Congestion Control for High Bandwidth-Delay Product Networks

XCP, the Explicit Control Protocol
SIGCOMM '02
by Diana Katabi, MIT-LCS, dk@mit.edu,
Mark Handley, ICSI, mjh@icsi.berkeley.edu
Charlie Rohrs, Tellabs, crhors@mit.edu

Presented by Guillaume Marceau for WPI CS577
on September 28, 2009

TCP is blind

TCP tries to discover the amount of bandwidth available, but it is blind
Packet drop are the only feedback, which ...

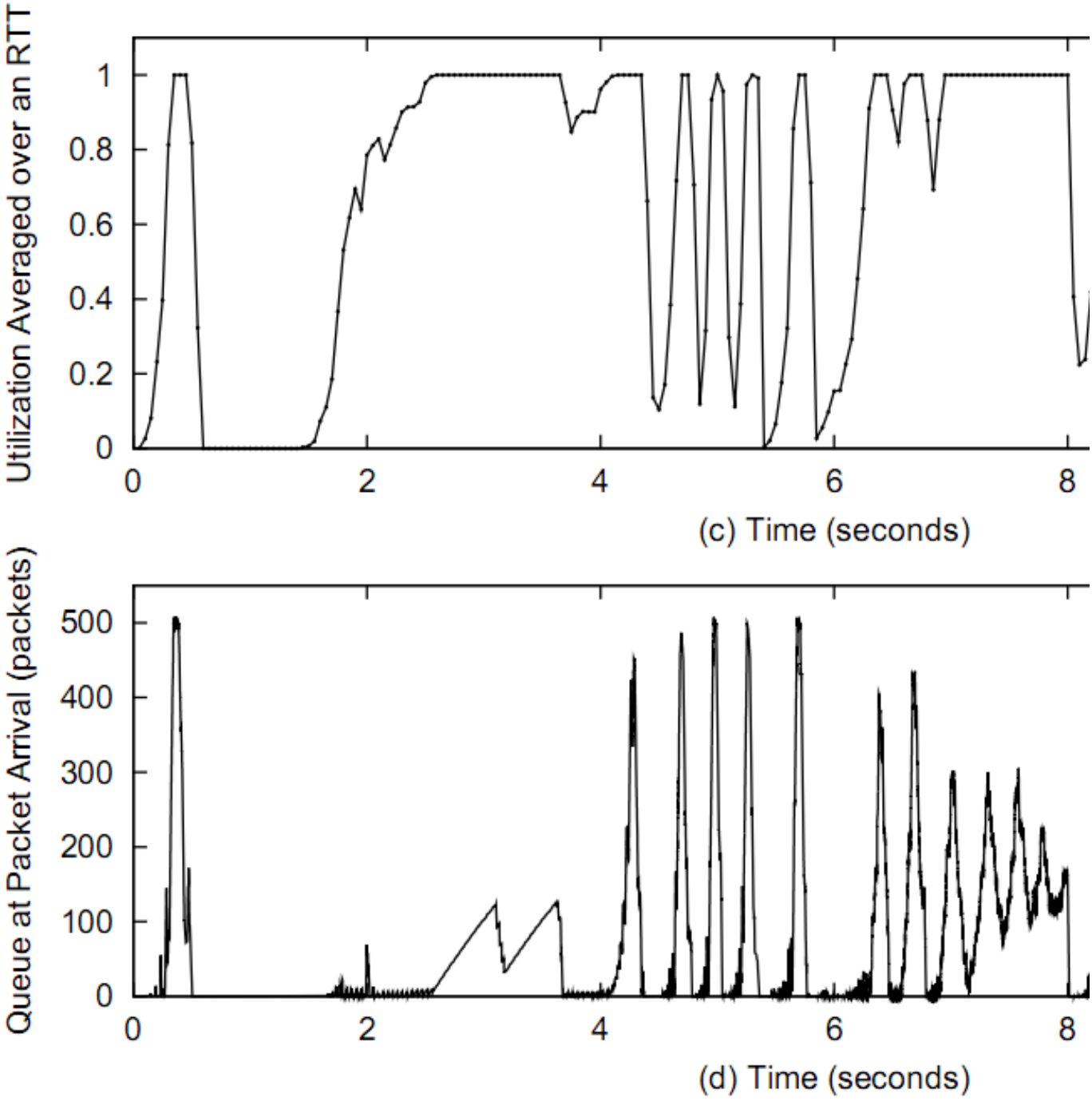
- ... is binary (drop/no drop)
- ... is ambiguous (caused by congestion or by faults)
- ... is delayed (must wait a full timeout before declaring a drop)

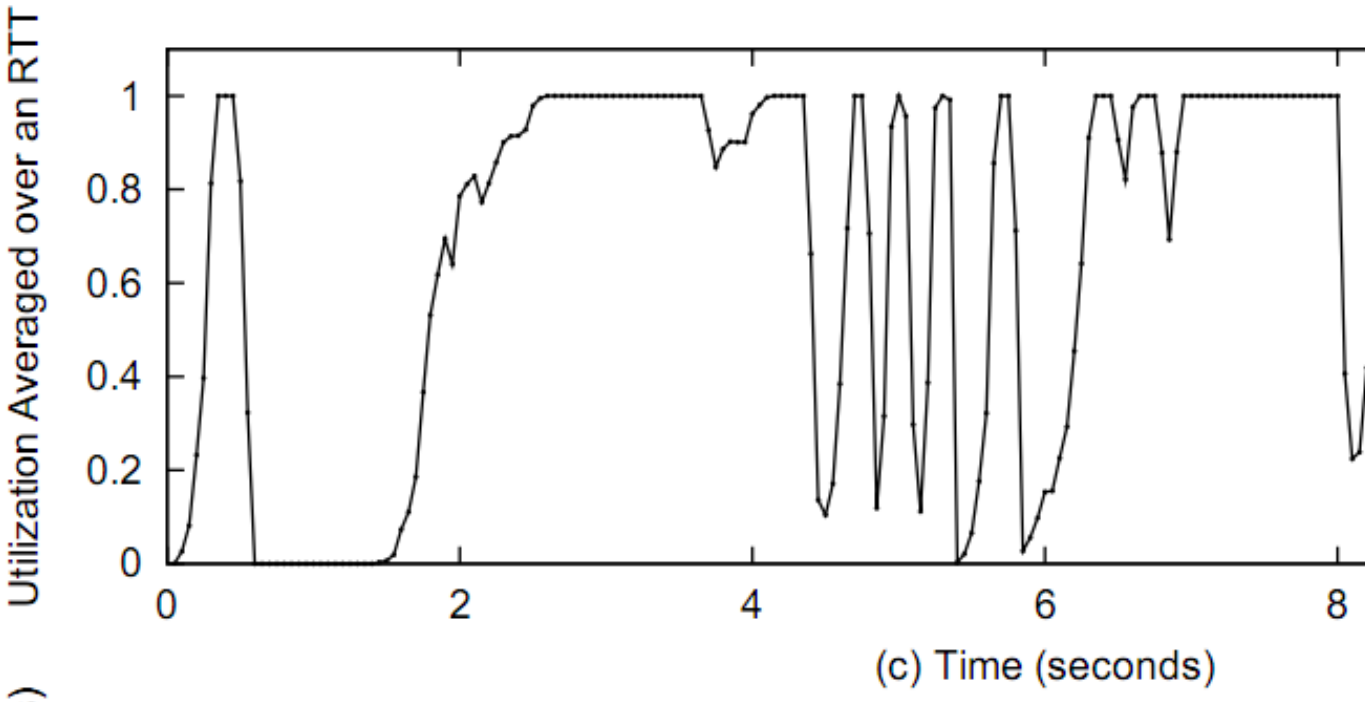
This has for consequence that ...

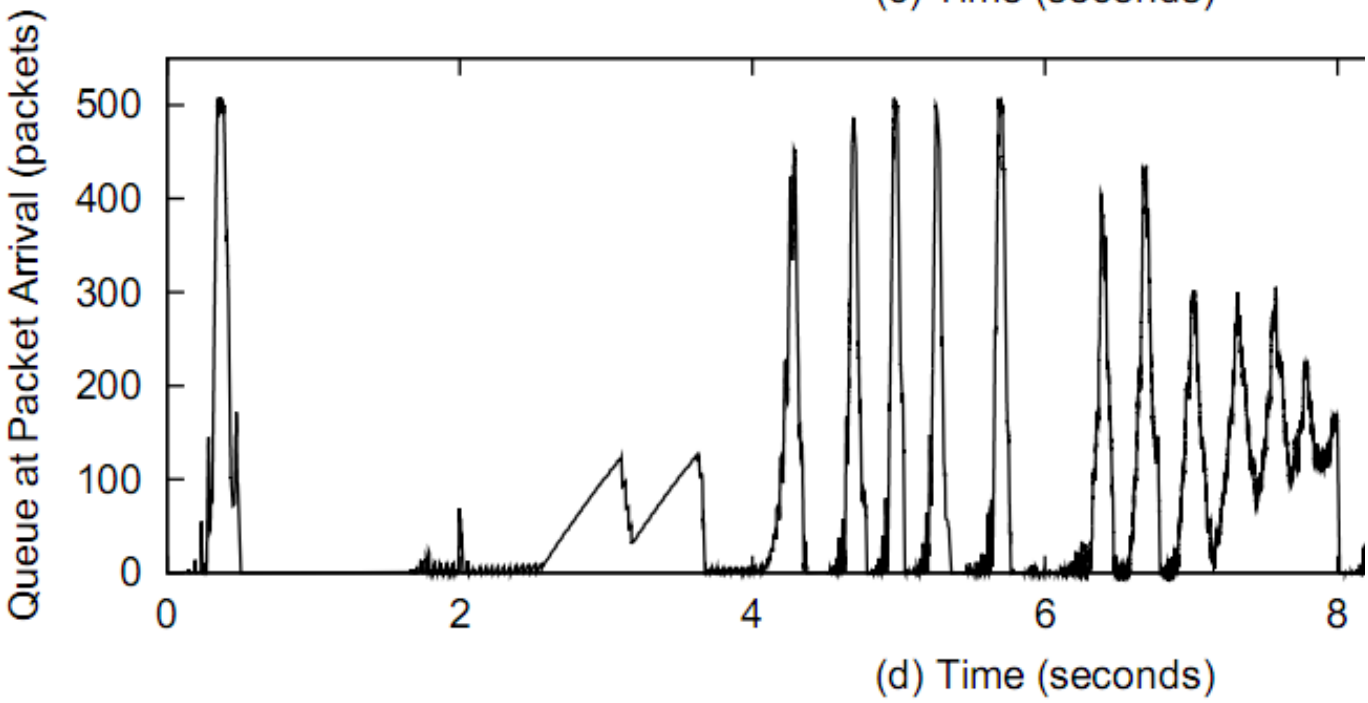
- TCP oscillates at high delay bandwidth product
- TCP's slow start is slow, which is painful at large b×d
- TCP is biased against flows with large rtt

Can't the routers just tell the sender how much bandwidth is available and be done with it?

TCP oscillates




Utilization Averaged over an RTT
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
(c) Time (seconds)


Queue at Packet Arrival (packets)
500
400
300
200
100
0
0
2
4
6
8
(d) Time (seconds)

Outline of the talk

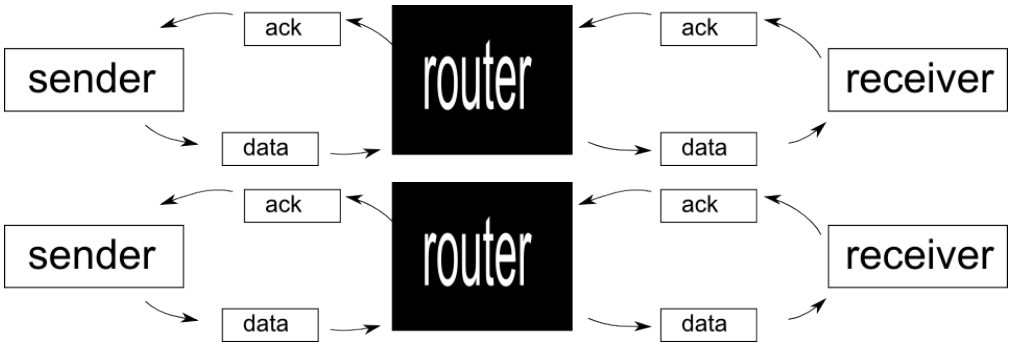
The problem with TCP

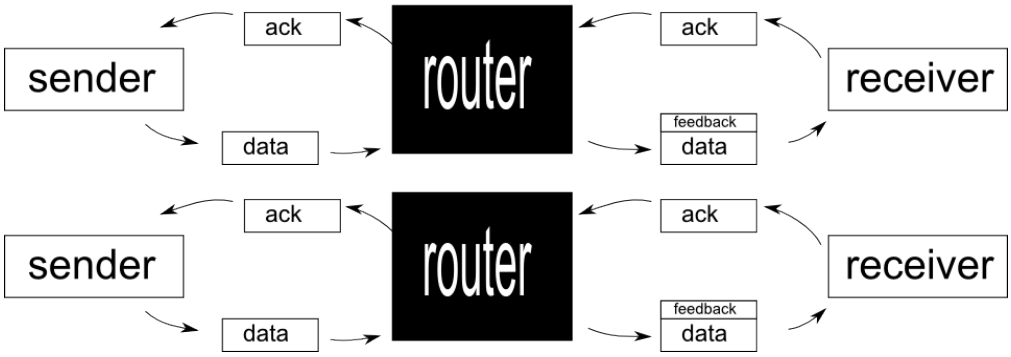
Intuitive overview XCP, with pictures ⟸

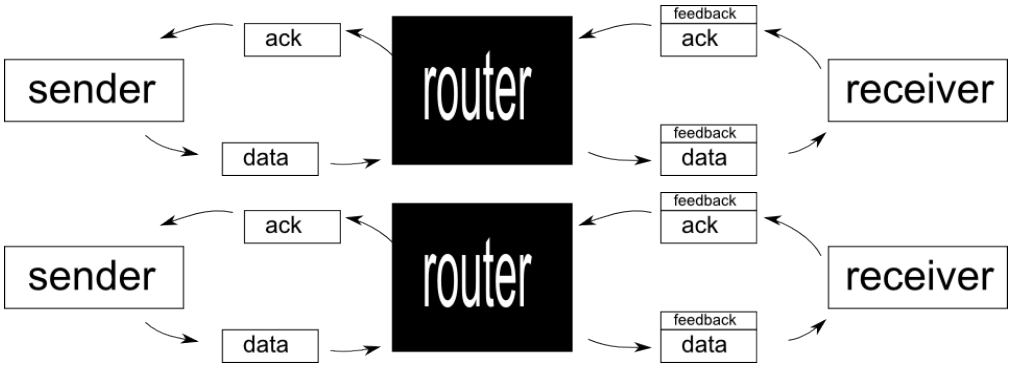
The math of XCP

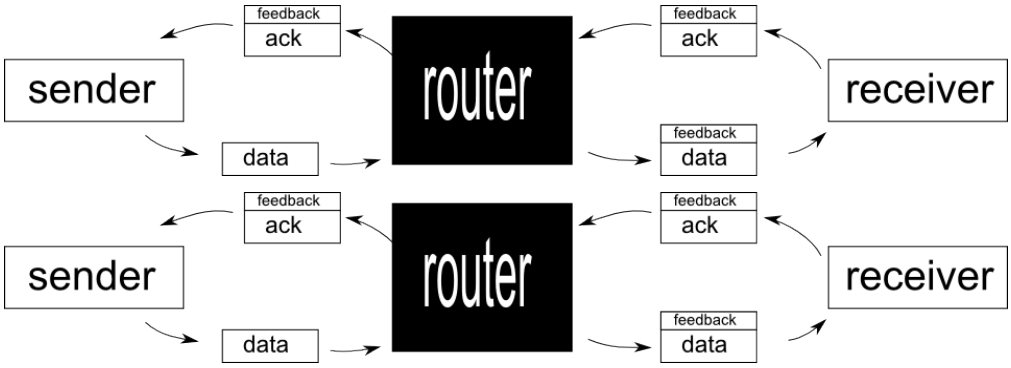
- The efficiency controller (in three steps)
- The fairness controller
  - Version I, Egalitarian
  - Version II, Proportional
  - Version III, Mix of both

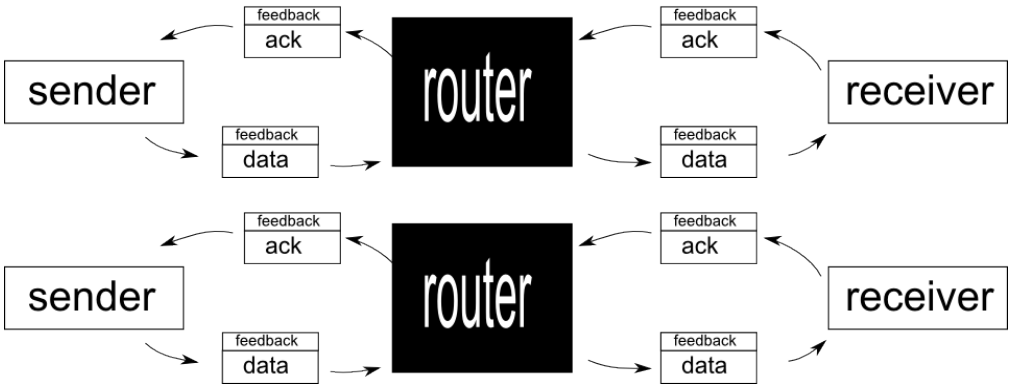
Charts of the simulation results

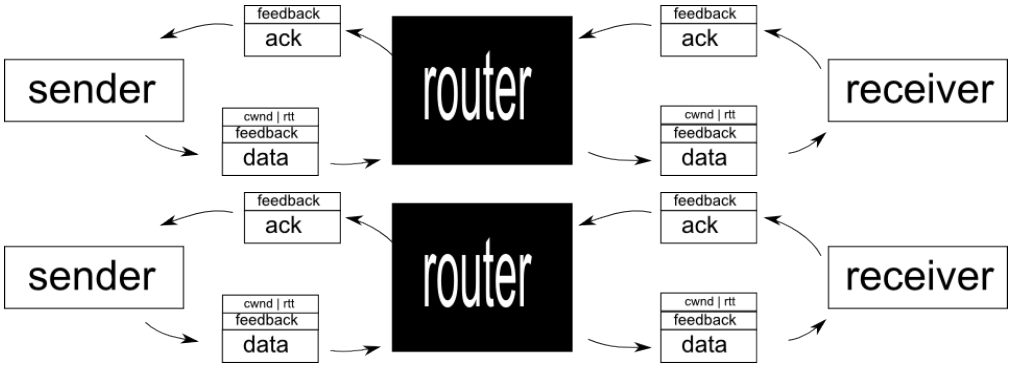

ack
ack
sender
router
receiver
data
data
ack
ack
sender
router
receiver
data
data

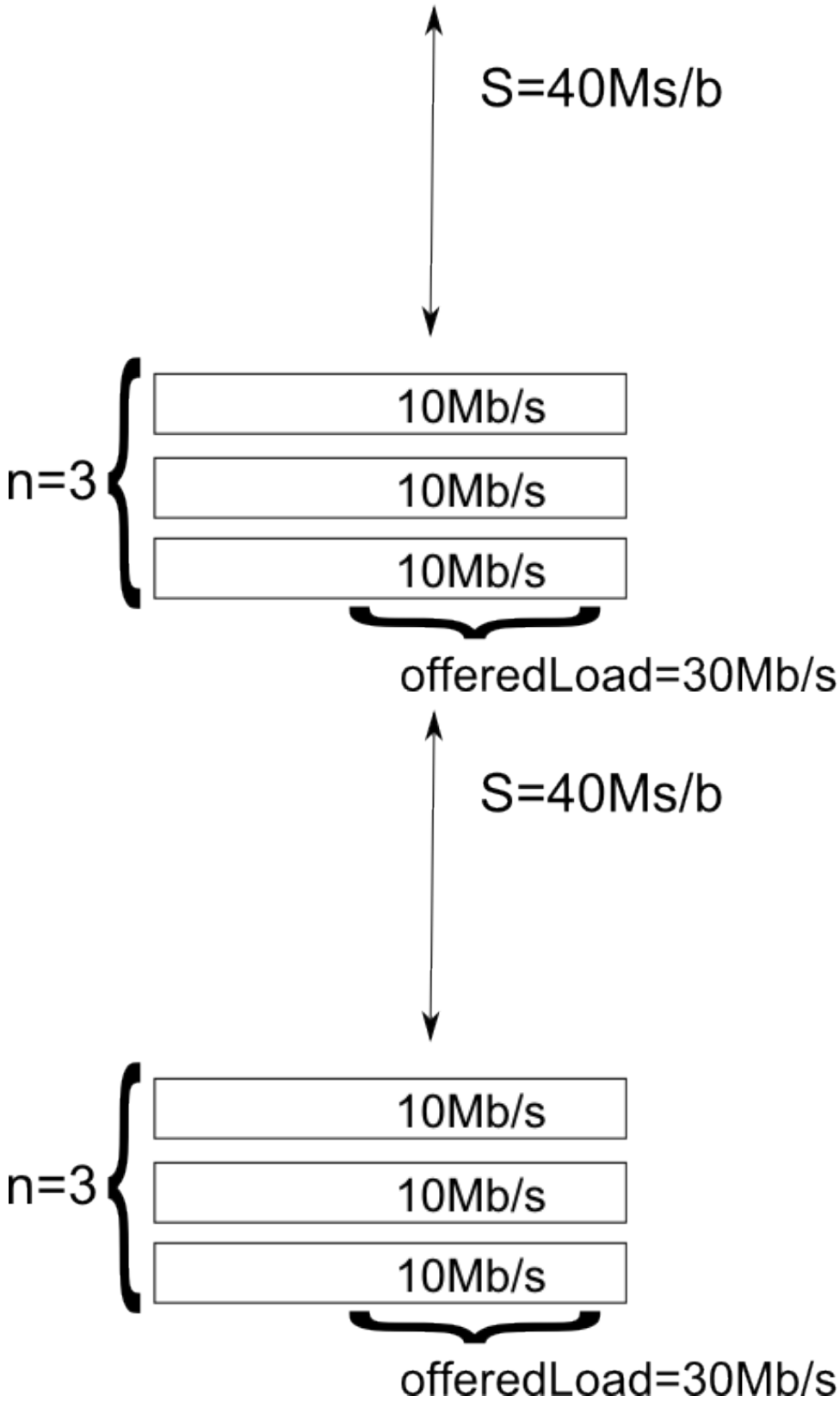

ack
ack
sender
router
receiver
feedback
data
data
ack
ack
sender
router
receiver
feedback
data
data

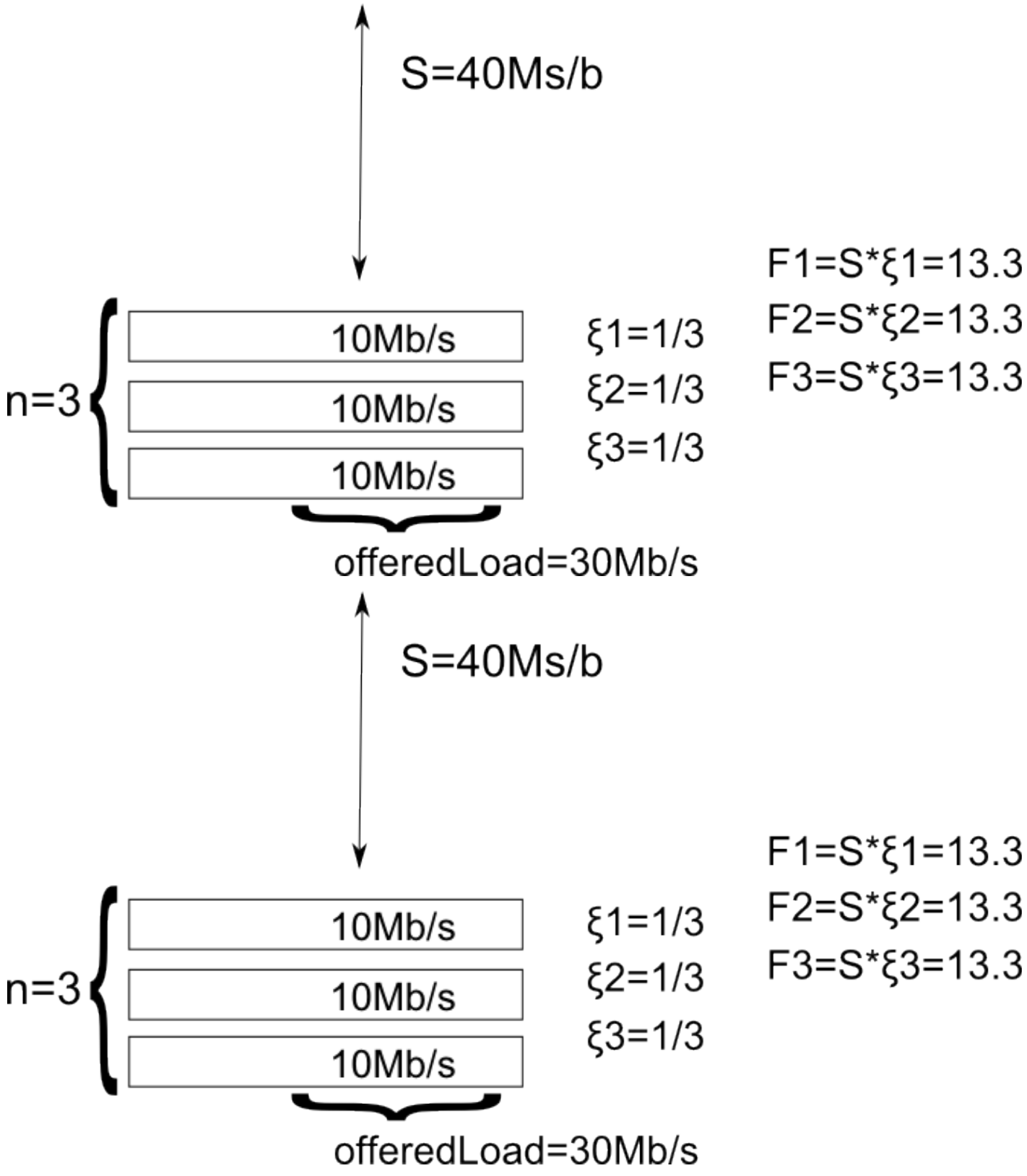

sender
ack
data
router
feedback
ack
feedback
data
receiver
sender
ack
data
router
feedback
ack
feedback
data
receiver


feedback
ack
feedback
ack
sender
router
receiver
data
feedback
data
feedback
ack
feedback
ack
sender
router
receiver
data
feedback
data

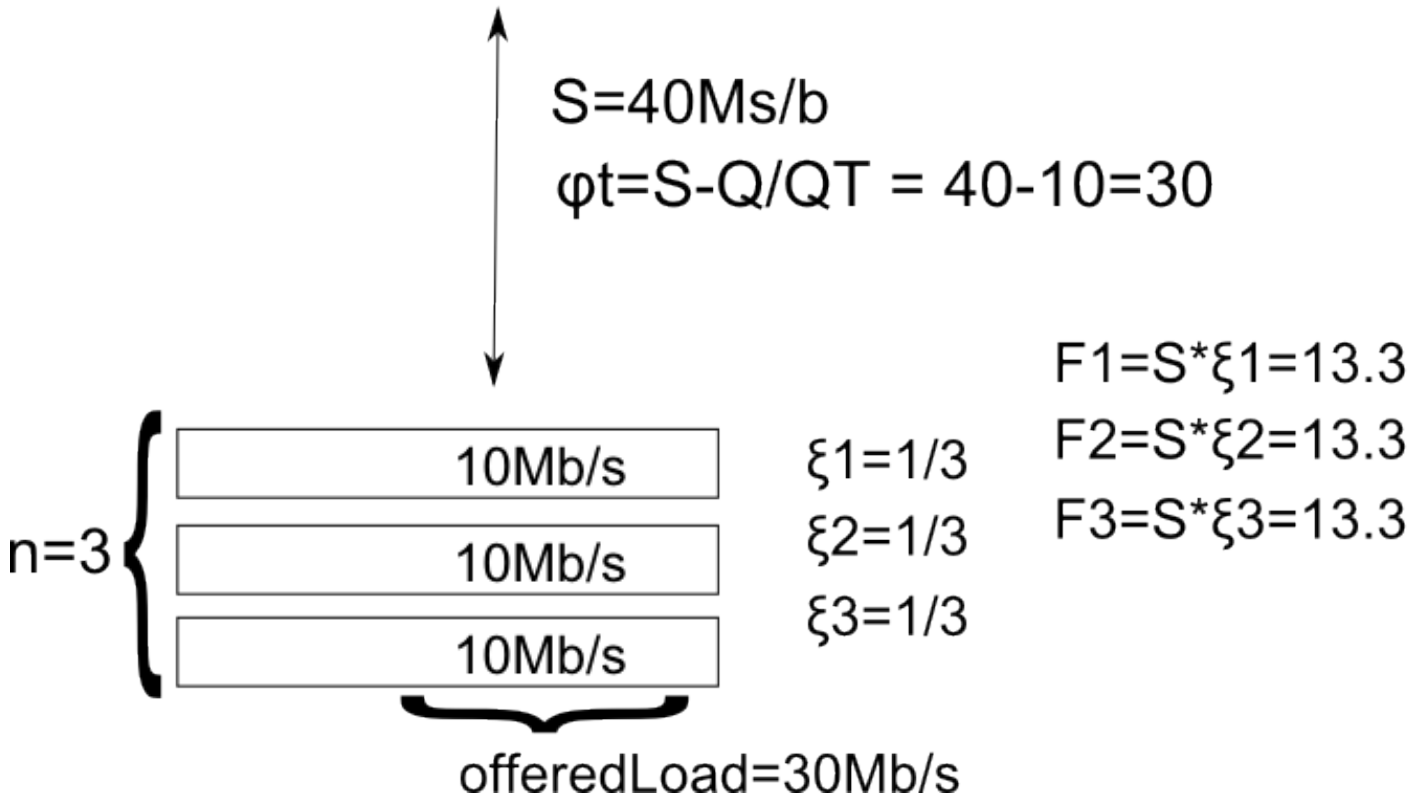

feedback
ack
feedback
ack
sender
router
receiver
feedback
data
feedback
data
feedback
ack
feedback
ack
sender
router
receiver
feedback
data
feedback
data

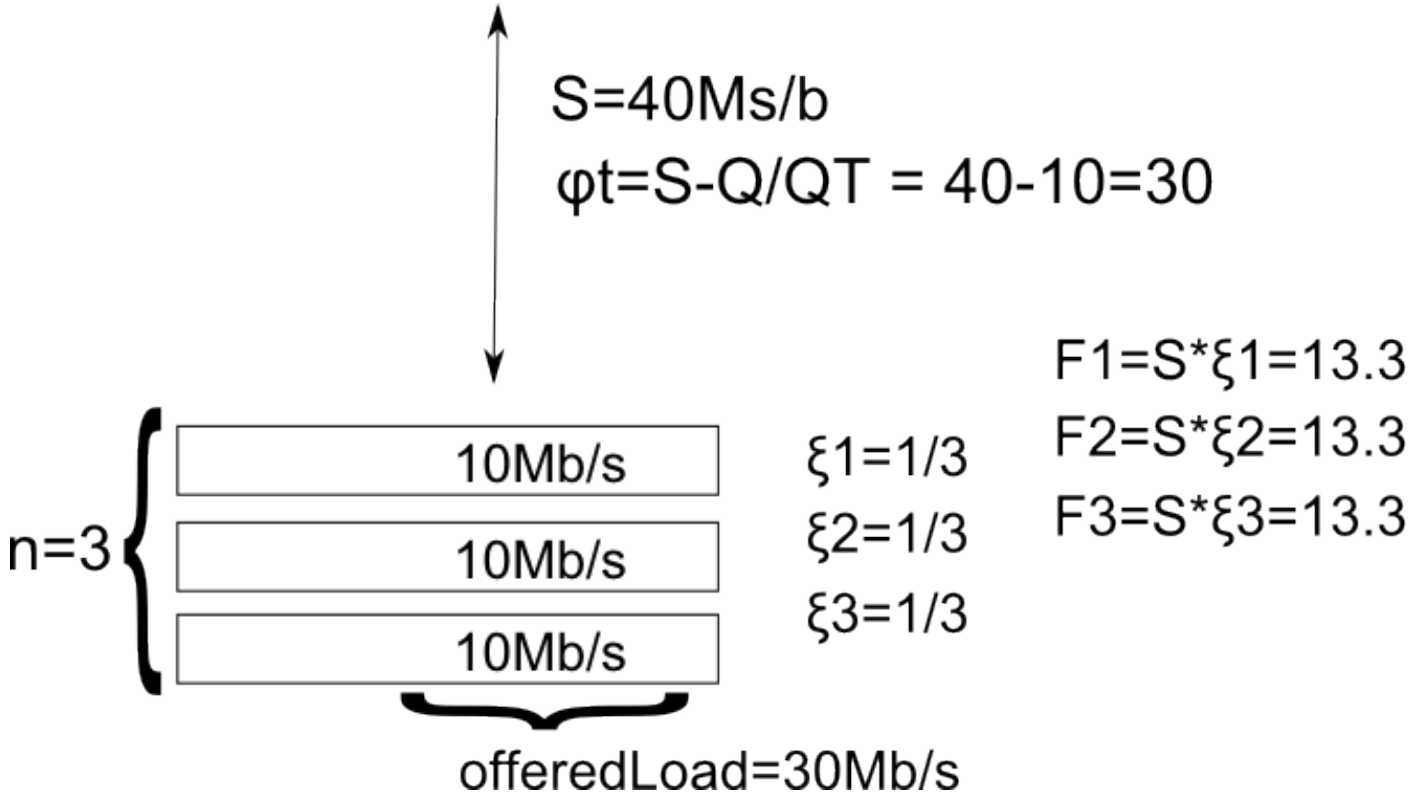

feedback
ack
router
feedback
ack
sender
receiver
cwnd | rtt
feedback
data
cwnd | rtt
feedback
data
feedback
ack
router
feedback
ack
sender
receiver
cwnd | rtt
feedback
data
cwnd | rtt
feedback
data

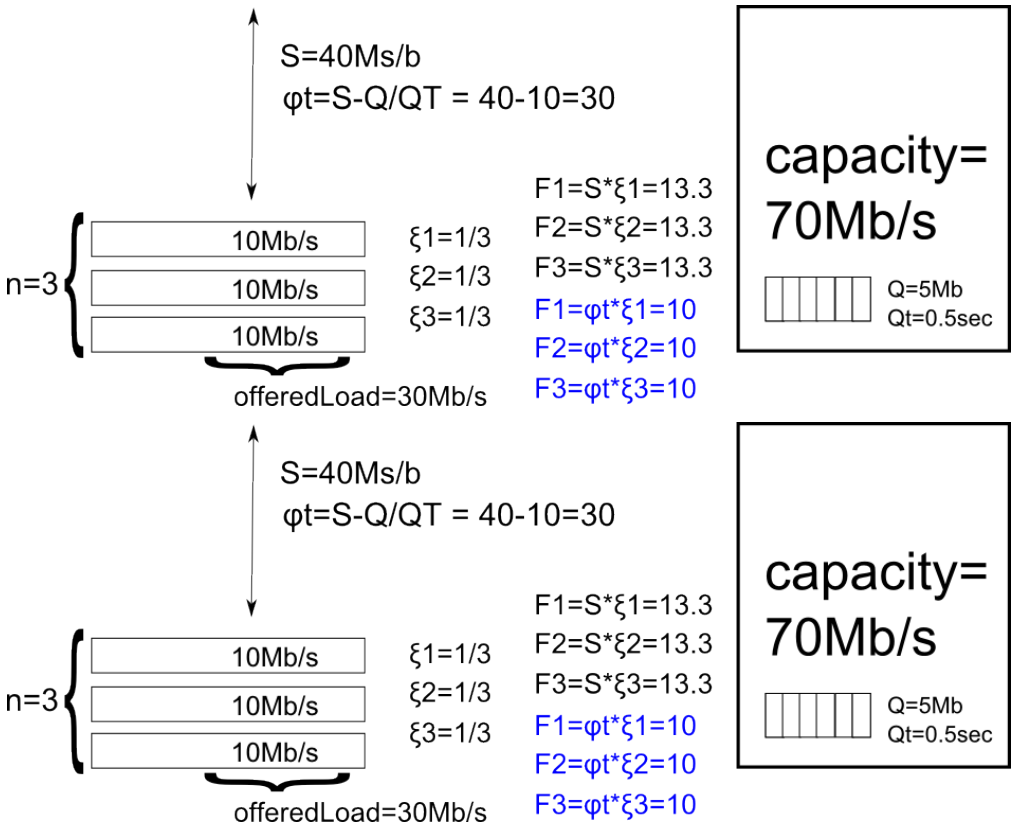

S=40Ms/b
n=3
10Mb/s
10Mb/s
10Mb/s
offeredLoad=30Mb/s
S=40Ms/b
n=3
10Mb/s
10Mb/s
10Mb/s
offeredLoad=30Mb/s


S=40Ms/b
F1=S*ξ1=13.3
F2=S*ξ2=13.3
F3=S*ξ3=13.3
10Mb/s
10Mb/s
10Mb/s
n=3
ξ1=1/3
ξ2=1/3
ξ3=1/3
offeredLoad=30Mb/s
S=40Ms/b
F1=S*ξ1=13.3
F2=S*ξ2=13.3
F3=S*ξ3=13.3
10Mb/s
10Mb/s
10Mb/s
n=3
ξ1=1/3
ξ2=1/3
ξ3=1/3
offeredLoad=30Mb/s


S=40Ms/b
F1=S*ξ1=13.3
F2=S*ξ2=13.3
F3=S*ξ3=13.3
n=3
10Mb/s
10Mb/s
10Mb/s
ξ1=1/3
ξ2=1/3
ξ3=1/3
offeredLoad=30Mb/s
S=40Ms/b
F1=S*ξ1=13.3
F2=S*ξ2=13.3
F3=S*ξ3=13.3
n=3
10Mb/s
10Mb/s
10Mb/s
ξ1=1/3
ξ2=1/3
ξ3=1/3
offeredLoad=30Mb/s


S=40Ms/b
φt=S-Q/QT = 40-10=30
F1=S*ξ1=13.3
F2=S*ξ2=13.3
F3=S*ξ3=13.3
n=3
10Mb/s
10Mb/s
10Mb/s
ξ1=1/3
ξ2=1/3
ξ3=1/3
offeredLoad=30Mb/s


S=40Ms/b
φt=S-Q/QT = 40-10=30
F1=S*ξ1=13.3
F2=S*ξ2=13.3
F3=S*ξ3=13.3
n=3
10Mb/s
10Mb/s
10Mb/s
ξ1=1/3
ξ2=1/3
ξ3=1/3
offeredLoad=30Mb/s


S=40Ms/b
φt=S-Q/QT = 40-10=30
n=3
10Mb/s
10Mb/s
10Mb/s
ξ1=1/3
ξ2=1/3
ξ3=1/3
F1=S*ξ1=13.3
F2=S*ξ2=13.3
F3=S*ξ3=13.3
F1=φt*ξ1=10
F2=φt*ξ2=10
F3=φt*ξ3=10
offeredLoad=30Mb/s
capacity=
70Mb/s
Q=5Mb
Qt=0.5sec
S=40Ms/b
φt=S-Q/QT = 40-10=30
n=3
10Mb/s
10Mb/s
10Mb/s
ξ1=1/3
ξ2=1/3
ξ3=1/3
F1=S*ξ1=13.3
F2=S*ξ2=13.3
F3=S*ξ3=13.3
F1=φt*ξ1=10
F2=φt*ξ2=10
F3=φt*ξ3=10
offeredLoad=30Mb/s
capacity=
70Mb/s
Q=5Mb
Qt=0.5sec

XCP is built in two parts

The Efficiency Controller, responsible for,

- S: the total amount of feedback given, as an aggregate of all flows
- ϕ: adjustments for standing queue and control stability
- Calling out to the fairness contoller to get the ξ_j
- H_feedback: spread the feedback across many packets

The Fairness Controller, responsible for,

- Computing ξ_j to acheive fainess, or any some performance objective

Outline of the talk

The problem with TCP

Intuitive overview XCP, with pictures

The math of XCP ⟸

- The efficiency controller (in three steps)
- The fairness controller
  - Version I, Egalitarian
  - Version II, Proportional
  - Version III, Mix of both

Charts of the simulation results

The Efficiency Controller

(How I understood it anyway, from the bottom up)

The Efficiency Controller, Step 1, Control throughput in aggregate

At its most simple, the feedback is the difference between the capacity and the usage.

```
S = capacity - offeredLoad  (* S is for Surplus [bytes/sec] *)
```

This is the aggregate feedback for all flow. But this is a control system, so you don't want to commit all at once (avoid shower burns).

```
ϕT = α · S      (*  ϕT is for Total Feedback [bytes/sec],
                    0 < α < 1 is a control damper,
                    think Zeno's router algorithm *)
```

Reserve some bandwidth to get rid of any standing queue in the buffer.

```
ϕT = α · S - β · Q / d  (* Q is the standing queue size [bytes],
                           d is the control delay      [seconds],
                           β another control damper       *)
```

The standing queue is defined as the minimum size of the queue during the previous control period.

The control delay is defined as the average round trip time of all the packets during the previous control period.

The Efficiency Controller, Step 2, Allocate the aggregate feedback

Ask some other module (the fairness controller) to split the feedback in some fair way.

$$\sum_{j=1}^{n} \xi_j = 1 \quad (* \ \xi_j \text{ is the proportion of throughput feedback assigned to the } j^{th} \text{ flow, } n \text{ is the number of flows} \ *)$$

Then split it equally across the packets.

$$H\_feedback_{T,j,i} = \left(\xi_j \cdot \phi_T\right) \Big/ E_j \quad (* \ E_j \text{ is the expected number of packets from flow j during the next control period} \ *)$$

Hack alert: They assume that the current packet is representative of the whole flow, or that it all averages out in the end. That is, assume that

$$E_j = \frac{cwnd_i}{s_i} \cdot \frac{d}{rtt_i} \quad (* \text{ The number of packet in the cwnd} \times \text{the number of cwnd's in the control period} \ *)$$

$$H\_feedback_{T,j,i} = \left(\xi_j \cdot \phi_T\right) \Big/ \left(\frac{cwnd_i}{s_i} \cdot \frac{d}{rtt_i}\right)$$

The Efficiency Controller, Step 3, Convert Δthroughput to Δcwnd

Rates are hard to measure. Beside, rates at the sender are controlled by the congestion window, might as well control the senders cwnd's, it's more direct.
Recall,

```
cwnd = bandwidth × delay  (* cwnds are set to the b-d product *)
```

Then converting the aggregate feedback from throughput to congestion window,

$$\phi_T = \alpha \cdot S - \beta \cdot Q / d$$

means multiplying by d on both sides,

$$\phi = \alpha \cdot d \cdot S - \beta \cdot Q$$

Similarly, convert the per-packet feedback from from throughput to congestion window,

```
H_feedback_{j,i} = H_feedback_{T,j,i} · rtt_i  (* assuming the packet is representative of the whole.
  aka, rtt_i ≈ rtt_j *)
```

$$H\_feedback_{j,i} = \left((\xi_j \cdot \phi_T) \Big/ \left(\frac{cwnd_i}{s_i} \cdot \frac{d}{rtt_i} \right) \right) \cdot rtt_i$$

$$H\_feedback_{j,i} = \xi_j \cdot \frac{\phi_T \cdot rtt_i^2 \cdot s_i}{d \cdot cwnd_i}$$

The Fairness Controller

(Choosing ξ_j wisely)

Version I, Egalitarian feedback

A first intuition of fairness might be, set all the ξ_j to the same value. Namely,

$$\xi_1 = \xi_2 = \ldots = \xi_n = \xi = \frac{1}{n}$$

But measuring n directly is hard. Let's try to find an indirect definition.

```
ϕT = Sum[H_feedback_{T,j,i}, {i, 0, L}]  (* L is the total number of packets
                                            through the router during control period *)
```

Substitute the definition of feedback$_{T,j,i}$. Since all the flows have the same ξ, we can drop the j subscript.

$$\phi_T = \sum_{i=0}^{L} \left(\xi \cdot \frac{\phi_T \cdot rtt_i \cdot s_i}{d \cdot cwnd_i} \right)$$

$$\phi_T = \frac{\xi \cdot \phi_T}{d} \cdot \sum_{i=0}^{L} \left(\frac{rtt_i \cdot s_i}{cwnd_i} \right)$$

$$\xi = \frac{d}{\sum_{i=0}^{L} \left(\frac{rtt_i \cdot s_i}{cwnd_i} \right)} \quad (* \text{ which is an odd way to measure } \tfrac{1}{n} *)$$

Version I, Egalitarian feedback (continued)

$$H\_feedback_i = \xi \cdot \frac{\phi_T \cdot rtt_i^2 \cdot s_i}{d \cdot cwnd_i}$$

$$H\_feedback_i = \frac{d}{\sum_{i=0}^{L} \left(\frac{rtt_i \cdot s_i}{cwnd_i}\right)} \cdot \frac{\phi_T \cdot rtt_i^2 \cdot s_i}{d \cdot cwnd_i}$$

$$H\_feedback_i = \frac{\phi_T}{\sum_{i=0}^{L} \left(\frac{rtt_i \cdot s_i}{cwnd_i}\right)} \cdot \frac{rtt_i^2 \cdot s_i}{cwnd_i} \quad (* \text{ the d cancel } *)$$

$$H\_feedback_i = \frac{\phi}{d \cdot \sum_{i=0}^{L} \left(\frac{rtt_i \cdot s_i}{cwnd_i}\right)} \cdot \frac{rtt_i^2 \cdot s_i}{cwnd_i} \quad (* \text{ with } \phi_T = \frac{\phi}{d} *)$$

Which is the formula in the paper for equal-Δthroughput feedback without shuffling.

Hack Alert: The left side of the dot refers to values from past packets, the right side of the dot refers the values taken from this packet's header.

Version II: Feedback proportional to throughput

$$\xi_j = \text{bandwidth}_j \Big/ \text{offeredLoad}$$

This is well-formed, $\sum \xi_j = 1$, since $\sum^{n} \text{bandwidth}_j = \text{offeredLoad}$. Then

$$\xi_j = \frac{\text{cwnd}_i}{\text{rtt}_i} \Big/ \text{offeredLoad}$$

Same slight of hand, assume the packet is representative of the whole.

$$\text{H\_feedback}_{j,i} = \xi_j \cdot \frac{\phi_T \cdot \text{rtt}_i^2 \cdot s_i}{d \cdot \text{cwnd}_i} \quad (*\text{recall the def. of feedback}*)$$

$$\text{H\_feedback}_i = \frac{\text{cwnd}_i}{\text{rtt}_i \cdot \text{offeredLoad}} \cdot \frac{\phi_T \cdot \text{rtt}_i^2 \cdot s_i}{d \cdot \text{cwnd}_i} (*\text{insert } \xi_j*)$$

$$\text{H\_feedback}_i = \frac{1}{\text{offeredLoad}} \cdot \frac{\phi_T \cdot \text{rtt}_i \cdot s_i}{d} \quad (*\text{cancel}*)$$

$$\text{H\_feedback}_i = \frac{1}{\frac{\sum s_i}{d}} \cdot \frac{\phi \cdot \text{rtt}_i \cdot s_i}{d^2} \quad (*\text{def. of load and } \phi_T*)$$

$$\text{H\_feedback}_i = \frac{\phi}{d \cdot \sum s_i} \cdot (\text{rtt}_i \cdot s_i) \quad (*\text{cancel and regroup } *)$$

Which is the formula in the paper for feedback $\propto$ throughput without shuffling.

Same Hack Alert: The left side of the dot refers to values from past packets, the right side of the dot refers the values taken FROM this packet's header.

Version III : A bit of both

Version I, egalitarian feedback (equal Δthroughput for everyone) doesn't make sense for negatives feedback — you might be asking a flow to go negative.

Version II, feedback ∝ throughput, doesn't make sense for positive feedback — you give the most boost to the already fast flows.

Thus Version III, mix of both, positive feedback is egalitarian, negative feedback is ∝ throughput.

$$\text{posFeedback}_i = \frac{\text{Max}[\phi, 0]}{d \cdot \sum_{i=0}^{L} \left(\frac{\text{rtt}_i \cdot s_i}{\text{cwnd}_i}\right)} \cdot \frac{\text{rtt}_i^2 \cdot s_i}{\text{cwnd}_i}$$

$$\text{negFeedback}_i = \frac{\text{Max}[-\phi, 0]}{d \cdot \sum s_i} \cdot (\text{rtt}_i \cdot s_i)$$

$$\text{H\_feedback}_i = \text{posFeedback}_i - \text{negFeedback}_i$$

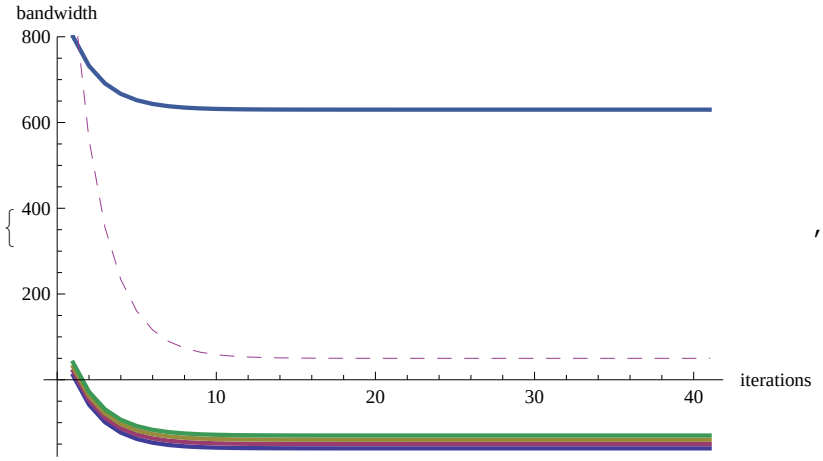
Good news! If you oscillate between the two you converge to fairness!

Bad news! The whole point of this protocol is to kill oscillations!

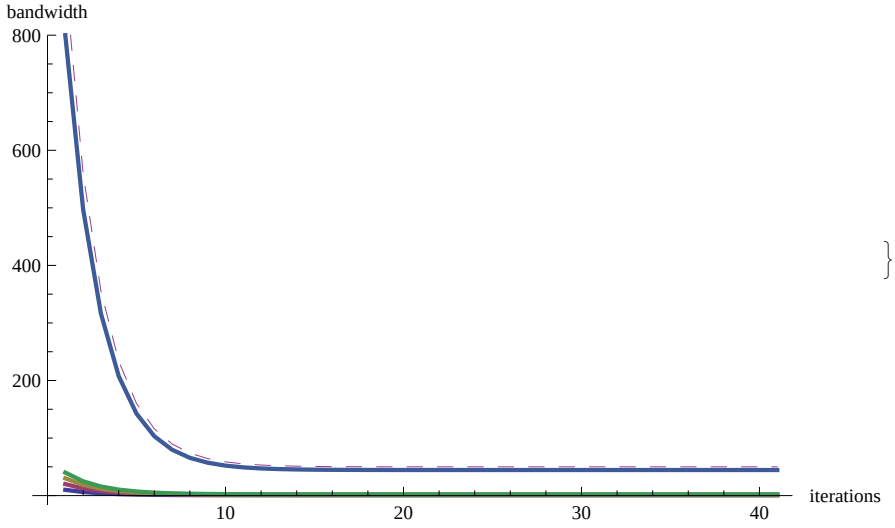
Exploring convergence via oscillations

```
ϕ[offeredLoad_, capacity_] := α (capacity - offeredLoad);
HfEgalitarian[offeredLoad_, capacity_, bandwidth_, n_] :=
  ϕ[offeredLoad, capacity] / n;
HfProportional[offeredLoad_, capacity_, bandwidth_, n_] :=
  ϕ[offeredLoad, capacity] bandwidth / offeredLoad;


length[lst_] := Dimensions[lst][[1]];

XCPEgalitarian[bandwidths_] :=
  Map[b ↦
    b + HfEgalitarian[Apply[Plus, bandwidths], capacity, b, length[bandwidths]],
   bandwidths];
XCPProportional[bandwidths_] :=
  Map[b ↦
    b + HfProportional[Apply[Plus, bandwidths], capacity, b, length[bandwidths]],
   bandwidths];

convergeXCPfn[xcpFn_, init_] := Module[{tbl = Table[Nest[b ↦ xcpFn[b], init, t], {t, 0, 40}]},
   Append[Transpose[tbl], Map[lst ↦ Apply[Plus, lst], tbl]]];


XCPPlot[xcpFn_] :=
  ListLinePlot[
   convergeXCPfn[xcpFn, init],
   PlotStyle → Append[Table[Thick, {i, length[init]}], {Thin, Dashing[0.02]}],
   PlotRange → {Automatic, {Automatic, Max[init, capacity]}},
   AxesLabel → {"iterations", "bandwidth"}
  ];
```

```
α = 0.4;
capacity = 50;
init = {10., 20., 30., 40., 800.};
{XCPPlot[XCPEgalitarian], XCPPlot[XCPProportional]}
```

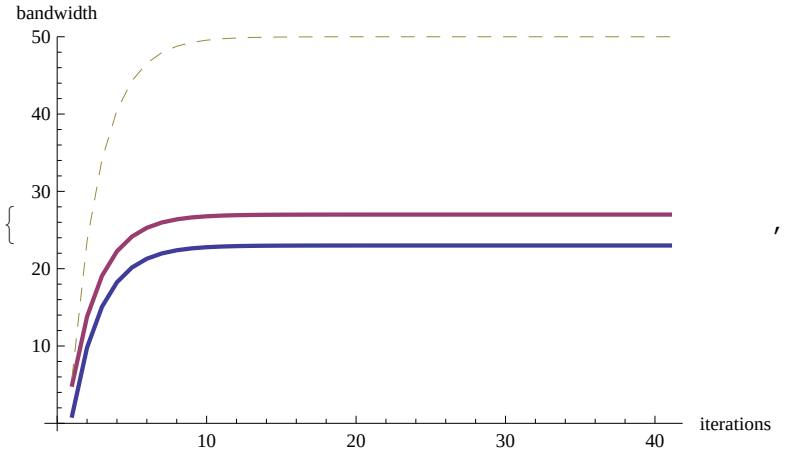


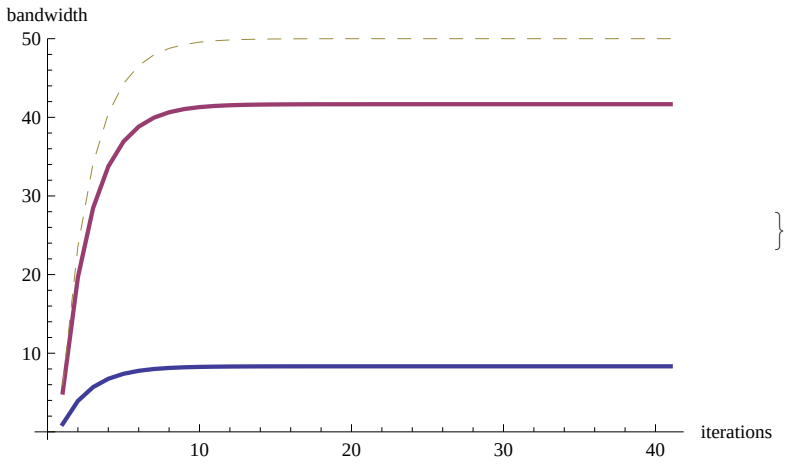
,



}

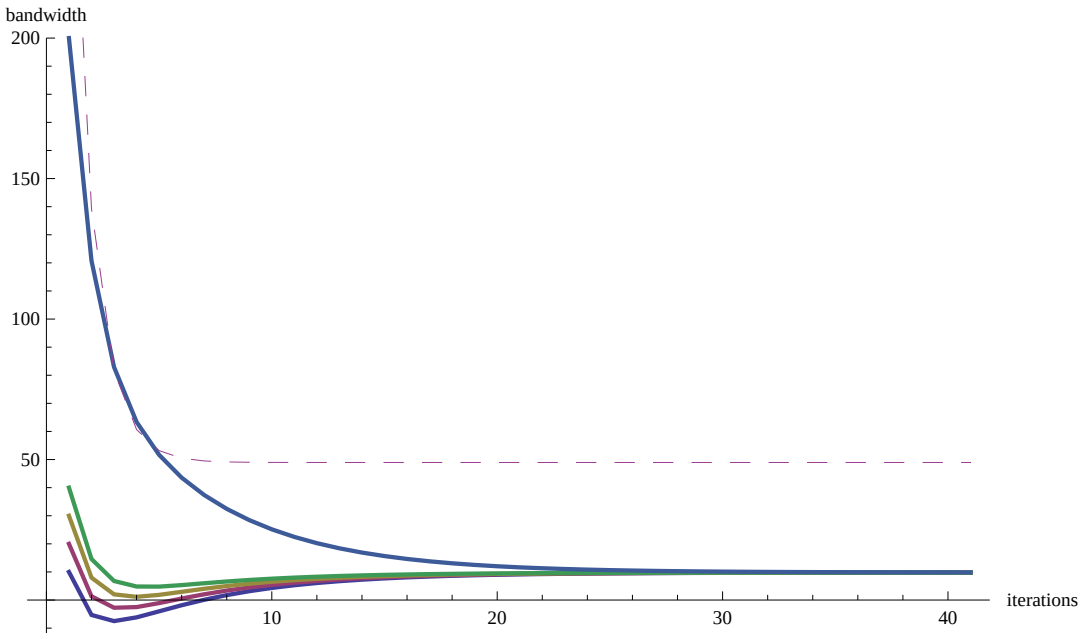
```
init = {1., 5.};
{XCPPlot[XCPEgalitarian], XCPPlot[XCPProportional]}
```



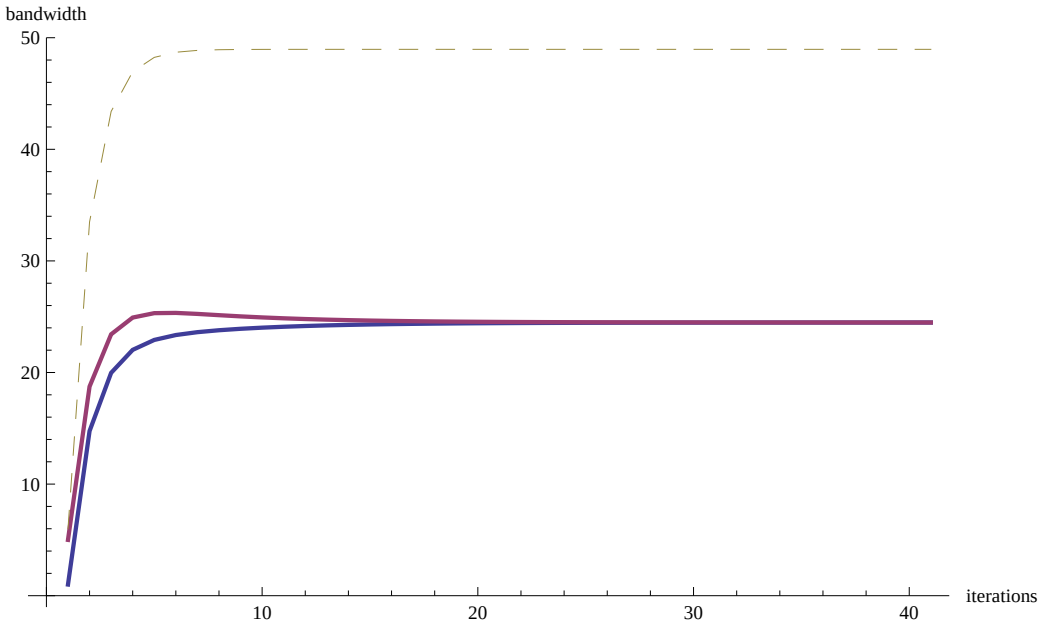



```
XCPBoth[bandwidths_] :=
  (capacity = capacity * 1.5;
   Module[{b2 = XCPEgalitarian[bandwidths]},
    capacity = capacity / 1.5 / 1.5;
    Module[{b3 = XCPProportional[b2]},
     capacity = capacity * 1.5;
     b3]]);
```

```
init = {10., 20., 30., 40., 200.};
XCPPlot[XCPBoth]
```




```
init = {1., 5.};
XCPPlot[XCPBoth]
```



Forcing a small amount of oscillation

The solution: re-introduce oscillation into the system in a controlled manner

```
h = Max[0, γ · y – Abs[ϕ]]
```

where

```
γ = 0.1            (* the size of the oscillation desired *)
y = offeredLoad · d (* the trafic during a control period*)
```

namely, unless we converged on a fair distribution, some minimum amount of traffic must always be trifled with, if not by ϕ, then by h.

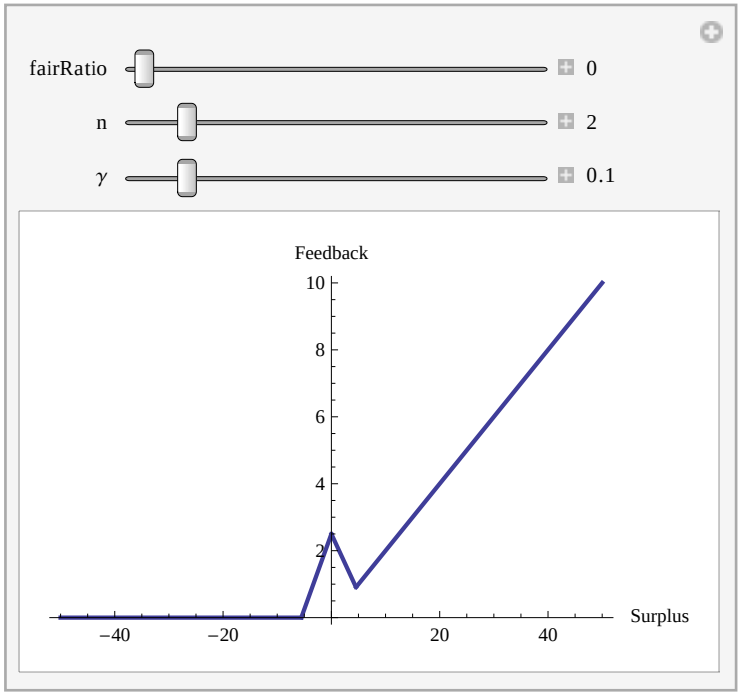
$$\text{posFeedback}_i = \frac{h + \text{Max}[\phi, 0]}{d \cdot \sum_{i=0}^{L} \left(\frac{\text{rtt}_i \cdot s_i}{\text{cwnd}_i}\right)} \cdot \frac{\text{rtt}_i^2 \cdot s_i}{\text{cwnd}_i}$$

$$\text{negFeedback}_i = \frac{h + \text{Max}[-\phi, 0]}{d \cdot \sum s_i} \cdot (\text{rtt}_i \cdot s_i)$$

$$\text{H\_feedback}_i = \text{posFeedback}_i - \text{negFeedback}_i$$

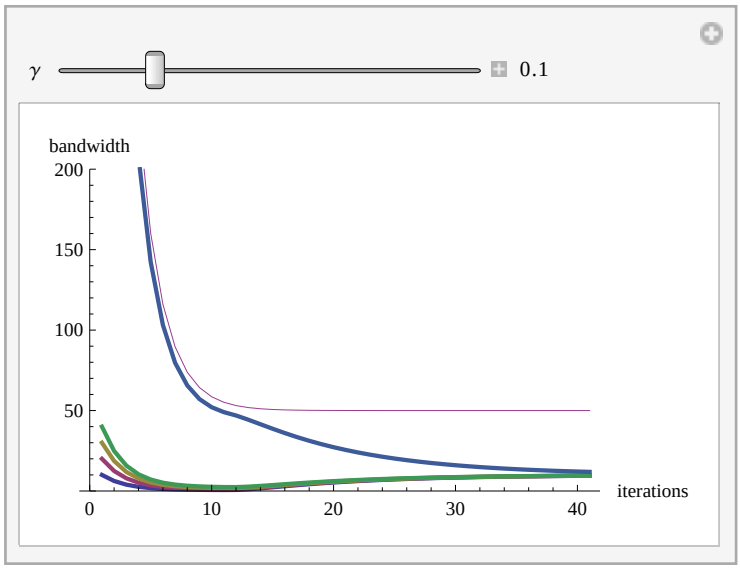
The magical *h* shuffle

```
h[γ_, offeredLoad_, capacity_] :=
  Max[0, γ offeredLoad - Abs[capacity - offeredLoad]];
ϕ[offeredLoad_, capacity_] :=
  α (capacity - offeredLoad);
neg[γ_, offeredLoad_, capacity_] :=
  h[γ, offeredLoad, capacity] + Max[0, -ϕ[offeredLoad, capacity]];
pos[γ_, offeredLoad_, capacity_] :=
  h[γ, offeredLoad, capacity] + Max[0, ϕ[offeredLoad, capacity]];
Hf[γ_, offeredLoad_, capacity_, bandwidth_, n_] :=
  pos[γ, offeredLoad, capacity] / n - neg[γ, offeredLoad, capacity] bandwidth/offeredLoad;
Hf'[γ_, surplus_, fairRatio_, n_] :=
  Hf[γ, capacity - surplus, capacity, fairRatio / n * capacity, n];

α = 1.0;
capacity = 50;
Manipulate[Plot[
  Hf'[γ, surplus, fairRatio, n],
  {surplus, -capacity, capacity},
  PlotStyle → Thick,
  AxesLabel → {"Surplus", "Feedback"}],
 {{fairRatio, 0}, 0, 4, 0.1, Appearance → "Labeled"},
 {{n, 2}, 1, 10, 1, Appearance → "Labeled"},
 {{γ, 0}, 0, 0.9, 0.05, Appearance → "Labeled"}]
```



Convergence to fairness

```
XCP[γ_, bandwidths_] :=
  Map[b ↦
    b + Hf[γ, Apply[Plus, bandwidths], capacity, b, length[bandwidths]],
   bandwidths];
convergeXCP[γ_, init_] := Module[{tbl = Table[Nest[b ↦ XCP[γ, b], init, t], {t, 0, 40}]},
   Append[Transpose[tbl], Map[lst ↦ Apply[Plus, lst], tbl]]];


init = {10. , 20., 30., 40., 200.};
α = 0.4;
Manipulate[
 ListLinePlot[
  convergeXCP[γ, init],
  PlotStyle → Append[Table[Thick, {5}], Thin],
  PlotRange → {Automatic, {0, 200}},
  AxesLabel → {"iterations", "bandwidth"}
 ],
 {{γ, 0.1}, 0, 0.5, Appearance → "Labeled"}]
```



Outline of the talk

The problem with TCP

Intuitive overview XCP, with pictures

The math of XCP

- The efficiency controller (in three steps)
- The fairness controller
  - Version I, Egalitarian
  - Version II, Proportional
  - Version III, Mix of both

Charts of the simulation results ⟸

Simulation results

Simulation topologies

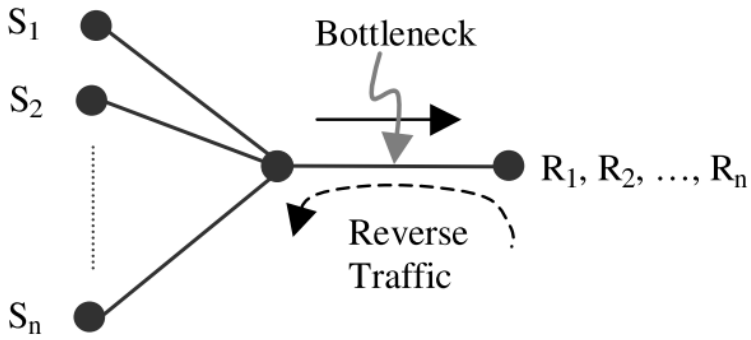


Figure 2: A single bottleneck topology.

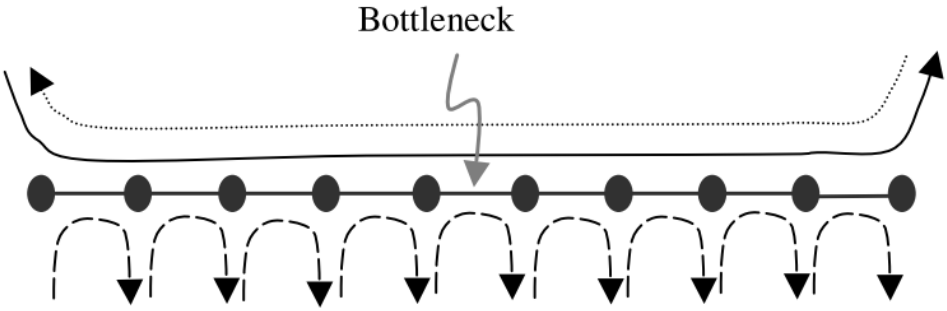


Figure 3: A parking lot topology.

Fixed: Bottleneck topology, 50 flows forwards, 50 flows reversewards, RTT is 80ms
Vary: Link bandwidth

Largest bandwidth-delay is 320 Megs · seconds (!)
Longest queue size is 2.3 Megs at 1Meg/s (2.3 seconds to clear)
150 Meg/s comes back later

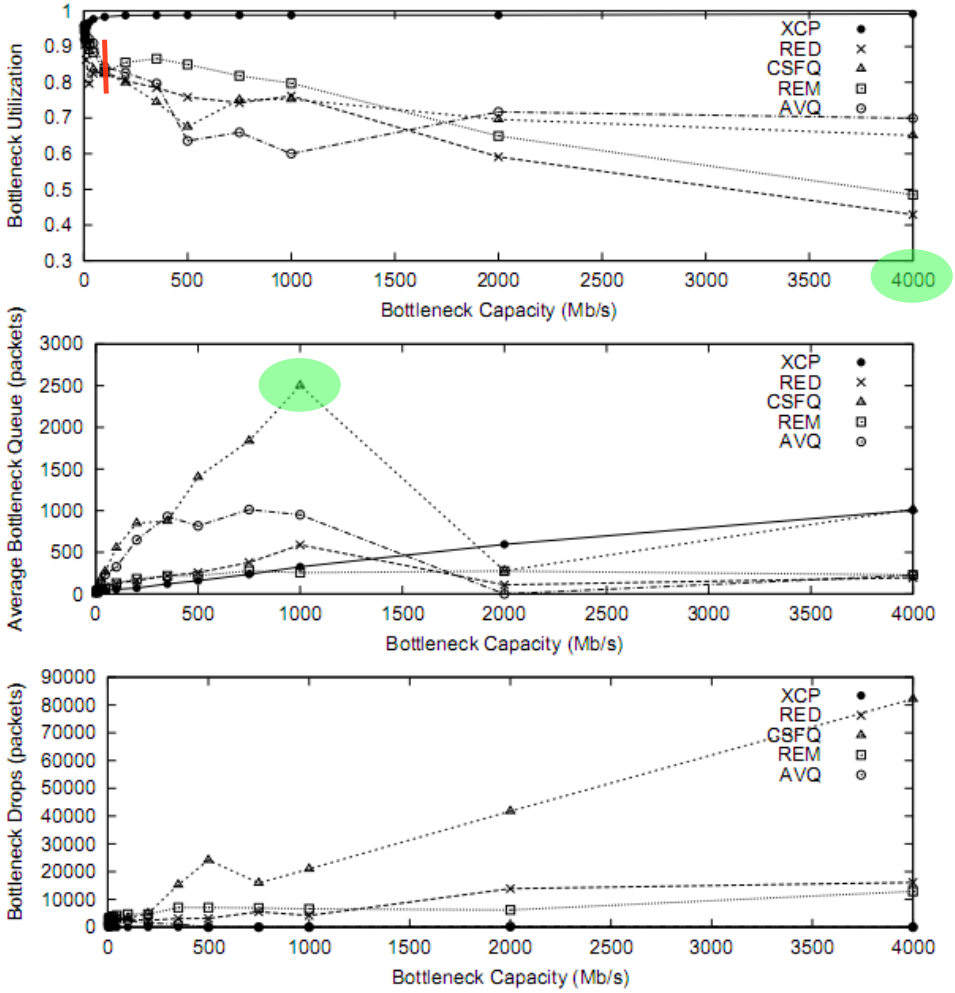


Figure 4: XCP significantly outperforms TCP in high bandwidth environments. The graphs compare the efficiency of XCP with that of TCP over RED, CSFQ, REM, and AVQ as a function of capacity.

Fixed: Bottleneck topology, 50 flows forwards, 50 flows reversewards, Link is 150Meg/s
Vary: RTT

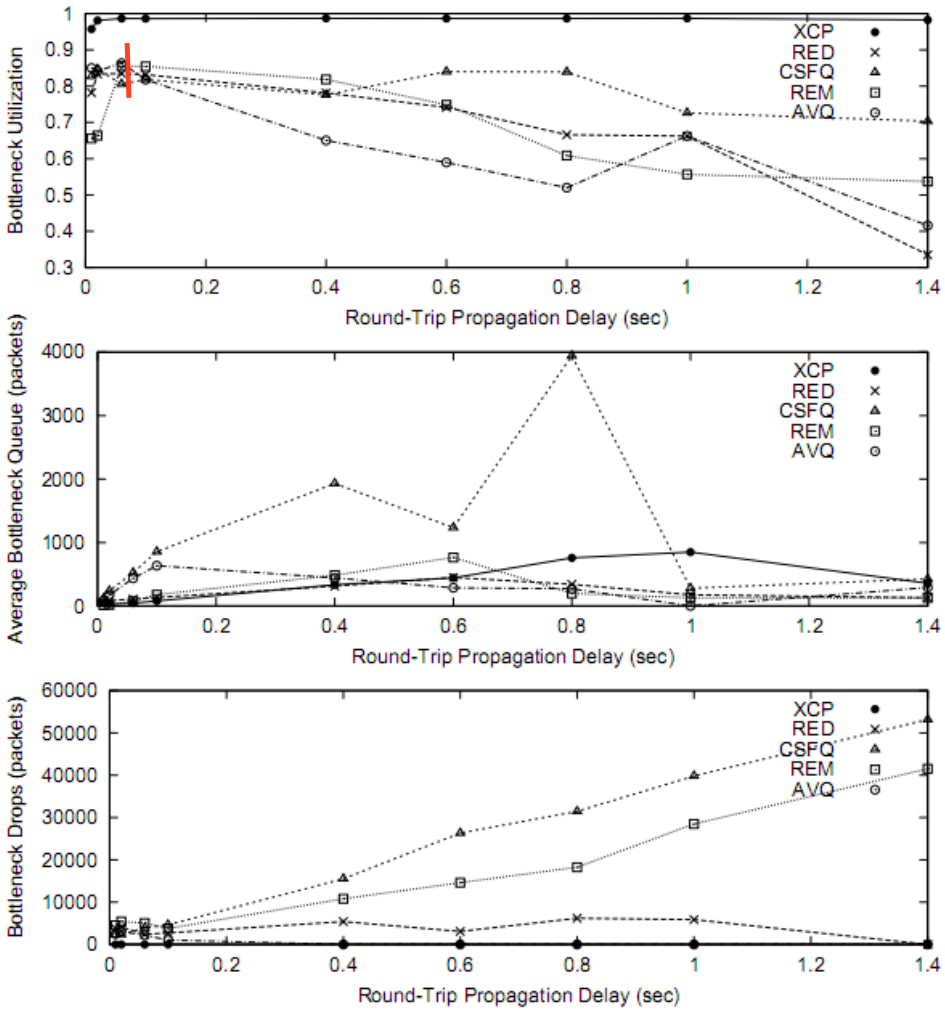


Figure 5: XCP significantly outperforms TCP in high delay environments. The graphs compare bottleneck utilization, average queue, and number of drops as round trip delay increases when flows are XCPs and when they are TCPs over RED, CSFQ, REM, and AVQ.

Fixed: Bottleneck topology, RTT is 80ms, Link is 150Meg/s
Vary: Number of flows

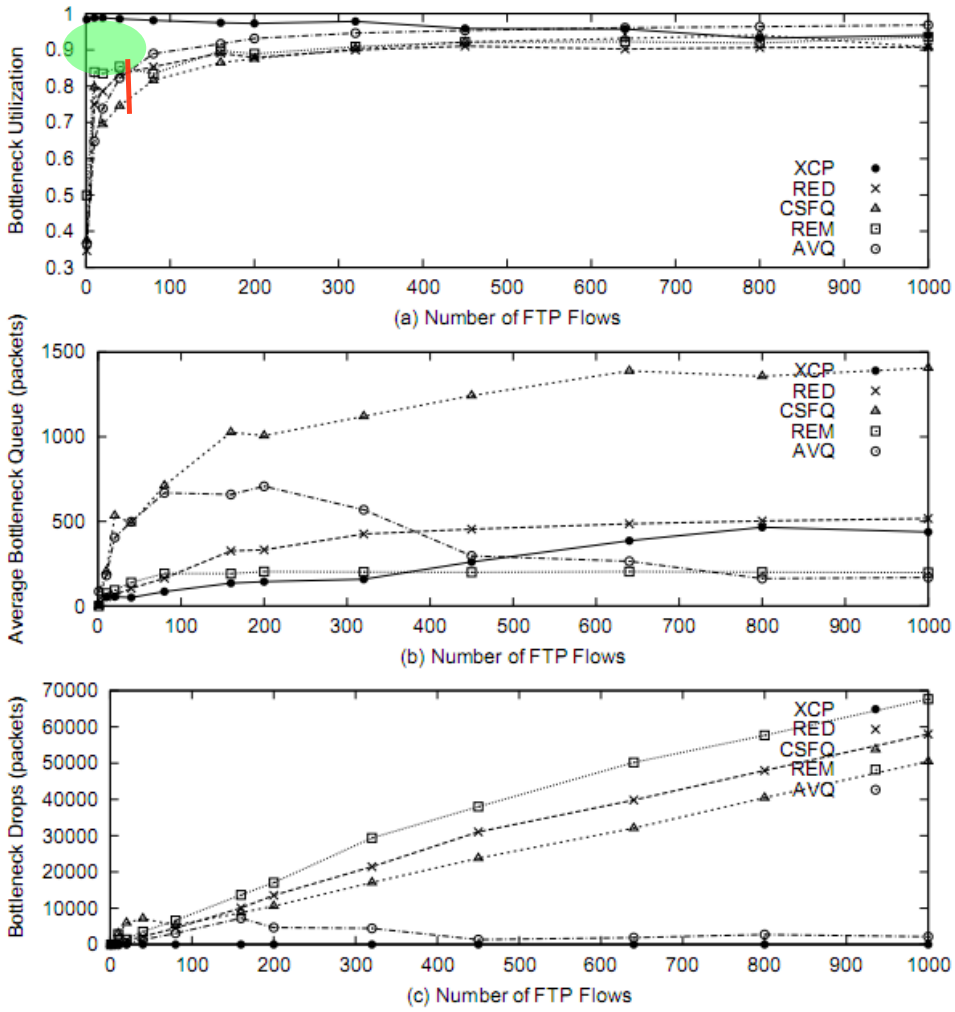


Figure 6: XCP is efficient with any number of flows. The graphs compare the efficiency of XCP and TCP with various queuing schemes as a function of the number of flows.

The war between mice and elephants

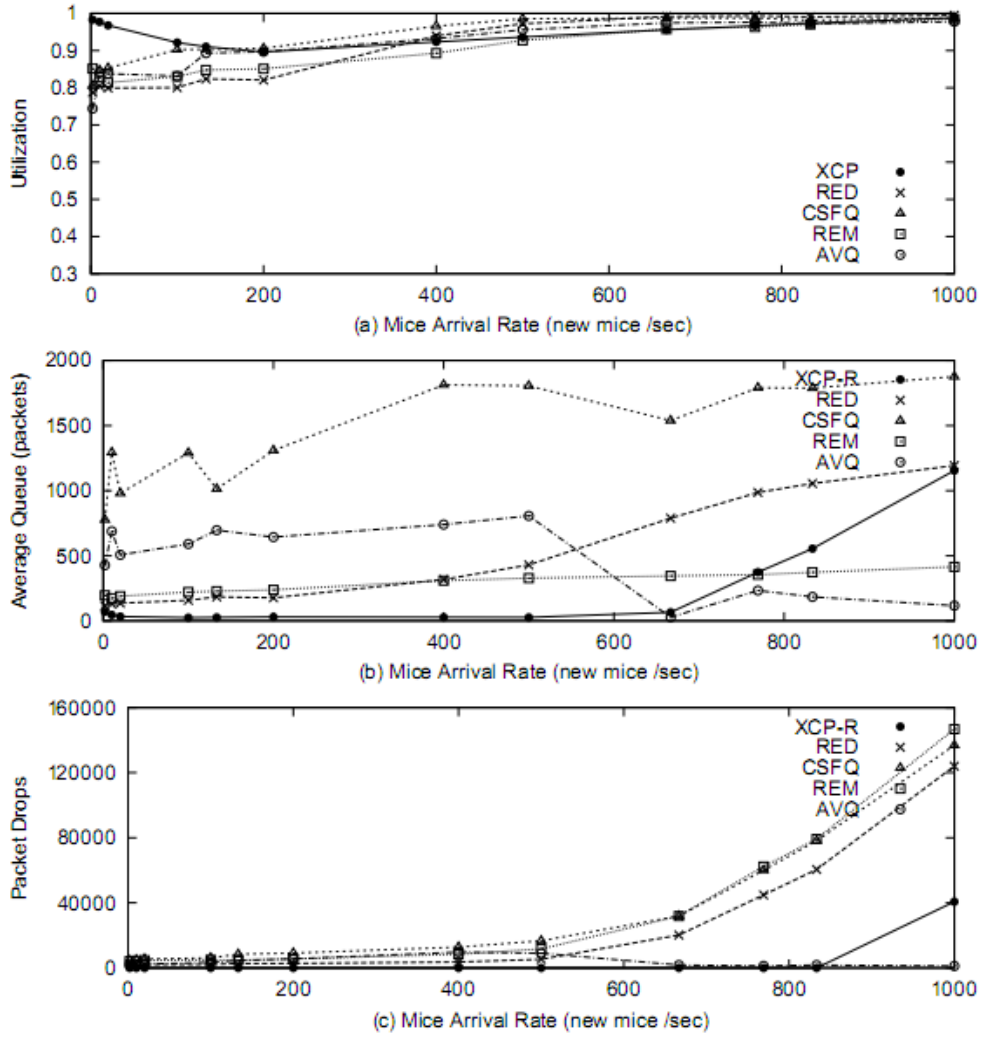


Figure 7: XCP is robust and efficient in environments with arrivals and departures of short web-like flows. The graphs compare the efficiency of XCP to that of TCP over various queuing schemes as a function of the arrival rate of web-like flows.

New Slide

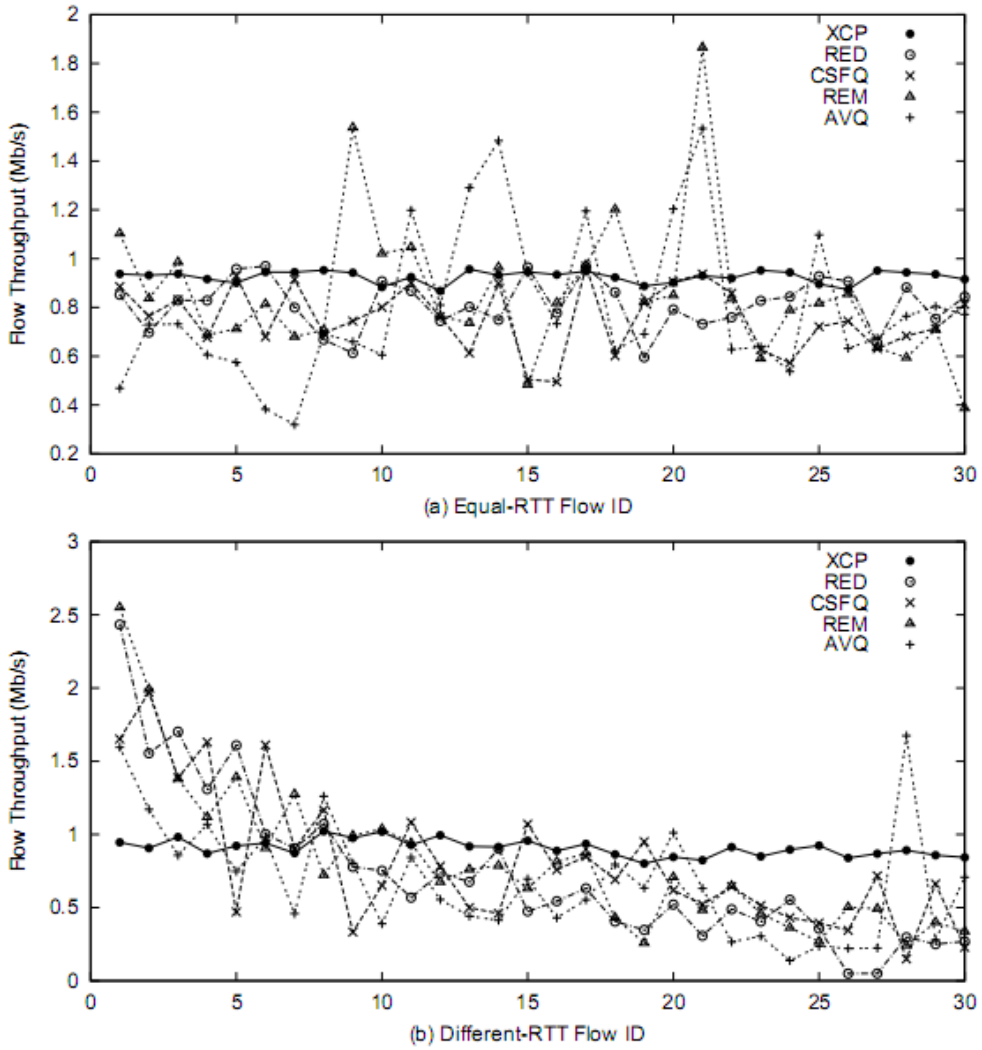


Figure 8: XCP is fair to both equal and different RTT flows. The graphs compare XCP's Fairness to that of TCP over RED, CSFQ, REM, and AVQ. Graph (b) also shows XCP's robustness to environments with different RTTs.

New Slide

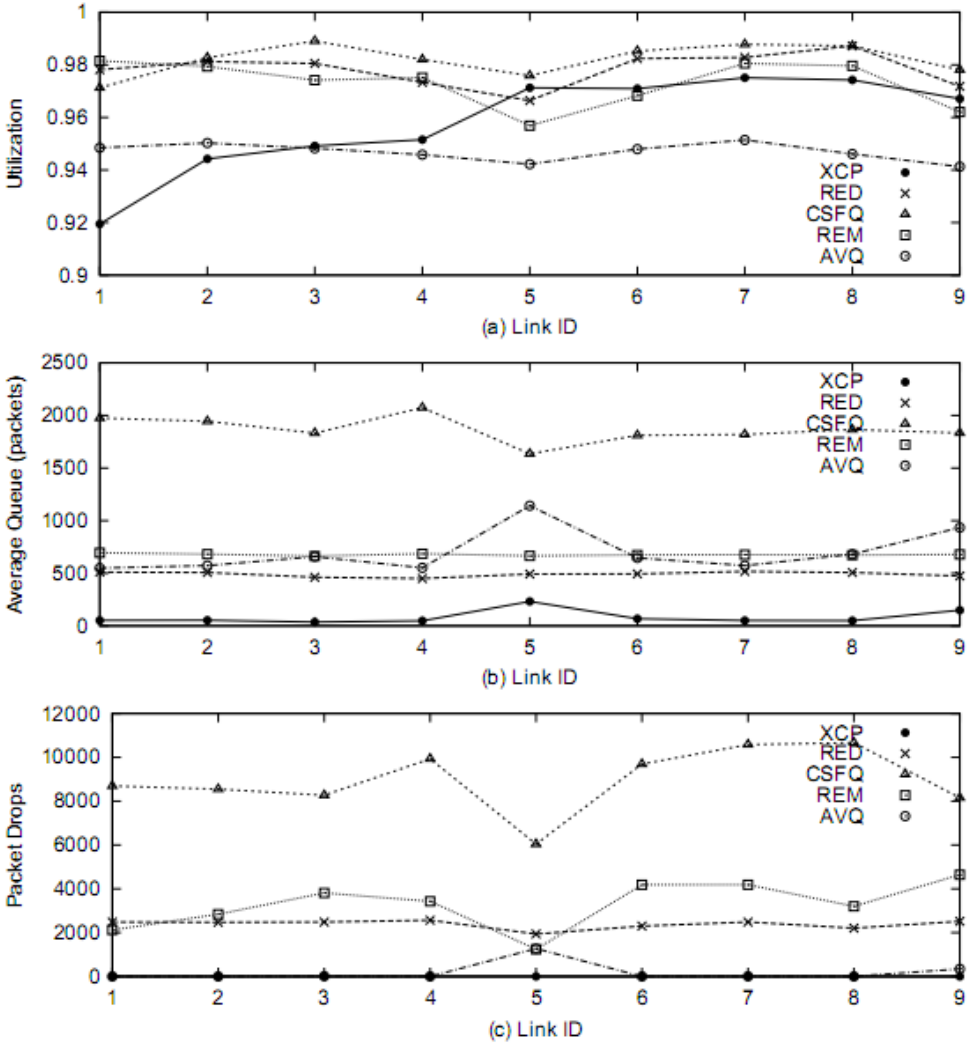


Figure 9: Simulation with multiple congested queues. Utilization, average Queue size, and number of drops at nine consecutive links (topology in Figure 3). Link 5 has the lowest capacity along the path.

Quick convergence to fairness

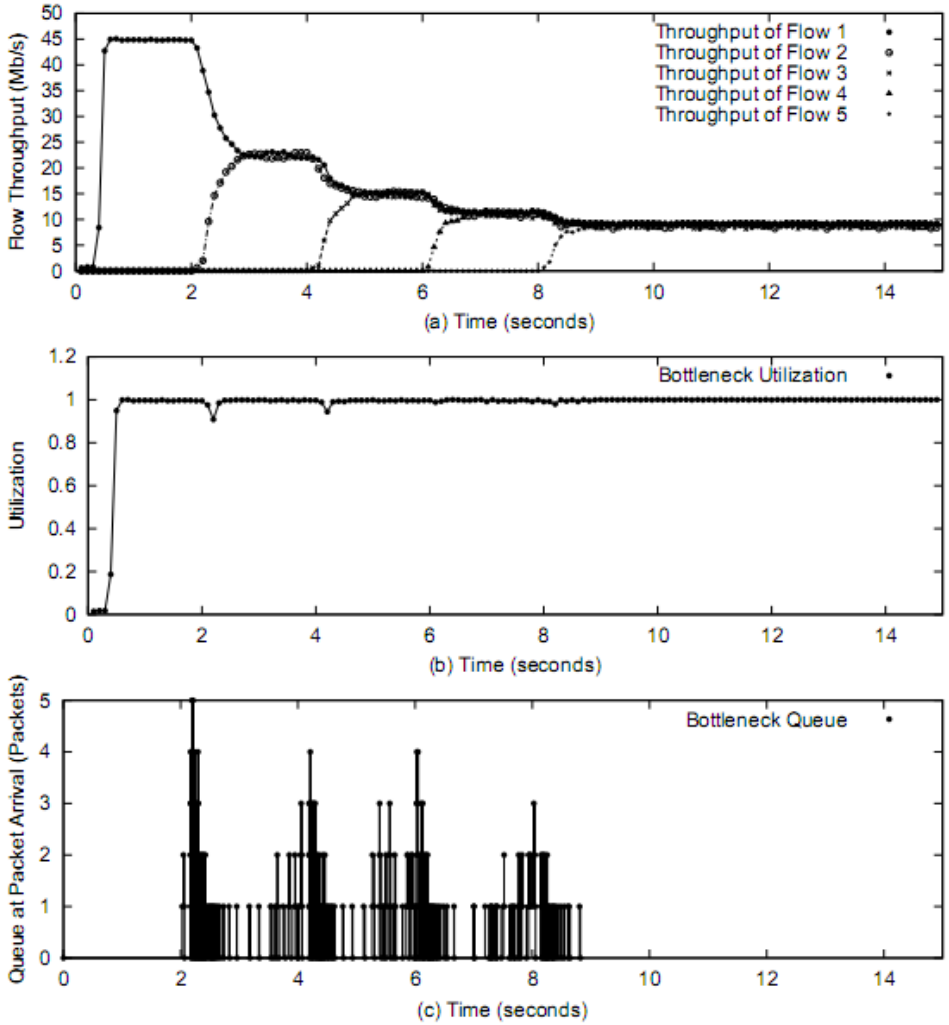


Figure 10: XCP's smooth convergence to high fairness, good utilization, and small queue size. Five XCP flows share a 45 Mb/s bottleneck. They start their transfers at times 0, 2, 4, 6, and 8 seconds.

Quick and stable asjustment to suddent changes

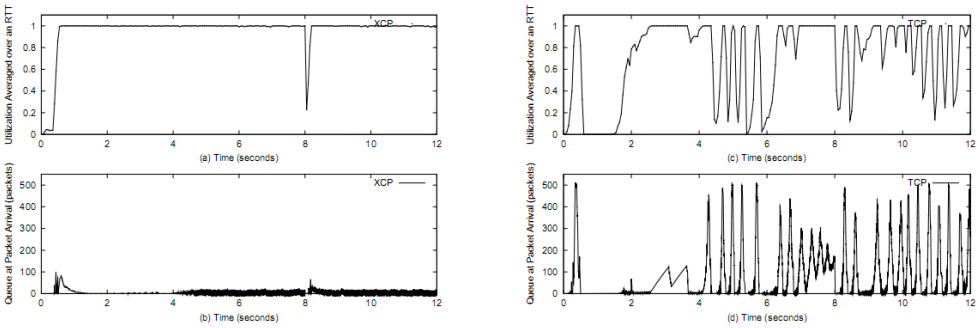


Figure 11: XCP is more robust against sudden increase or decrease in traffic demands than TCP. Ten FTP flows share a bottleneck. At time $t = 4$ seconds, we start 100 additional flows. At $t = 8$ seconds, these 100 flows are suddenly stopped and the original 10 flows are left to stabilize again.

Non-fair bandwidth allocation, assign $\xi_i \propto \$$

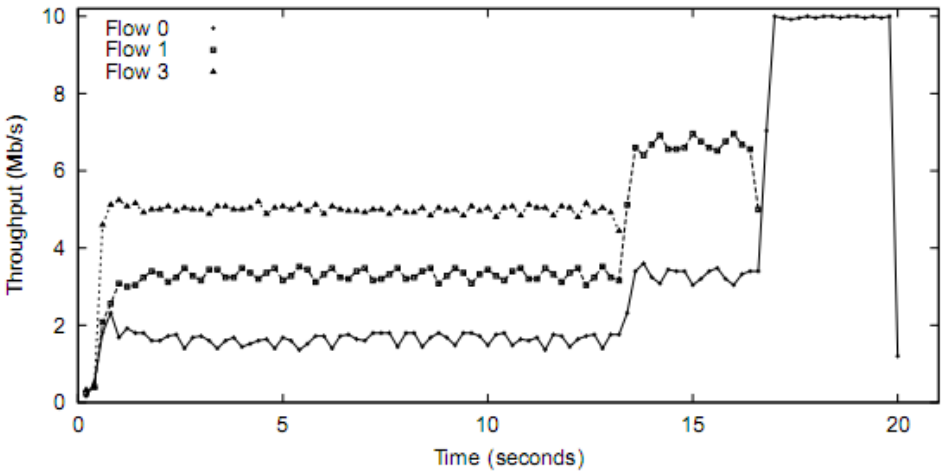


Figure 12: Providing differential bandwidth allocation using XCP. Three XCP flows each transferring a 10 Mbytes file over a shared 10 Mb/s bottleneck. Flow 1' s price is 5, Flow 2' s price is 10, and Flow 3' s price is 15. Throughput is averaged over 200 ms (5 RTTs).

XCP is TCP-friendly

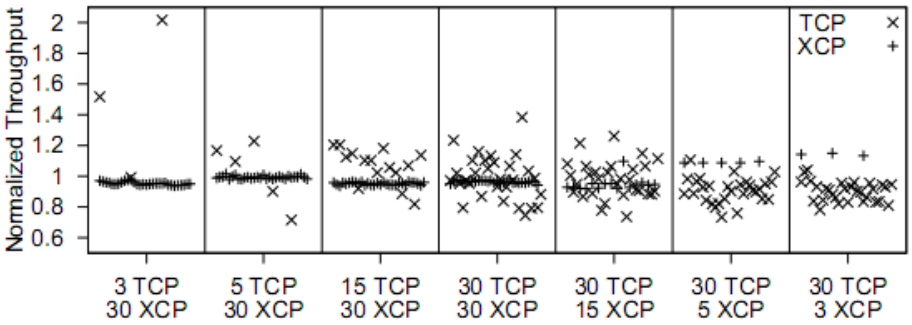


Figure 13: XCP is TCP-friendly.

Robustness to high variance to round trip time

This is not a good challenge of this weakness of the design of XCP. Large variance in RTT will make it difficult for XCP to adjust to the dynamic of flows which are far away from the average RTT. Yet, there are no dynamics in this simulation.

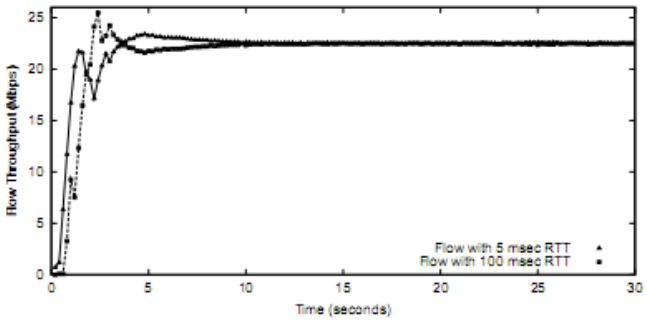


Figure 16: XCP robustness to high RTT variance. Two XCP flows each transferring a 10 Mbytes file over a shared 45 Mb/s bottleneck. Although the first flow has an RTT of 20 ms and the second flow has an RTT of 200 ms both flows converge to the same throughput. Throughput is averaged over 200 ms intervals.