-fairness is not straightforward. A naïve aggregation of TCP flows on different paths would consume a TCP-fair-share on each path. Such aggregate would simply obtain a larger unfair share of bandwidth than other single-path TCP flows if the different paths share any bottleneck link.

In the coming chapters, we will address issues of multipath TCP-fairness, describe approaches to achieve them, and evaluate two novel mechanisms that achieves multipath TCP-fairness.

1.2 TCP Fairness conditions

TCP-compatible applies to a non-TCP congestion controlled flow when it "behaves under congestion like a normal TCP flow"[15]. Protocols commonly meet this requirement by using some form of AIMD (Additive Increase Multiplicative Decrease) congestion window management similar to TCP window increase and backoff strategies.

TCP-fairness/TCP-friendliness is a more relaxed requirement. A flow is TCP-fair if its arrival rate does not exceed the arrival of a conformant TCP connection in the same circumstances. Put another way, a TCP-fair flow sharing a bottleneck link with N TCP flows should receive less than or equal to $\frac{1}{N+1}$ of bandwidth available. More specifically, the maximum sending rate T_f (upper bound) for a TCP-fair flow is characterized by the following:

$$T_f \leq \frac{1.5\sqrt{2/3} \times B}{R\sqrt{p}} \tag{1.1}$$

where p is the packet drop rate, R the round-trip time, and B the packet size for a traditional TCP flow[13]. The crucial difference of the above two conditions is best demonstrated by an example. An equation congestion controlled TCP-fair flow (eg. TFRC) does not show similar behavior as a TCP flow would in the same conditions and therefore is not TCP-compatible. However, it is TCP-fair as it consumes less or equal bandwidth compared to a TCP flow.

1.3 Novel definitions of multipath TCP-fair

With the prospect of multipath protocols rapidly becoming a reality, we believe a definition of TCP-fairness over multiple paths is necessary. The existing simple definition is insufficient because of the ambiguity introduced by the use of multiple paths. What is the fair share of a flow that is using two independent paths? We consider two definitions:

Definition 1 ***Lenient multipath TCP-fairness.***
A leniently multipath TCP-fair protocol is one in which no ensemble of flows consumes more bandwidth on a single link than a single traditional TCP over that path would consume.

Definition 2 ***Stringent multipath TCP-fairness.***
A stringently multipath TCP-fair protocol is one in which no ensemble of flows consumes more total bandwidth than the maximum sum of the TCP-fair-share of one link and any spare bitrate of the remaining links.

The lenient definition is best understood to mean, "a multipath TCP-fair ensemble may consist of N TCP flows on N bottleneck independent paths." The condition is violated if any single TCP flow in the network loses more bandwidth to the multipath flow than it would have lost to a traditional flow. We define P as the set of possible paths, M as the bitrate consumed by a multipath flow, M_p as the bitrate consumed along a particular path $p \in P$ by a multipath flow, and T_p as the bitrate consumed on a path $p \in P$ by a traditional TCP flow.

$$\forall p \in P : M_p \leq T_p \tag{1.2}$$

$$M \leq \sum_{p \in P} M_p \tag{1.3}$$

The stringent definition can be expressed as an additional restriction to the lenient definition. To ensure fair competition on each path, the ensemble must not use more than its TCP-fair-share bandwidth on any given path [1] (see equations 1.2 and 1.3).

In addition, the total bandwidth used is bounded by the best combination of the TCP-fair -share capacity of any one path plus the spare capacity available on the other paths. Excess capacity on a given path is represented by S_p, available because all flows using the links of p are bottlenecked elsewhere. Thus the additional restriction is defined as follows:

$$M \leq \arg\max_{p \in P}(M_p + \sum_{q \in P \setminus p} S_q) \tag{1.4}$$

The stringent and lenient definitions each seem reasonable in appropriate contexts. If a network is a multi-homed customer to multiple ISPs, the customer's fairness preference depends on the placement of bottlenecks. If the bottlenecks are within its ISPs, the customer would prefer the lenient definition, which would allow its flows to use both ISP's underpowered networks. If the bottlenecks are in the access links to the ISPs, the customer may prefer the stringent definition, in order to ensure that its multipath transfers do not consume more than their fair-share, displacing its other single-path flows. This preference is especially likely in the case that the customer is itself an ISP or large organization with many users.

We believe that bottlenecked access links are common. Further, since previous work in multipath transport has focused on lenient fairness (where it has considered fairness at all), we focus primarily on stringent fairness.

1.4 Contributions

The contributions of this thesis are:

1. Lenient and Stringent definitions of multipath TCP-fairness;
2. A priority based multipath transport protocol using TCP-LP;
3. A bilevel based multipath transport protocol using TFRC; and
4. Analysis of the proposed approaches and previous work that is shown to be unfair.

[1]We define fair-share relative to an idealized TCP that does not suffer from advertised window restrictions or other artificial impediments.

Chapter 2

Approaches

Having established definitions for multipath TCP-fairness, we now consider three approaches to achieve them.

In all cases, we assume that multiple paths are available for use, but details of these paths are unknown. Just as today, an end-host knows little about the path to another end-host (except what it may determine through experimentation), a host considering multipath transport may select from N paths, but knows nothing specific about each of the paths. Some of these paths may be the same or similar. Paths may or may not share bottleneck links.

Due to the focus on fairness of this thesis, some practical details of retransmission, timeout handling and stream reassembly have been omitted. The interested reader looking for a good discussion of these issues is referred to the techniques used in mTCP [12] and pTCP [10]. The tactics under study perform loss detection on a per-subflow basis, similar to mTCP, are therefore not troubled by reordering between streams.

2.1 Multiple TCPs

The most straightforward way to obtain multipath transport is to stripe the packets of a single TCP across all available paths. This simplistic approach performs poorly when the paths have varying round-trip times [9]. Out-of-order ACKs obtained from faster paths falsely indicate congestive loss to TCP.

Striping the transport stream across multiple, independent TCPs, eliminates this complication. However, because little is known about the true "independence" of paths, further work must be done to meet even the lenient definition of multipath TCP-fairness . In addition to independent striping, end-points may attempt to explicitly detect when flows share a bottleneck, and coalesce the shared flows, as is done by mTCP [12]. Assuming the detection mechanism is reliable, shared bottleneck detection is leniently multipath TCP-fair , because no subflows will share a bottleneck.

However, bottleneck detection is *insufficient* to meet the stringent definition. Even without shared bottlenecks, an mTCP-style ensemble using congested paths could consume $\sum_{p\in P} T_p$ instead of $\arg\max_{p\in P}(M_p + \sum_{q\in P\setminus p} S_q)$.

In order to meet the stringent definition, a multipath transport might use an ensemble consisting of multiple "fractional" TCPs ($\frac{1}{N}$-TCPs). Two tactics might be used to create fractional TCPs: reduced-aggression TCP and fractional equation-based TCP.

A reduced-aggression TCP is a TCP whose parameters have been changed to reduce its rate of increase, similar to the techniques used by Hacker et al. [16] for making better use of high-bandwidth paths.[1] TCP can be forced to compete less aggressively with other TCPs by reducing the AIMD increase rate to $1/N$ (by multiplying the round-trip time by N). While this technique has been shown to create less aggressive TCP flows, the bandwidth consumption of the resulting flows is not related to $1/N$ the bandwidth of a standard TCP flow.

An ensemble of N flows with an additive increase parameter of $1/N$ will have increase behavior similar to that of TCP, incrementing its congestion window by one per RTT. The decrease behavior, however, is very different. Because each fractional flow responds to loss independently, a loss will cause only one flow to decrease by half. The ensemble will thus decrease only to $\frac{(N-1)+\frac{1}{2}}{N}$. With $N = 4$, for instance, the overall decrease caused by one loss on the path would be $\frac{7}{8}$ instead of $\frac{1}{2}$.

Without further improvements, this technique does not create true $1/N$ flows, and does not meet the fairness criteria. While the technique looks effective for consuming spare bandwidth in controlled situations, choosing the proper value of N to balance between less aggression and more effective use of the network appears challenging [16].

Fractional equation-based TCP seems more promising. The TCP response function [17] provides an upper bound on the sending rate T of a TCP flow based on the packet size s, round-trip time R, loss rate p, and retransmit timeout value t_{RTO}:

$$T = \frac{s}{R}\sqrt{\frac{2p}{3} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1 + 32p^2)} \tag{2.1}$$

This response function forms the basis for equation-based flow control [14]. A simple way to meet the stringent fairness criteria is to transmit over N paths at $1/N$ times the TCP response function. When these paths are independent, each subflow will take $1/N$ of its fair bandwidth on its path. Together, these sum to at most the fair-share bandwidth on the path from which a traditional TCP flow would have received the most bandwidth. The per-link fairness criterion is met trivially, as each subflow consumes only a fraction of its fair-share on its path. Even if all paths are shared, the combined bandwidth can at most equal the bandwidth computed by the original TCP response function.

This approach, however, may vastly underutilize the network. Consider five independent paths, one of which has far greater capacity than the rest. All paths contains a single existing TCP flow. Stringent multipath TCP-fairness allows the multipath flow to consume a TCP-fair-share of the large path's bandwidth. A fractional TCP

[1]In the thesis, we refer, perhaps improperly, to the multipath version of the reduced aggression TCPs as the "Hacker" scheme. This is slightly different from the original proposal by Hacker et al., which applied primarily to a single path.

approach would consume just 1/5 of the large path's bandwidth plus 1/5 of each of the small paths, which could be much less.

2.2 Multi-Priority Congestion Control

A background transport protocol is designed to be unobtrusive to traditional TCP flows—it should only use spare capacity. TCP-LP [18] (TCP Low Priority) and TCP-Nice [19] are two examples. Both use delay measurements to detect congestion and back off before traditional TCP reacts to the congestion. In addition, they react quickly to utilize excess bandwidth when it is available.

Assuming the existence of a perfect background transport protocol, a multipath congestion control algorithm can meet the stringent fairness requirements by striping across N paths, one of which is a traditional TCP stream, while the rest are background streams. Since a background stream is intended to consume only spare bandwidth, this scheme ensures that the first stringent criterion is met – the TCP-fair-share capacity of a single link is consumed by the traditional stream, while the background streams consume only spare capacity. The stringent restriction is trivially met because the combination of a single TCP flow and any number of background flows on a single link is not supposed to be more aggressive than the TCP flow itself. This technique is similar in spirit, but in a different context, to Hacker et al.'s use of a TCP flow with an additional number of reduced aggression TCP flows.

In reality, background transport protocols are not perfect. Background streams must emit periodic probes in order to adapt to the dynamic spare capacity. When a traditional TCP flow shares a bottleneck link with background flows, the background flows may consume an unfair portion of the link bandwidth due to their probes. Section 3.1.1 explains how a Multi-Priority TCP can work around this limitation and maintain stringent multipath TCP-fairness.

2.3 Bilevel Congestion Control

Bilevel Congestion Control uses a single "master" congestion control mechanism to determine the overall sending rate, and divides that among a number of subflows running their own congestion control. The send rates of the subflows determine the ratio in which traffic is allocated to different subpaths (Figure 2-1). Because Bilevel Congestion Control has a single master mechanism that reacts to congestion in a TCP-fair way, it is fair to competing flows even when all subflows share a single bottleneck. Consider a Bilevel Congestion Control scheme based on TCP. The real congestion window for each subflow is relative to its share of the subflows' total windows:

$$\text{actualcwnd}_i = \text{cwnd}_M \cdot \frac{\text{cwnd}_i}{\sum_{j=1}^{N} \text{cwnd}_j} \tag{2.2}$$

Subflows in a Bilevel TCP scheme independently perform loss detection. When a subflow detects a loss, it reduces its window, and reports the loss to the master TCP,

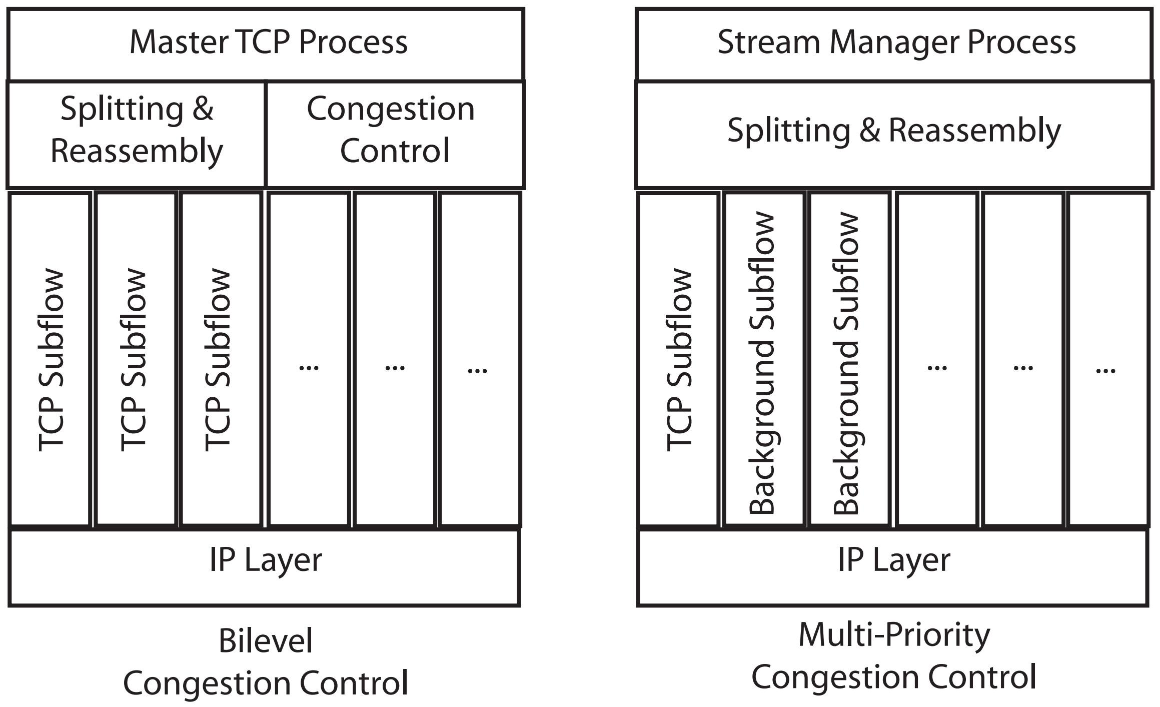

Figure 2-1: The architectures of Bilevel TCP and Multi-Priority TCP . Bilevel TCP uses congestion control at two levels to ensure that each subflow, and the flow as a whole are controlled. Multi-Priority TCP uses one normal TCP as well as a set of background flows to ensure that the ensemble as a whole will not unduly displace standard TCP flows.

which reduces its own window:

$$\begin{aligned}\text{cwnd}_i &= \frac{\text{cwnd}_i}{2}\\ \text{cwnd}_M &= \frac{\text{cwnd}_M}{2}\end{aligned}$$

Similarly, on success, both master and child increase their windows by $\frac{1}{\text{cwnd}}$ (note that each maintains its own cwnd). As a result, when a packet is lost on a subflow, the master process reduces the *overall* sending rate by half, *and* shifts traffic away from the link on which the loss occurred. A key feature of this design is that we allow congestion windows to become fractional, so that excessively poor links do not excessively penalize the ensemble.

Given N links, each with capacity C and k existing flows, the TCP-fair-share bandwidth $T = \frac{C}{k+1}$. A Bilevel TCP that treats the ensemble as a huge collection of links will obtain $\frac{NC}{Nk+1} \leq \frac{2C}{k+1}$, which is within a small constant factor of its guaranteed fair-share, and when k is large or N is small, it would be close to exact fairness.

We chose to implement a Bilevel Congestion Control scheme based on TFRC rather than TCP because it was simpler to add the required functionality. Chapter 3 presents the actual implementation details of Bilevel TCP.

We will see in Chapter 4 that in practice, although it is multipath TCP-fair, Bilevel TCP tends to consume somewhat less than its fair-share bandwidth when competing with traditional TCP flows.

Chapter 3

Implementation

This section describes an *ns-2* based implementations of a Multi-Priority congestion controlled protocol that uses TCP-LP as its background transport mechanism, and a Bilevel congestion controlled protocol that uses TFRC as its base congestion control mechanism. Our implementations of both implement standard reliable transport features, including the handling of retransmissions, duplications, and reordering. Packet reordering on each path is dealt with within subflows, so packet assembly of the ensemble as a whole is straightforward. An extra field in TCP headers of subflows for packet assembly is required, but the overhead remains small (0.5% for 64-bits in 1.5Kb packets). Because the focus of this thesis is fairness, such details are beyond the scope of this work. We refer the interested reader to the assembly techniques used in pTCP [10] and mTCP [12] for a good discussion of such issues. We describe the implementation details that are relevant to fairness and congestion control below.

3.1 Multi-Priority TCP

We have implemented a Multi-Priority TCP in *ns-2* in which data is striped over one traditional TCP flow and multiple TCP-LP flows. We believe that TCP-Nice would be equally suitable, and the choice of using TCP-LP in this thesis was based solely upon the availability of its *ns-2* simulation code.

TCP-LP's basic mechanism to avoid competing with traditional TCP flows is a rapid backoff strategy. TCP-LP shrinks its congestion window in response to either increased delay or a single loss. TCP-LP will immediately reduce its congestion window size to one in response to two congestion indications within a short period of time (3 RTTs by default). Traditional TCPs take $\log_2$(cwnd) RTTs to reduce their window to one, as they halve their window once per RTT. By racing to the bottom, TCP-LP gets out of the way before traditional TCPs respond.

Once it has reduced its window to one, TCP-LP emits a data packet once per RTT to measure the delay. From the previously measured minimum delay it determines whether the current delay is indicative of congestion. If so, the sender does not increase its congestion window size in response to the ACK. If the the ACK arrives quickly enough, the congestion is deemed to have subsided, and TCP-LP grows its window.

TCP-LP's periodic probes constitute a departure from an ideal background transport mechanism. These probes consume bandwidth, and could lead to significant unfairness in a Multi-Priority TCP scheme that employed numerous background flows.

3.1.1 Packet Borrowing

Because of the probe packets, N TCP-LP flows would send N probe packets per RTT. Let cwnd_B be the congestion window size of the "foreground" or "base" flow of Multi-Priority TCP. This constant traffic brings the total traffic generated by our ensemble to a maximum of $\mathrm{cwnd}_B + N$ packets per RTT. Therefore when a traditional TCP flow shares a bottleneck with many Multi-Priority TCP subflows, they may consume an unfair portion of available bandwidth and displace the competing TCP flow. Figure 3-1 demonstrates the problem, when Multi-Priority TCP competes with a single TCP on a single physical path. As Multi-Priority TCP increases the number of background subflows, the ensemble consumes an ever increasing fraction of the bandwidth on the path. Regardless of the number of flows in the ensemble, the multipath TCP-fairness definitions demand that the Multi-Priority TCP flow consume only 50% of the bandwidth in this case.

We address the non-ideal nature of background transport by accounting for probe traffic in the "foreground" TCP. When a background stream has reduced it window size to one, any packets that it sends are taken from the traditional TCP by decrementing its congestion window. If the response to the probe indicates that the background flow may ramp back up, it does so.

3.1.2 Packet Returning

While increasing fairness to competing TCP flows, we have reduced the aggressiveness of Multi-Priority TCP by taking packets from the base flow. Every packet that a TCP-LP flow takes from the base flow had the potential to increase the window of the base flow by $1/\mathrm{cwnd}_B$. When an ACK returns to the TCP-LP flow, its window only grows if it senses that the link is not congested, determined by the reduction of delay on the sending path of the probe packet. If the TCP-LP flow had not taken the packet, the base flow would have sent a data packet and received an ACK, thereby increasing the its window by $1/\mathrm{cwnd}_B$. To restore normal TCP behavior, if a TCP-LP flow receives a slow ACK and does not increase its own window because of delay considerations, the base flow's window is increased by $1/\mathrm{cwnd}_B$.

Packet borrowing is not an absolutely precise way to account for the probe traffic of TCP-LP. First, the probe packet was actually sent along the TCP-LP path, which is likely to be different from the path used by the traditional TCP. Treating its acknowledgment like the acknowledgment of a packet from the traditional stream is a small departure. These probe packets are, by definition, being sent on paths that are likely to be congested. Their return is likely to be slower than on the traditional TCP's path, meaning that the traditional TCP is penalized slightly in its ability to grow. In order to minimize any error introduced, the total number of probe packets

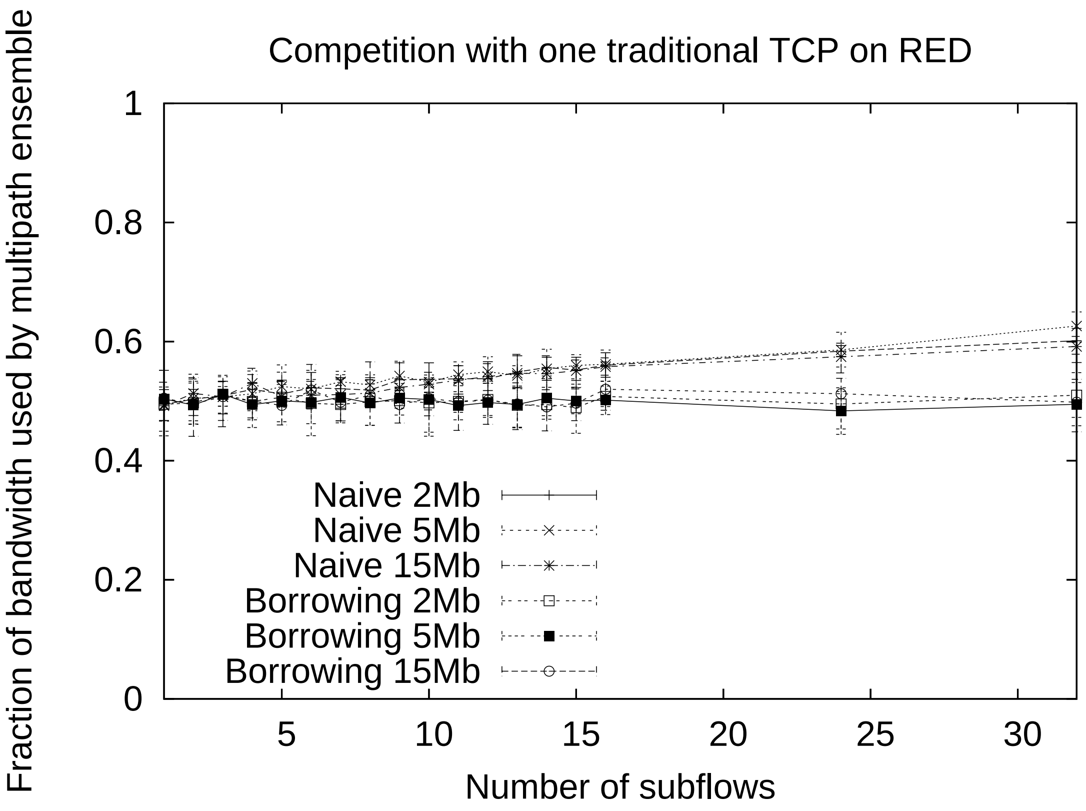

Figure 3-1: On drop-tail queues, Multi-Priority TCP streams based on a naive aggregation of TCP-LP flows with a TCP flow can be unfair to TCP streams, especially at lower bandwidths. Each additional TCP-LP sends additional probe packets that consume TCP's rightful share. Packet borrowing allows an even sharing of the path, regardless of path capacity or the number of TCP-LP subflows.

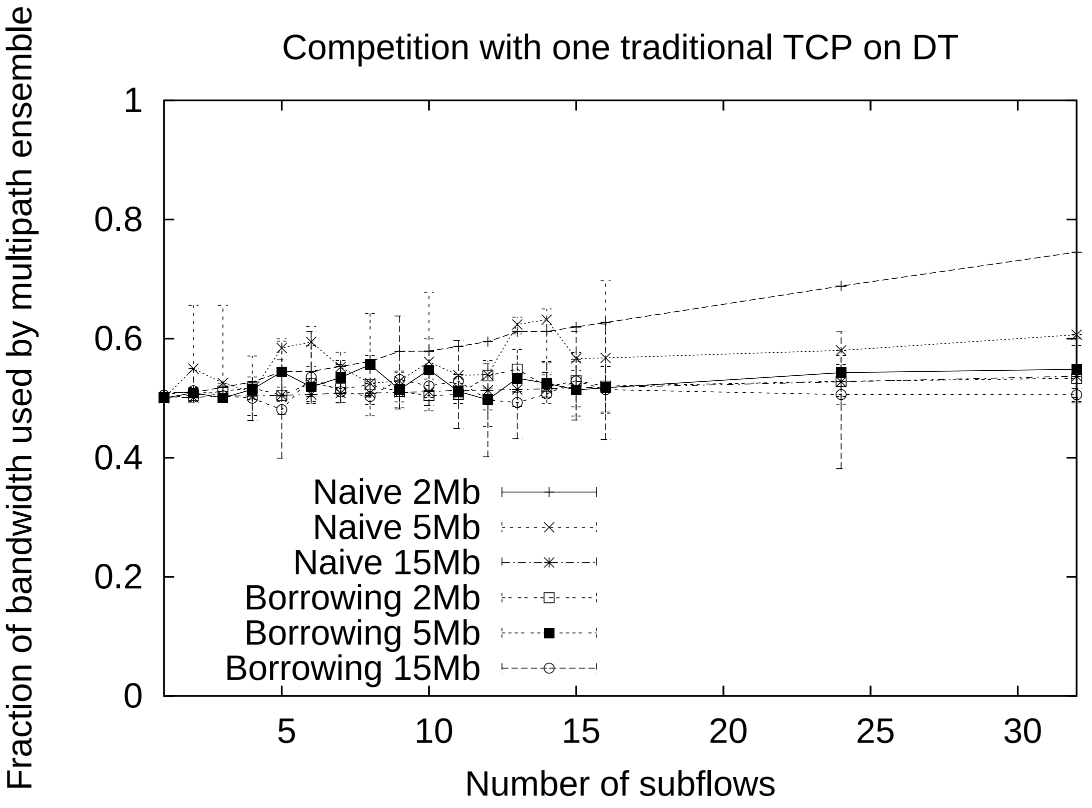

Figure 3-2: On RED queues, Multi-Priority TCP streams based on a naive aggregation of TCP-LP flows with a TCP flow are unfair to TCP streams regardless of link speed, due to mostly fair queue drops among flows. Each additional TCP-LP sends additional probe packets that consume TCP's rightful share. Packet borrowing allows an even sharing of the path, regardless of the number of TCP-LP subflows.

sent by the ensemble of background flows is rate-limited to one packet per RTT of the traditional flow, regardless of the number of probing TCP-LPs.

Figure 3-1 shows that the packet borrowing technique fixes our Multi-Priority TCP implementation's aggression, regardless of the number of flows used. Although Packet Borrowing fixes the non-ideal nature of TCP-LP as a background flow, the technique is more broadly applicable. For example, had we developed Multi-Priority TCP with TCP-Nice, we would have defined any packet that is emitted when the congestion window just returned to one from fractional (below one) as a "probe" packet, and performed the same accounting as described above.

3.1.3 RED

Random Early Detection [20] is a simple active queue management scheme that drops packets with increasing probability as router queues grow in size. Although there is still a lack of consensus that RED is an unqualified benefit, there is evidence that RED improves fairness and lowers latency, particularly for short flows. Unfortunately, the bottleneck detection mechanism of mTCP relies on drop tail queues in order to find correlated losses. Since TCP-LP's congestion avoidance mechanism depends on latency, rather than correlated loss, experiments in this thesis were all performed with RED deployed, and results show little difference with traditional drop tail queues. Experimental graphs with drop tail queues are omitted due to similarity and clarity.

3.2 Bilevel TFRC

Our implementation of Bilevel Congestion Control uses TCP Friendly Rate Control (TFRC) [17] as its base congestion-control mechanism[1]. TFRC is an equation-based flow control protocol that is meant to be TCP-fair, while providing a smoother sending rate than TCP's burstier pattern. TFRC uses the TCP response function (2.1) to calculate a fair sending rate T, based on the packet size s, round-trip time R, loss rate p, and retransmit timeout value t_{RTO}.

We chose to implement Bilevel Congestion Control with TFRC rather than TCP because it provides a simpler way of combining flow control information from subflows. To implement a Bilevel TCP system, one must define the ways in which the upper-level TCP responds to each event that responds on the subflows (e.g., ACKs, losses, timeouts, etc.). TFRC, on the other hand, suggests a simple way to combine individual flow control into a superflow, which we will describe below. In addition, providing reliability on top of the unreliable TFRC proved easier than separating TCP's reliability and congestion control mechanisms.

Bilevel TFRC is controlled by one master TFRC process, which controls a number of TFRC subflows. Each subflow performs independent congestion control and loss calculations which determine their share of the overall sending rate. The new TCP-

[1]Our implementation uses a modified RFC3448-compliant TFRC for *ns-2* [21] instead of the original *ns-2* implementation of TFRC.

fair sending rate R'_i of subflow i (of N subflows) is:

$$R'_i = R_M \frac{R_i}{\sum_f^N R_f} \tag{3.1}$$

where R_M is the previous sending rate of the master process and R_i the previous sending rate of subflow i. The total sending rate (the master's sending rate), R'_M, is calculated as a combination of the individual loss and send rates from its children using the TCP Response function 2.1. The combined loss rate f^M_{lost} is described in the next section. Other than the total send rate, retransmissions are also handled exclusively by the master process.

3.2.1 Loss Estimation

TFRC employs a feedback mechanism in which the receiver sends a feedback packet to the sender once per RTT that contains information about the current receiving rate T_r, frequency of loss f_{lost}, and number of lost packets *losses* within the last RTT. From these values, the sender can check whether its current sending rate T_r is TCP-fair by comparing T_r to T_F, where T_F is the fair share rate calculated by the TCP response function with the new f_{lost} and RTT values.

The TFRC receiver calculates the loss frequency f_{lost}, by calculating a weighted average of packets that arrive in each of the past several *loss intervals*. A loss interval is the longest sequence of packets, at least one RTT in length, that contains no losses that are more than one RTT apart. TFRC counts the (weighted) average number of packets that successfully arrive in each loss interval (s_i):

$$\hat{s}_{(1,N)} = \frac{w_i s_i}{\sum_{i=1}^N w_i} \tag{3.2}$$

With weights defined by:

$$w_i = \begin{cases} 1 & \text{if } 1 \leq i \leq N/2 \\ 1 - \frac{i-N/2}{N/2+1} & \text{if } N/2 < i \leq N \end{cases}$$

This treats all losses in a single loss interval as a single loss event. The reported loss estimate is then $1/\hat{s}_{(1,N)}$, which considers the last N loss intervals. The TFRC paper recommends $N = 8$, yielding weights of 1.0, 1.0, 1.0, 1.0, 0.8, 0.6, 0.4 and 0.2 respectively.

To find an accurate loss estimate f^M_{lost} for the master TFRC process, we have to combine loss reports from each child. A naïve approach one can take is to combine them by simply summing up loss frequencies. Such an approach may greatly overestimate f^M_{lost}. When all subflows share the same bottleneck, f^M_{lost} should be similar to each f^i_{lost}, rather than $\sum_{i=1}^n f^i_{lost}$ because the master flow should choose a rate equal to the rate determined for the given path.

An improved approach sums the loss frequencies, each weighted by the fraction of subflow's send rate to the total send rate. The product of the send rate and the loss rate of a child gives the real probability of losing a packet in the ensemble by using that particular child:

$$f^M_{lost} = \sum_i^N f^i_{lost} \frac{R_i}{\sum_f^N R_f} \tag{3.3}$$

3.2.2 Reliability

TFRC is an unreliable protocol. Bilevel TFRC implements reliability in the master TFRC process in order to produce a reliable protocol that can be compared to other approaches. A Bilevel TFRC receiver adds a list of NACKs to each feedback packet sent by each child so that the Bilevel TCP sender can perform retransmissions. Bilevel TCP tries to use a different path to retransmit a packet than the path upon which it was originally transmitted. It is important to perform retransmission at the highest possible layer. If the responsibility for sending a given packet is irrevocably given to a particular subflow, a link failure on the path used by that subflow could stall the transfer indefinitely.

TFRC's lack of built-in reliability was one reason we chose it for our implementation. Conventional TCPs tightly link congestion control and reliability, making it difficult to implement alternate-path retransmission.

Bilevel TFRC receivers send feedback on each of the paths they are using, resulting in potentially duplicated NACKs. These duplicate NACKs can arise at very different times because they are sent over different paths. The time for an acknowledgement for a retransmitted packet to arrive at the sender is 1 RTT, and at least an additional 0.5 RTT required for the last duplicate NACK to be received by the sender. Therefore a Bilevel TFRC sender does not expire NACKs until 1.5 RTTs after it has sent a retransmission. After 1.5 RTTs, the sender treats an incoming NACK as a new loss report, and schedules another retransmission.

3.2.3 Statistics collection

In both the original *ns-2.27* TFRC implementation [17] and the newer RFC3448-compliant TFRC for *ns-2* [21], the TFRC receiver stores information regarding the arrival of previous packets for throughput estimation. For each packet, the receiver records whether the packet has arrived, and its timestamp and RTT information. Unfortunately, both implementations use a fixed size array to store this information, and do not deal with wraparound correctly. After wraparound, the loss estimator will incorrectly consider packets that arrived in the *previous* run through the array to have arrived in the current iteration, causing the receiver to incorrectly believe that lost packets were received. Eventually, the receiver will estimate $f_{lost} = 0$. Using the default parameters that store information for $100,000$ packets, on a 5Mb link with 1Kb-sized packets, it only takes $\frac{100,000}{5,000} = 20$ seconds to wraparound, causing TFRC to consume an ever increasing fraction of bandwidth.

Our simulations include an extension to the RFC3448-compliant TFRC implementation to support correct wraparound. Our modified TFRC resets array slots that are older than one RTT, as TFRC only examines the most recent RTT for throughput estimation.

Chapter 4

Evaluation

This section compares and contrast the behaviors of Bilevel TFRC and Multi-Priority TCP. Each is intended to meet the stringent multipath TCP-fairness definition while making effective use of spare network capacity. In addition to the two novel multipath TCP-fair protocols, we have developed *ns-2* simulation code for $1/N$ fractional TFRC and the parallel TCP scheme as described by Hacker [16]. Finally, we analyze the likely behavior of mTCP, which is not amenable to *ns* simulation. mTCP depends on shared bottleneck detection, the efficacy of which depends on wide variety of variables, such as queue sizes, packet drop schemes (AQM or drop-tail), and cross-traffic. We generally describe mTCP's best-case behavior in which we assume that it shared-bottleneck detection is successful.

4.1 Shared bottlenecks

The first test of fairness for a multipath TCP scheme is its ability to cooperate with traditional TCP flows when independent paths are unavailable. To test this degenerate case, we simulated a traditional TCP flow along a single link, and then added a multipath TCP flow to the link. Multipath TCP-fairness (whether stringent or lenient) would demand that the multipath flow consume no more bandwidth than the traditional flow. Figure 4-2 shows that both approaches perform well. The Hacker scheme is less fair because the RTT-penalized flows are not true $1/N$ flows or back-

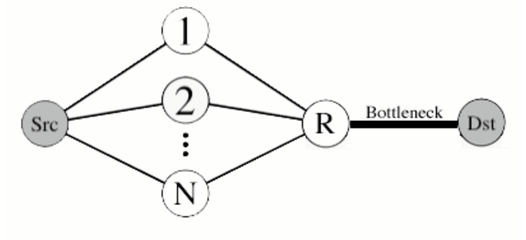

Figure 4-1: A network with N (16, in our experiments) subflows sharing a bottleneck link between the source and the sink.

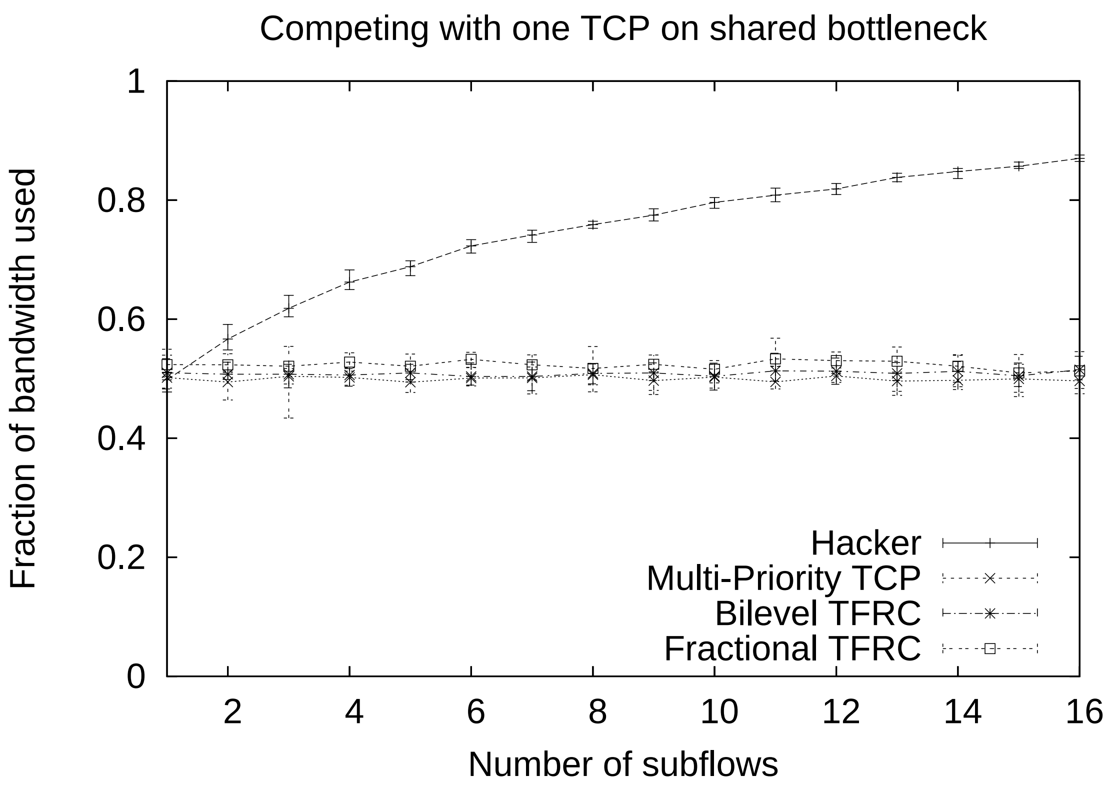

Figure 4-2: The percentage of bandwidth used by a multipath flow using an increasing number of subflows. The multiflow competes with a single TCP flow through a single bottleneck. Multi-Priority TCP and Bilevel TFRC show nearly perfect fairness, while the Hacker scheme becomes less fair as more subflows are used.

ground flows. They are less aggressive than traditional TCP flows, but each additional flow takes more bandwidth from the competing TCP flow. In this experiment, if the detection mechanism of mTCP succeeds, it would only utilize one TCP flow and mTCP would also be fair.

4.2 Independent paths

The opposite degenerate case for multipath transfer is when every available path is truly independent with respect to bottlenecks. For these experiments, we simulate the topology shown in Figure 4-3. The network has 16 physically distinct paths from the source to the destination, all with equal capacity. In the first experiment, we begin with an empty network, and then allocate multipath subflows to more and more of the paths to determine how effectively each scheme is able to use the spare capacity in the network. Figure 4-4 shows the differences between a theoretically optimal scheme, Multi-Priority TCP, and Bilevel TFRC. Multi-Priority TCP and the Hacker scheme are both effective, their non-traditional TCP subflows act almost precisely like TCP when they are used on empty paths. mTCP would perform similarly.

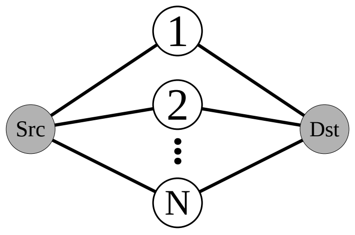

Figure 4-3: A network with N (16, in our experiments) physically distinct paths between the source and the sink.

In contrast to the other schemes, Bilevel TFRC loses some of its effectiveness as additional links are added. With sixteen links, Bilevel TFRC is only able to use 80% of the total network capacity. Some of this inefficiency arises because Bilevel TFRC experiences small oscillations in the way it allocates traffic to subflows, causing one flow to experience losses while the remaining flows are slightly under-filled. While its performance is still 13 times better than an unmodified TCP, the Bilevel TFRC scheme clearly has room for improvement when dealing with a large number of subflows.

One noteworthy feature of Figure 4-4 is that the fractional TFRC scheme consumes slightly more bandwidth than schemes that are, in effect, multiple independent TCP flows. That is, each $1/N$ flow is acting like an entire TCP (and then some). The explanation is that when a path is underutilized, TFRC will experience no losses, and its send rate will grow. Therefore, although each fractional TFRC flow is sending at only $1/N$ of its nominal send rate, that nominal send rate is growing to approximately N times the true individual path bandwidths. This finally leads to congestion, and an equilibrium is found. We believe the final difference is due to TFRC's smoother evolution, which reduces bursty losses even when the router buffers are large enough.

The last experiment showed that all multipath protocols were mostly effective at consuming spare network resources on multiple paths. We now consider how TCP competition affects behavior on independent paths. We begin where the last experiment ended, with a 16 flow multipath transfer on the topology of Figure 4-3. We show how the bandwidth consumed by the multipath transfer changes as traditional TCP flows are added as competition on an increasing number of paths. Figure 4-5 shows how much bandwidth each protocol uses when competing with other TCP flows. With 16 independent paths, a single fair-share TCP flow would obtain 1/32 of the total system bandwidth (1/2 of one path). The Y axis in the figure is normalized to fair-share TCP units. The Multipath-TCP-Fair line shows how an optimal protocol should behave.

Bilevel TFRC is slightly more conservative when competing with existing flows, and that this conservatism reduces its effectiveness at soaking up excess bandwidth compared to Multi-Priority TCP . Both schemes converge to using a single TCP flow's worth of bandwidth as the number of competing TCP flows increases toward the number of available physical paths, meeting exactly the additional restriction of the stringent multipath TCP-fairness definition.

The Hacker scheme does not meet the definition of stringent fairness. On one

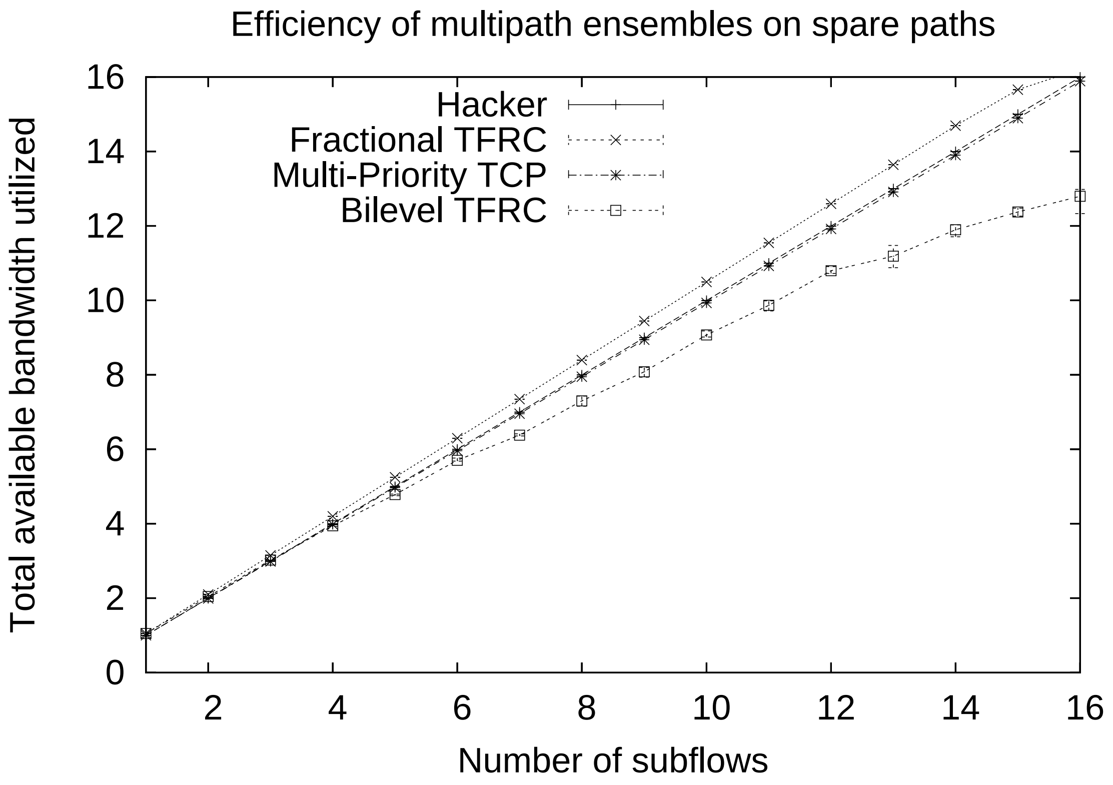

Figure 4-4: The fraction of single-link TCP throughput achieved by a multiflow as the number of subflows increases in a network with 16 physically distinct paths. Each subflow is assigned to a separate path. An optimal scheme would scale linearly with the number of subflows, up to the number of paths.

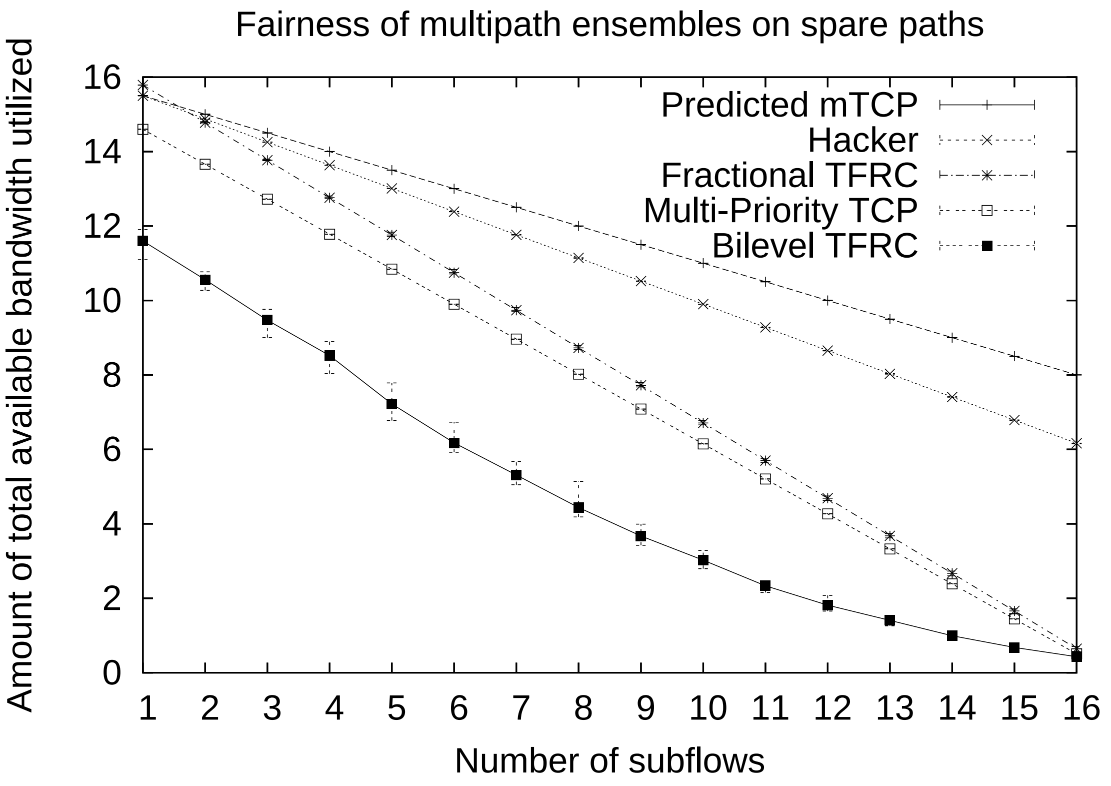

Figure 4-5: As more competing TCP flows are added to the network, both Multi-Priority TCP and Bilevel TFRC converge to using a single TCP flow's worth of bandwidth.

path, the Hacker scheme would compete fairly with its base TCP flow. However on all other paths, a stream with a long virtual RTT would compete with a traditional stream. These virtual RTT flows would consume a steady fraction of traffic from the traditional flows. When all paths contain TCP competition, we see that the scheme would use 15 times the stringently fair allowed throughput on each path. The Hacker scheme *does* meet the definition of lenient fairness at this point, but adding additional flows would compromise lenient fairness once any two flows share a same path.

mTCP is not intended to achieve stringent fairness. In this scenario, assuming correct bottleneck detection, mTCP would stripe across no more than 16 TCPs, each on a different path. This behavior meets our lenient fairness definition.

4.3 Heterogeneous Paths

The cases so far examined what happens when a single path is unwittingly used multiple times or when there are multiple, identical paths. We now examine the behavior of each protocol in more challenging conditions, when the available paths differ in some way.

These experiments consider bandwidth, delay, and loss. Each experiment uses two

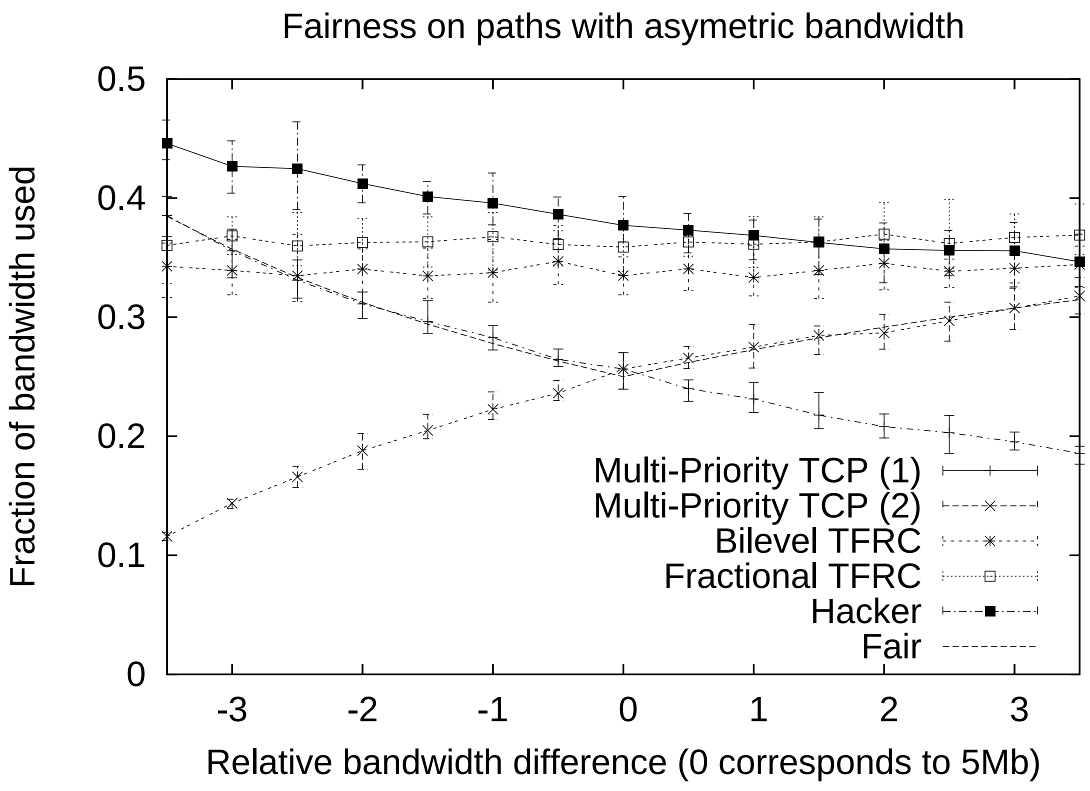

Figure 4-6: Bandwidth consumption on two paths, each with one prior TCP flow. The first path has a capacity of 5Mb. The second's varies from 1.5Mb to 8.5Mb. A stringently multipath TCP-fair flow will consume no more than half of the larger path.

paths, and the multipath protocols are configured to use two subflows, one on each path. On each path there is a competing TCP flow. To illustrate the effect of heterogeneous parameters, we graph the bandwidth utilized by the multipath ensemble as we change the variable parameter on the second path while keeping the variable constant on the first path.

4.3.1 Bandwidth

Both Multi-Priority TCP and the Hacker scheme both have a notion of a "primary" TCP flow and less aggressive subflows. When the primary flow is allocated to the wrong path, each of these schemes receives less bandwidth. Figure 4-6 shows the bandwidth used with two paths whose bandwidth varies. Each path has a single competing TCP flow. An incorrect placement (such as the bottom left points on the Multi-Priority TCP flow) results in a very suboptimal throughput. Bilevel TFRC and mTCP do not suffer this problem, because all of their subflows are identical. To augment this situation in Multi-Priority TCP, it is possible that a "switching" scheme be implemented that allows switching paths between the primary flow and a low-priority subflow. Due to already available RTT and loss information in all subflows,

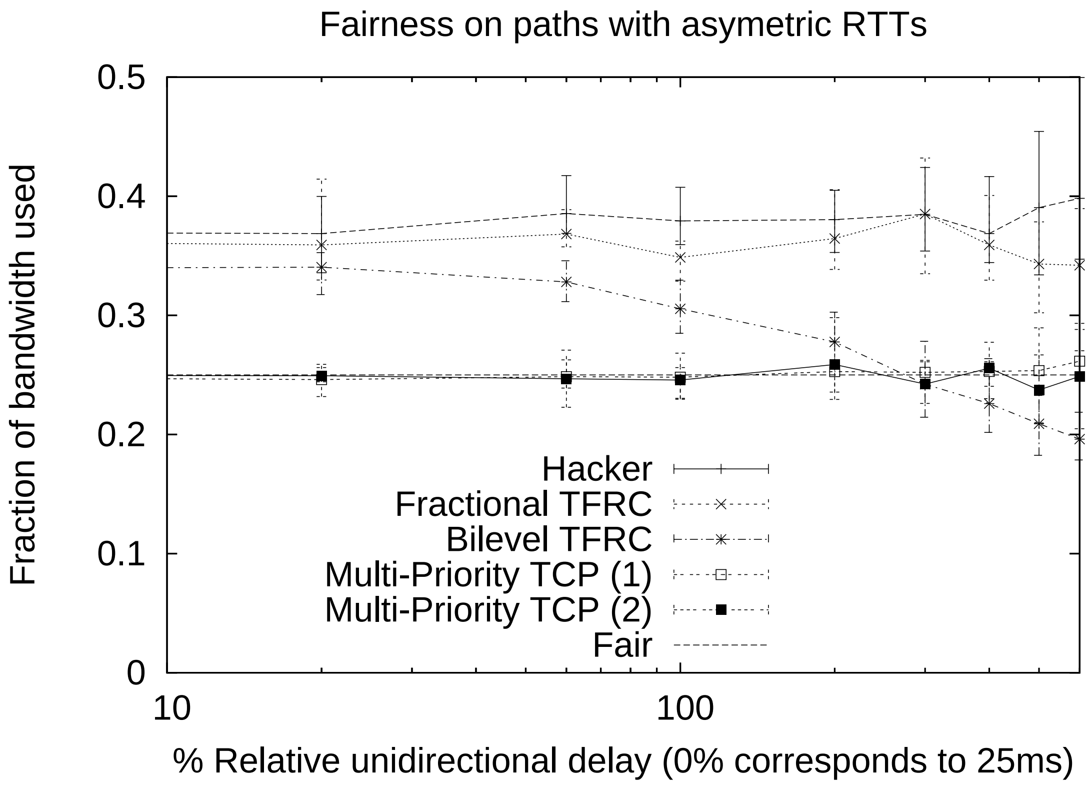

Figure 4-7: Bandwidth consumption on two paths, each with one prior TCP flow. The first path has a one-way delay of 25ms. The second's relative delay ranges from 10% to 600% (28ms to 150ms). A stringently multipath TCP-fair flow will consume no more than half of the first path (25% of the total traffic).

one can find the path with maximum bitrate potential by the TCP response function (2.1).

Bilevel and fractional TFRC, however, suffer from a different problem. Bilevel TFRC is only a constant-factor approximation to stringent fairness. Stringent fairness demands that the throughput of the multipath flow be bounded by $\frac{1}{2}$ of any single path. In this scenario Bilevel TFRC competes for bandwidth as if there were a single link and two other flows, so it takes $\frac{1}{3}$ of the bandwidth instead of $\frac{1}{4}$. In the worst case, the extra competition is bounded by a factor of two.

4.3.2 Delay

Figure 4-7 shows the effects of heterogeneous delay paths on the various multipath protocols. Evidenced by the flat trends on different delays, we see that Multi-Priority TCP and the Hacker scheme are not affected by asymmetric delay, due to their seperate subflow congestion control. In the figure, Multi-Priority TCP remains very close to multipath TCP-fairness at all delay values. Although the Hacker scheme uses reduced aggression flows, is not stringently multipath TCP-fair and consumes

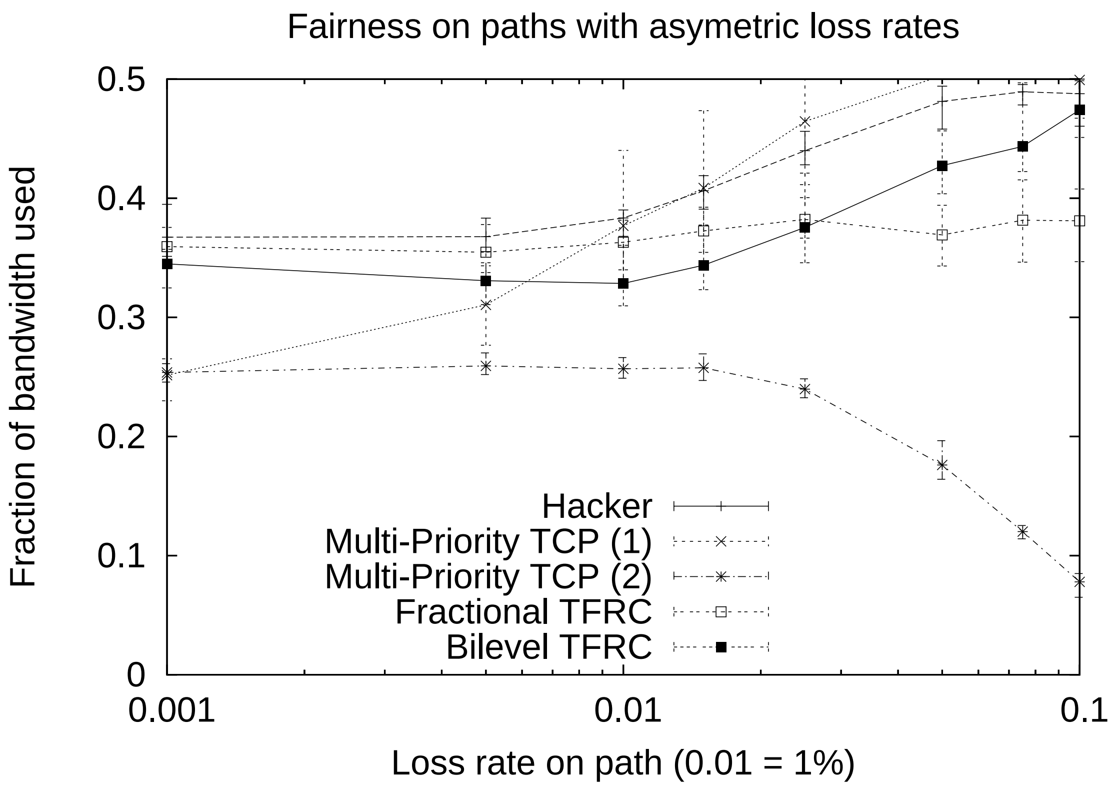

Figure 4-8: Bandwidth consumption on two paths, each with one prior TCP flow. The first path has no non-congestive loss. The second's varies from 0.1% to 10% loss. A stringently multipath TCP-fair flow will consume no more than half of the first path (25% of the total traffic at 0.1% loss to around 40% of total traffic at 10% loss).

an unfair level of bandwidth.

Most of the protocols are unaffected by asymmetric delay, as evidence by their flat trends as delay changes. Bilevel TFRC, however, is quite affected when there are large latency differences (over 300%) on its two paths, and is unable to use an appropriate fraction of bandwidth. As mentioned above, Bilevel TFRC tends to consume 1/3 of the two paths. When a large RTT difference exists, Bilevel TFRC actually becomes more "fair" to the competing flows, with its usage at 0% RTT difference at 33%, reaching down to 25% at 300% RTT difference, and further to 20% at 600% RTT difference. This is likely a result of using a master congestion control mechanism that depends heavily on RTT to determine its send rate. There is no single RTT that is suitable in this situation. Unfortunately, because TCP itself depends on RTT, any multipath TCP friendly mechanism will use RTT in some way.

4.3.3 Loss

Figure 4-8 shows the effects of heterogeneous loss paths on the various multipath protocols. In this case, we introduced artificial, non-congestive loss onto the paths, as indicated by the loss rates in the figure. Traditional TCP flows perform poorly under these conditions, so we expect the multipath schemes to perform more aggressively in this case.

The Hacker scheme obtains the most bandwidth under this scenario, which is not surprising. Our prior analysis showed that this scheme behaves like a TCP with a reduced backoff value. The same mechanism is used in the $CETEN_A$ technique for reducing the effects of non-congestive losses [22]. Fractional TFRC is slightly unfair at low loss rates and is slighly too forgiving at higher loss rates. At high loss rates, due to its fractional sending rate, it is unable to utilize half of the non-lossy link. Multi-Priority TCP (1) seems to be increasingly unfair at high losses, but some of this effect is accountable to the fact that the competing TCP flow could not utillize all available bandwidth due to non-congestive losses. Thus the background subflows of Multi-Priority TCP will consume the spare capacity on the lossy link, due to the link's low congestive delay. However if the Multi-Priority TCP non-background subflow is allocated to the lossy link (Multi-Priority TCP (2)), at high losses (10%) the ensemble can only utilize a sub-optimal 8% of the available throughput. The reason being its background subflow does not consume bandwidth on the non-lossy link, and the total throughput of the lossy link is only 16% of the link pair at 10% loss. Bilevel TFRC, as mentioned in previous experiments, consumes 1/3 of the pair of the symmetric links at steady state. Here, Bilevel TFRC acts slightly unfair at lower loss rates (similar to Fractional TFRC), but as differences in loss rates increase, it starts shifting traffic away from the lossy link and obtains its stringently multipath TCP-fair bitrate mainly from the non-lossy link.

4.4 Subflow Placement

The examination of asymmetric bandwidth showed that both Multi-Priority TCP and the Hacker scheme suffer when they choose to allocate their primary flow on the "wrong" link. Bilevel TFRC flows can also suffer from a similar problem, if all of its subflows are allocated to the same physical path. Figure 4-9 shows a nearly worst-case allocation of Bilevel TFRC subflows to two paths. The top path has a competing TCP flow, and all but one of the Bilevel TFRC subflows are assigned to this congested path. Figure 4-10 shows the effect of adding more and more subflows to the bad path, while retaining only one subflow on the uncongested path. While Bilevel TFRC never violates the stringent fairness criteria, it *does* shift more and more of its traffic to the already-congested link, resulting in inefficient utilization of the network under a worst-case flow allocation.

In practice, we believe this scenario is unlikely for several reasons. First, we expect multipath routing to select from available paths randomly, or (hopefully) with a bias toward selecting dissimilar paths. In either case, there is no reason to expect that all but one of many possible paths traverses the same bottleneck.

Figure 4-11 examines in more detail the problem of subflow allocation for Multi-Priority TCP. Two paths are available, and Multi-Priority TCP uses one subflow on each path. We then introduce competitive TCP flows, either on the traditional

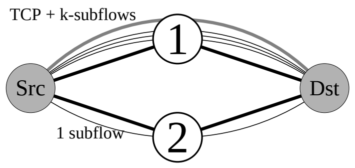

Figure 4-9: A network that shows what happens when adding more Bilevel TFRC subflows to a congested path.

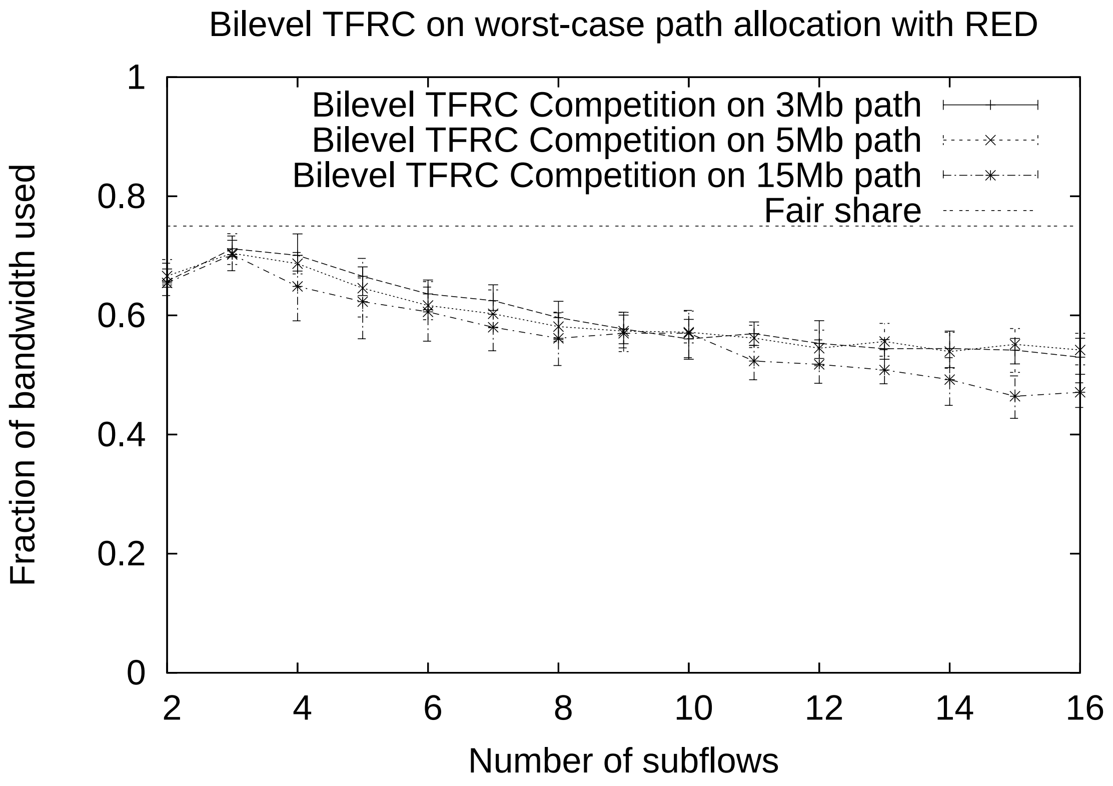

Figure 4-10: The fraction of optimal bandwidth used by all flows in the system decreases as more Bilevel TFRC subflows are added to the link occupied by a TCP flow. The problem is exacerbated at higher rates because the impact of packet loss is more severe in a long-fat network. As the number of shared subflows grows, the ratio will converge to 50%, and the alternate path will be completely unused.

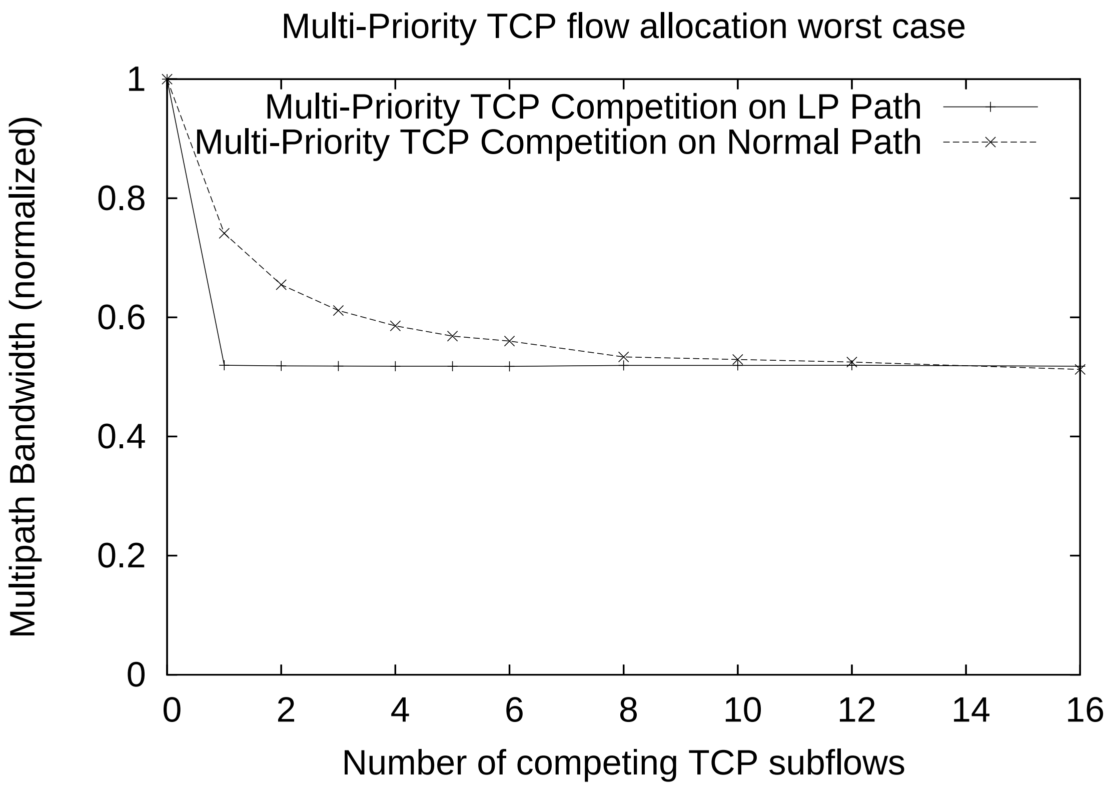

Figure 4-11: Bandwidth obtained by Multi-Priority TCP as competitive TCP flows are introduced on one of two available paths. Multi-Priority TCP is able to retain a larger fraction of its original bandwidth when its TCP-like subflow competes with the newly introduced traffic.

path or on the background path. When TCP flows are introduced on the path that competes with Multi-Priority TCP's aggressive flow, Multi-Priority TCP retains more total bandwidth. For example, with two competitive TCP flows, Multi-Priority TCP retains 1/3 of the congested path and all of the empty path, for a total of 2/3 of the total bandwidth. When the TCP flows are introduced in competition with Multi-Priority TCP's background flow, they take all of the bandwidth on that path. This leads to the flat line at 50%, regardless of how many competing TCP flows are present. We intend to improve Multi-Priority TCP by dynamically changing which subflow uses traditional congestion control. This would optimize its use of available bandwidth without compromising fairness. A similar technique could improve the Hacker scheme.

Chapter 5

Conclusion

5.1 Related Work

This thesis is concerned primarily with the *fairness* aspect of multipath transport. There is a large body of work that approaches multipath transport from the perspective of lower-level routing. Chen's thesis [8] contains a good overview of these approaches, with references to other work in the area. Chen's technique for actually using the bandwidth, MPTCP, increases the size of the loss-induced back-off for secondary connections—similar in spirit, but not effect, to use of TCP-LP in this thesis.

Iyengar et al. [11] approach multihoming using SCTP (the Stream Control Transmission Protocol), and present several approaches to reduce the impact of reordering between subflows. Approaches in this thesis avoid this by having stronger subflow loss detection, and unlike this work, the SCTP multihoming techniques assume independent bottlenecks.

The congestion detection techniques used by Zhang et al. in mTCP [12] are similar to those proposed by Rubenstein et al. [23], except that Zhang's technique uses ongoing TCP packets as probes instead of explicitly injecting probe traffic. mTCP is concerned only with lenient fairness, and only operates under drop-tail queuing.

A number of researchers have examined the effects of using parallel TCP transfers, primarily in the Grid context of trying to obtain high bandwidth. Hacker et al. examined the effects of parallel connections in a lossy wide-area network [9]. They find that parallel TCP increases the throughput on an unloaded network by reducing the impact of random (non-congestive) losses. Their more recent work examines the use of less aggressive TCP connections to consume bandwidth on a single high-capacity path while attempting to remain TCP-friendly [16], as discussed in Chapter 2. While the goal of their work is to efficiently utilize high bandwidth-delay-product links, the technique used is similar to some techniques used in this thesis for multipath environments.

Snoeren used link-layer striping over heterogeneous wireless links (Wide-area Multilink PPP) [24]. The focus was on adapting to the extremely variable performance of low-bandwidth wireless links, and so was able to assume that all bottlenecks were known and independent.

The design of Multi-Priority TCP is based around TCP-LP [18], but a similar technique, TCP-Nice [19], would serve equally well. Both protocols endeavor to be unobtrusive to normal TCP traffic, and could serve equally well as background TCPs for use in an Multi-Priority TCP scheme. The choice of TCP-LP in this thesis was based solely upon the availability of its *ns-2* simulation code.

Finally, many peer-to-peer systems such as Chord [25] or BitTorrent [26] make use of concurrent transfers among multiple peers. While the mechanisms and nodes involved in these transfers are different from node-to-node multipath transfers, many of the same issues of fairness arise in the peer-to-peer context [27].

5.2 Conclusion

Multipath transfers are becoming an increasingly popular topic of research and mechanism for using extra network links. To avoid the pitfalls of prior approaches, this thesis presented two new, formal definitions of multipath TCP-fairness, and proposed two novel mechanisms, Multi-Priority congestion control (Multi-Priority TCP) and Bilevel congestion control (Bilevel TCP), that meet the most stringent of multipath TCP-fairness definitions. As demonstrated by preceding experiments, the proposed approaches achieve near-optimal usage of network resources while remaining fair to single path traditional TCP flows, while earlier approaches did not.

In particular, compared to Bilevel TCP, Multi-Priority TCP appears to be more stringently multipath TCP-fair while being able to effectively utilize spare bandwidth. The only seemingly drawback of Multi-Priority TCP is its dependence on path allocation, which affects its ability to utilize spare bandwidth but not fairness. To augment this situation in Multi-Priority TCP, it is possible that a "switching" scheme be implemented that allows switching paths between the primary flow and a low-priority subflow. Due to already available RTT and loss information in all subflows, one can easily find the path with maximum bitrate potential by the TCP response function (2.1).

While these schemes appear effective, there is considerable future work remaining. Multipath schemes should be robust to configuration parameter changes, the number of paths the protocol uses, the presence or absence of shared bottlenecks, and different queuing disciplines and traffic management techniques. While techniques presented are a step in this direction—being more resilient than previous approaches, none of the existing or new techniques we examined is perfectly robust. Simultaneously, this thesis advocates for a strong definition of multipath fairness that other researchers and developers can use to guide their implementations.

We believe it is likely that more and more applications will take advantage of multipath transfers to improve their reliability and resilience to interruptions. Our hope is that by presenting two reasonable definitions of multipath TCP-fairness, and showing that each is achievable in practice, this work will serve as a catalyst for a community discussion about the correct meaning of multipath fairness, and a springboard for future research into multipath transfer mechanisms.

Bibliography

[1] “Ns,” http://www.isi.edu/nsnam/ns/.

[2] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, “The end-to-end effects of internet path selection,” *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 289–299, 1999.

[3] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, “Resilient overlay networks,” in *Proc. of the 18th ACM Symposium on Operating Systems Principles*, Oct. 2001, pp. 131–145.

[4] B. Raghavan and A. C. Snoeren, “A system for authenticated policy-compliant routing,” in *Proc. ACM SIGCOMM Conference*, Sept. 2004, pp. 167–178.

[5] X. Yang, “NIRA: A New Internet Routing Architecture,” in *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Karlsruhe, Germany, Aug. 2003.

[6] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, “A measurement-based analysis of multihoming,” in *Proc. ACM SIGCOMM Conference*, Karlsruhe, Germany, Aug. 2003.

[7] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall, “Improving the reliability of internet paths with one-hop source routing,” in *Proc. 6nd Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004, pp. 183–198.

[8] J. Chen, “New approaches to routing for large-scale data networks,” Ph.D. dissertation, Rice University, June 1999.

[9] T. J. Hacker, B. D. Athey, and B. D. Noble, “The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network,” in *Proc. 16th IEEE-CS/ACM International Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2001.

[10] H.-Y. Hsieh and R. Sivakumar, “pTCP: An end-to-end transport layer protocol for striped connections,” in *IEEE International Conference on Network Protocols (ICNP)*, Paris, France, Nov. 2002.

[11] J. R. Iyengar, K. C. Shah, P. D. Amer, and R. Stewart, “Concurrent multipath transfer using SCTP multihoming,” in *Proc. SPECTS*, San Jose, CA, July 2004.

[12] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang, “A transport layer approach for improving end-to-end performance and robustness using redundant paths,” in *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2004, pp. 99–112.

[13] S. Floyd and K. Fall, “Promoting the use of end-to-end congestion control in the internet,” *IEEE/ACM Trans. Netw.*, vol. 7, no. 4, pp. 458–472, 1999.

[14] S. Floyd, M. Handley, J. Padhye, and J. Widmer, “Equation-based congestion control for unicast applications,” in *Proc. ACM SIGCOMM Conference*, Aug. 2000.

[15] B. B. et al, “Recommendations on queue management and congestion avoidance in the internet,” *RFC 2309*, p. 16, 1998. [Online]. Available: citeseer.ist.psu.edu/braden97recommendations.html

[16] T. J. Hacker, B. D. Noble, and B. D. Athey, “Improving throughput and maintaining fairness using parallel TCP,” in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004.

[17] J. Padhe, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP throughput: A simple model and its empirical validation,” in *SIGCOMM Symposium on Comm. Architectures and Protocols*, Aug. 1998.

[18] A. Kuzmanovic and E. W. Knightly, “Tcp-lp: A distributed algorithm for low priority data transfer,” in *Proc. IEEE INFOCOM*, San Francisco, CA, Apr. 2003.

[19] A. Venkataramani, R. Kokku, and M. Dahlin, “System support for background replication,” in *Proc. 5th USENIX OSDI*, Boston, MA, Dec. 2002.

[20] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[21] S. Landström and L.-Å. Larzon, “Rfc3448 compliant tfrc for ns-2.27,” Luleå University of Technology, Uppsala University, Tech. Rep., Oct. 2004.

[22] W. M. Eddy, S. Ostermann, and M. Allman, “New techniques for making transport protocols robust to corruption-based loss,” *Computer Communication Review*, vol. 34, no. 5, Oct. 2004.

[23] D. Rubenstein, J. Kurose, and D. Towsley, “Detecting Shared Congestion of Flows Via End-to-end Measurement,” in *Proc. ACM SIGMETRICS Conference*, June 2000.

[24] A. C. Snoeren, “Adaptive inverse multiplexing for wide-area wireless networks,” in *Proc. IEEE Conference on Global Communications (GlobeCom)*, Rio de Janiero, Brazil, Dec. 1999, pp. 1665–1672.

[25] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for Internet applications,” in *Proc. ACM SIGCOMM Conference*, Aug. 2001.

[26] B. Cohen, “Incentives build robustness in BitTorrent,” May 2003, http://bitconjurer.org/BitTorrent/bittorrentecon.pdf.

[27] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, “Designing a DHT for low latency and high throughput,” in *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, Mar. 2004.