Practical Congestion Control for Multipath Transport Protocols

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

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

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

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.

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. When a flow shifts its traffic onto less congested paths, then the loss rate on the less congested path will increase and that on the more congested path will decrease; the overall outcome with many multipath flows is that the loss rates across an interconnected network of paths will tend to equalize. This is a form of load balancing, or more generally *resource pooling* [14], described further in §2.

Multipath congestion control should be designed to achieve a fair allocation of resources. For example, if a multipath flow has four paths available and they all happen to go through the same bottleneck link, and if we simply run TCP's congestion avoidance independently on each path, then this flow will grab four times as much bandwidth as it should. In fact, the very idea of "shifting traffic from one path to another" in the previous paragraph presupposes that there is some fair total traffic rate, and that extra traffic on one path should be compensated for by less traffic on the other. The problem of fairness is made even harder by round-trip-time dependence. For example, in figure 1, suppose that path A_1 has a loss rate of 4% and a round trip time of 10ms, and path A_2 has a loss rate of 1% and a round trip time of 100ms. Resource pooling suggests that flow A should send all its traffic on the less congested path A_2, but a simple-minded interpretation of TCP fairness says that

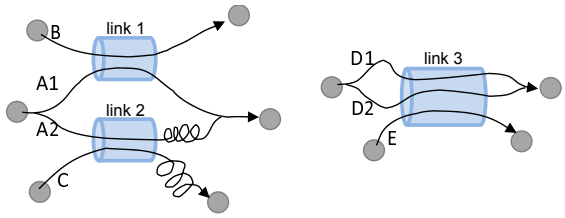


Figure 1: Test scenarios for multipath congestion control

throughput should be proportional to $1/\mathsf{RTT}\sqrt{p}$ where p is the packet drop probability, which would give A_1 five times higher throughput than A_2. In §4 we describe a more appropriate notion of fairness.

In this paper, we propose a multipath congestion control algorithm for balancing traffic across multiple paths. It is a modification of TCP: it maintains a window on each path that it uses, it increases the window on a path in response to acknowledgements on that path, and it decreases the window on a path in response to drops on that path. The amount by which it increases or decreases depends on the windows and round trip times of all the other paths in use. By choosing the increase and decrease appropriately, the algorithm is able to achieve a reasonable degree of resource pooling. Furthermore it explicitly guarantees fairness with current TCP, and it does this without needing any explicit knowledge about which paths share bottleneck links. We believe it is sufficiently well-behaved to be deployable in today's Internet. We present experimental evaluation in §5.

Previous work has proven that resource pooling and bottleneck fairness can be achieved by simple increase / decrease rules [4, 7]. That work is based on differential equation models of congestion control. When we tried to interpret those equations as a discrete packet-based system, we found that flows were highly "flappy": they would use one path almost exclusively for a while, then flip to another path, then repeat. The reason for this is explained further in §2, and the modifications we make to prevent flap in §3. Furthermore the theoretical work was based on rate-based congestion control, and it does not attempt to play fair with existing TCP. Our fix for this is described in §4. This paper, and our proposed multipath congestion control algorithm, is a practical response to these two concerns with the theory.

In §7 we describe some paths for future work.

Design goals

Before we get into details about algorithm design, let us outline how we would like a multipath flow to behave.

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.*

To see how these two work together, consider bottleneck link 3 in figure 1, and suppose for simplicity that all RTTs are equal. Goal 2 says that the throughput of flow E should be no worse than if flow D only had one path, i.e. half the link's capacity. Goal 1 says that the throughput of flow D should be at least what it would get if it had only one path, i.e. half the link's capacity. The two goals together require that flows D and E share the capacity equally.

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.*

2. FLAPPINESS & RESOURCE POOLING

Previous theoretical work on multipath congestion control [4, 7] suggests that goals 1–3 can be satisfied by a simple extension of TCP's congestion control, in the case where all flows have common RTT. Consider a multipath flow with available paths $r = 1, \ldots, N$, and suppose it uses window-based flow control on each. Let w_r be the window size on flow r, and let $w = \sum_r w_r$ be the total window size. Use the following rule:

ALGORITHM: FULLY COUPLED

- increase w_r by $1/w$ per ack on path r;
- decrease w_r to $\max(w_r - w/b, 1)$ per loss event on path r; if there is truncation then do a timeout but no slow-start; use $b = 2$ to mimic TCP.

When all the paths traverse a common bottleneck (and neglecting the truncation at $w_r = 1$), each loss causes the same back-off of total window as would happen for a regular TCP flow, and each ack causes the same window increase as would happen for a regular TCP flow, so goals 1 & 2 are satisfied. For more general topologies, here is a more mathematical argument. Suppose that the loss rate on path r is p_r. In equilibrium, the increases and decreases of w_r should balance out, so

$$(1 - p_r)\frac{1}{\hat{w}} \approx p_r \frac{\hat{w}}{b}$$

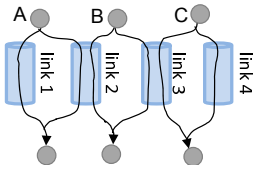


Figure 2: A scenario where multipath congestion control can achieve resource pooling

where $\hat{w}$ is the equilibrium value of w. If some path is more congested than another, say $p_x > p_y$, then the window decreases on path x will outweigh the increases, resulting in zero throughput on path x, i.e. the flow will only use the paths with minimum drop probability p_{min}. By the balance equation, and making the approximation that $1 - p_{\text{min}} \approx 1$, we get $\hat{w} = \sqrt{b/p_{\text{min}}}$.

Using $b = 2$, the total window size for a flow is $\sqrt{2/p_{\text{min}}}$, and this fact guarantees goals 1 & 2. Consider for example bottleneck link 3 in figure 1: all three paths experience the same drop probability, therefore both flows get the same total window size, and since RTTs are assumed to be equal both flows get the same throughput.

The scenario in figure 2 shows an interesting outcome of goal 3 and the formula for $\hat{w}$. The goal implies that flows will tend to equalize the congestion on all available paths. Each flow therefore sees the same smallest loss rate, so each flow gets the same window size hence the same throughput. This outcome, where the capacity of a collection of links is shared equally between the flows that use them, is referred to as "resource pooling". It means for example that if one of the links experiences a loss in capacity then it can shed load to the other links, and therefore its loss rate does not increase by as much as it otherwise would.

This algorithm has the obvious problem at $b = 2$ that whenever the smaller window decreases, it is truncated all the way to $w_r = 1$, hence the algorithm spends most of its time using either one path or the other, rarely both. We call this "flappy". However, flappiness is not simply a consequence of setting $b = 2$. Suppose there is a flow using two paths, with $b = 4$, and consider the somewhat contrived scenario that the two paths share a single bottleneck link and that there is a loss whenever $w_1 + w_2 = 100$. The left hand plot in figure 3 shows how the window sizes w_1 (horizontal axis) and w_2 (vertical axis) might evolve. Starting at $w_1 = w_2 = 1$, both windows will increase until $w_1 = w_2 = 50$. Suppose that path 1 experiences a drop and w_1 decreases. The two windows will then grow until $w_1 + w_2 = 100$ again, and w_2 will grow at a faster rate because it is sending at a higher rate. (This shows itself as radial increase lines in the figure.) Just one more drop on path 1 is enough to push w_1 down to 11 packets. At this point it will take very many drops on path 2 for the two windows to

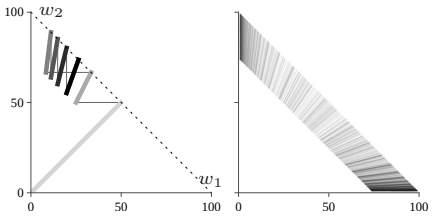


Figure 3: Window size dynamics for a two-path flow running FULLY COUPLED with $b = 4$

equalize again; the plot shows the state after three further drops on path 2. The right hand plot in figure 3 shows the same diagram after several thousand drops, and it omits the decrease lines.

The conclusion is that the FULLY COUPLED algorithm is unstable in the region where $w_1 \approx w_2$, tending to flip towards one of the extremes. Once it reaches an extreme say $w_1 \approx 0$ it is captured there for a long time, because it takes a long run of consecutive drops on the other path to allow w_1 to increase significantly. Over a very long timescale it tends to equalize traffic over the two paths, by symmetry; a closer look at simulation output indicates that it flips from one extreme to the other at random intervals.

The top two plots in figure 4 show window dynamics for $b = 4$ when packet loss occurs with fixed probability (as though there is a high degree of statistical multiplexing, so that the values of w_1 and w_2 do not significantly affect the loss rate). In the left hand plot the loss rates are equal, and the density of the plot shows capture at $w_1 \approx 0$ or $w_2 \approx 0$. In the right hand plot the loss rate on path 1 is lower, and the flow is mainly captured at $w_2 \approx 0$ as we would like.

There seem to be two drivers of flappiness. (i) The flow attempts to move traffic off congested paths and onto uncongested paths. If one path has a loss rate that is just a touch higher than another path, then the first path is still more congested, and the math says that traffic will move away from it. Even when two paths have the same loss rate, chance fluctuations will mean that from time to time the first path suffers a few extra drops, so it will look momentarily more congested, so the flow will flip to the other path. To overcome this, it will be necessary either to accept less perfect resource pooling, or to use smoothed estimates of loss which will result in a more sluggish response. (ii) The FULLY COUPLED algorithm also has the problem of capture: if flow 1 experiences a few drops, flow 2 needs to experience several times more drops to bring the traffic rates back into balance.

3. REDUCING FLAPPINESS

We now propose an algorithm that reduces flappiness, at the cost of slightly worse resource pooling. We will assume that all paths have equal round trip time throughout this section (the case of general round trip times is in section 4). A simple proposal is to use:

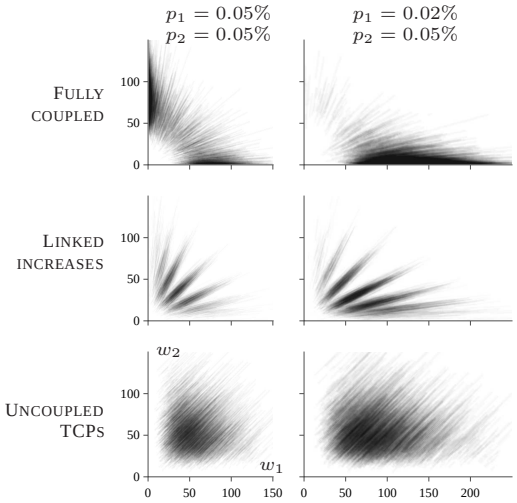


Figure 4: Window size dynamics for a two-path flow.

ALGORITHM: LINKED INCREASES
- increase w_r by a/w per ack on path r;
- decrease w_r by $w_r/2$ per loss event on path r

where a should be chosen to control the overall aggressiveness of the flow, as set out in section 4.

To understand the motivation for this algorithm, consider first what happens when we run separate TCP congestion control on each of the paths:

ALGORITHM: UNCOUPLED TCPS
- increase w_r by $1/w_r$ per ack on path r;
- decrease w_r by $w_r/2$ per loss event on path r.

The bottom left plot in figure 4 shows the evolution of window size with UNCOUPLED TCPS. The increase lines are all parallel, since w_1 and w_2 increase at exactly the same rate, one packet per round trip time. There is no problem with flappiness. On the other hand, the bottom right hand plot shows that this algorithm is not very good at moving traffic away from a congested link.

LINKED INCREASES takes the increase rule from FULLY COUPLED and the decrease rule from UNCOUPLED TCPS. It is intended to combine the resource pooling of the former with the unflappability of the latter. The decreases are like those for regular TCP, so this algorithm does not suffer from truncation problems any more than TCP does. The increases can be bounded to be no more than that for a single-path TCP, by choosing a appropriately, which should prevent the algorithm from being too aggressive. This is just one of many possibilities—a more general class is described in section 7.

The middle row of figure 4 plots window sizes for LINKED INCREASES with $a = 1$. The system is less flappy, spending most of its time near the centre of the plot. It is interesting that only a few discrete radial lines

are shown. A little thought shows this is natural: each line has twice the gradient of the preceding line, and the system moves from one line to another whenever a backoff occurs. For the plot we jittered the lines to give a better idea of frequency; in fact the lines are perfectly radial. This system does move traffic away from the congested path; in the right hand plot the system spends more of its time on the lower gradient lines, as can be seen from their increased density.

The equilibrium throughputs may be calculated by observing that in equilibrium, the increases and decreases on each flow should balance out, hence

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

where $\hat{w}_r$ is the equilibrium window size on path r and $\hat{w} = \sum_r \hat{w}_r$. Again, make the approximation that p_r is small so $1 - p_r \approx 1$. This gives $\hat{w}_r = 2a/p_r\hat{w}$. We can then solve for $\hat{w}$ to find $\hat{w} = \sqrt{2a\sum_r 1/p_r}$.

This formula lets us see how much traffic is moved away from the congested path. It shows that total window size $\hat{w}$ is split between the paths in proportion to $1/p_r$; the UNCOUPLED TCPs algorithm by contrast splits its total window size in proportion to $1/\sqrt{p_r}$, according to the standard TCP throughput formula. A $1/p_r$ allocation is more biased in favor of small values of p_r than a $1/\sqrt{p_r}$ allocation, therefore LINKED INCREASES does a better job at balancing congestion. It does not completely meet goal 3, which would require a complete bias against any path whose loss rate is not minimal.

Note that the aggressiveness parameter a should depend on the RTTs. To see this, consider bottleneck 3 in figure 1: paths D_1 and D_2 experience the same loss rate, so they get the same window size, so the total throughput is $\hat{w}(1/RTT_1 + 1/RTT_2)/2$. If path 2 changes to have a very long round trip time then throughput will halve, unless a is increased; the formula for $\hat{w}$ can be used to choose a. However, if a is too large then the flow might be too aggressive on one of its paths, failing goal 2. The next section explains how to adjust a safely.

4. RTT COMPENSATION

We now consider how to adjust LINKED INCREASES to compensate for dissimilar round trip times. We propose a modification to the increase rule:

ALGORITHM: RTT COMPENSATOR
- increase w_r by $\min(a/w,\ 1/w_r)$ per ack on path r;
- decrease w_r by $w_r/2$ per loss event on path r.

The choice of a determines the total throughput, and a formula is given below.

Goal 1, which requires that total throughput should not suffer by using multipath, places a constraint on the equilibrium window sizes $\hat{w}_r$, and thereby on the choice

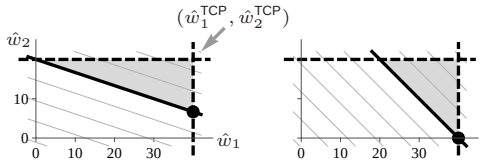


Figure 5: The three design goals place constraints on what the equilibrium window sizes should be.

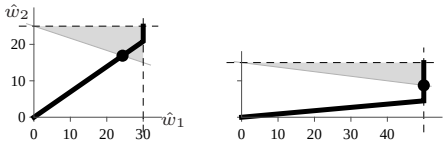


Figure 6: By choosing a we control the equilibrium window sizes.

of a. The equilibrium total throughput is $\sum_r \hat{w}_r/\mathsf{RTT}_r$. A single-path flow experiencing drop probability p_r would have window size $\hat{w}_r^{\mathsf{TCP}} = \sqrt{2/p_r}$ and throughput $\hat{w}_r^{\mathsf{TCP}}/\mathsf{RTT}_r$. The constraint is thus

$$\sum_r \frac{\hat{w}_r}{\mathsf{RTT}_r} \geq \max_r \frac{\hat{w}_r^{\mathsf{TCP}}}{\mathsf{RTT}_r} \qquad (1)$$

where $\max_r$ denotes the maximum over all paths r that the flow can use. (Note that $\hat{w}_r^{\mathsf{TCP}}$ is defined based on the loss rate p_r that is observed when the flow is using multiple paths. If it really only used path r then the loss rate would be higher, so its throughput would be lower than $\hat{w}_r^{\mathsf{TCP}}/\mathsf{RTT}_r$.)

Goal 2 requires that the multipath flow should not take up any more capacity on path r than would a single-path flow. i.e. $\hat{w}_r \leq \hat{w}_r^{\mathsf{TCP}}$. This is guaranteed by the cap in the increase term.

Figure 5 illustrates these constraints for a flow with two paths. The axes show $\hat{w}_1$ and $\hat{w}_2$, and the point $(\hat{w}_1^{\mathsf{TCP}}, \hat{w}_2^{\mathsf{TCP}})$ is shown with an arrow. The two bounds from goal 2 say that we want an equilibrium point left and below the dashed lines. The slanted lines are lines of constant throughput $\hat{w}_1/\mathsf{RTT}_1 + \hat{w}_2/\mathsf{RTT}_2$ (the slope depends on the round trip times, and the figure shows two generic cases); goal 1 says that we want a fixed point above the bold slanted line. Overall, the two goals require that the equilibrium point should lie in the shaded region. Goal 3 tells us where in the shaded region: it says there should be as little traffic as possible on the more congested path. In the figure we have drawn $\hat{w}_1^{\mathsf{TCP}} > \hat{w}_2^{\mathsf{TCP}}$, hence $p_1 < p_2$, hence the goal is to make $\hat{w}_2$ as small as possible subject to the other two goals. The optimal point is indicated in the diagram by a large dot.

The parameter a controls how aggressively to increase window size, hence the overall throughput. The equilibrium window sizes satisfy

$$\hat{w}_r = \min\{(\hat{w}_r^{\mathsf{TCP}})^2 a/\hat{w}, \hat{w}_r^{\mathsf{TCP}}\}; \qquad (2)$$

the first term is what we calculated for the LINKED INCREASE algorithm, only written in terms of $\hat{w}_r^{\mathsf{TCP}}$ rather than p_r, and the second term is what we get when the

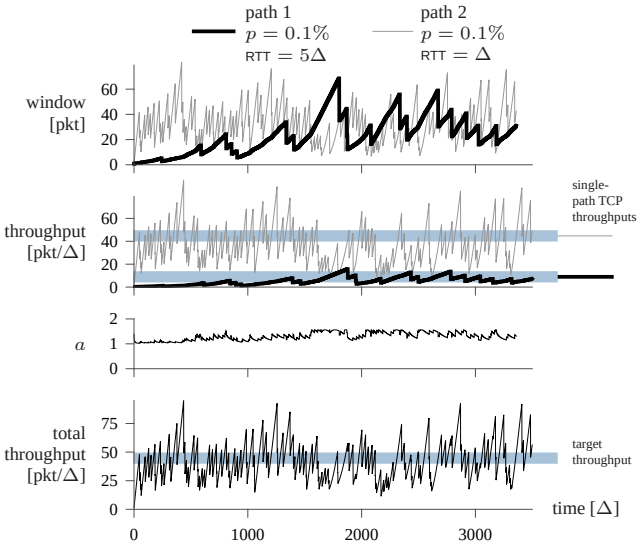


Figure 7: RTT compensation, with equal loss rates

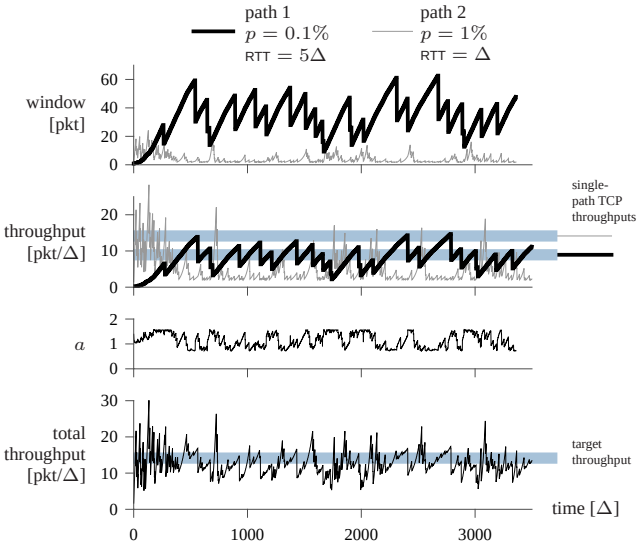


Figure 8: RTT compensation, with unequal loss rates

window cap is in operation. The bold lines in figure 6 shows the range of equilibrium window sizes that we can achieve by changing a. A natural choice is to choose a to just achieve goal 1, and the equilibrium window sizes that result are indicated by large black dots. These do not coincide exactly with the optimal choice according to goal 3, because as discussed in section 3 we sacrificed some congestion balancing for the sake of improved stability. By simultaneously solving (1) & (2) with the inequality replaced by an equality, we find after some algebra that

$$a = \hat{w}\Big(\frac{\max_r \sqrt{\hat{w}_r}/\mathsf{RTT}_r}{\sum_r \hat{w}_r/\mathsf{RTT}_r}\Big)^2.$$

This formula for a obviously requires that we measure the round trip times. It also involves $\hat{w}_r$ and $\hat{w}$, the equilibrium window sizes; we propose that these terms should be estimated simply by the current window size. It may be that some sort of smoothing might help; this is a matter for further work.

5. EXPERIMENTAL EVALUATION

We have tested LINKED INCREASES in two simulators and with a Linux implementation. The first simulator (*cwndsim*) is rtt-based, and only models the evolution of the congestion windows. Drops in this simulator are exogenous, i.e. the multipath flows do not drive the loss rate. This simple simulator mainly serves to validate the theory.

Our second simulator (*htsim*) is packet-based. In contrast with *cwndsim*, it models links with delays and queues, and can simulate arbitrary network topologies. This simulator enables us to investigate the effects of endogenous drops and varying rtts on the properties of the algorithms.

5.1 Experiments with cwndsim

In *cwndsim* we simulate a two-path flow, and track the congestion windows on each path: once per RTT (or per minimum RTT, if the RTTs are different) we apply either an increase or decrease to each window. The RTTs and the loss rate are constant, and every packet is equally likely to be dropped. We use the formula for a from §4, and we use the current window size w_r as a proxy for the equilibrium window size $\hat{w}_r$.

We describe here two runs of this simulator, corresponding to the two generic cases shown in figure 6. In the first run we have $p_1 = p_2 = 0.1\%$ and $\mathsf{RTT}_1 = 5\Delta$ and $\mathsf{RTT}_2 = \Delta$. (The base time unit Δ is immaterial.) Since the loss rates are equal we expect equal window sizes on each path, hence 5 times higher throughput on path 1. The formula predicts $a = 1.39$, so we expect $a/w < 1/w_r = 1/0.5w$ for each path and the increase cap should not come into play. Figure 7 shows a simulation trace: it shows the two window sizes, and a, and the two throughputs, and the total throughput. We see that the total throughput is very close to what a single-path TCP would get at $p = 0.1\%$ (the straight line in the bottom panel). We also see that a varies a little. If by chance $w_2 > w_1$ then w_2 gets most of the increase and the total throughput suffers so a is made larger; if by chance $w_2 < w_1$ then a is made smaller.

In the second run we leave $\mathsf{RTT}_1 = 5\mathsf{RTT}_2$ as before but now $p_1 = 0.1\%$ and $p_2 = 1\%$. If it weren't for the rate cap we would expect window sizes to be inversely proportional to loss rate, giving $a = 1.106$, which means that the increase on path 1 should be capped, which means a actually needs to be slightly higher to compensate. This corresponds to the right hand case in figure 6. The simulation trace in figure 8 shows that the algorithm succeeds in keeping total throughput close

to the target rate, and that traffic is pushed onto the less congested path as much as it can be given the constraints of goals 1 & 2.

5.2 Experiments with htsim

We built *htsim* to support large scale modelling of TCP-like congestion control algorithms. It works at packet level and models the network paths as a sequence of queues (with fixed processing capacity and fixed buffers) and pipes (with a fixed amount of delay). *htsim* can scale to large numbers of flows and to flows with high rates.

Implementation. Our multipath code in htsim implements the congestion window coupling, but does not implement the connection level receive buffer. Receive-buffer blocking is an important area of further study.

Congestion windows are maintained in bytes. The implementation mostly follows the companion internet-draft [11]. The key differences to the draft are:

- The increase parameter "a" is an integer, not a double. To implement this, the formula computing a is multiplied by a scaling coeficient (500), and the formula computing the window increase is divided by the same coeficient.
- Rounding errors affect the window increase computation when the total congestion window is large. These errors affect multipath more than TCP in a few corner cases, so it's wise to fix them. Say for simplicity the increase is computed as a/b. Our fix simulates fractional increases of the congestion window by probabilistically returning a/b+1 with probability a%b / b, or a/b otherwise.

Setup. We use *htsim* in many network setups, including ones where a few flows compete at a bottleneck. Phase effects can appear in such cases, and even more so in simulations where there are few sources of randomness affecting packet spacing [2]. The standard way to avoid them is to use active queue management algorithms, such as Random Early Discard (see [2]). RED does remove phase effects and is fair to bursty traffic, but is notoriously difficult to configure. For our purposes it suffices to remove the phase effects, so we built a simple solution to achieve this. Our solution makes the last three positions in a queue randomly drop packets with probability 1/3. This sufficed to remove phase effects in our experiments.

We use three simple network topologies to test the algorithms, aiming to cover the three goals we've set to achieve (Improve Throughput, Do No Harm, Balance Congestion):

- **Multipath Dumbbell** is an instantiation of the left-hand topology in Figure 1. There are two bottleneck links, one multipath flow consisting of two subflows, each crossing one of the bottleneck links. At each bottleneck there is a configurable number of TCP flows. Each flow arriving at the bottleneck is paced by a feeder link. The feeder links have twice the capacity of the bottleneck link, infinite buffering and add zero delay besides buffering. They are used to avoid excessive burstiness caused by simultaneous packet transmissions of the same host. The multipath dumbbell topology is useful in two main ways. It allows us to understand the effects of buffer driven losses on algorithm behavior, as well as those of the correlation between congestion window and RTT. Secondly, it allows us to see whether multipath does increase throughput compared to single path TCP, and whether the throughput it gets is fair to TCP.
- **Shared Bottleneck** is an instantiation of the right hand topology in Figure 1, with feeder links as above. Here, the multipath subflows have a common bottleneck, and are competing there with a configurable number of TCP flows. This topology allows us to examine bottleneck fairness.
- **Fence** is an instantiation of the topology in Figure 2, with 5 links and 4 multipath flows. Each multipath subflow is paced by a feeder link. We use this topology to examine the level of resource pooling each algorithm provides.

5.2.1 Flappiness

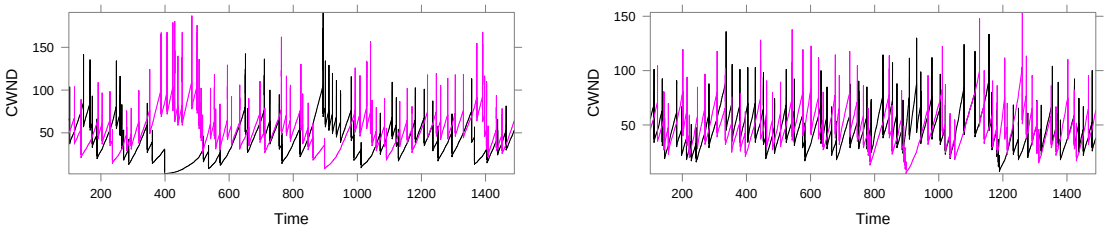
In §2 we have showed that FULLY COUPLED is flappy when the loss rates of the its subflows are equal and constant. In real networks, when there are low levels of statistical multiplexing, the multipath subflows will influence the loss rate. Here, we wish to find out whether FULLY COUPLED is still flappy in such cases.

Consider the multipath dumbbell topology, with one multipath flow competing with a single TCP flow at each bottleneck. If the multipath flow puts more traffic on the top path, the loss rate of the bottom path is reduced; hence, the multipath flow will tend to send more of its traffic through the bottom path. Intuitively, it feels that the algorithm should be less flappy: the loss rates act as springs pushing towards equal window allocation on the two subflows.

In Figure 9(a), we present one run of the FULLY COUPLED algorithm in the dumbbell topology; the simulation setup is presented in Table 1 in the appendix (experiment FC1).

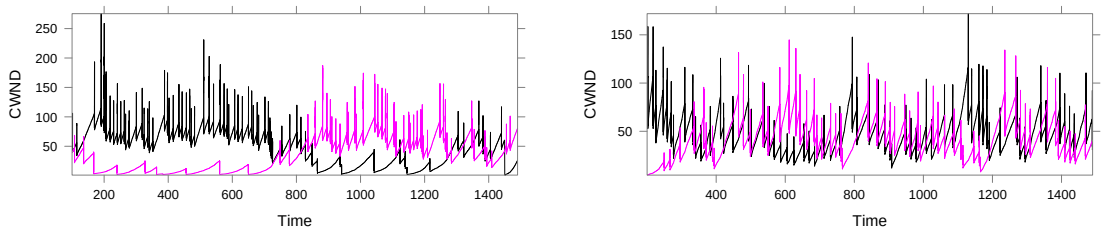
We find that the algorithm is still flappy, even in this case, albeit not as flappy as before. The algorithm seems to be overreacting to loss indications, hence arriving at the edges; from there, it gets more quickly to equi-allocation, but this equilibrium is unstable.

LINKED INCREASES is not flappy with fixed loss rates. Is it flappy when it drives the loss rates? We ran the same experiment (params in line LI1 of table 1) and



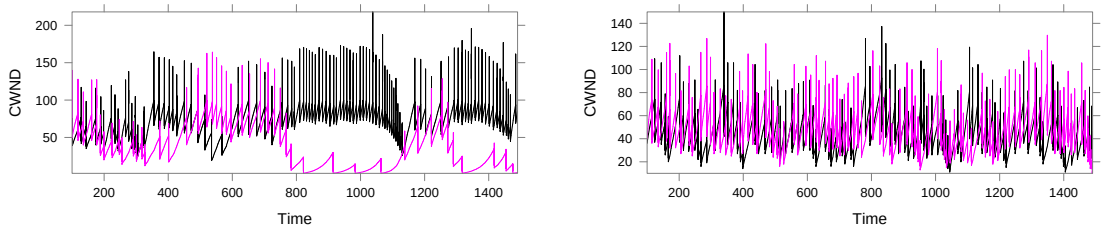
(a) FULLY COUPLED (b) LINKED INCREASES

Figure 9: Flappiness in the Dumbbell Topology, 1 TCP at each bottleneck



(a) FULLY COUPLED (b) LINKED INCREASES

Figure 10: Flappiness in the Dumbbell Topology, 5 TCPs at each bottleneck



(a) FULLY COUPLED (b) LINKED INCREASES

Figure 11: Flappiness in the Fence Topology

plotted the results in Figure 9(b). We find that LINKED INCREASES is not flappy.

To simulate more "exogenous" drops, we ran 5 TCPs at each bottleneck and repeated the two experiments above. The evoluation of cwnd against time is given in Figure 10(a). The results are not surprising: when the multipath flows cannot influence the loss rates too much, FULLY COUPLED congestion control is flappier, while LINKED INCREASES is stable.

Next, we examined flappiness in the Fence topology, with 4 multipath flows competing for 5 links. We plot the evolution of the congestion windows of the middle flows in Fig. 11(a) for FULLY COUPLED and Fig. 11(b) for LINKED INCREASES.

The results show that FULLY COUPLED manages to equalize throughput and loss rates across links (more details are given in §5.2.3), but it does so over a large timeframe; most of the time, the algorithm allocates nearly all of its window to a single path.

This behavior is not surprising: the multipath flows are competing against eachother; when one flow "pushes" harder the other flow will become less aggressive, and these effects will trickle through the network. As opposed to the dumbbell, where the TCPs act as fixed-strength springs that stabilize the multipath flow, the multipath subflows are strings with varying strength. Hence, we expect more flappy behavior than in the dumbbell case.

FULLY COUPLED is flappy, but is flappiness bad? In low statistical multiplexing environments and on short timescales flappiness causes unnecessary variability in throughput to the competing flows. For the multipath flows, it can result in reduced throughput if round trip times on the paths are different.

5.2.2 Fairness and Throughput

Computing a. Theory predicts we can estimate mean

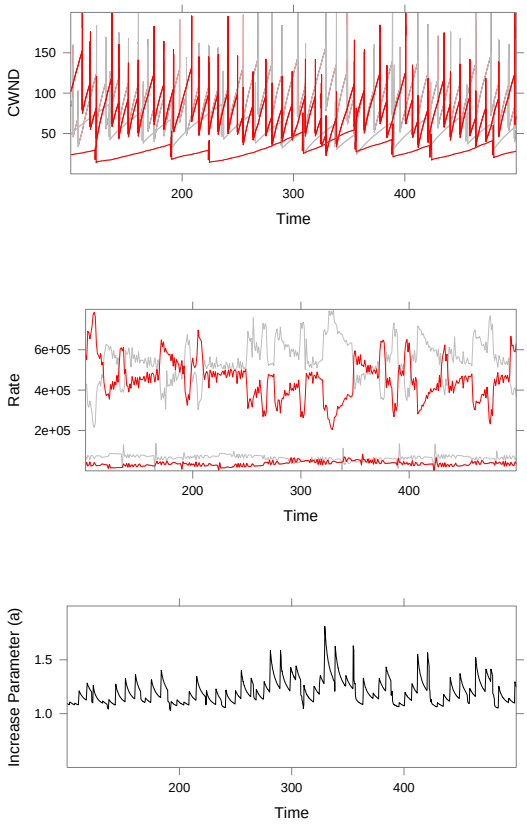


Figure 12: Throughput with Endogenous Drops, Experiment 1

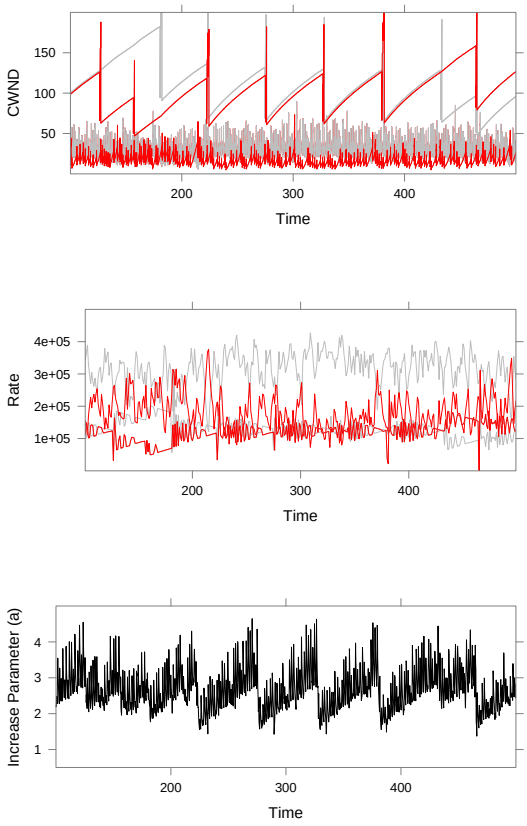


Figure 13: Throughput with Endogenous Drops, Experiment 2

congestion windows and round trip time with their instantaneous values. The obvious way to implement this is to compute a for each ack on each subflow, but this is very expensive.

We chose to compute "a" only when congestion windows grow to accomodate one more packet. We used instantaneous congestion windows and the smoother round trip time estimate, computed similarly to TCP.

We ran one experiment in the dumbbell topology, with the top link having small RTT (100ms) and high capacity (1MB/s) and the bottom link with high RTT (500ms) and low capacity (100KB/s). This experiment corresponds to the left scenario in Figure 6. The results are plotted in Figure 12, using red lines for multipath subflows and gray lines for TCP. The rate was sampled once per second to get a smoothed signal. However, this introduces ocasional aliasing issues in the display, making instantaneous rate wrongly appear higher than the bottleneck.

The average rate of the multipath flow is 500KB/s, the top TCP's rate is 535KB/s and the bottom TCP's rate is 65KB/s. The multipath flow allocates more window to the first subflow, since it has a lower loss rate (1/1256 vs. 1/354).

We plot the evolution of "a" in Figure 12(c) (the value plotted is the real floating point value; the code only maintains the integer approximation of $500a$).

We are interested to see how far the instantaneous value is from a prescient version, where we knew the congestion window and loss rates in advance. For this, we compute the mean round trip times and congestion windows on the two paths in our experiment, sampled when cwnd grows by one packet. This sampling ensures there is no sampling bias when the window is large.

The "perfect" value of a is 1.22, while the average value of a in our run is 1.24. The difference is very small: less than 2%. This supports our choice to compute a based on current values of cwnd and RTT.

We re-ran a similar experiment in the dumbbell topology, increasing the capacity of the second link (params are given in Table 1, row ED2).

The rates for this run are: top TCP 315KB/s, multipath 305KB/s, bottom TCP 130KB/s. The loss rates are 1/354 for link 1, and 1/451 for link 2. In this case, the multipath connection pushes more traffic through the less congested link 2.

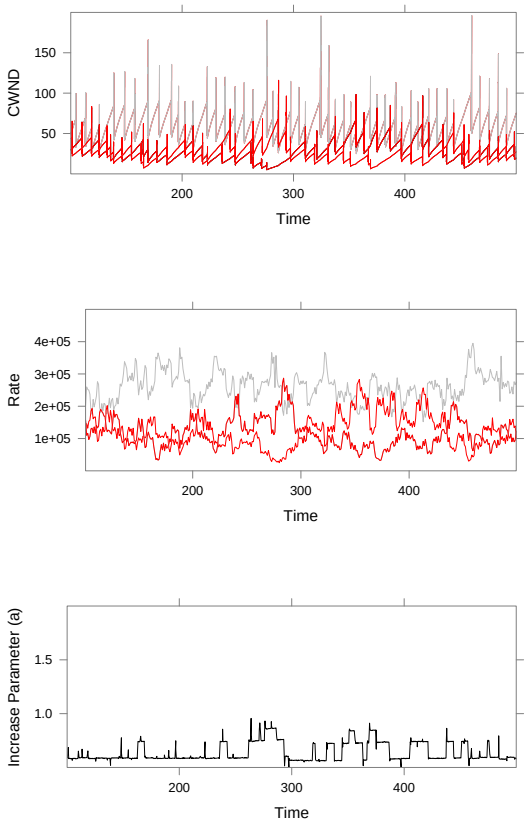


Figure 14: Throughput with Shared Bottleneck, Experiment 1

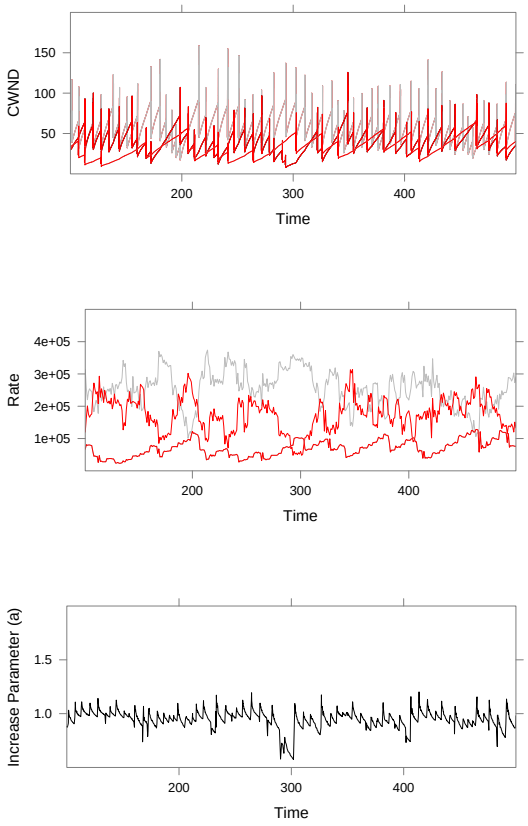


Figure 15: Throughput with Shared Bottleneck, Experiment 2

The details of the run are plotted in Fig. 13. The multipath algorithm sends almost half of its traffic through the bottom link, and its increases are capped to those of TCP to ensure goal 2 is met (this is obvious in Fig. 13(a)).

Because of this, the algorithm is more aggressive: the average value of a is 2.712, while the optimal value is 2.694.

Parameter Exploration. We fix the speed and RTT of link 1 to 400KB/s and 100ms, and vary the speed of link 2 from 400KB/s to 3.2MB/s (8x) and the round trip time from 12ms to 800 ms (1/8 to 8x). We ran one experiment with the dumbbell topology using every combination of parameters. The results are given in the appendix, in Table 3.

Typically, the throughput achieved by the multipath connection varied less than +/-5% from the maximum throughput achieved by the best TCP. Larger deviations were observed when there were many retransmit timeouts , underutilized links (buffers too small), or high RTT links (it takes much longer for the connection to achieve steady rate.)

We also compared the throughput obtained by multipath with the maximum throughput that could be achieved if we ran a single path flow over the best path. The detailed results are given in the appendix in Table 4. On average, multipath increased throughput by 15% and almost always improves throughput compared to single path transport.

Bottleneck Experiments. We ran two experiments using the shared bottleneck topology, with the multipath flow competing against a single TCP flow (for parameters, see Table 2 in the appendix).

In the first experiment, both multipath subflows have the same round trip time (Fig. 14). TCP gets 259KB/s, and multipath gets 241KB/s. The ideal value of a is 0.57, and the average of the real value is 0.66, with an error of roughly 10%. The higher error in computing a is due to the short term variations of RTT around the average; as the RTTs are the same, and a depends on their ratio, the sampling bias of RTT1/RTT2 influences the end result.

In the second experiment, we increase the RTT of one multipath subflow by 250ms. As both subflows have the same loss rate, the windows allocated to each subflow are still roughly equal, but a is increased to 0.88 to compensate. The end result is that the multipath flow gets

245KB/s, and TCP gets 255KB/s.

Summary. Multipath TCP is able to always match the throughput TCP would get on the best of its paths, and is able to satisfy goals 1 and 2 in practice. When loss rates are fixed, multipath gets as much throughput as a single TCP running on the best path would. Adding more subflows increases the pool of paths multipath can choose from.

When multipath influences the loss rates, adding more paths does increase throughput beyond running an independent TCP flow on the best path. This was clear in the multipath dumbbell experiments, where using both paths always gave more throughput than using a single path.

5.2.3 Resource Pooling

We chose the simple 5 link fence topology to test resource pooling. One obvious indicator of resource pooling is the ability of the algorithms to equalize throughput. The second is the ability to spread congestion.

We ran two experiments: one with links of equal capacity and delay, and one where the middle link has half the capacity of the other links but the same delay.

We report the ratio between the rates of the flows for flows 1 and 2 (3 and 4 are mirrored due to the setup), and also the loss rates of links 1, 2, and 3:

	F.COUPLED	LINKED.INC	UNCOUPLED
Rates	1.12	1.33	1.5
link 1	1/5300	1/3500	1/2500
link 2	1/5800	1/2000	1/775
link 3	1/5700	1/1200	1/150

FULLY COUPLED comes closest to equalizing the loss rates and throughputs, LINKED INCREASES is second, and UNCOUPLED TCPS is worse yet.

In the second experiment, we reduce the capacity of the middle link to 200 packets/s. The results are:

	F.COUPLED	LINKED.INC	UNCOUPLED
Rates	1.06	1.72	2.2
link 1	1/3400	1/4500	1/2400
link 2	1/3200	1/2200	1/765
link 3	1/3300	1/950	1/28

Contrasting these numbers with the previous ones, we see that LINKED INCREASES does move a lot of traffic away from the congested link 3 to the other links. UNCOUPLED TCP does not spread congestion at all.

Finally, we ran experiments with more multipath flows in the same topology. Running with 8 or 12 multipath flows equally distributed among the links gives similar results to the ones we have presented, so we skip these here.

5.3 Implementation

We implemented the proposed congestion control algorithm in a user-land port of the Linux 2.3.23 stack. The implementation differs from htsim in the way in maintains the congestion window: packets, rather than bytes. We ran this algorithm on the UCL HEN testbed, using dummynet to shape traffic.

We first used fixed loss rates and round trip times, and compared our implementation to the simulator. The results agree for a range of parameters (though not for loss rates above 5%, when timeouts become significant).

We then ran experiments in which packet drops are caused by queue overflow, i.e. the loss rate is not fixed, rather it varies depending on the traffic that is sent. We used the topology on the left of figure 1, to test how the algorithm behaves when competing with standard TCP flows. We let link 1 be 10Mb/s with delay 10ms, we let link 2 be 2Mb/s with delay 250ms, and we let the return paths have negligible delay. The RTT difference means that link 2 is much less congested, which invites flow A to send most of its traffic on A_2, but we expect RTT compensation to kick in and keep most of A's traffic off link 2 so that flow C does not suffer. We obtained these throughputs, in Mb/s:

	FULLY COUPLED[1]	LINKED INCREASES	UNCOUPLED TCPS
flow B	7.1	5.3	4.8
flow A	3.3	5.4	5.8
flow C	0.6	0.6	0.6

We find that LINKED INCREASES gives flow A the throughput it would have received had it only used link 1; it shifts some of A's traffic onto link 2 and this allows flow B to get slightly better throughput.

Resource pooling. Our next experiments test resource pooling using the topology in figure 2. Each link has capacity 10Mb/s and each path has average RTT 80ms. Over 5 minutes running time, we measured these average throughputs in Mb/s:

	F.COUPLED	LINKED.INC	UNCOUPLED
flow A	13.2	13.2	14.7
flow B	13.5	10.4	8.5
flow C	12.5	14.5	15.0

Under perfect resource pooling the entire 40Mb/s should be shared equally between the flows. We find that FULLY COUPLED almost equalizes the throughputs, LINKED INCREASES does slightly worse and UNCOUPLED TCPS is worse still.

Resource pooling also implies the ability to cope with traffic surges. To simulate the effect of a surge of extra traffic on link 1, we reduced the capacity of link 1 from 10Mb/s to 5Mb/s. The loss rates at link 1 are:

	F.COUPLED	LINKED.INC	UNCOUPLED
link 1=10Mb/s	0.02%	0.03%	0.09%
link 1=5Mb/s	0.05%	0.06%	0.28%

We also tried reducing the capacity of link 2 while fixing the other links at 10Mb/s, resulting in these loss rates at link 2:

[1]The FULLY COUPLED algorithm was not designed to compensate for different RTTs, and moreover the compensation technique we described in §4 cannot be applied to it.

	F.COUPLED	LINKED.INC	UNCOUPLED
link 2=10Mb/s	0.02%	0.13%	0.23%
link 2=5Mb/s	0.05%	0.18%	0.65%

We see that for UNCOUPLED TCPs the loss rate triples when a link experiences a surge, but the other two algorithms are better able to shift traffic away from the congested link so the loss rate merely doubles. (It is not meaningful to compare absolute numbers because UNCOUPLED TCPs has a more aggressive increase than the other two algorithms, so it causes a higher loss rate in all cases.)

6. RELATED WORK

There has been a gap in the literature between theoretical work on multipath congestion control and proposed multipath transport protocols. This work is the first to marry the two: it takes ideas from theory and shows how they can be made work in practice.

6.1 Theoretical work

There have been four proposals for multipath congestion control algorithms: an original proof of concept by Kelly et al. [8], a translation of techniques from optimization and control theory by Wang et al. [13], and algorithms derived from fluid models by Kelly and Voice [7] and Han et al. [4]. In the latter three pieces of work, it was assumed that fluid models are an appropriate description of the system, and the work was to analyse the stability of the fluid models.

We simulated the algorithms suggested by these fluid models, and found surprising behaviour: even when the stability analysis says the system should be stable, the algorithms behaved erratically, flipping from sending almost all traffic on one path to sending almost all traffic on a different path, and the flips were non-periodic. In section we have described this behaviour, and explained why it arises; we further explain how the fluid models should be interpreted in [15]. In practice, we are concerned with the behaviour of an individual flow, not with aggregates; our algorithms address these concerns.

Another issue is that the proposed fluid models are for an Internet in which a user's traffic rates are determined by the congestion he/she sees, whereas in the current Internet it is his/her window size that is determined by congestion, and traffic rates are determined by window size and round trip time. We describe how to adapt the multipath congestion control algorithm so that it plays nicely with todays protocols (or indeed with any other benchmark for fairness that we might set).

6.2 Multipath TCP in practice

There have been many practical attempts to do multipath transport to date [16, 9, 5, 6, **?**]. The majority of these focus on the mechanisms required to implement multipath transmission, such as splitting sequence numbers across paths, attributing losses to the right path, implementing re-ordering, flow control, and so on. There is little consideration for the congestion control aspects of multipath transport. Some of these approaches try to detect a shared bottleneck and to ensure bottleneck fairness; none of them considers resource pooling (Goal 3), and most of them fail to achieve Goal 2 (fairness to other flows).

mTCP [16] performs independent congestion control on each path, and maintains dupacks on each subflows to avoid fast retransmit due to reordering. To detect shared congestion, they test correlation between fast retransmit intervals. There is little analysis to show whether this correlation is sufficient to reliably detect shared bottlenecks in all cases.

R-MTP [9] targets wireless links in particular, probes bandwidth available for each subflow periodically and adjusts rate accordingly. To detect congestion uses packet interarrival times, and jitter, and associates increased jitter with mounting congestion. This solution only works for cases where wireless links are the bottleneck.

pTCP [5] is another solution where independent TCP Reno congestion control is applied on each path. As R-MTP, it assume wireless is the bottleneck to ensure TCP-friendliness.

Concurrent Transfer Multipath over SCTP (or CMT) [6] performs independent TCP Reno congestion control on each path, and does not consider bottleneck fairness.

6.3 Layer 3 Multipath

Multipath routing can be implemented underneath the transport layer too. In this case, at transport layer there is a single connection (with its congestion control and reliability mechanisms), but on the wire packets will take different paths.

ECMP (equal cost multipath) is a solution used today to balance traffic across multiple paths inside an ISP's network [12]. Splitting a transport flow across multiple paths is deemed as bad practice for ECMP, because packet re-ordering causes unnecesarry retransmissions at the transport layer, and significantly reduces throughput. Consequently, entire flows are "routed" on the same path, and load balancing is achieved in aggregate.

However, in a few particular scenarios network layer multipath can improve performance, even when the transport is unmodified TCP. Horizon [10] is one approach where multipath transmission is used below the transport layer in mesh wireless networks. In Horizon, the network nodes maintain congestion state information for each possible path towards the destination, and hop-by-hop backpressure is applied to achieve near optimal throughput.

6.4 Application Layer Multipath

BitTorrent [1] is a succesful example of application layer multipath, where different chunks of the 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. Since the receiving application does not need the chunks to arrive in order, download order does not matter at all. BitTorrent runs independent congestion control on each path, and achieves job level resource pooling in many-to-one transfers. Multipath congestion control could be used to "link" congestion control for BitTorrent, allowing it to achieve rate-level resource pooling.

7. CONCLUSIONS

Starting from the three goals of *improve throughput*, *do no harm*, and *balance congestion* from section 1, we derived a practical multipath congestion control algorithm suitable for deployment on today's Internet in a protocol such as Multipath TCP[3]. The particular mechanism we derived is not the only possible solution; indeed our choice to link only the increases in the TCP algorithm is a compromise between the undesirable flappy behaviour of a fully linked algorithm and an uncoupled subflows algorithm which would not pool network resources to relieve congestion. However, once this choice was made, we have shown how the goals themselves constrain the desired equilibrium balance between the traffic on the multiple subflows, and hence lead naturally to a single solution for determining the additive increase constant. This solution was evaluated in simulation and a full TCP implementation, and does satisfy the goals.

Other variants of our algorithm are worth investigation though. Our current solution results in $w_r \propto 1/p_r$, which provides better resource pooling than unlinked flows, where $w_r \propto 1/\sqrt{p_r}$. It might be that an algorithm that gives $w_r \propto 1/p_r^\kappa$ might give even better resource pooling, with $\kappa > 1$, and our analysis of how to set the additive increase constant carries through almost unchanged. However, it is an open question whether this can be achieved without causing excessive flappiness.

Finally, although our algorithm achieves TCP fairness, we do not believe that current TCP dynamics are anywhere near optimal. The approach we have taken is broadly applicable, and could be used to determine how a multipath variant of an improved congestion control algorithm should behave.

APPENDIX

A. SIMULATION PARAMETERS

Unless specified otherwise, in all the experiments we set buffers on the bottleneck link to be the delay-bandwidth product of that link.

Exp. ID	Link1	Link 2	CC	TCP Count
FC1	500KB/s 100ms	500KB/s 100ms	FULLY COUPLED	1/1
FC2	500KB/s 100ms	500KB/s 100ms	FULLY COUPLED	5/5
LI1	500KB/s 100ms	500KB/s 100ms	LINKED INCREASES	1/1
LI2	500KB/s 100ms	500KB/s 100ms	LINKED INCREASES	5/5
ED1	1MB/s 100ms	100KB/s 500ms	LINKED INCREASES	1/1
ED2	500KB/s 50ms	250KB/s 500ms	LINKED INCREASES	1/1

Table 1: Parameters for Experiments in the Multipath Dumbbell Topology

Exp. ID	Link	RTT2	CC	TCP Count
BN1	500KB/s 50ms	0ms	LINKED INCREASES	1
BN2	500KB/s 50ms	250ms	LINKED INCREASES	1

Table 2: Parameters for Experiments in the Bottleneck Topology

RTT2 — BW2	400KB/s	800KB/s	1.6MB/s	3.2MB/s
12ms	-6	-6	-11	1
25ms	-9	-9	-2	0
50ms	-1	-4	1	-1
100ms	6	1	-4	3
200ms	1	-1	4	-4
400ms	2	13	1	1
800ms	-4	9	40	-8

Table 3: Throughput Comparison with competing TCP

RTT2 — BW2	400KB/s	800KB/s	1.6MB/s	3.2MB/s
12ms	59	18	3	6
25ms	43	17	9	4
50ms	40	17	8	3
100ms	40	16	3	5
200ms	43	10	6	-1
400ms	49	14	0	1
800ms	44	6	18	-24

Table 4: Throughput Comparison with TCP on Best Path

Parameters for experiments run on the dumbbell topology are given in Table 1. Parameters for experiments run on the bottleneck topology are in Table 2.

B. SENSITIVITY ANALYSIS

Parameters for experiments run on the bottleneck topology, with link 1 running at 400KB/s and delay 100ms, are given in Table 3.

We compared multipath throughput to that of running a single TCP on the best path (i.e. getting half of the bigger bottleneck link). The relative throughput increases are given in Table 4.

The bottom right result is weird: in this case, the RTT is so high that the real TCP gets much less than half of the links bandwidth (i.e. the link is underutilized).

C. REFERENCES

[1] B. Cohen. Incentives build robustness in bittorrent, 2003.

[2] S. Floyd and V. Jacobson. Traffic phase effects in packet-switched gateways. *SIGCOMM Comput. Commun. Rev.*, 21(2):26–42, 1991.

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

[4] 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.

[5] H.-Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 83–94, New York, NY, USA, 2002. ACM.

[6] 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.

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

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

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

[10] B. Radunović, C. Gkantsidis, D. Gunawardena, and P. Key. Horizon: balancing tcp over multiple paths in wireless mesh network. In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 247–258, New York, NY, USA, 2008. ACM.

[11] C. Raiciu, M. Handley, and D. Wischik. Coupled Multipath-Aware Congestion Control. Internet-draft, IETF, 2009.

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

[13] W.-H. Wang, M. Palaniswami, and S. H. Low. Optimal flow control and routing in multi-path networks. *Perform. Eval.*, 52(2-3):119–132, 2003.

[14] D. Wischik, M. Handley, and M. B. Braun. The resource pooling principle. *CCR*, 38(5), 2008.

[15] D. Wischik, M. Handley, and C. Raiciu. Control of multipath tcp and optimization of multipath routing in the internet. In *NET-COOP*, pages 204–218, 2009.

[16] 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 *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 8–8, Berkeley, CA, USA, 2004. USENIX Association.