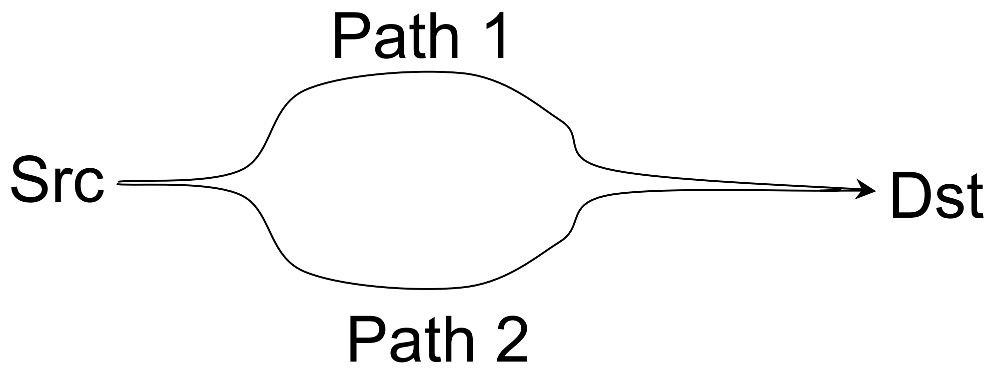
Linked Congestion Control

Costin Raiciu, UCL

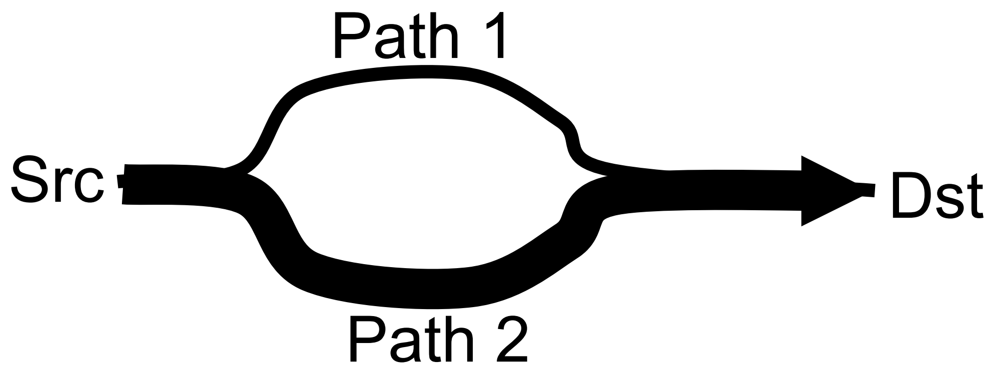
Multipath TCP at work

- Source can use both 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 both 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

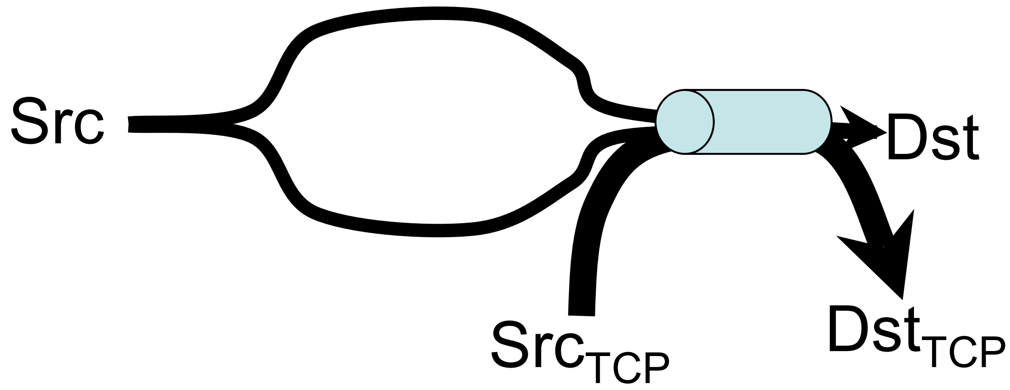
Aims

- **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

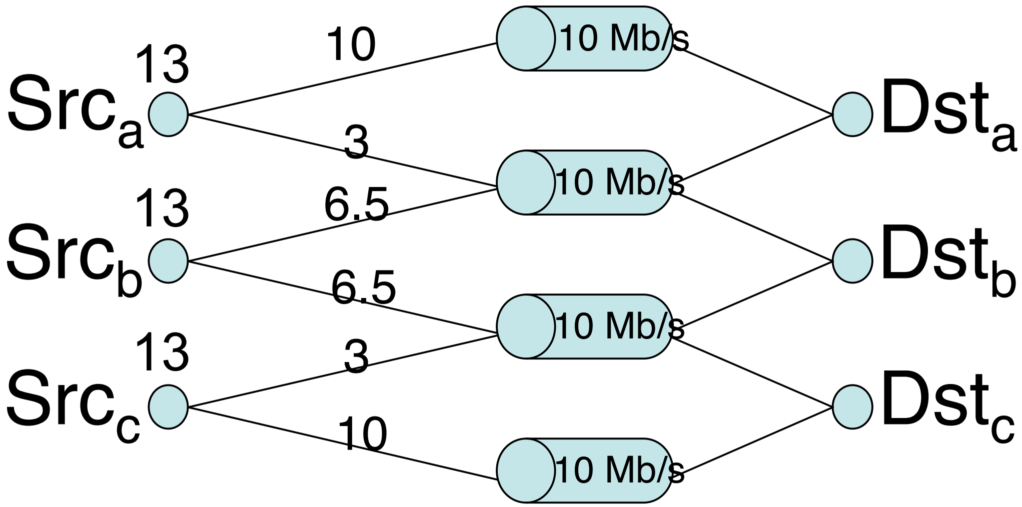
Aims

- **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



This Talk

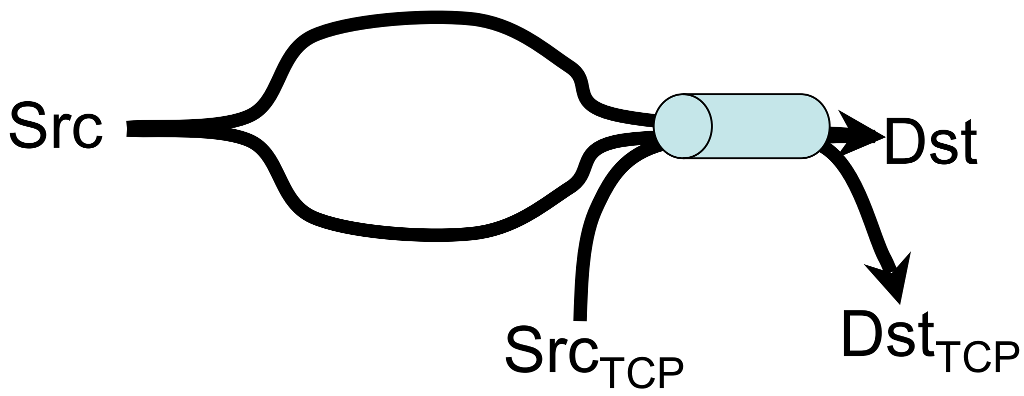
- Show that goals can be met
- Present a simple, safe, deployable protocol
  – Achieves reasonable resource pooling
- There are probably other solutions that
  – Get better resource pooling
  – Are possibly safe to deploy
  – We just don’t know them yet

Default

- Use independent TCP CC on each path

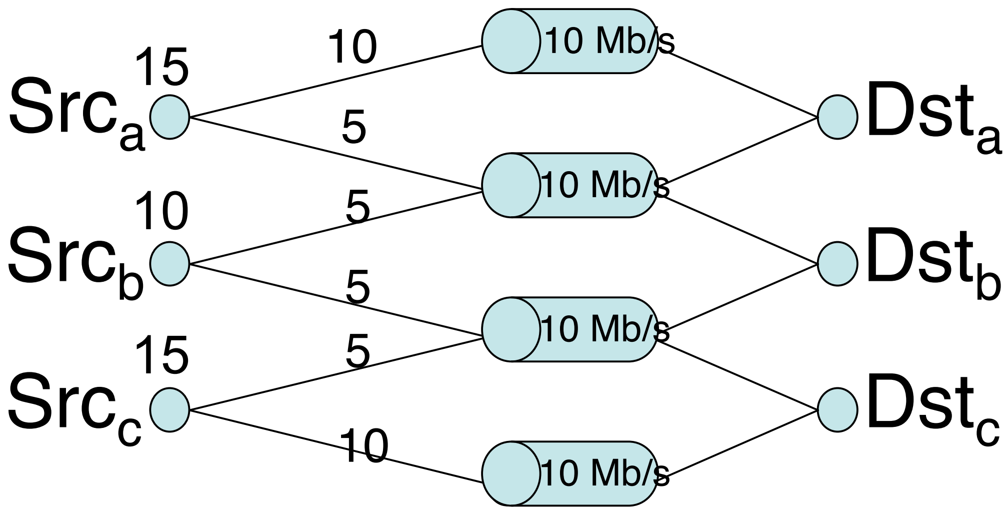
Default

- Use independent TCP CC on each path
- Problem: bottleneck fairness



Default

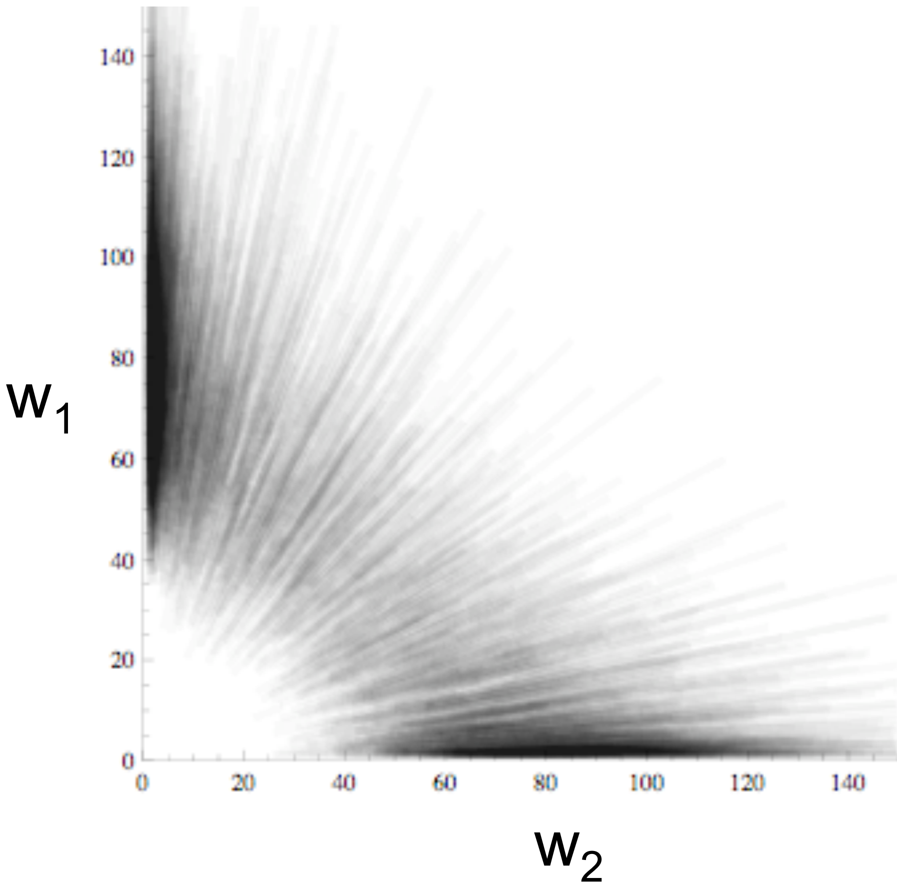
- Use independent TCP CC on each path
- Problem: bottleneck fairness
- Problem: resource pooling



Solution:
Couple Congestion Controllers

- w_r - congestion window on subflow r
- $w = sum(w_r)$
- **Fully Coupled** algorithm
  - Increase w_r by 1 / w per ack on subflow r
  - Decrease w_r by w / 2 per drop on subflow r
- Behaves like a single TCP

Fully Coupled is Flappy

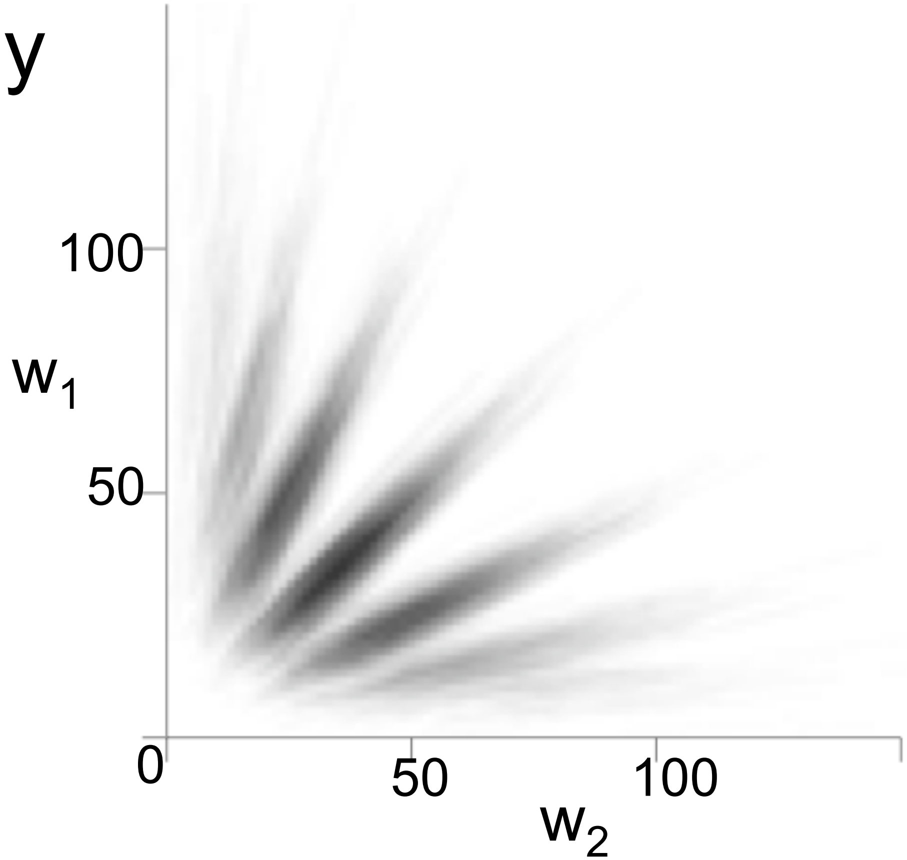


Better Solution

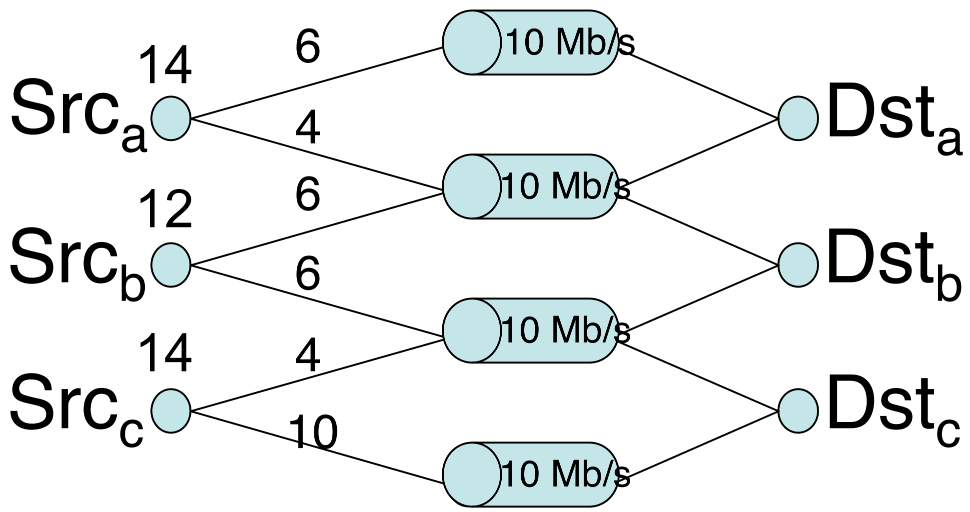
- **Linked Increases** Algorithm
  - Increase w_r with a / w for each ack on subflow r
  - Decrease w_r by w_r / 2 for each drop on subflow r
- **a** is a parameter that controls aggressiveness

Linked Increases

- Not Flappy

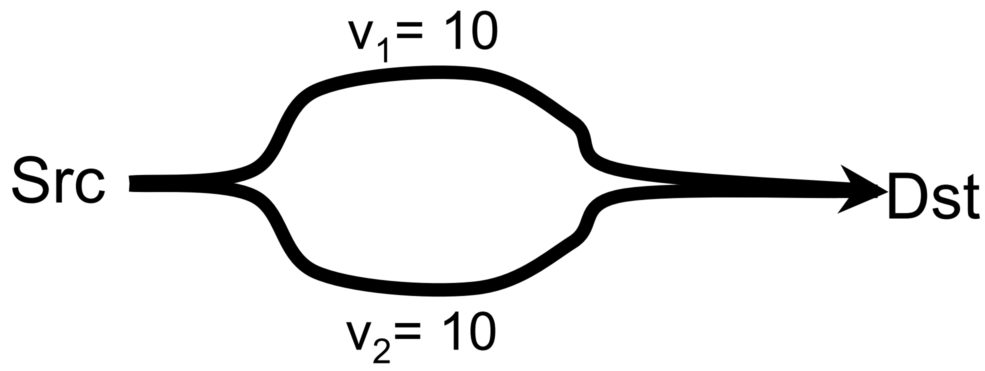


Resource Pooling of Linked Increases Algorithm



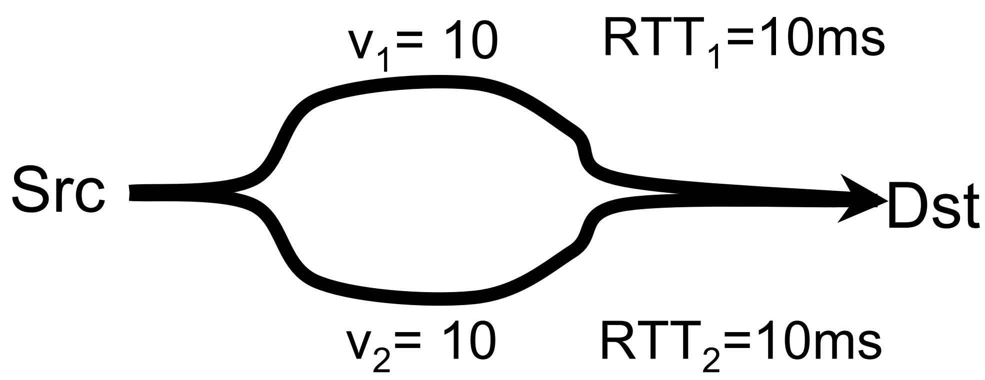
Effect of RTT

- Assume equal drop rates: $p_1 = p_2$



Equal RTTs

- Assume equal drop rates: $p_1=p_2$



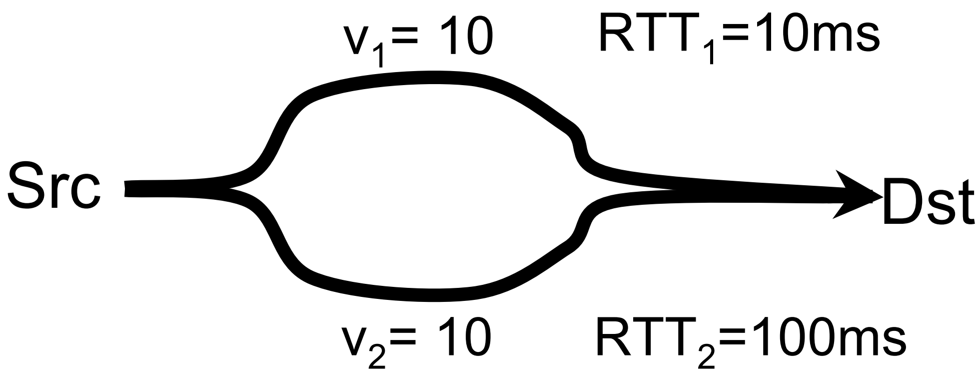
=> 1000pkts/s

=> 1000pkts/s

Rate = 2000 pkts/s

Dissimilar RTTs

- Assume equal drop rates: $p_1 = p_2$



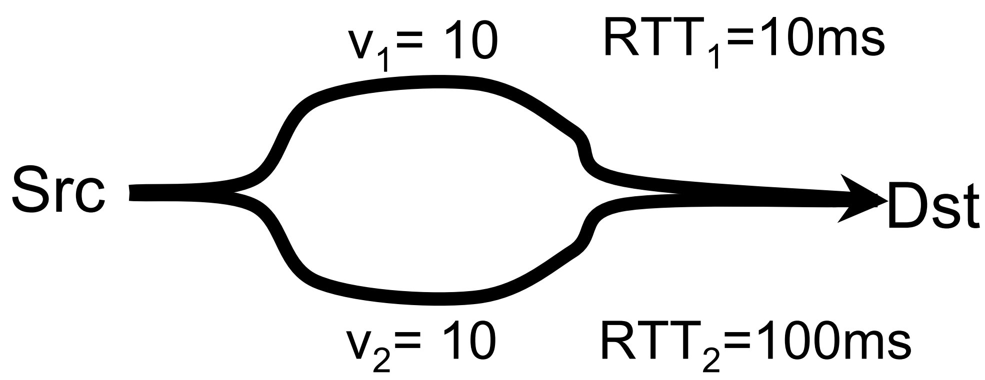
=> 1000pkts/s

=> 100pkts/s

Rate = 1100 pkts/s

Dissimilar RTTs

- Assume equal drop rates: p_1=p_2

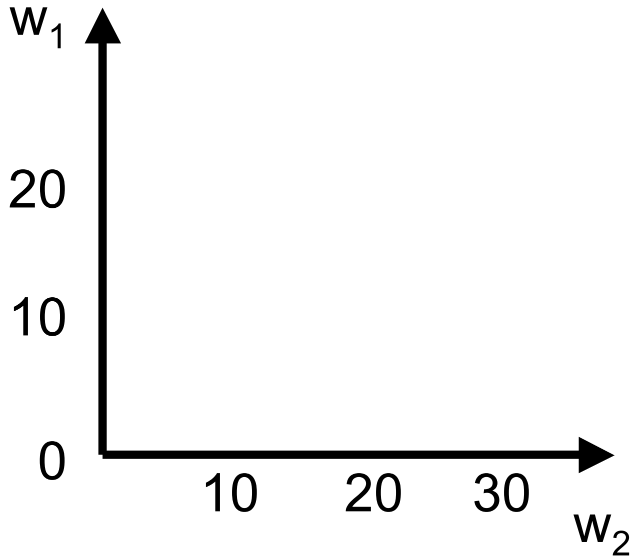


Rate = 1100 pkts/s

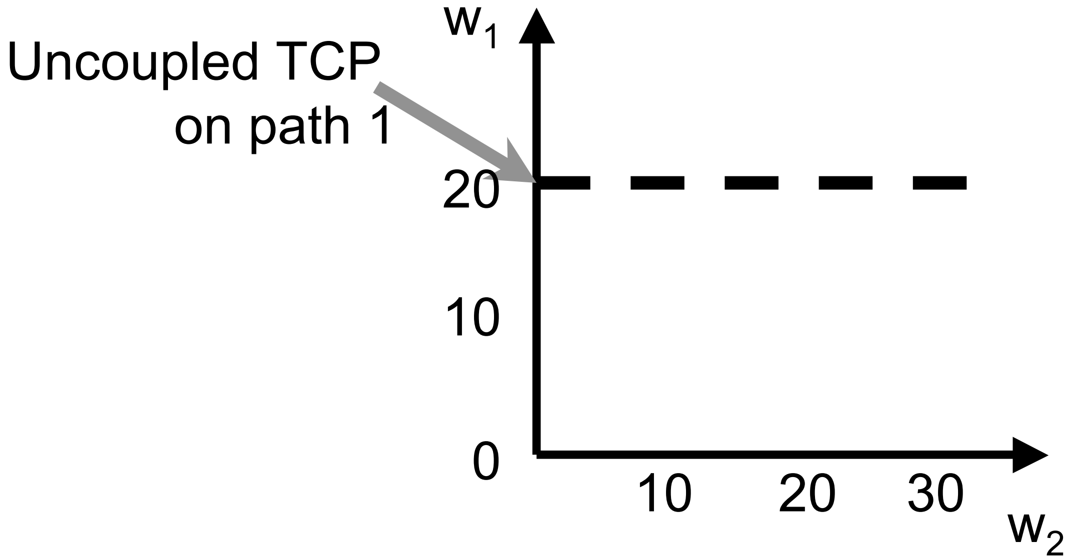
A TCP on path 1 would get 2000pkts/s
Multipath is doing worse!

Dissimilar RTTs

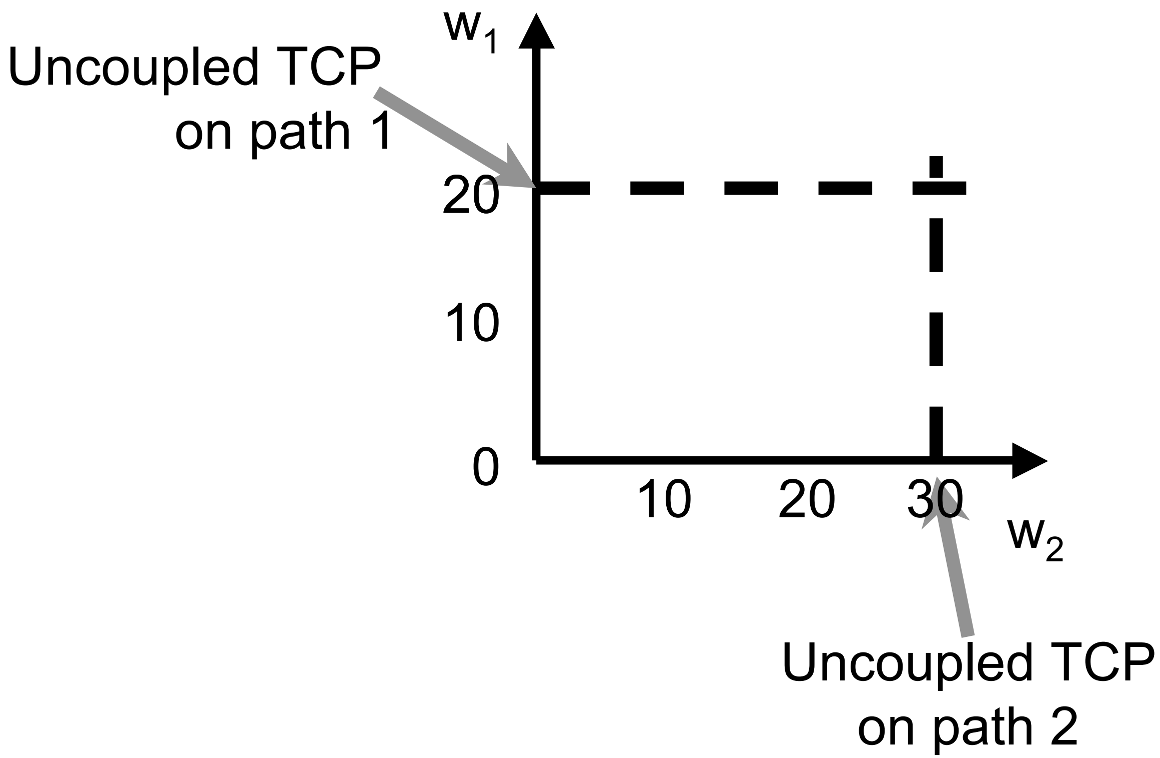
Assume $p_1 > p_2$, so $w_1 < w_2$



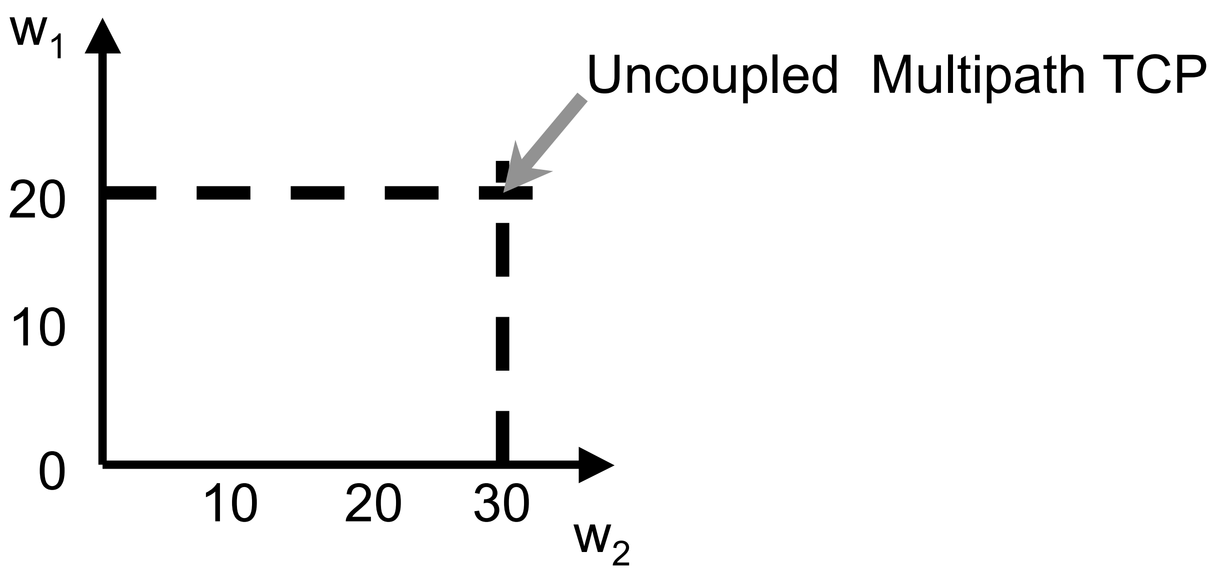
Dissimilar RTTs



Dissimilar RTTs

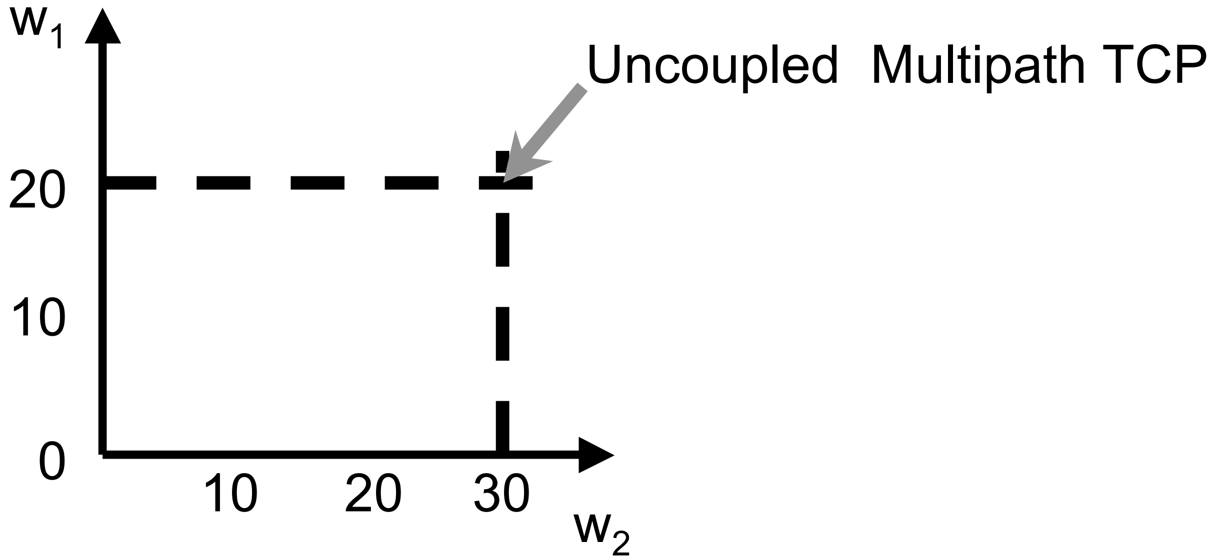


Dissimilar RTTs

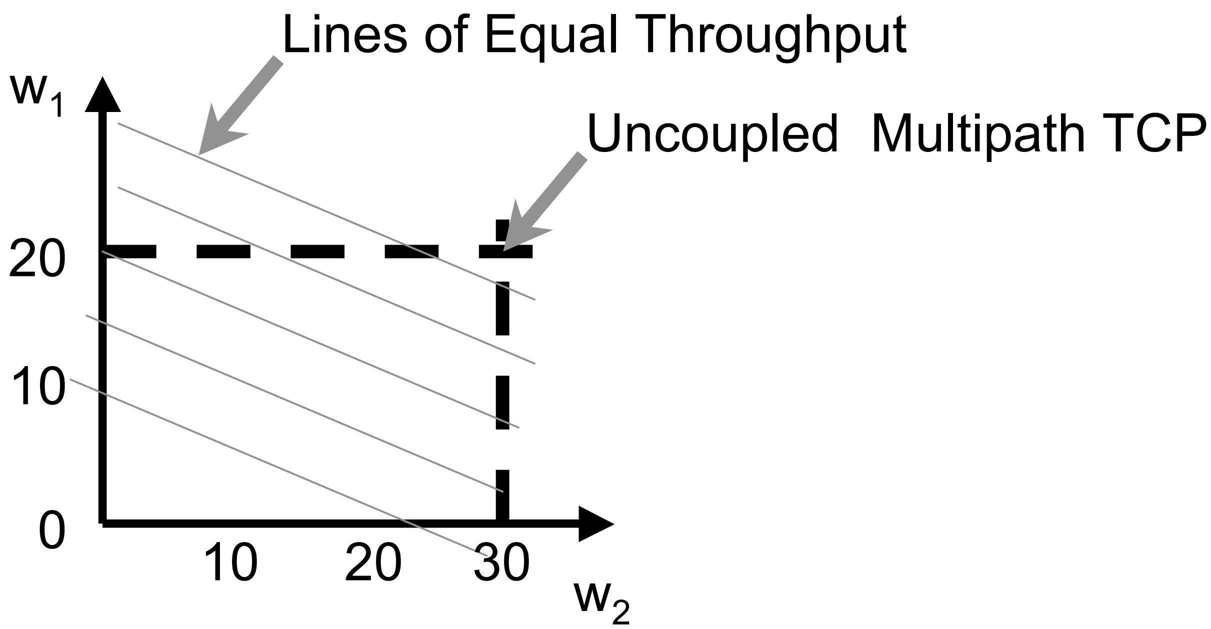


Dissimilar RTTs

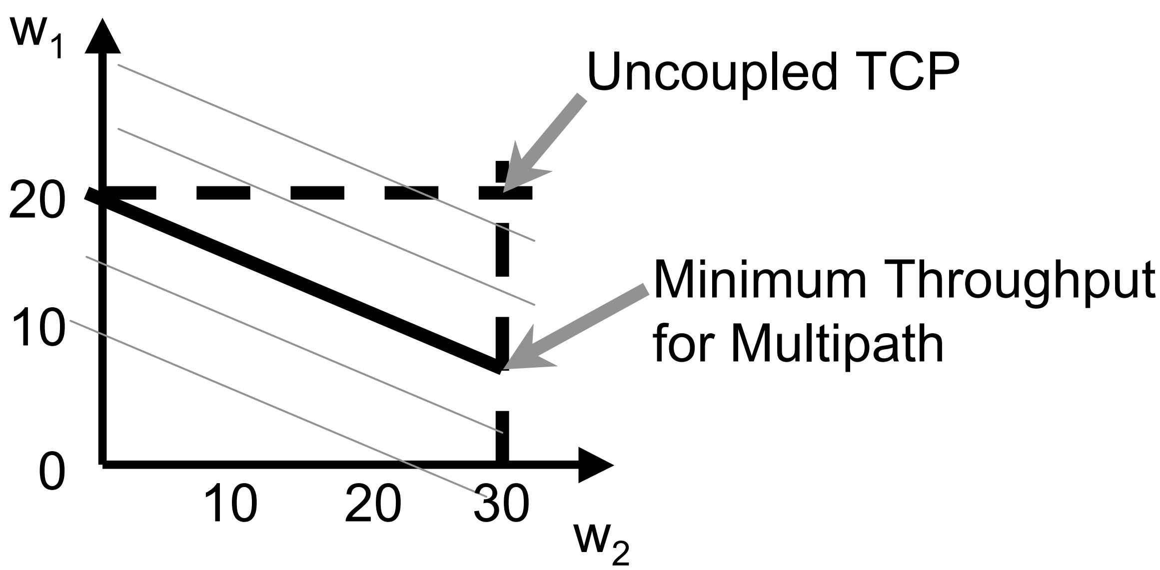
Assume $RTT_1 < RTT_2$, and $TCP_1 > TCP_2$



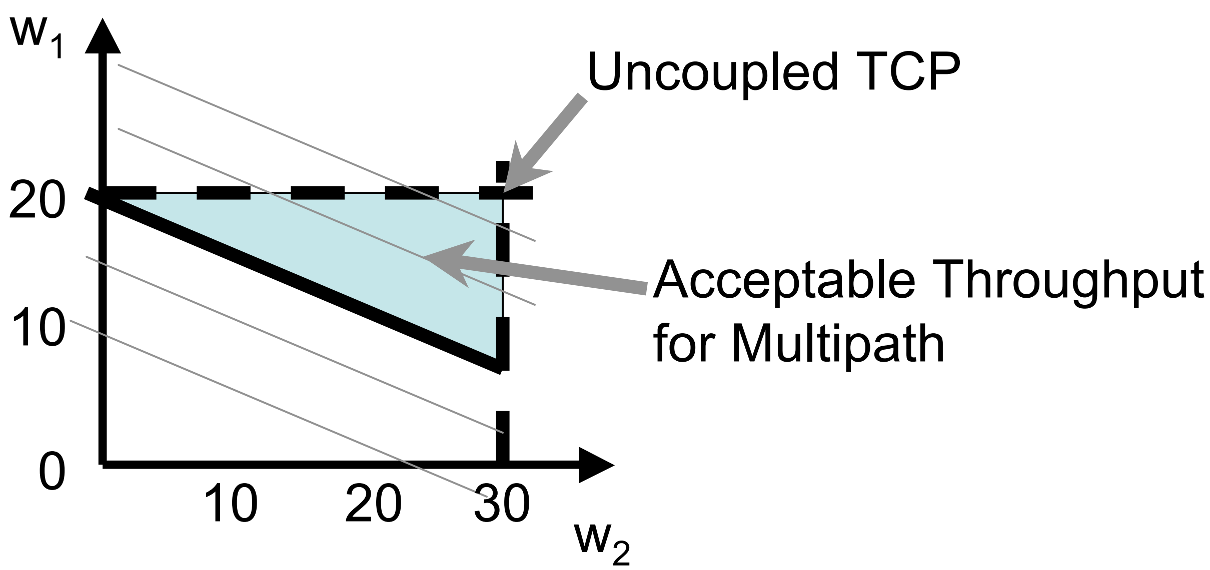
Dissimilar RTTs



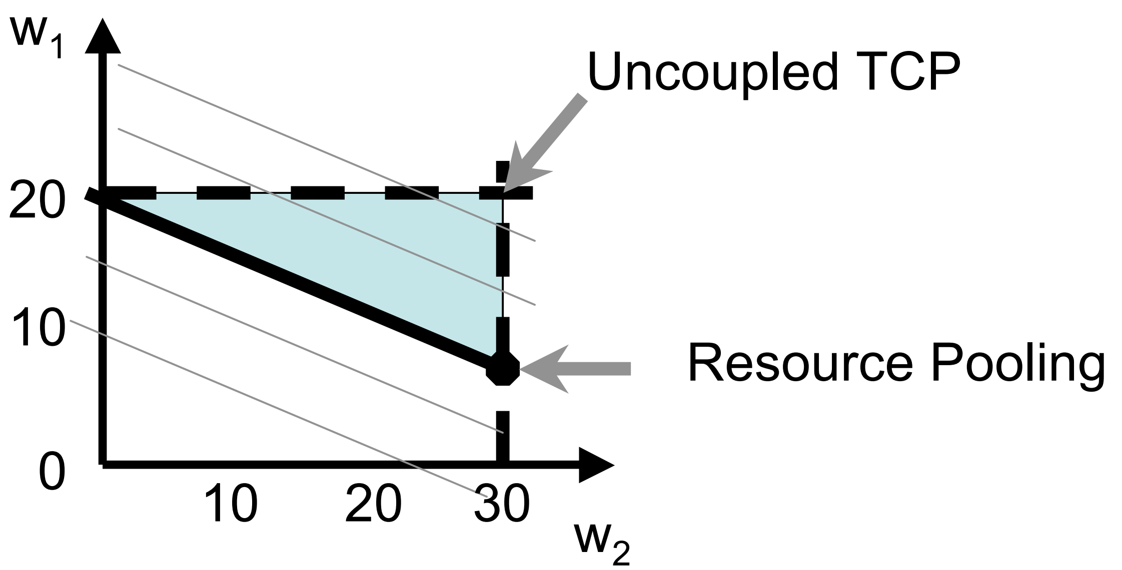
Dissimilar RTTs



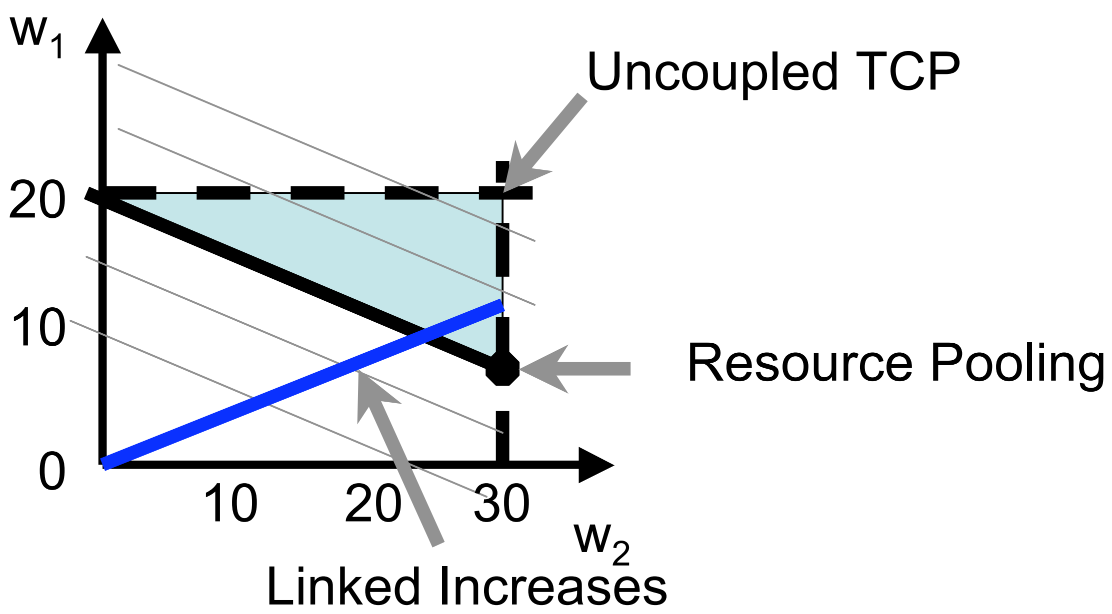
Dissimilar RTTs



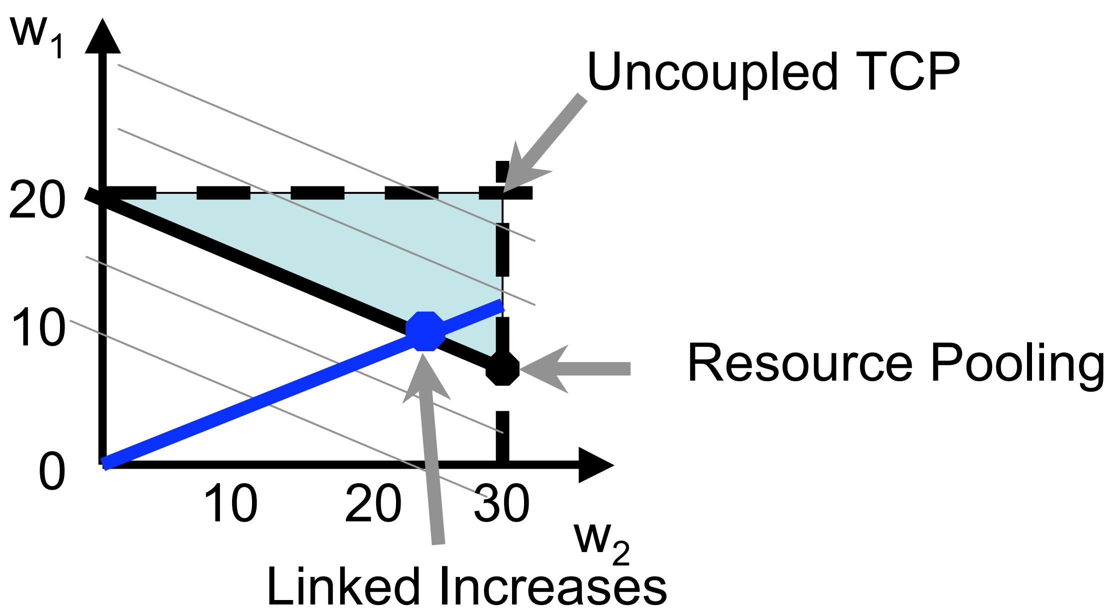
Dissimilar RTTs



Dissimilar RTTs

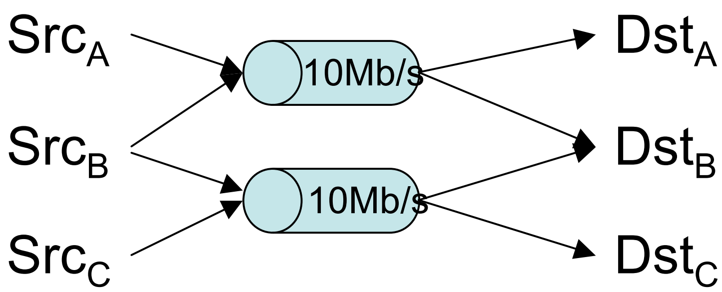


Dissimilar RTTs



It Works

- Experiment



- Results (Mb/s)

	Coupled	Linked Inc.	Uncoupled
Src_A	7.1	5.3	4.8
Src_B	3.3	5.4	5.8
Src_C	0.6	0.6	0.6

Conclusions

- We must couple congestion control loops to get resource pooling and bottleneck fairness
- It is not hard to do so
  - Must remove flappiness
  - Must take into account RTT fairness
- Our proposal
  - Simple and works
  - We have a working implementation
- Other solutions possible

It Works

- Simulation Run: p_1=p_2=1/1000, 5 RTT_1=RTT_2