

Linked Congestion Control

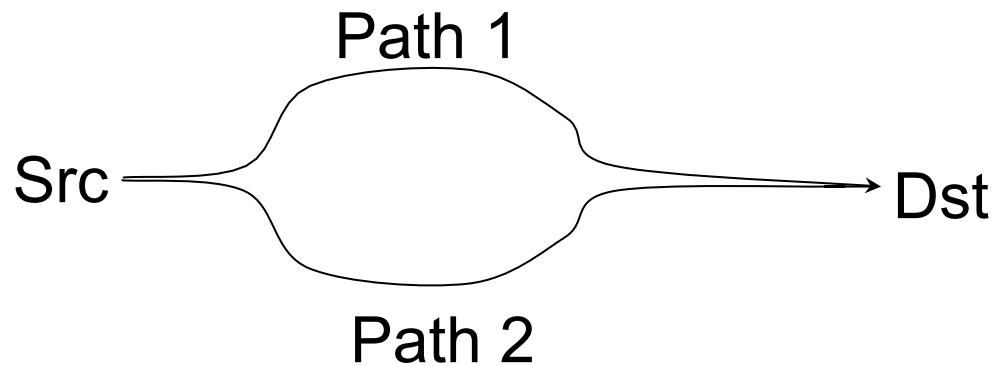
Costin Raiciu, Damon Wischik, Mark Handley
UCL

Where we are today

- We've studied practical aspects of multipath congestion control for 1.5 years
 - Solved issues with previous theoretical work (flappiness, RTT bias)
- Linked Increases algorithm
 - draft-raiciu-mptcp-congestion-00
 - Detailed analysis in tech report

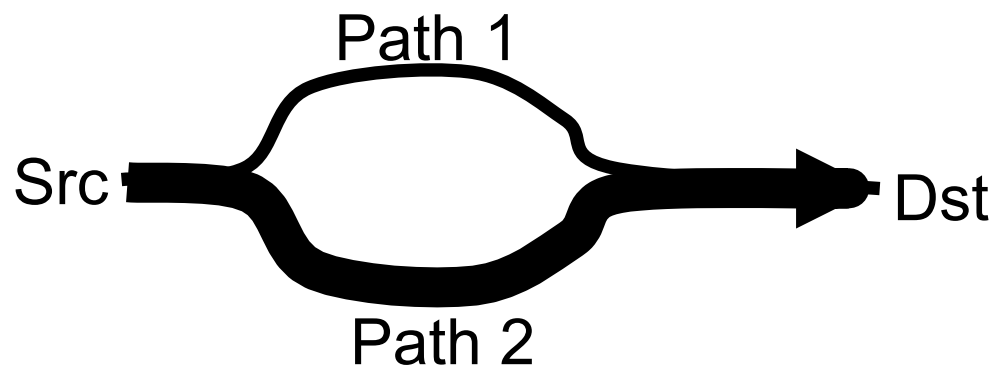
Multipath TCP at work

- Source can use multiple paths to send traffic
- How should it allocate traffic to the two paths?
 - Using a window based protocol
 - Playing fair with TCP



Multipath TCP at work

- Source can use multiple paths to send traffic
- How should it allocate traffic to the two paths?
 - Using a window based protocol
 - Playing fair with TCP



Aims

- **Goal 1** (improve throughput): when compared to using the best single path

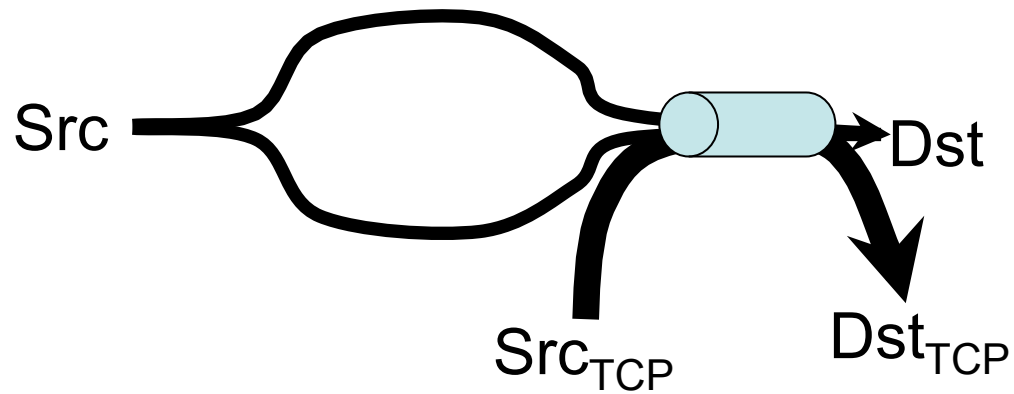
Aims

- **Goal 1** (improve throughput): when compared to using the best single path
- **Goal 2** (do no harm): on any available path, take at most the same throughput a single TCP would

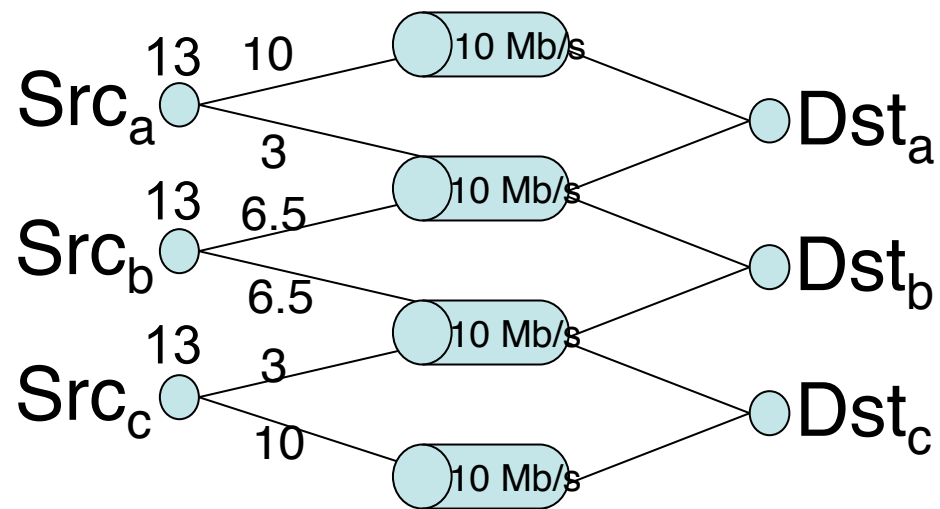
Aims

- **Goal 1** (improve throughput): when compared to using the best single path
- **Goal 2** (do no harm): on any available path, take at most the same throughput a single TCP would
- **Goal 3** (balance congestion) move traffic onto least congested links as long as goals 1 and 2 are met

Goals 1&2 Imply Bottleneck Fairness



Goal 3 Implies Resource Pooling



Can we use existing algorithms?

- Independent TCP on each subflow
 - Breaks goals 2 and 3
- Theoretical work (Kelly et al)
 - Flappy – tends to allocate all traffic to a single subflow
 - Breaks goal 1 due to RTT dependence

Linked Increases Algorithm

- Preserves the basic window-based AIMD behavior that has kept the Internet running for ~20years
- Tweaks the increase phase

What is changing?

- We only change behavior in congestion avoidance phase
- All other algorithms are unchanged, and will run independently per subflow
 - Slow start
 - Fast retransmit/fast recovery
 - SACK
 - RTT estimation
 - ...

Linked Increases Algorithm

- On each ack, TCP NewReno increases window by:

$$\frac{\text{bytes_acked} \cdot \text{mss}}{\text{cwnd}}$$

Linked Increases Algorithm

- On each ack, TCP NewReno increases window by:

$$\frac{bytes_acked \cdot mss}{cwnd}$$

- On each ack on path i , increase $cwnd_i$ by

$$\frac{\alpha \cdot bytes_acked \cdot mss_i}{tot_cwnd}$$

Linked Increases Algorithm

- On each ack, TCP NewReno increases window by:

$$\frac{\text{bytes_acked} \cdot \text{mss}}{\text{cwnd}}$$

- On each ack on path i , increase cwnd_i by

$$\frac{\alpha \cdot \text{bytes_acked}_i \cdot \text{mss}_i}{\text{tot_cwnd}}$$

Linked Increases Algorithm

- On each ack, TCP NewReno increases window by:

$$\frac{bytes_acked \cdot mss}{cwnd}$$

- On each ack on path i , increase $cwnd_i$ by

$$\frac{\alpha \cdot bytes_acked \cdot mss_i}{tot_cwnd}$$

Tuning α

- We know loss rates and rtt's
 - We know the throughput a TCP would get on the best path
 - We can compute α by solving a simple equation

Tuning α

- Formula

$$\alpha = tot_cwnd \frac{\max_i \frac{cwnd_i \cdot mss_i^2}{rtt_i^2}}{\left(\sum_i \frac{cwnd_i \cdot mss_i}{rtt_i} \right)^2}$$

Is this practical?

- Compute α only when cwnd grows by one mss
 - Gives good precision at low cost
- We can do all operations with integers

Capping Increases

- α can be arbitrarily large
- On certain paths this may make multipath subflows be more aggressive than TCP

Capping Increases

- α can be arbitrarily large
- On certain paths this may make multipath subflows be more aggressive than TCP
- To avoid this, just cap!

$$\min\left(\frac{\alpha \cdot \text{bytes_acked} \cdot \text{mss}_i}{\text{tot_cwnd}}, \frac{\text{bytes_acked} \cdot \text{mss}_i}{\text{cwnd}_i}\right)$$

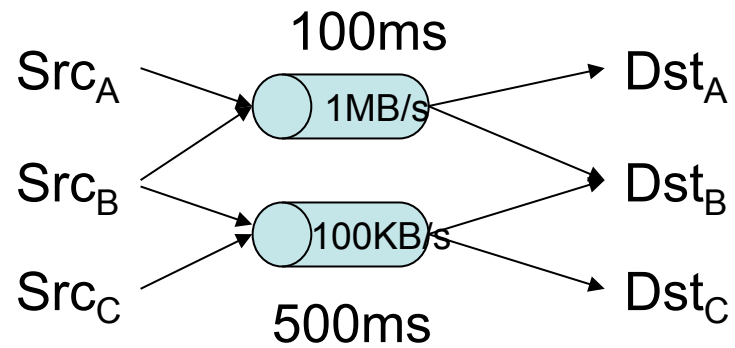
Emergent Behavior

- Linking the increase allocates proportionally more window to subflows with lower loss rates
- Tuning α scales the total window such that the desired throughput is achieved

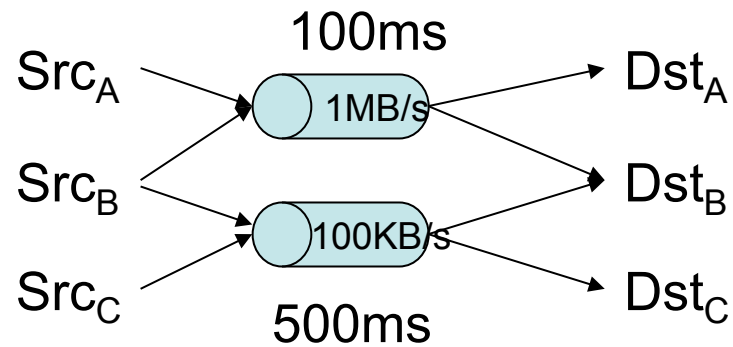
Linked Increases Implementations

- Implemented in
 - Simple cwnd simulator, RTT based
 - Packet-level simulator [available soon]
 - Userland Linux stack [available on demand]
- Ran extensive experiments
 - Linked Increases gets throughput within +/-10% of best TCP

Experiment: Throughput



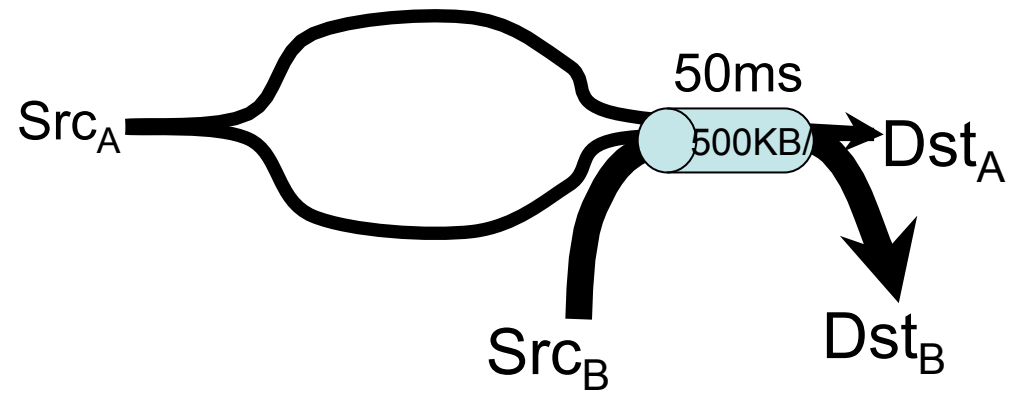
Experiment: Throughput



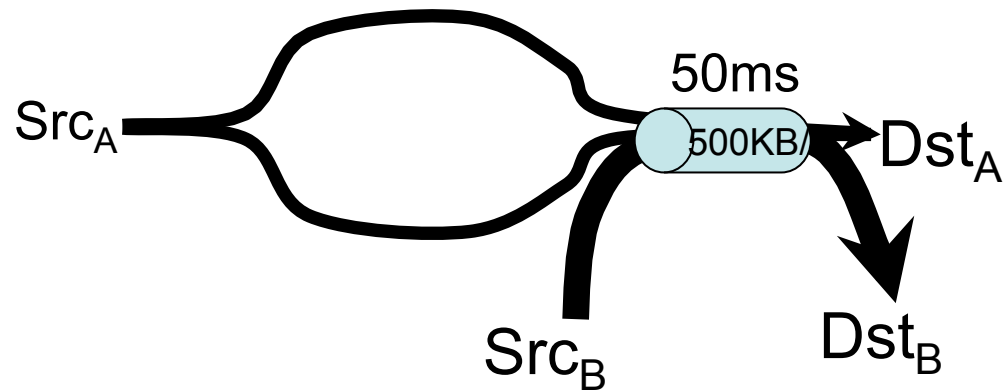
Throughput:

- Src_A 520KB/s
- Src_B 510KB/s
- Src_C 71KB/s

Experiment: Bottleneck



Experiment: Bottleneck



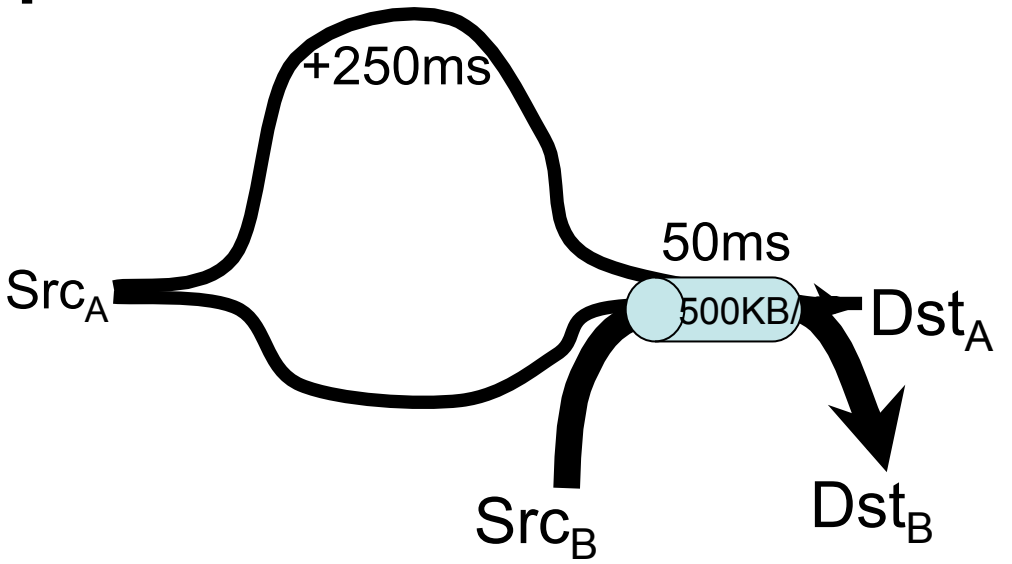
$$cwnd_1 = cwnd_2$$

$$\alpha = 0.66$$

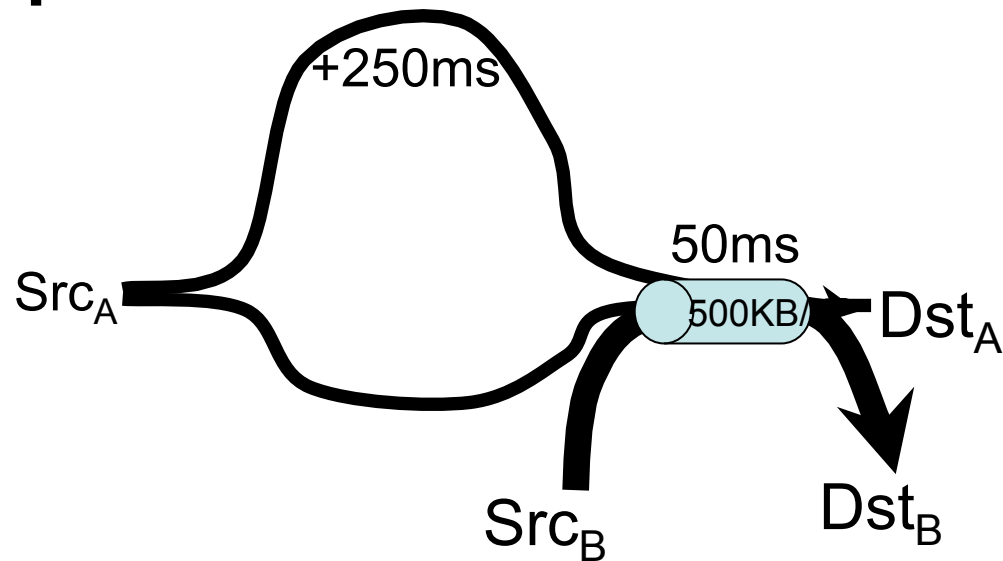
Throughput:

Src_A	240KB/s
Src_B	260KB/s

Experiment: Bottleneck [2]



Experiment: Bottleneck [2]

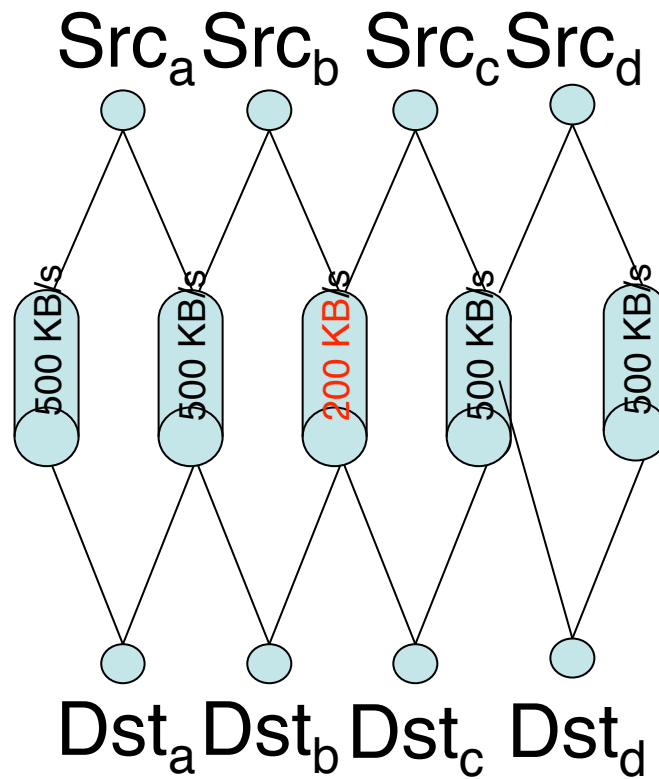


$$cwnd_1 = cwnd_2$$

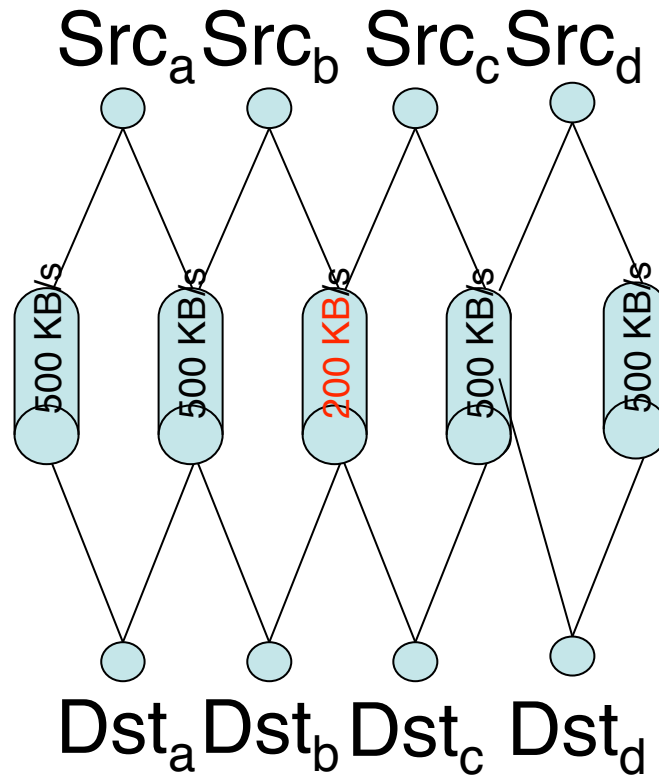
$$\alpha = 0.89$$

Result: Src_A 255KB/s
Src_B 245KB/s

Resource Pooling Experiment



Resource Pooling Experiment



Throughput (KB/s):	625	475	475	625
Uncoupled TCP:	750	350	350	750
Perfect:	550	550	550	550

Summary

- We must couple congestion control loops to get resource pooling and bottleneck fairness
- Linked Increases [draft-raiciu-mptcp-congestion-00] achieves both
 - Simple and works
 - We have a working implementation
- Is this draft ready to become a working group document?