Balancing Resource Pooling and Equipoise in Multipath Transport

Damon Wischik
University College London
d.wischik@cs.ucl.ac.uk

Costin Raiciu
University College London
c.raiciu@cs.ucl.ac.uk

Mark Handley
University College London
m.handley@cs.ucl.ac.uk

ABSTRACT

By simultaneously using multiple paths through the Internet, multipath transport protocols have the potential to greatly improve performance, resilience and flexibility. Further, by linking the congestion behavior of the subflows of a connection, it is possible to move traffic away from congested paths, allowing network capacity to be pooled and better handling surges in traffic. In this paper we show that existing algorithms that achieve such resource pooling have a few problems, which stem from the differences between fluid flow models and packet-level behavior. Further, these algorithms are poor at detecting and using capacity when congestion levels vary rapidly, as they do in the Internet.

We propose the principle of *equipoise* as a balance to resource pooling, and present a class of algorithms that achieve different degrees of resource pooling and equipoise. We show how to dynamically adapt the aggressiveness of multipath flows so they compete fairly in the current Internet.

We use a combination of real deployment and packet-level simulation to show that the emergent behavior of these algorithms is robust, the algorithms are fair to existing TCP, and achieve both equipoise and resource pooling.

1. INTRODUCTION

Multipath transport protocols have the potential to greatly improve the performance and resilience of Internet traffic flows. The basic idea is that if flows are able to simultaneously use more than one path through the network, then they will be more resilient to problems on particular paths (e.g. transient problems on a radio interface), and they will be able to pool capacity across multiple links. These multiple paths might be obtained for example by sending from multiple interfaces, or sending to different IP addresses of the same host, or by some form of explicit path control.

Although Multipath TCP has been suggested many times over the years (first in [9]), it was really only with the recent development of fluid-flow models[11][6] that the main potential benefits have become clear. In short, multipath-capable flows should be designed so that they shift their traffic from congested paths to uncongested paths, so that the Internet will be better able to accommodate localized surges in traffic and use all available capacity. In effect, multipath congestion control means that the end systems take on a role that is normally associated with routing, namely moving traffic onto paths that avoid congestion hotspots, at least to whatever extent they can given the paths they have available.

The idea is very timely. Smart phones are becoming ubiquitous and incorporate multiple radios. Experience shows that the performance of each of these can be very variable, especially while moving. Multipath transport's ability to use more than one radio simultaneously *for the same connection* has the potential to improve both reliability and performance. In addition, multi-homing of server farms has also become ubiquitous. To balance load at such multi-homed sites, network operators play traffic engineering games with BGP routing. BGP, though, is a very slow and imprecise tool for this purpose; in contrast multipath transport can react on timescales of seconds and can precisely and automatically balance load. Finally, in the core of the network multipath transport protocols can move traffic away from hotspots in the face of congestion caused by failures such as fiber cuts, or overload caused by flash crowds or DDoS attacks. In the hope of gaining these benefits the IETF has recently started a working group to standardize multipath extensions to TCP[4].

In this paper we investigate the congestion control dynamics of just such a multipath transport protocol. We start from the understanding of resource pooling provided by the previous work on fluid flow models and show that although they capture the steady-state behavior well, they miss crucial packet-level dynamics, even in steady-state scenarios. When these dynamics are combined with dynamic background load, solutions that previously appeared optimal no longer do so.

To address these issues, we will first introduce the concept of *equipoise* (or equal balance) and show that solutions exhibiting equipoise tend to be more robust when network conditions vary rapidly. Through theory and simulation we will then derive a more robust multipath congestion control algorithm and demonstrate that it works well even in dynamic network conditions with vastly differing RTTs between the paths. Finally we will demonstrate that our algorithm works well under real network conditions when used as part of a full Multipath TCP protocol stack.

2. THE STATE OF THE ART

Theoretical models for multipath congestion control were first proposed by [12], and subsequently by [20], [11] and [6]. The latter two proposals [11, equation (21)] and [6, equation (14)] are particularly interesting because they use the same mechanism as TCP: they adapt a congestion window at the sender, in response to congestion information transmitted via ACK packets.

These models suggest that it is possible for a multipath sender to split traffic between multiple paths and to control how traffic is balanced between those paths, on the same timescale as TCP congestion control. The common conclusion is that it is possible to get many of the benefits of load-dependent routing, but to do so stably on short timescales using control mechanisms located at the transport layer of the end systems.

The specific proposals from [11] and [6] are minor variations[1] of the following algorithm:

ALGORITHM: COUPLED SCALABLE

- For each ACK on path r, increase window w_r by a.
- For each loss on path r, decrease window w_r by w_{total}/b.

Here w_r is the window size on path r, and w_{total} is the total window size across all paths; a and b are constants.

Both [11] and [6] analyse a fluid-model approximation to this and related algorithms. That is, they write down a collection of differential equations describing an arbitrary network topology with a fixed number of long-lived flows, and they analyse the behaviour of the dynamical system. A suitable equation for the evolution of w_r is

$$\frac{dw_r(t)}{dt} = \frac{w_r(t)}{\mathsf{RTT}_r}\Big(\big(1-p_r(t)\big)a - p_r(t)\frac{w_{\text{total}}(t)}{b}\Big)^{+[w_r(t)=0]} \tag{1}$$

where $p_r(t)$ is the packet loss probability on path r at time t. The supserscript means if $w_r(t) = 0$ then take the positive part. The fluid model predicts[2] that simply by coupling the congestion windows, we get three important benefits: load balancing, fairness, and stability.

Load balancing. Traffic moves away from the more congested paths until either the congestion levels on the paths equalize, or no traffic remains to be moved from the more congested paths. When there are several paths with minimal congestion, the algorithm has no preference for any particular split between them.

These properties may be seen from (1). The equilibrium point, i.e. the point where $dw_r/dt = 0$ for every subflow r, has $w_r = 0$ for every path r such that $p_r > p_{\text{min}}$, where p_{min} is the minimum of the p_r. It also has total window size $w_{\text{total}} = ab(1 - p_{\text{min}})/p_{\text{min}}$.

Resource pooling. The overall effect of load-balancing is that a set of disjoint bottleneck links can behave as if they were a single pooled resource. This is known as 'resource pooling'. Figure 1 illustrates. Consider a scenario with four links traversed by three multipath flows, and suppose that because of topological constraints each flow has access to only two of the links. The capacities are $C_1 = 100$, $C_2 = 250$, $C_3 = 180$ and $C_4 = 50$ pkt/s, and the common round trip time is 100ms. The left hand diagram shows how capacity would be shared by running uncoupled TCP congestion control on each subflow (except that we have scaled the window increase parameter by $1/4$ so that each subflow is half as aggressive as a normal TCP). The right hand diagram shows how capacity would be shared by COUPLED SCALABLE: congestion is equalized at the four links, and the three flows achieve the same throughput. In effect, the network is behaving as if the four links constituted a single link shared by three single-path flows.

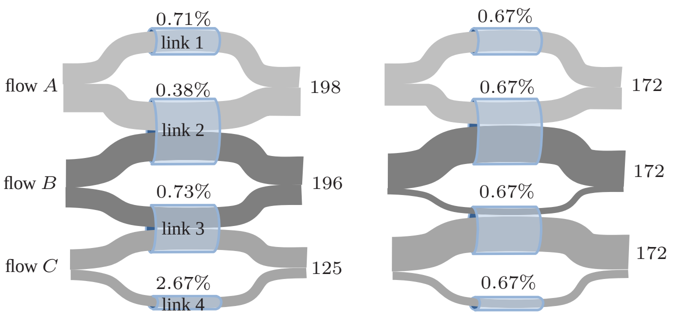


Figure 1: A scenario where regular TCPs will not balance load or share capacity fairly, but COUPLED SCALABLE will. The numbers above the links are packet drop probabilities, and the numbers to the right are aggregate throughputs in pkt/s.

For intuition about how COUPLED SCALABLE achieves this pooling of resources, suppose it starts out with throughputs as per the left hand diagram. Flow C experiences higher congestion on link 4, so it shifts some traffic into link 3. This causes congestion on link 3 to increase, so flow B shifts some traffic onto link 2, and so on.

Resource pooling does depend on the subflows taking different paths. There are a range to ways to ensure this happens, but if the endpoints are multihomed the simplest is to use multiple IP addresses at each endpoint. With reasonable probability[1], this will give paths through the network that are distinct for much of the way.

Fairness. COUPLED SCALABLE takes a fair share of resources at a bottleneck link, even if several subflows pass through that same link. Fairness means there is no need for shared bottleneck detection, as used by [16].

To see why it is fair, note from (1) that at equilibrium $w_{\text{total}} = ab(1 - p_{\text{min}})/p_{\text{min}}$, hence the total window depends only on the level of congestion and not on the number of paths or their intersections.

We assume for now that all RTTs are equal, so fair window size means fair throughput. Note that COUPLED SCALABLE is not fair against TCP NewReno because its response to congestion is intrinsically different. In Section 5 we will show how to achieve the benefits of multipath while accounting for different RTTs and maintaining 'legacy fairness'.

Stability. Fluid model analysis of (1) shows that parame-

[1]The two proposals differ slightly in how to respond when the different subflows have different RTTs. For convenience, we shall in this section only consider the case where all subflows have the same RTT. The algorithm in [11] is specified in terms of rates, but we have recast it here in terms of congestion windows. Both papers assume that qeueueing delay is negligible.

[2]Note that these are theoretical predictions; the rest of our paper asks how to obtain these benefits in practice.

ters a and b can be chosen to make the network stable, i.e. to ensure that once the equilibrium point has been reached, any deviations are damped down and do not grow into oscillations. This suggests that there will not be any route flap or synchronization between flows, since these effects would result in a deviation from equilibrium.

3. RESOURCE POOLING IN THE FACE OF FLUCTUATING CONGESTION

Our goal, when starting this work, was to take the ideas from §2 and implement them in the TCP protocol in a manner that is acceptable for standardization. We expected that any difficulties would be with the protocol embedding, not the congestion control dynamics. Although there are a range of interesting protocol questions to be answered, the more challenging issue has turned out to be the dynamics. The key challenge is that congestion levels fluctuate in ways not accounted for by the theory in §2. In order to obtain the benefits of resource pooling in the face of fluctuating congestion, we needed to make significant changes to the dynamics. There are two main sources of fluctuations:

- Packet drops are discrete random events. Even if the packet loss probability remains constant, there will from time to time be chance bursts of loss on one path or another, hence the short-timescale observed loss probability will not be constant. The fluid models however use a real number $p_r(t)$ to represent congestion, so they do not take account of the random nature of the signal.
- Typically, the background load in a network is variable. When there is a persistent change in the congestion on a path, e.g. a change that lasts longer than several RTTs, the flow should quickly adapt. The fluid model analysis however deals with a steady-state network and does not give any guidance about how fast it is safe to react.

A multipath flow ought to adapt to persistent changes in congestion by moving its traffic away from more-congested paths—but if it is hardly using those paths then it will be slow to learn and adapt if and when they decongest. We refer to this as *capture* by the less-congested paths. Furthermore, if multipath is deployed in the Internet and resource pooling actually works, then we should expect that there will often be balanced levels of persistent congestion, which means that transient fluctuations could be enough to trigger capture.

In this section we investigate capture in a practical variant of COUPLED SCALABLE. We show that capture plus transient fluctuations in congestion tend to make the algorithm flap from one path to another, and that this effect can prevent the algorithm from achieving resource pooling even when persistent congestion levels are stable. We also point out a protocol problem with timeouts that arises from capture. These problems, combined with the difficulty that capture brings in responding to fluctuations in persistent congestion, lead us to a new design principle for multipath congestion control, the principle of *equipoise*.

The Zen of resource pooling

In order for multipath congestion control to pool resources effectively, it should not try too hard to pool resources. Instead of using only the paths that currently look least-congested it should instead maintain equipoise, i.e. it should balance its traffic equally between paths to the extent necessary *to smooth out transient fluctuations in congestion and to be ready to adapt to persistent changes.*

In §4 we will examine a spectrum of algorithms with different tradeoffs between resource pooling and traffic-balancing, in order to quantify the phrase ‘to the extent necessary’.

3.1 The algorithm under test

Throughout this paper we care specifically about deployability in the current Internet, so our starting point will be a modification of COUPLED SCALABLE to make it fit better with TCP NewReno. We make two changes.[3]

Change 1. In the absence of loss, for each ACK received COUPLED SCALABLE increases w_r by a giving exponential growth in window size, whereas TCP NewReno grows the congestion window linearly. We can easily adapt the multipath algorithm to grow windows linearly: simply increase w_r by a/w_{total} per ACK on path r, rather than increasing by a. TCP NewReno also increases its window w by a/w per ACK, so this will be fair to TCP even when several subflows go through a single bottleneck link.

Change 2. In COUPLED SCALABLE the window w_r is meant to decrease by w_{total}/b per loss on path r. If $w_{\text{total}}/b > w_r$ then the decrease has to be truncated; if there are two paths and we have chosen $b = 2$ to mimic TCP NewReno, then the smaller congestion window will always be truncated to 0. To avoid problems of truncation, we will multiply the decrease term by w_r/w_{total}. We shall also multiply the increase term by the same amount; the algebra below shows that this gives us resource pooling and fairness. These two changes give us:

ALGORITHM: COUPLED
- Each ACK on path r, increase window w_r by aw_r/w_{total}^2.
- Each loss on path r, decrease window w_r by w_r/b.

In experiments in this section, we use $a = 1$ and $b = 2$ to mimic TCP.

A fluid-model approximation for this is

$$\frac{dw_r(t)}{dt} = \frac{w_r(t)}{\mathsf{RTT}_r}\Big(\big(1-p_r(t)\big)\frac{aw_r(t)}{w_{\text{total}}(t)^2} - p_r(t)\frac{w_r(t)}{b}\Big)^{+[w_r(t)=0]}$$

In equilibrium, there can be no traffic on paths r for which p_r is not minimal, for if there were then that derivative would

[3]Simulation results (not reported) show that the changes do not affect our observations about capture.

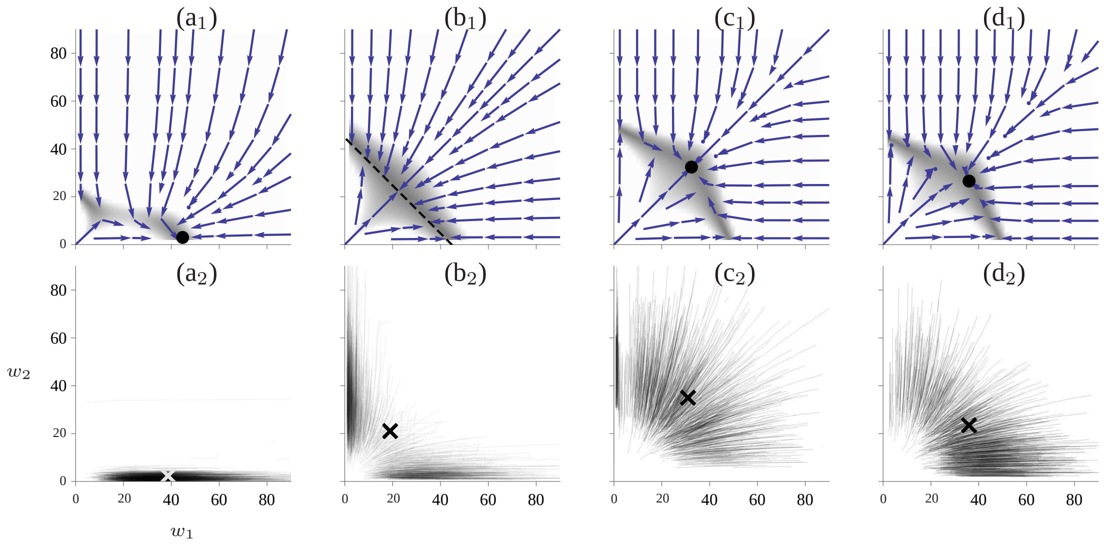


Figure 2: Window size dynamics for a two-path flow. The top row shows the fluid model predictions: the arrows show trajectories of $(w_1(t), w_2(t))$ and the darker areas indicate that w_1 and w_2 are changing slowly. The bottom row shows simulation results: there is ink at every state (w_1, w_2) occupied during the course of the simulation, so the density of the ink indicates how much time the algorithm spends in each state.

be negative. At the equilibrium value of w_{total}, the increase and decrease on active paths must balance out, hence

$$(1 - p_{\text{min}})\frac{a}{w_{\text{total}}^2} = p_{\text{min}}\frac{1}{b}$$

where p_{min} is the lowest congestion level over all paths, hence $w_{\text{total}} = \sqrt{ab(1 - p_{\text{min}})/p_{\text{min}}}$. When p_{min} is small this is approximately $\sqrt{ab/p_{\text{min}}}$, which agrees with what TCP NewReno would get on the least congested path. Note that the total window size depends only on p_{min} and not on the number of paths or their overlap, hence this algorithm allocates window size fairly. Since the increase and decrease terms are both less aggressive than COUPLED SCALABLE, for which we know that the fluid model is stable, we conjecture that this fluid model is also stable.

3.2 Flappiness

In simulations we found that a COUPLED flow very rarely uses all its paths simultaneously. It switches between paths, but not periodically: rather it is captured on one path for a random time, then it flaps to the other, and so on. Properly speaking this is not oscillation, since the flaps are not periodic, rather it is an example of bistability.

Consider first the bottom left plots in Fig. 2 labelled (a_2) and (b_2). These come from a simulation where two subflows of a multipath flow experience random loss with probability p_r. We refer to such random losses as 'exogeneous drops' because they are outside the influence of the flow itself. This plot graphs the window of one subflow, w_1, against the window of the other subflow, w_2. As the simulation proceeds, the windows increase linearly and backoff multiplicatively leaving a trace in the plot. The plot is thus a form of histogram—the simulation spends most of its time in the areas of the plot where the ink is darkest. An $\times$ marks the average window sizes.

Plots (a_2) and (b_2) differ in one respect only. In (a_2) the loss probability p_2 is greater than the loss probability p_1, whereas in plot (b_2) the drop probabilities are equal.

The top plots in Fig. 2 show the predictions of the fluid model. The arrows show how the model predicts (w_1, w_2) will evolve, given an arbitrary starting point. Where the model predicts a unique equilibrium exists, a black dot is shown.

The fluid model shown in (a_1) and the simulation in (a_2) agree: all the traffic shifts to path 1. In Fig. 2(b) they disagree: the fluid model says that any point with $w_1 + w_2 = \sqrt{2/p_1}$ is an equilibrium point, but the simulation shows that the flow avoids those equilibria where $w_1 \approx w_2$.

What causes flappiness? There are two related causes of flappiness. (1) The algorithm has a capture effect. If w_1 happens to be larger than w_2 at some point in time, then it takes several drops on path 1 to counter the effect of a single drop on path 2. This means the flow spends some time captured on path 1. Another way to express this is that when w_2 is small the flow does not probe path 2 frequently, and it does not attempt to increase aggressively. (2) Random fluctuations in congestion mean that over short timescales the losses seen by each subflow are never precisely equal. COUPLED mistakes this for a persistent difference in congestions, so it load-balances its traffic onto the less congested path.

To see these effects more clearly, consider a toy model: suppose that the two subflows of a COUPLED flow go through the same bottleneck link, and suppose that when $w_1 + w_2 = 100$ a packet is dropped, and that the probability it is dropped from subflow r is proportional to w_r. Fig. 3(a) shows the evolution of w_1 and w_2 over the first 4 drops, first two drops on path 2 then 2 drops on path 1, and it shows clearly the

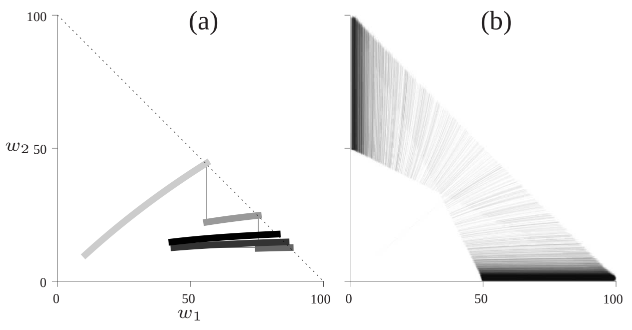


Figure 3: Window size dynamics for a two-path flow.

capture effect. Fig. 3(b) shows just the increases over 5000 drops, with the densities of the lines indicating the fraction of time spent with a given combination of window sizes, and it shows that the overall outcome is flappiness.

Capture is a robust finding. It might be thought that capture will not arise in practice, because if the flow flaps onto a path then the congestion on that path will increase (we call this 'endogenous drops'), and this will be enough to push the flow away.

Fig. 2(c) shows this is only partly true. Consider a scenario with two paths with bandwidth-delay-product of 96 packets, a COUPLED flow using both links, and an additional single-path TCP flow on each of the links. Unlike with exogenous drops, the endogenous fluid model[4] does show a unique equilibrium point, to which the drift arrows show convergence.

Despite this, the simulation results show that the flow does not converge to equilibrium, though the capture effect is not as pronounced as with the exogeneous drops in Fig. 2(b).

We expect even more flappiness when the multipath flow in question is competing with many flows, since more competition means that p_r is less sensitive to w_r, which brings us closer to the exogenous drops case.

Capture can prevent resource pooling. There are some situations where a COUPLED multipath flow can get captured on one path but not on the other (unlike the example in Fig. 2(b), where the multipath flow spends an equal amount of time captured at each extreme). It means that the flow spends a disproportionate amount of time on the path on which it has been captured, which means that that path has excessively high average congestion, which means that resource pooling has failed.

Fig. 2(d) illustrates the problem. The scenario is like Fig. 2(c): two links, one multipath flow and two single-path flows. The bandwidth-delay-product of link 1 is 98 packets and that of link 2 is 89 packets. The fluid model predicts an equilibrium point which is just off-center, at which congestion is balanced. However, the simulation results show that the algorithm spends much of its time captured by link 1: the average of w_2 is 13% lower than the fluid model predicts, and the consequence is that $p_1 = 0.055\%$ while $p_2 = 0.047\%$.

[4]The fluid model comes from the following approximation. Let x_r be the rate of the single-path flow on link r, and solve two extra equations: the TCP throughput equation for the single-path flow, $x_r = \sqrt{2}/\mathsf{RTT}_r\sqrt{p_r}$; and an equation that says the link is kept fully utilized, $(1-p_r)(x_r + w_r/\mathsf{RTT}_r) = C_r$ where C_r is the link speed. This gives a solution for p_r as a function of w_r.

3.3 Timeouts with small windows

For the most part, the precise extensions to the TCP protocol to support multipath do not greatly impact the dynamics. However, one effect cannot be ignored: when a subflow suffers a retransmit timeout due to the ACK clock being lost, other subflows that were performing well may stall. This is because the receive buffer needed to put data from the different subflows back in order can fill up if data remains missing for an entire retransmit timeout (RTO) period. There are a number of ways to mitigate this; selective acknowledgments help a little, and adaptively reducing the dup-ack threshold in the absense of reordering [21] can reduce timeouts. However, despite these, the main problem is simply that with very small windows there are not enough packets in flight to trigger a fast retransmission, meaning that a timeout is inevitable. As a result, algorithms that reduce a subflow to a very small window tend to much more suscepible to timeouts, and these timeouts risk stalling the other subflows and degrading performance overall.

4. UNDERSTANDING THE DESIGN SPACE

According to §2 a multipath flow should shift its traffic onto the least-congested paths in order to achieve resource pooling, but according to §3 it ought to maintain equipoise over its available paths. In order to explore the tension between equipoise and resource pooling, we shall consider the following family of algorithms parameterized by ϕ:

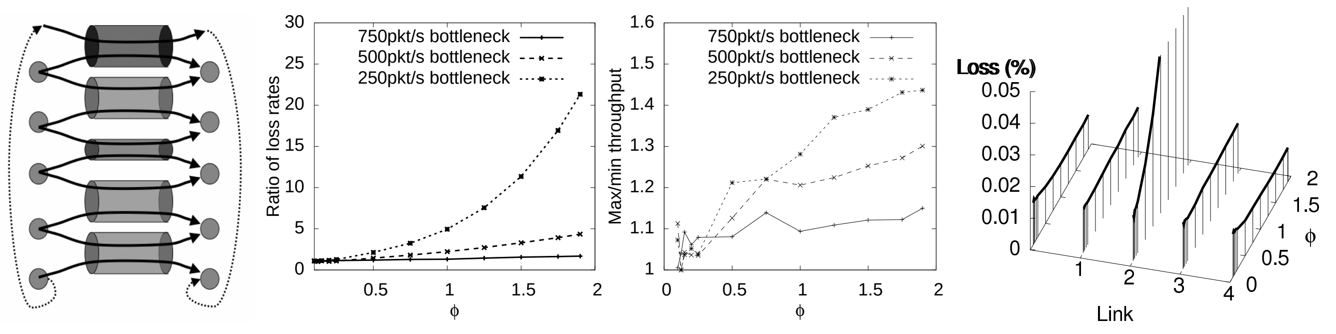
ALGORITHM FAMILY: SEMICOUPLED(ϕ)
- Each ACK on subflow r, increase the window w_r by $a^{2-\phi} w_r^{1-\phi} / w_{\text{total}}^{2-\phi}$.
- Each loss on subflow r, decrease the window w_r by w_r/b.

Use $b = 2$ to mimic TCP. The a parameter controls aggressiveness, and is discussed in §5.

With $\phi = 0$, SEMICOUPLED reduces to COUPLED [5], and as we saw in §3 this achieves resource pooling but does not maintain equipoise. It seems to be too sensitive to minor or short-term changes in loss rates, and it moves too much traffic off the more congested path.

With $\phi = 2$, SEMICOUPLED has an increase term of $1/w_r$ and a decrease term of $w_r/2$, which corresponds to running separate TCP congestion control on each subflow. We shall refer to this as UNCOUPLED congestion control. Since the subflows are controlled separately, there is no tendency for traffic to flap from one subflow to another, i.e. this algorithm maintains equipoise. However it does a poor job of balancing load and resource pooling, as seen in Fig. 1.

[5]although a in SEMICOUPLED corresponds to $\sqrt{a}$ in COUPLED



(a) Torus topology (b) Loss rates ratio for differing values of ϕ (c) Throughput ratio for differing values of ϕ (d) Loss rates on each link for differing values of ϕ (bottleneck=250pkt/s)

Figure 4: Impact of ϕ on long term Resource Pooling

We shall explore whether it is possible to achieve both resource pooling and equipoise, by studying what happens when we vary ϕ in the range $[0, 2]$. We do not claim that by carefully choosing ϕ we can obtain a perfect algorithm, merely that this is an interesting spectrum with which to begin an exploration.

The case $\phi = 1$ is *a priori* appealing since the increase and decrease terms can be computed using basic arithmetic. The equilibrium window size can be computed by calculating when increase and decrease are balanced, as in §3.1; when $\phi = 1$ and $1 - p_r \approx 1$ the equilibrium window sizes are

$$w_r \approx \sqrt{ab}\frac{1/p_r}{\sqrt{\sum_s 1/p_s}}. \qquad (2)$$

4.1 Evaluating the SEMICOUPLED(ϕ) Family

The ϕ-parameterized family of algorithms allows us to study the balance between resource pooling and equipoise. The fluid-flow models cannot capture this, so instead we use packet-level simulation. We developed our own high-performance event-based simulator called *htsim* that can scale from single flows to many thousands of flows and Gb/s link-speeds, allowing us to examine a wide variety of scenarios. *htsim* models TCP NewReno very similarly to *ns2*. All the simulations in this paper were run with *htsim*.

Resource Pooling in the Steady State

First we shall investigate how well the different algorithms in the SEMICOUPLED(ϕ) family balance load and pool resources in a stable environment of long-lived flows. We have examined many topologies and scenarios, but Fig. 4(a) shows a torus topology that illustrates the effects particularly nicely. It consists of five bottleneck links, each traversed by two multipath flows. This topology is a good test of resource pooling because it demonstrates the knock-on nature of load balancing while at the same time having no end points or special cases which complicate the choice of metric.

To start with, all five bottleneck links have equal capacities of 1000 pkts/second, equal RTTs of 100ms, and the bottleneck buffers are one bandwidth-delay product. To see the effectiveness of resource pooling we progressively reduce the capacity of the middle link to 750, 500 and 250 packets/s. We wish to see the extent to which multipath TCP can move traffic away from the middle link towards the top and bottom links. The best metric here is the ratio of loss rates between the middle link and the top link—an algorithm that pools resources completely will equalize these loss rates, giving a ratio of one.

We also examine the aggregate throughputs of each of the multipath flows. Although this is a less effective metric (each flow traverses two links, so congestion is conflated), it serves to verify that the overall outcome is acceptable for the users of these flows, not just for the network operator.

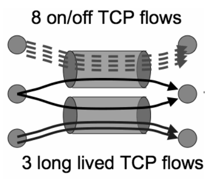
Fig. 4(b) shows the ratios of loss rates and Fig. 4(c) shows the ratio of best-to-worst throughput. Each data point is from a single long run (10,000 simulated seconds). As ϕ decreases we see that resource pooling (as exhibited by the loss rate ratio) improves steadily and approaches perfect resource pooling as ϕ approaches zero. The ratio of throughputs also decreases steadily, but the graph gets a little noisy as ϕ approaches zero due to increased flappiness.

Fig. 4(d) shows the absolute loss rates on all the links for different values of ϕ. The z-axis is truncated to emphasize the effect as ϕ approaches zero (losses on the middle flow actually extend linearly to 0.25% loss when $\phi = 2$). The figure clearly shows the way resource pooling diffuses congestion across the network as ϕ approaches zero.

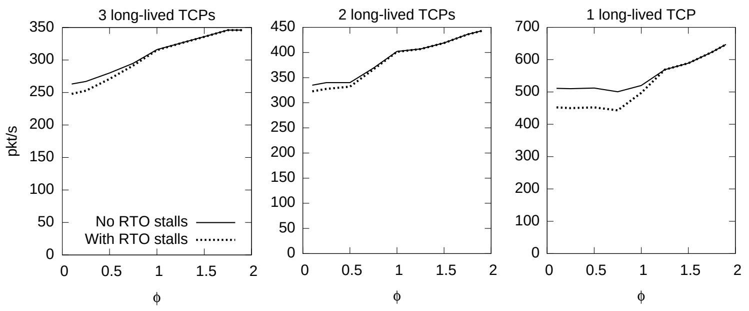
Dynamic Background Load and Equipoise.

In stable steady-state scenarios it is clear that the COUPLED (i.e, $\phi = 0$) algorithm achieves close to optimal resource pooling, albeit with the possibility of flappiness. However, the Internet rarely resembles such a steady-state scenario—usually there is a constant background of short flows that come and go. This background traffic can change the situation considerably.

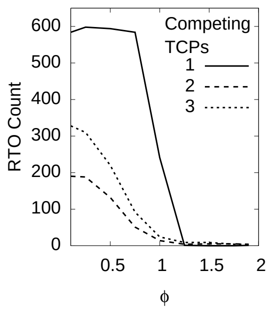
Consider a scenario where the background traffic is constantly changing but where it is still feasible for multipath flows to balance load. If the multipath flows succeeded in balancing the load on average, then for short periods of time first one path then the other path will be more congested. This resembles the flappiness issue we described earlier, ex-



(a) Topology



(b) Effect of ϕ on throughput



(c) Retransmit Timeouts

Figure 5: Impact of ϕ on throughput with dynamic background load

cept that the differences in the loss rates between the paths may be larger, and they may reverse faster as flows slow-start or terminate. The capture effect exhibited by the COUPLED algorithm can be a problem here. If one path is more congested for a short while, then a COUPLED flow will move as much traffic as it can off that path. However, if the congestion suddenly reduces, as happens when competing flows terminate, then COUPLED will not respond rapidly to rebalance the network because it has too small a window and a very low increase rate on the previously congested path. Thus, although $\phi = 0$ is good at steady-state load balancing, its lack of equipoise makes it less effective at dynamic load balancing.

To illustrate this, consider the very simple scenario in Fig. 5(a). Three long lived flows provide background traffic on the bottom path. On the top path eight short-lived flows with idle and active periods with a mean of 5 seconds provide a rapidly changing background of slow-starting and terminating flows. These particular parameters were chosen to ensure both links are fully utilized for all values of ϕ, and to provide roughly equal congestion levels on the two paths.

Directly measuring short-term resource pooling by measuring loss rates is difficult because we need to measure short time periods and each period contains few losses. We can, however, measure the opportunity cost that the multipath flow pays when it becomes captured on a path that is no longer the best. This is shown in Fig. 5(b). The left of the three graphs shows the scenario from Fig. 5(a), whereas the middle and right graphs show the same scenario with less background traffic (2 and 1 long-lived TCP respectively on the bottom link, plus an appropriate level of short flows on the top link).

The reduction in throughput as $\phi \to 0$ is due to this lack of equipoise and the resulting inability to rapidly change from prefering one path to prefering the other. For each plot, two curves are shown. The solid line is the raw results from *htsim*. However these do not model when the receive buffer fills up at the receiver during a retransmit timeout on one path, leading to the other path being stalled. Fig. 5(c) shows the number of RTOs for each of the three scenarios. Clearly the lack of equipoise with smaller values of ϕ is also increasing the probability of timeouts. The dashed lines in Fig. 5(b) show the effect on throughput if the receiver buffer is small enough that each of these timeouts also causes the better path to stall. A real implementation would lie between the two curves.

With three long-lived TCP flows on the bottom link (left plot) there is always a short-lived flow on the top link able to use the spare capacity the multipath flow fails up take up, so the link utilization is always 100%. With one long-lived TCP on the bottom link (right plot), when the scenario is evenly balanced there are too few small flows on the top path to always take up the bandwidth that the multipath flow fails to utilize. In fact in the right hand plot, with $\phi = 1.9$ the utilization is only 68%; this falls to 47% for $\phi = 0.1$. Thus with low levels of highly variable competing traffic, low values of ϕ not only reduce throughput, but also can be less effective at utilizing the network. This can be regarded as a failure to effectively pool resources.

This is even more clear if we examine why the curves levels off in the middle and right plots for low values of ϕ. The reason is that these flows are sufficiently captured by the large window they obtain on the lower path that they almost never increase their window on the top flow above one packet, even when there is no traffic on that link at all for many seconds. Thus these curves level off at their fair share of the lower path - 333 pkts/s in the middle plot and 500 pkts/s in the right plot.

Our conclusion is that while both the fluid flow models and steady-state simulations point towards $\phi = 0$ being optimal for resource pooling, this does not apply to the dynamic conditions seen in most real networks. Under such circumstances it is better to use an algorithm with $\phi > 0$. We have examined a wide range of scenarios, both in simulation and with a full implementation. Unfortunately there is no single clear “sweet spot”. However, values in the middle of the range give robust behavior, pooling capacity quite effectively across a wide range of steady and dynamic conditions.

Of these robust values, $\phi = 1$ stands out because it corresponds to a simple algorithm that can be implemented easily in an operating system kernel without needing floating point arithmetic or expensive calculations. As a result, we will

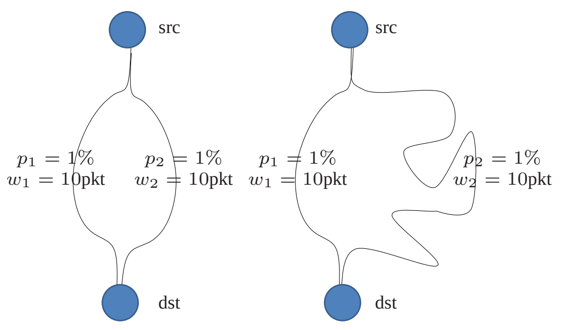


Figure 6: SEMICOUPLED$(\phi = 1)$ allocates equal window sizes when congestion is equal, regardless of RTT.

use the SEMICOUPLED$(\phi = 1)$ algorithm as the basis for the remainder of this paper.

5. FAIRNESS AND DEPLOYABILITY

So far we have examined a spectrum of algorithms with the aim of understanding how well they balance load and pool resources, and concluded that algorithms that exhibit better equipoise are more robust (in particular to timeouts) and better able to cope with dynamic operating conditions. However, we have not examined whether these algorithms are fair to competing traffic, or even whether they perform better than a single-path TCP. There are two effects to consider:

- Fairness when several subflows of a multipath flow compete with a single TCP flow at a shared bottleneck. For example, it is easy to see that two uncoupled subflows using TCP-like AIMD parameters would get twice the throughput of a single TCP flow.
- Fairness when the RTTs seen by the subflows differ significantly. For example, COUPLED always prefers a less congested path, even when that path has high RTT and hence low throughput.

For an example of why the RTT can be a problem, consider the two-path SEMICOUPLED$(\phi = 1)$ flow shown in Fig. 6 where both paths have the same packet drop probability $p = 1\%$. Use the constants $a = 1$ and $b = 2$. Equation (2) says that the equilibrium window sizes are $w_1 = w_2 = 1/\sqrt{p} = 10$ packets, regardless of RTT. The total throughput the flow gets does depend on RTT: it is $x = w_1/\mathsf{RTT}_1 + w_2/\mathsf{RTT}_2$. For example,

$$\begin{array}{ll} \mathsf{RTT}_1 = 10\text{ms}, \mathsf{RTT}_2 = 10\text{ms} & \text{gives } x = 2000 \text{ packets/s} \\ \mathsf{RTT}_1 = 10\text{ms}, \mathsf{RTT}_2 = 100\text{ms} & \text{gives } x = 1100 \text{ packets/s.} \end{array}$$

But a single-path TCP using only the low-RTT path would get throughput of 2000 packets/s, so there is no incentive to run SEMICOUPLED$(\phi = 1)$ using the higher-RTT path.

Although the two issues above have separate causes, they are part of the same fairness problem, and we will address them together. Before we can do so though, we must decide what our fairness goals are.

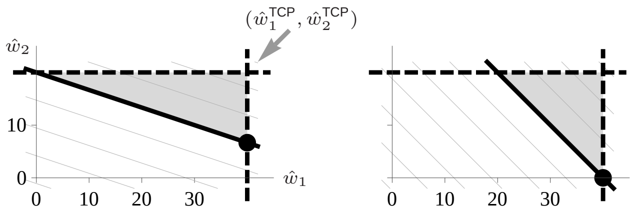


Figure 7: The three design goals place constraints on what the equilibrium window sizes should be.

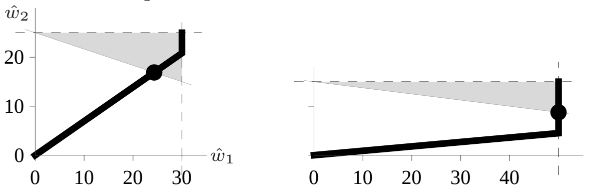


Figure 8: By choosing a we control the equilibrium window sizes.

5.1 Fairness Goals

Our overall aim is to compete with TCP in a way that does not starve competing traffic, but that still gives sufficient performance advantage to provide incentives for deployment. This leads to three specific goals that any multipath congestion control algorithm should aim to satisfy:

Goal 1 (Improve throughput) *A multipath flow should perform at least as well as a single-path flow would on the best of the paths available to it. This ensures that there is an incentive for deploying multipath.*

Goal 2 (Do no harm) *A multipath flow should not take up any more capacity on any one of its paths than if it was a single path flow using only that route. This guarantees that it will not unduly harm other flows.*

Goal 3 (Balance congestion) *A multipath flow should move as much traffic as possible off its most-congested paths, subject to meeting the first two goals.*

To understand how these goals interact, consider a two-path congestion control with equilibrium[6] window sizes $\hat{w}_1$ and $\hat{w}_2$ on its two paths. Fig. 7 shows constraints on $\hat{w}_1$ and $\hat{w}_2$ given some arbitrary combination of loss rates and RTTs. The vertical dashed lines show the equilibrium window sizes $\hat{w}_1^{\mathsf{TCP}} = \sqrt{2/p_1}$ that a regular TCP flow would achieve on path 1, and the horizontal line shows the same for path 2. (From the figure we can deduce that $p_1 < p_2$ since we see that $\hat{w}_1^{\mathsf{TCP}} > \hat{w}_2^{\mathsf{TCP}}$.) Goal 2 (do no harm) requires that the multipath flow should not use more capacity on path r than would a single-path TCP flow, i.e. that $\hat{w}_r \le \hat{w}_r^{\mathsf{TCP}}$ for every path r.

The total throughput of the multipath flow is $\sum_r \hat{w}_r/\mathsf{RTT}_r$. The diagonal lines in Fig. 7 are lines of equal throughput; their gradient is given by the ratio of the RTTs. The left figure shows a case where $\mathsf{RTT}_1 > \mathsf{RTT}_2$, and so even though

[6]Our method for compensation only works when the equilibrium window sizes are unique. It does not apply to algorithms like COUPLED.

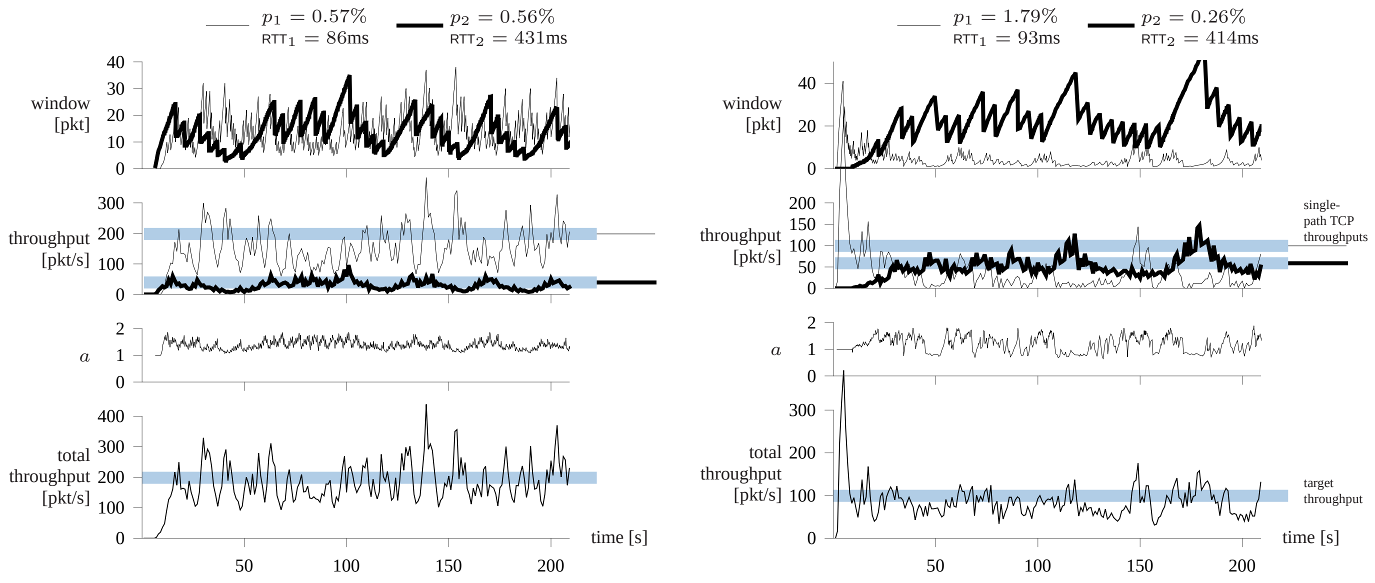


Figure 9: RTT compensation, with equal and unequal loss rates

$\hat{w}_2^{\text{TCP}}$ is smaller, a TCP on path 2 would achieve higher throughput than one on path 1. In the right figure the RTTs are equal, so path 1 gives better throughput. The solid diagonal line is the line of equal throughput equivalent to that of the better of the single path TCP flows. The region below this line and below the dashed lines satisfies Goal 2 (do no harm)—points in this region achieve no more throughput on either path than TCP would on that path, even if the bottleneck turns out to be common to both subflows. The region above the solid diagonal line satisfies Goal 1 (improve throughput), because points in this region achieve more throughput than TCP would on the better of the two paths.

Thus points on the solid diagonal line and inside the dashed lines satisfy both Goal 1 and Goal 2. Any congestion control algorithm with its equilibrium on this line is acceptable. The total throughput at any point on this line is

$$\sum_r \frac{\hat{w}_r}{\text{RTT}_r} = \max_r \frac{\hat{w}_r^{\text{TCP}}}{\text{RTT}_r} \tag{3}$$

where $\max_r$ denotes the maximum over all paths r that the flow can use. Of these points, the specific solution that best satisfies Goal 3 is shown by the dot, as this puts the least traffic on path 1 which has the higher drop probability.

For the SEMICOUPLED$(\phi = 1)$ algorithm, the equilibrium point is at $w_r \propto 1/p_r$ (from Equation 2), where the constant of proportionality depends on a. Hence by changing a we can get different equilibrium points on a radial line, shown in Fig. 8. The algorithm will satisfy all three goals if we choose a so that it lies on the bottom edge of the shaded triangle. In the left figure this can be achieved purely by scaling a, but in the right figure we must also cap the increases on path 1 so as to still satisfy Goal 2 on that path.

This leads us to a final algorithm, a version of SEMICOUPLED$(\phi = 1)$ with RTT compensation.

ALGORITHM: RTT COMPENSATOR

- Each ACK on path r, increase w_r by $\min(a/w_{\text{total}},\ 1/w_r)$.
- Each loss on path r, decrease w_r by $w_r/2$.

The parameter a controls how aggressively to increase window size, hence it controls the overall throughput. To see the effect of a, observe that in equilibrium the increases and decreases on each subflow should balance out. Neglecting for the moment the cap $1/w_r$, balance implies that

$$(1 - p_r)\frac{a}{\hat{w}_{tot}} = p_r \frac{\hat{w}_r}{2}$$

where $\hat{w}_{\text{tot}}$ is total equilibrium window summed over all paths. Making the approximation that p_r is small enough that $1 - p_r \approx 1$, we find $\hat{w}_r = 2a/(p_r \hat{w}_{\text{tot}})$. We could equivalently express this in terms of $\hat{w}_r^{\text{TCP}}$ rather than p_r, giving $\hat{w}_r = (\hat{w}_r^{\text{TCP}})^2 a/\hat{w}_{\text{tot}}$.

When we also take into account the window cap (as in the right side of Fig. 8), we see that the equilibrium window sizes must satisfy

$$\hat{w}_r = \min\Big\{\frac{(\hat{w}_r^{\text{TCP}})^2 a}{\hat{w}_{\text{tot}}}, \hat{w}_r^{\text{TCP}}\Big\}. \tag{4}$$

By simultaneously solving (3) & (4), we find after some algebra that

$$a = \hat{w}_{\text{tot}} \frac{\max_r \hat{w}_r/\text{RTT}_r^2}{(\sum_r \hat{w}_r/\text{RTT}_r)^2}.$$

This formula for a obviously requires that we measure the round trip times. It also involves $\hat{w}_r$ and $\hat{w}_{tot}$, the equilibrium window sizes. In practice, experiments show these can be approximated by the instantaneous window sizes with no adverse effect. We chose to compute a only when congestion windows grow to accommodate one more packet, rather than every ACK on every subflow. We used a smoothed round trip time estimate, computed similarly to TCP.

5.2 Evaluation: high statistical multiplexing

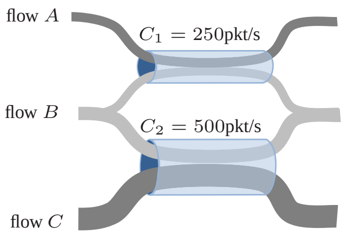


Figure 10: 'Fair' resource allocation in a two-link scenario with dissimilar link rates.

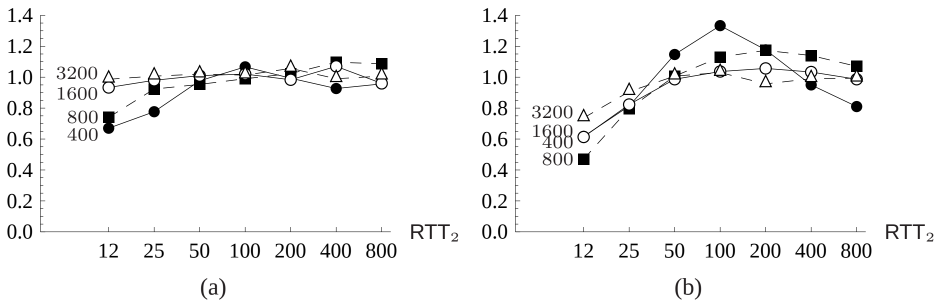


Figure 11: The ratio of flow B's throughput to the better of flow A and C, from figure 10, as we vary C_2 (shown as line labels) and RTT_2. The top plot shows the RTT COMPENSATOR algorithm, the bottom plot shows it with a fixed at 1.

We simulated a simple topology with two bottleneck links, each carrying 1000 single-path TCP flows, and we added a multipath flow able to use either or both of the links. There are so many other flows that the multipath flow's actions have hardly any effect on the loss rates, so these simulations are effectively testing RTT COMPENSATOR under constant levels of congestion. We describe here two runs of this topology, corresponding to the two generic cases shown in figure 8. The propagation delays for both runs are 50ms and 250ms.

The left hand plot in Fig. 9 shows the first run. We let the link speeds be $C_1 = 200000$pkt/s and $C_2 = 40000$pkt/s, chosen so to achieve equal loss rates ($p_1 \approx p_2 \approx 0.57\%$). Taking account of queueing delay the observed round trip times are $\mathsf{RTT}_1 = 86$ms and $\mathsf{RTT}_2 = 431$ms. The top panel shows the window sizes on each path; since the drop probabilities are roughly equal the multipath flow gets a roughly equal window on each path, in accordance with Goal 3 and the principle of equipoise. The second panel shows the throughput that the multipath flow gets on each path, and the horizontal lines show the average throughput that the single-path TCP flows are getting; we see that the multipath flow is not taking more than this on either path, hence Goal 2 is satisfied. The bottom panel shows total throughput for the multipath flow, and the horizontal line shows the larger of the two single-path TCP throughputs; we see that the multipath flow takes roughly this much in total, hence Goal 1 is satisfied. We also see that a varies a little: when by chance $w_2 > w_1$ then w_2 gets most of the window increase and the total throughput suffers so a is made larger to compensate; when by chance $w_2 < w_1$ then a is made smaller.

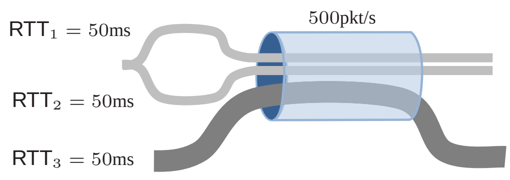
The right hand plot in Fig. 9 shows the second run. We let the link speeds be $C_1 = 100000$pkt/s and $C_2 = 60000$pkt/s, chosen so as to achieve $p_1 \approx 1.79\%$ and $p_2 \approx 0.26\%$. The observed round trip times are $\mathsf{RTT}_1 = 93$ms and $\mathsf{RTT}_2 = 414$ms. Because $p_1 > p_2$, resource pooling indicates that the multipath flow should mainly use path 2. But because path 1 has a smaller RTT, a single-path flow gets better throughput on path 1 than on path 2. In order to meet Goal 1 the multipath flow sends as much as it can on path 2 without falling foul of Goal 2, and then it takes some further capacity from path 1. In this way the multipath flow can do as well as the best single-path flow, while still achieving some degree of resource pooling. Since the multipath flow is hitting its limit on path 2, we can deduce that the window increase on path 2 is persistently hitting its cap.

5.3 Evaluation: low statistical multiplexing

We also simulated scenarios with low statistical multiplexing. When the multipath flow shifts its traffic, then the drop probabilities change and so the throughput of the hypothetical single-path TCP flows in our three fairness goals will also change.

To get an idea of what happens, consider the topology shown in Fig. 10. The obvious resource pooling outcome would be for each flow to get throughput of 250 pkt/s. The simulation outcome is very different: flow A gets 130pkt/s, flow B gets 305pkt/s and flow C gets 315pkt/s; the drop probabilities are $p_1 = 0.22\%$ and $p_2 = 0.28\%$. (Here the propagation delays are $\mathsf{RTT}_1 = 500$ms and $\mathsf{RTT}_2 = 50$ms.) After some thought we realize that the outcome is very nearly what the fairness goals dictate: the multipath flow aims to satisfy Goal 1, but the comparison it is making in that goal is to 'what a single-path TCP flow would achieve when $p_2 = 0.28\%$' rather than to 'what it would actually get if it used only link 2'. The issue is that the multipath flow does not take account of how its actions would affect the drop probabilities when it calculates its rate. It is difficult to see any practical alternative. Nonetheless, the outcome in this case is still better for flows A and B than if flow B used only link 1, and it is better for flows B and C than if flow B used only link 2.

Parameter exploration. We now repeat the same experiment, but with $C_1 = 400$pkt/s, $\mathsf{RTT}_1 = 100$ms, and a range of values of C_2 and RTT_2. The top plot in Fig. 11 shows how well flow B achieves its goal of doing at least as well as the better of flow A and C. It is within a few percent in all cases except where the bandwidth delay product on link 2 is very small; in such cases there are problems due to timeouts. We also experimented with RTT and fairness compensation turned off (by setting $a = 1$): then flow B gets as little as



$\mathsf{RTT}_1 = 50\text{ms}$
500pkt/s
$\mathsf{RTT}_2 = 50\text{ms}$
$\mathsf{RTT}_3 = 50\text{ms}$

Figure 12: Testing the fairness of multipath TCP at a shared bottleneck

47% and as much as 133% of the better of flows A and C.

Over all of these scenarios, flow B always gets better throughput by using multipath than if it used just the better of the two links; the average improvement is 15%. The multipath flow also gets more throughput than if it used only its best path.

Fairness at a shared bottleneck. We also tested the fairness of multipath TCP in the topology shown in Fig. 12. With the parameters shown, the multipath flow gets 241pkt/s and the single path flow gets 259pkt/s. The multipath algorithm achieves fairness by tuning a, rather than by any sort of shared bottleneck detection. In this case the average value of a is 0.57; there is some short-term variability and sampling bias due to fluctuations in measured RTT.

We then repeated the experiment but with $\mathsf{RTT}_1 = 250$ms. The multipath flow gets 245pkt/s and the single path flow gets 255pkt/s. The two multipath subflows still see the same packet drop probability, so they get the same window size, but the algorithm has increased a to an average value of 0.88 to compensate for the fact that path 1 has a bigger RTT.

6. OVERALL EVALUATION

From the fluid-flow models we learnt how the equilibrium throughput of an algorithm relates to the loss rates and RTTs on the different paths. Packet-level simulation, examining how resource pooling is affected by fluctuating congestion, led us to the SEMICOUPLED($\phi = 1$) algorithm. We then used a fluid model to calculate how to set the gain parameter a in order to be fair and to compensate for differing RTTs. Simulation showed that this works well. To gain greater confidence, through, we must move beyond simulation and implement the algorithms in a complete multipath TCP stack.

The main differences between our implementation and our simulator are that the implementation implements a full reorder buffer at the receiver and it uses integer arithmetic for the computation of the congestion window as floating point instructions are not permitted in the Linux kernel. We compute a using the current window size (compensating for artificial window inflation during fast-retransmit) and use TCP's existing smoothed RTT estimator.

We cross-validated the implementation against the simulator using dummynet to generate random loss (exogenous drops); the results agree for a broad range of parameters, though they start to diverge somewhat for loss rates above 5% when timeout behavior starts to dominate. We also ran experiments in which packet drops are caused by queue overflow (endogenous drops), validating the simulation results on RTT fairness, resource pooling and equipoise.

It is more interesting to look at likely deployment scenarios and examine how the full multipath TCP performs. We will examine two here: a device wirelessly connected via WiFi and 3G simultaneously, and a server multihomed via two different network links. These cases are very different. The wireless case is mostly about robustness and verifying that the algorithms work well, even in the face of very variable wireless links with vastly different characteristics and rapidly changing congestion. Under such circumstances we care less about balancing the load, and more about getting good reliable throughput. In contrast, the server case is primarily about the effectiveness of load balancing.

6.1 Multipath Wireless Client

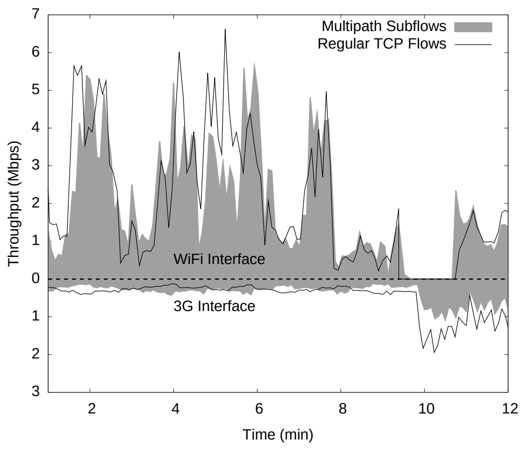
Modern mobile phones and devices such as Apple's iPad often have multiple wireless interfaces such as WiFi and 3G, yet only one of them is used at any given time. With more and more applications requiring Internet access, from mail to video streaming, multipath can improve mobile users' experience by allowing the simultaneous use of both interfaces. This shields the user from the inherently variable connectivity available over wireless networks.

We ran tests using a laptop equipped with a 3G USB interface and a 802.11 network adapter, both from work and at home. The tests we present were run in a university building that provides reasonable WiFi coverage on most floors but not on the staircases. 3G coverage is acceptable, but is sometimes heavily congested by other users.

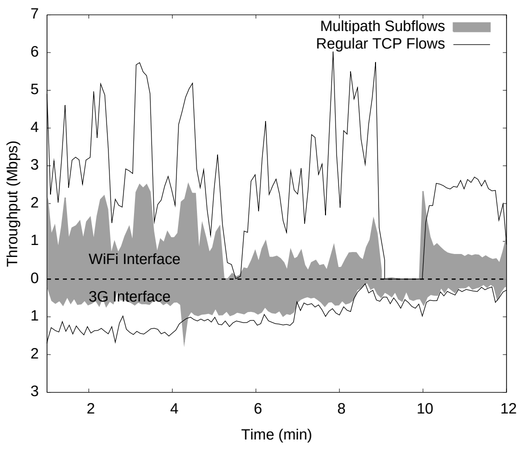
3G and WiFi have quite different link characteristics. WiFi provides much higher throughput and short RTTs, but we observe its performance to be very variable with quite high loss rates as we see a lot of interference in the 2.4GHz band. 3G tends to vary on longer timescales and we found that it is overbuffered leading to RTTs of well over a second. This provides a good real-world test of the adaptation of a to cope with differing RTTs.

The experiment starts with one TCP running over the 3G interface and one over WiFi, both downloading data from an otherwise idle university server that implements the multipath extensions. A multipath flow then starts, using both 3G and WiFi interfaces, downloading data from the same server. Fig. 13(a) shows the data rates received over each link (each point is a ten-second average). WiFi is shown above the dashed line, 3G is shown inverted below the dashed line, and the total throughput of the multipath flow can be seen clearly from the vertical range of the grey region.

During the experiment the subject moves around the building. Both WiFi and 3G are used by the multipath connection during the experiment, and it is easy to see that the overall throughput correctly matches quite closely that of the TCP flow on the faster WiFi link up until minute 9. Due to the large RTT and low loss on the 3G flow, a coupled algo-



(a) RTT Compensation



(b) RTT Compensation off, a=1

Figure 13: Throughput of multipath and regular TCP running simultaneously over 3G and WiFi. The 3G graph is shown inverted, so the total multipath throughput (the grey area) can be seen clearly.

rithm would by default prefer to send that way. To achieve good throughput, the RTT COMPENSATOR algorithm has increased a, and then has to cap the increases on the 3G subflow to avoid being unfair.

At 9 minutes the subject walks down the stairs to go to the coffee machine on a different floor—there is no WiFi coverage on the stairs, but the 3G coverage is better there so the connection adapts and takes advantage. When the subject leaves the stairwell, a new WiFi basestation is acquired, and multipath quickly takes advantage of it.

This single trace shows the robustness advantages of multipath, and it also shows that it does a good job of utilizing very different link technologies simultaneously without harming competing traffic on those links.

From this trace it is difficult to see the importance of RTT compensation. To show this, we re-ran the same experiment, taking the same path around the building and down the stairs to the coffee machine, but we switched off RTT compensation by setting $a = 1$, thus reverting to the basic SEMI-COUPLED($\phi = 1$) algorithm. Fig. 13(b) shows the results. The overbuffered 3G link is preferred, and a large window is maintained over 3G. The coupling then causes the WiFi path to be much less aggressive, and so the multipath flow receives much less throughput that it should. This illustrates quite clearly the necessity of RTT compensation.

6.2 Server Load Balancing

Multihoming of important Internet servers has become ubiquitous over the last decade; no company reliant on network access for their business can afford to be dependent on a single upstream network. However, balancing traffic across these links is difficult, as evidenced by the hoops operators jump though using BGP techniques such as prefix splitting and AS prepending. Such techniques are coarse-grain, very slow, and stress the global routing system.

Multipath transport can balance load, but if it requires all flows to be upgraded to do so, then this would be less useful. Can multipath transport still balance load if only a minority of the flows support it?

We ran our multipath TCP implementation on a server dual-homed with 100Mbps links and on a set of client machines. As these machines are all local, we used dummynet to add 20ms of latency to emulate a wide-area scenario.

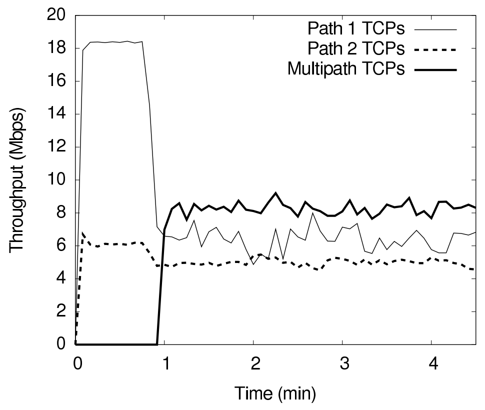
We ran several sets of experiments; we present two here. The aim is to load the network asymmetrically with regular TCP traffic (Linux 2.6 kernel running NewReno), and then see how well a few multipath TCP flows can re-balance it. In the first experiment there are 5 TCP flows on one link and 15 on the second link. In the second experiment there are 15 TCP flows on one link and 25 on the other. We let conditions stabilize so we can see how unbalanced the starting conditions are, and then after one minute we start ten multipath TCP flows and observe the throughput.

Figures 14(a) and 14(b) show the average TCP throughput on each link and the average multipath TCP throughput. Individual flows vary a little from the average on short timescales, but within each category all the flows achieve roughly the same throughput when measured over multiple loss intervals.

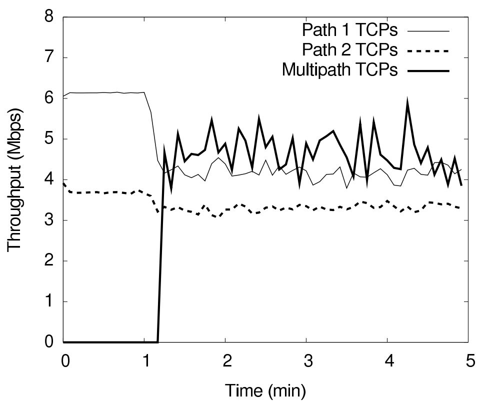
It is clear that even a small fraction of multipath flows ($^1/_3$ in the first case, $^1/_5$ in the second) can significantly help in balancing load. In neither case is the balance perfect—only the COUPLED algorithm could do that and it would not work well in the wireless case. However it is close enough for all practical purposes, and the multipath flows are within about 10% of their target rate. In effect multipath transport enables the links to be used as a single pooled resource, in the way originally envisaged by the fluid models.

7. RELATED WORK

We have already discussed in §2 the theoretical work on multipath congestion control, and in particular the use of fluid flow models to demonstrate resource pooling. In this paper our intent has been to understand the issues that arise



(a) 5 and 15 TCP Flows



(b) 15 and 25 TCP Flows

Figure 14: Server Load Balancing: 10 Multipath Flows Balance Traffic across Links

in bringing resource pooling to the Internet.

There has been much work on building practical multipath transport protocols [9, 22, 14, 8, 10, 3, 16, 4], though none of this work addresses the problem we have studied of how to achieve resource pooling.

Most prior work focuses on the protocol mechanisms needed to implement multipath transmission, with key goals being robustness to long term path failures and to short term variations in conditions on the paths. The main questions are how to split sequence numbers across paths (i.e. whether to use one sequence space for all subflows or one per subflow with an extra connection-level sequence number), how to do flow control (subflow, connection level or both), how to ack, and so forth. Our implementation uses the current multipath TCP protocol specification [4].

In most existing proposals, there is little consideration for the congestion control aspects of multipath transport. Some do try to detect a shared bottleneck to ensure bottleneck fairness; none of them considers resource pooling, and most of them fail to achieve fairness to other flows. Let us highlight the congestion control characteristics of these proposals.

pTCP [8], CMT over SCTP[10] and M/TCP [16] use uncoupled congestion control on each path, and are not fair to competing single-path traffic in the general case.

mTCP [22] also performs uncoupled congestion control on each path. In an attempt to detect shared congestion it computes the correlation between fast retransmit intervals on different subflows. It is not clear how robust this detector is.

R-MTP [14] targets wireless links: it periodically probes the bandwidth available for each subflow and adjusts the rates accordingly. To detect congestion it uses packet interarrival times and jitter, and infers mounting congestion when it observes increased jitter. This only works when wireless links are the bottleneck.

The work in [7] examines fairness at shared bottlenecks; that work was also motivated by fluid flow models. The idea is to constrain the aggregate multipath TCP flow to grow as a TCP would, by spreading the one packet per RTT increase over multiple subflows using an "aggressiveness manager" which attempts to be fair to TCP. It is not clear from the paper what the resulting behavior is. In addition, the proposed algorithm does not perform RTT compensation, which will be necessary for good performance in scenarios such as Fig. 13(a).

Network layer multipath. ECMP[18] achieves load balancing at the flow level, without involving end-systems. It sends all packets from a given flow along the same route in order that end-systems should not see any packet re-ordering. To do this it needs to look at transport-layer parts of the packet header, so it is not a pure network-layer solution. ECMP and end-system multipath differ in the path choices they have available, and it is not clear which is more useful or even if they are compatible.

Horizon [15] is a system for load balancing at the network layer, for wireless mesh networks. Horizon network nodes maintain congestion state and estimated delay for each possible path towards the destination; hop-by-hop backpressure is applied to achieve near-optimal throughput, and the delay estimates let it avoid re-ordering.

Application layer multipath. BitTorrent [2] is an example of application layer multipath. Different chunks of the same file are downloaded from different peers to increase throughput. BitTorrent works at chunk granularity, and only optimizes for throughput, downloading more chunks from faster servers. Essentially BitTorrent is behaving in a similar way to uncoupled multipath congestion control, albeit with the paths having different endpoints. While uncoupled congestion control does not balance flow rates, it nevertheless achieves some degree of load balancing when we take into account flow sizes [13, 19], by virtue of the fact that the less congested subflow gets higher throughput and therefore fewer bytes are put on the more congested subflow. This is called 'job-level resource pooling' as opposed to 'rate-level resource pooling'.

8. CONCLUSIONS AND FUTURE WORK

We have demonstrated a working multipath congestion control algorithm. It brings immediate practical benefits: in §6 we saw it seamlessly balance traffic over 3G and WiFi radio links, as signal strength faded in and out. It is safe to use: the fairness mechanism from §5 ensures that it does not harm other traffic, and that there is always an incentive to turn it on because its aggregate throughput is at least as good as would be achieved on the best of its available paths. It should be beneficial to the operation of the Internet, since it balances congestion and pools resources as promised in §2, at least in so far as it can given topological constraints and the requirements of fairness.

Our main theoretical finding is that if a multipath congestion control tries myopically to balance congestion, then it is not robust to transient noise in congestion feedback nor to dynamically varying background load. We formulated the *principle of equipoise*, which says that these problems may be resolved by balancing traffic across paths, to some suitable extent. Our proposed congestion control algorithm makes a reasonable compromise between myopic resource pooling and balance.

We believe our multipath congestion control algorithm is safe to deploy as part of the IETF's ongoing efforts to standardize Multipath TCP[4] or with SCTP, and it will perform well. This is timely, as the rise of multipath-capable smart phones and similar devices has made it crucial to find a good way to use multiple interfaces more effectively. Currently such devices use heuristics to periodically choose the best interface, terminating existing connections and re-establishing new ones each time a switch is made. Combined with a transport protocol such as Multipath TCP or SCTP, our congestion control mechanism avoids the need to make such binary decisions, but instead allows continuous and rapid rebalancing on short timescales as wireless conditions change.

Our congestion control scheme is designed to be compatible with existing TCP behavior. However, existing TCP has well-known limitations when coping with long high-speed paths. To this end, Microsoft incorporate Compound TCP[17] in Vista and Windows 7, although it is not enabled by default, and recent Linux kernels use Cubic TCP[5]. We believe that Compound TCP should be a very good match for our congestion control algorithm. Compound TCP kicks in when a link is underutilized to rapidly fill the pipe, but it falls back to NewReno-like behavior once a queue starts to build. Such a delay-based mechanism would be complementary to the work described in this paper, but would further improve a multipath TCP's ability to switch to a previously congested path that suddenly has spare capacity. We intend to investigate this in future work.

9. REFERENCES

[1] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman. A measurement-based analysis of multihoming. In *Proc. ACM SIGCOMM '03*, 2003.

[2] B. Cohen. Incentives build robustness in BitTorrent, 2003.

[3] Y. Dong, D. Wang, N. Pissinou, and J. Wang. Multi-path load balancing in transport layer. In *Proc. 3rd EuroNGI Conference on Next Generation Internet Networks*, 2007.

[4] A. Ford, C. Raiciu, M. Handley, and S. Barre. TCP Extensions for Multipath Operation with Multiple Addresses. Internet-draft, IETF, 2009.

[5] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42(5), 2008.

[6] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley. Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the Internet. *IEEE/ACM Trans. Networking*, 14(6), 2006.

[7] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda. Multipath Congestion Control for Shared Bottleneck. In *Proc. PFLDNeT workshop*, May 2009.

[8] H.-Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proc. MobiCom '02*, pages 83–94, New York, NY, USA, 2002. ACM.

[9] C. Huitema. Multi-homed TCP. Internet draft, IETF, 1995.

[10] J. R. Iyengar, P. D. Amer, and R. Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Trans. Netw.*, 14(5):951–964, 2006.

[11] F. Kelly and T. Voice. Stability of end-to-end algorithms for joint routing and rate control. *CCR*, 35(2), Apr. 2005.

[12] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49, 1998.

[13] P. Key, L. Massoulié, and D. Towsley. Path selection and multipath congestion control. In *Proc. IEEE Infocom*, May 2007. Also appeared in proceedings of IEEE ICASSP 2007.

[14] L. Magalhaes and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. *ICNP*, page 0165, 2001.

[15] B. Radunović, C. Gkantsidis, D. Gunawardena, and P. Key. Horizon: balancing TCP over multiple paths in wireless mesh network. In *Proc. MobiCom '08*, 2008.

[16] K. Rojviboonchai and H. Aida. An evaluation of multi-path transmission control protocol (M/TCP) with robust acknowledgement schemes. *IEICE Trans. Communications*, 2004.

[17] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP approach for high-speed and long distance networks. In *Proc. IEEE INFOCOM 2006*, pages 1–12, April 2006.

[18] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991 (Informational), Nov. 2000.

[19] B. Wang, W. Wei, J. Kurose, D. Towsley, K. R. Pattipati, Z. Guo, and Z. Peng. Application-layer multipath data transfer via TCP: schemes and performance tradeoffs. *Performance Evaluation*, 64(9–12), 2007.

[20] W.-H. Wang, M. Palaniswami, and S. H. Low. Optimal flow control and routing in multi-path networks. *Performance Evaluation*, 52(2–3), 2003.

[21] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A reordering-robust TCP with DSACK. In *Proc ICNP 03*, 2003.

[22] 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 '04*, 2004.

APPENDIX

A. A LINK-FAIR ALGORITHM

In this appendix we describe a modification to RTT COMPENSATOR that achieves per-link fairness rather than per-path fairness.

A.1 Introduction

The constraints of the RTT COMPENSATOR congestion controller are to ensure that a multipath flow gets as much aggregate throughput as a single-path TCP flow would on the best of the paths available to it (so there is an incentive for users to deploy multipath), while not using any more capacity on any path than a single-path TCP flow would (so that it does not unduly harm other flows). Given these constraints, the controller aims to move traffic off the most-congested paths.

To express the constraints formally, we first introduce some notation. Suppose the multipath flow is able to use paths in a finite set R. On path $r \in R$, let RTT_r be the round trip time, let w_r be the instantaneous window size, let $\hat{w}_r$ be the equilibrium window size, let p_r be the packet loss probability, and let $\hat{w}_r^{\mathsf{TCP}}$ be the equilibrium window size that a single-path TCP flow would get on that path when faced with p_r. Let $x_r = w_r/\mathsf{RTT}_r$ etc. The constraints of RTT COMPENSATOR may be written

$$\sum_{r\in R} \hat{x}_r = \max_{r\in R} \hat{x}_r^{\mathsf{TCP}} \tag{5}$$

and

$$\hat{x}_r \le \hat{x}_r^{\mathsf{TCP}} \quad \text{for all } r \in R. \tag{6}$$

Figure 15 illustrates a scenario where these constraints might be deemed inadequate. According to (5) the multipath flow should get aggregate throughput $\hat{x}_1 + \hat{x}_2 + \hat{x}_3 = 5$. Since $p_2 < p_1$, we want as much of the aggregate traffic to go on paths 2 and 3 as possible, with no preference between the two (though equipoise suggests we should balance them). The diagram suggests that to avoid unduly harming any other flows that might be using the links, we should ensure

$$\hat{x}_1 \le 5 \quad \text{and} \quad \hat{x}_2 + \hat{x}_3 \le 3, \tag{7}$$

which suggests that the optimal allocation is $(\hat{x}_1, \hat{x}_2, \hat{x}_3) = (2, 1.5, 1.5)$. Yet (5) & (6) only impose

$$\hat{x}_1 + \hat{x}_2 + \hat{x}_3 = 5,$$
$$\hat{x}_1 \le 5, \quad \hat{x}_2 \le 3 \quad \text{and} \quad \hat{x}_3 \le 2$$

which allows $(\hat{x}_1, \hat{x}_2, \hat{x}_3) = (0, 3, 2)$. This latter allocation is pathwise-fair, as per the stated goal of RTT COMPENSATOR, but it means that the multipath flow takes a total of 5 units of capacity on link 2, more than any of the single-path TCP flows would.

The remedy is to impose a stronger version of (6). Instead of (5) & (6) we propose the objectives

Goal 4 (Improve throughput)

$$\sum_{r\in R} \hat{x}_r \ge \max_{r\in R} \hat{x}_r^{TCP}$$

Goal 5 (Do no harm) *For all $S \subseteq R$,*

$$\sum_{r\in S} \hat{x}_r \le \max_{r\in S} \hat{x}_r^{TCP}.$$

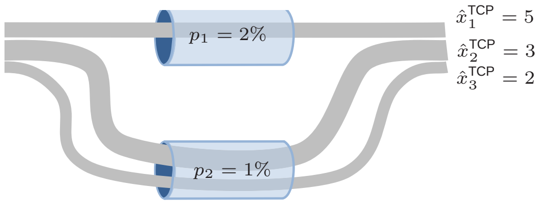

$p_1 = 2\%$
$\hat{x}_1^{\mathsf{TCP}} = 5$
$\hat{x}_2^{\mathsf{TCP}} = 3$
$\hat{x}_3^{\mathsf{TCP}} = 2$
$p_2 = 1\%$

Figure 15: A multipath flow with three subflows.

This last collection of inequalities ensures that the multipath flow takes no more capacity on any link or collection of links than would a single-path TCP flow on one of the paths $r \in R$. To see this, let S be the set of paths that use the links in question: then the left hand side of (6) is the total throughput of the multipath flow on those links, and the right hand side is the throughput of a single-path flow through those links. In the scenario of Figure 15, Goal 5 requires

$$\begin{aligned} &\hat{x}_1 \le 5 && \\ &\hat{x}_1 + \hat{x}_2 \le 5 && \hat{x}_2 \le 3 \\ &\hat{x}_1 + \hat{x}_3 \le 5 && \hat{x}_3 \le 2 \\ &\hat{x}_1 + \hat{x}_2 + \hat{x}_3 \le 5 && \hat{x}_2 + \hat{x}_3 \le 3. \end{aligned}$$

This full set of inequalities is more stringent than (7). However, one or more of the additional constraints would be appropriate if there were further shared bottleneck links in addition to those shown here. We take the view that it is better to guarantee doing no harm for any possible configuration of shared bottlenecks, rather than attempt to detect shared bottlenecks. See also Section A.3 for an explanation of why detection of shared bottlenecks is conceptually problematic.

A.2 A link-fair algorithm

Consider the following algorithm, an extension of RTT COMPENSATOR:

ALGORITHM: LINK-FAIR

- Each ACK on subflow r, increase the window w_r by

$$\min_{S\subseteq R\,:\,r\in S} \frac{\max_{s\in S} w_s/\mathsf{RTT}_s^2}{\left(\sum_{s\in S} w_s/\mathsf{RTT}_s\right)^2}. \tag{8}$$

- Each loss on subflow r, decrease the window w_r by $w_r/2$.

Note that if we took the minimum in (8) over $\{r\}$ and R, rather than over all subsets S for which $r \in S$, the increase term would match exactly the increase in RTT COMPENSATOR. This reflects the fact that RTT COMPENSATOR guarantees fairness over individual paths and over the whole, whereas LINK-FAIR guarantees fairness over all possible shared bottlenecks.

The complexity of computing the minimum in (8) is linear in the number of paths, if window sizes are kept in the order of (9) below.

The following lemmas concern the properties of an equilibrium point, i.e. a set of window sizes $\hat{w}_r$ satisfying the balance equations

$$\min_{S\,:\,r\in S} \frac{\max_{s\in S} \hat{w}_s/\mathsf{RTT}_s^2}{\left(\sum_{s\in S} \hat{w}_s/\mathsf{RTT}_s\right)^2} = p_r \frac{\hat{w}_r}{2} \quad \text{for each } r \in R.$$

They show that an equilibrium point satisfies the two fairness goals.

Lemma 1 *Any equilibrium point satisfies Goal 5.*

Lemma 2 *Let the equilibrium window sizes be ordered such that*

$$\frac{\sqrt{\hat{w}_1}}{RTT_1} \le \frac{\sqrt{\hat{w}_2}}{RTT_2} \le \cdots \le \frac{\sqrt{\hat{w}_n}}{RTT_n}. \tag{9}$$

Then

$$\sum_r \frac{\hat{w}_r}{RTT_r} = \frac{\hat{w}_n^{TCP}}{RTT_n}. \tag{10}$$

Furthermore, for all r,

$$\frac{\hat{w}_r^{TCP}}{RTT_r} \le \frac{\hat{w}_n^{TCP}}{RTT_n}. \tag{11}$$

Proof of Lemma 1. First we will find an expression for $\hat{w}_r^{\mathsf{TCP}}$ in terms of the $\hat{w}_r$. For $S \subseteq R$, define

$$i(S) = \frac{\max_{r\in S} \sqrt{\hat{w}_r}/\mathsf{RTT}_r}{\sum_{r\in S} \hat{w}_r/\mathsf{RTT}_r}.$$

Rewriting the balance equations in terms of $i(\cdot)$, and in terms of $\hat{w}_r^{\mathsf{TCP}} = \sqrt{2/p_r}$,

$$\min_{S\,:\,r\in S} i(S)^2 = p_r \frac{\hat{w}_r}{2} = \frac{\hat{w}_r}{(\hat{w}_r^{\mathsf{TCP}})^2}$$

hence

$$\hat{w}_r^{\mathsf{TCP}} = \sqrt{\hat{w}_r} \max_{S\,:\,r\in S} 1/i(S). \tag{12}$$

Now, take any $T \subseteq R$. We shall verify the equation in Goal 5 for set T. Rearranging the definition of $i(T)$,

$$\begin{aligned}\sum_{r\in T} \hat{w}_r/\mathsf{RTT}_r &= \frac{\max_{r\in T} \sqrt{\hat{w}_r}/\mathsf{RTT}_r}{i(T)} \\ &= \max_{r\in T} \frac{1}{\mathsf{RTT}_r} \sqrt{\hat{w}_r}/i(T) \\ &\le \max_{r\in T} \frac{1}{\mathsf{RTT}_r} \sqrt{\hat{w}_r} \max_{S\,:\,r\in S} 1/i(S) \\ &= \max_{r\in T} \frac{1}{\mathsf{RTT}_r} \hat{w}_r^{\mathsf{TCP}} \quad \text{by (12).}\end{aligned}$$

Since T was arbitrary, we have proved Goal 5. □

Proof of Lemma 2. With the window sizes ordered as in the statement of the lemma, we can rewrite (12):

$$\begin{aligned}\frac{\hat{w}_r^{\mathsf{TCP}}}{\mathsf{RTT}_r} &= \frac{\sqrt{\hat{w}_r}}{\mathsf{RTT}_r} \max_{S\,:\,r\in S} \frac{\sum_{r\in S} \hat{w}_r/\mathsf{RTT}_r}{\max_{r\in S} \sqrt{\hat{w}_r}/\mathsf{RTT}_r} \\ &= \frac{\sqrt{\hat{w}_r}}{\mathsf{RTT}_r} \max_{u\ge r} \frac{\sum_{t\le u} \hat{w}_t/\mathsf{RTT}_t}{\sqrt{\hat{w}_u}/\mathsf{RTT}_u}\end{aligned} \tag{13}$$

which at $r = n$ reduces to (10).

We prove (11) by induction. It is obviously true at $r = n$. Suppose it is true at $r+1, \ldots, n$. Let

$$\gamma_r = \max_{u\ge r} \frac{\sum_{t\le u} \hat{w}_t/\mathsf{RTT}_t}{\sqrt{\hat{w}_u}/\mathsf{RTT}_u}$$

so that (13) can be rewritten

$$\begin{aligned}\frac{\hat{w}_r^{\mathsf{TCP}}}{\mathsf{RTT}_r} &= \frac{\sqrt{\hat{w}_r}}{\mathsf{RTT}_r}\gamma_r \\ &= \frac{\sqrt{\hat{w}_r}}{\mathsf{RTT}_r} \max\Big(\frac{\sum_{t\le r} \hat{w}_t/\mathsf{RTT}_t}{\sqrt{\hat{w}_r}/\mathsf{RTT}_r}, \gamma_{r+1}\Big) \\ &= \max\Big(\sum_{t\le r} \frac{\hat{w}_t}{\mathsf{RTT}_t}, \frac{\sqrt{\hat{w}_r}}{\mathsf{RTT}_r}\gamma_{r+1}\Big).\end{aligned}$$

By (10) the first term is $\le \hat{w}_n^{\mathsf{TCP}}/\mathsf{RTT}_n$. By the ordering of the window sizes, the second term is $\le \gamma_{r+1}\sqrt{\hat{w}_{r+1}}/\mathsf{RTT}_{r+1}$, which is nothing other than $\hat{w}_{r+1}^{\mathsf{TCP}}/\mathsf{RTT}_{r+1}$, and by the induction hypothesis this too is $\le \hat{w}_n^{\mathsf{TCP}}/\mathsf{RTT}_n$. This completes the induction step. Thus we have proved (11). □

A.3 Ineffective means for achieving fairness

Weighted increase. Our algorithm LINK-FAIR uses weighted increases, and we proved it is fair. The proposal in [7] also uses weighted increases: it increases w_r by a_r^2/w_r per ACK, so that $\hat{w}_r = a_r \hat{w}_r^{\mathsf{TCP}}$, and it adjusts the weights a_r to ensure that $\sum_r a_r = 1$. This guarantees Goal 5, since for any $S \subseteq R$

$$\begin{aligned}\sum_{r\in S} \frac{\hat{w}_r}{\mathsf{RTT}_r} &= \sum_{r\in S} a_r \frac{\hat{w}_r^{\mathsf{TCP}}}{\mathsf{RTT}_r} \\ &\le \Big(\sum_{r\in S} a_r\Big) \max_{r\in S} \frac{\hat{w}_r^{\mathsf{TCP}}}{\mathsf{RTT}_r} \\ &\le \max_{r\in S} \frac{\hat{w}_r^{\mathsf{TCP}}}{\mathsf{RTT}_r}.\end{aligned}$$

It achieves Goal 4 if and only if

$$\hat{x}_r^{\mathsf{TCP}} < \max_t \hat{x}_t^{\mathsf{TCP}} \quad \Longrightarrow \quad a_r = 0. \tag{14}$$

The 'proportion manager' in [7] chooses the path weights adaptively: it measures the average $\hat{w}_r^{\mathsf{TCP}}$ over some long enough interval, and decreases a_r on paths where $\hat{w}_r^{\mathsf{TCP}}$ is small i.e. where p_r is large. This has the same outcome as COUPLED, namely it moves traffic onto paths with the least congestion. Since it adapts a_r based on $\hat{w}_r^{\mathsf{TCP}}$ rather than on $\hat{x}_r^{\mathsf{TCP}}$, this algorithm does not in general achieve (14), therefore it does not achieve Goal 4. If the proportion manager adapted a_r in favor of paths with large $\hat{x}_r^{\mathsf{TCP}}$ rather than paths with large $\hat{w}_r^{\mathsf{TCP}}$, it would achieve Goal 4 but it would not in general prefer less-congested paths, so it would do a poor job of resource pooling.

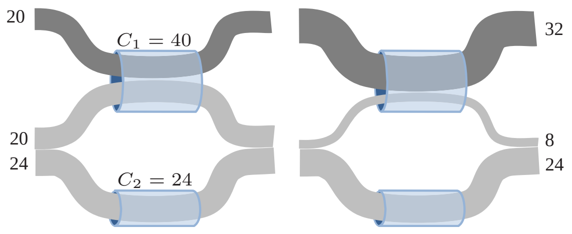


Figure 16: A multipath flow may take a 'fair share' at each bottleneck, but nevertheless be unfair in aggregate (left). It would be better to take a fair share of the pooled bottleneck (right).

Shared bottleneck detection. It is not sufficient for a multipath flow to detect shared bottleneck links, and to ensure it takes a fair share of capacity at each link. To see this, consider the scenario in the left of Figure 16. The multipath flow uses two disjoint links, and takes a 'fair' share of capacity at each of the links—with the consequence that it takes an unfair share of the network's aggregate capacity. A more desirable outcome is shown in the right hand diagram. In this scenario the multipath flow enables the network to share capacity as though the two links were a single pooled resource, and so it is more natural to seek fairness across the resource pool than over individual links.

Our Goal 5 entails fairness across resource pools. In fact it says that a multipath flow should be fair (with respect to TCP) across any link *or collection of links*, therefore it entails fairness whatever the topology of paths and resource pools.

It has been proposed that a multipath flow should detect shared congestion between its paths, and reduce its aggressiveness on paths that share congestion [22]. In one sense, the two links in this scenario do share congestion, because they are part of the same resource pool: if there is a traffic surge on link 1, then congestion will increase on link 1, so some of the multipath traffic should shift onto link 2, so congestion will increase on link 2. We believe it is extremely unlikely that shared congestion can be detected by correlated fast retransmits (as used by [22]) when the congestion is shared across a bottleneck resource pool rather than at an individual link.