

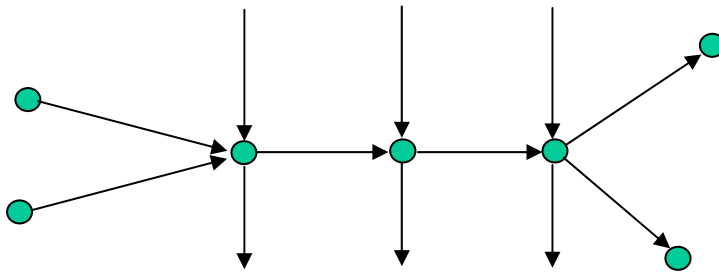
Recent Developments in TCP

D.J.Leith
Hamilton Institute
NUI Maynooth

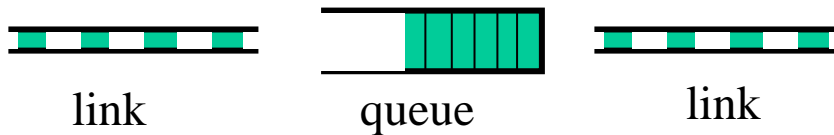


The Need for Congestion Control

A network consists of a number of sources and sinks that communicate via network links ('wires') and routers ('queues').



Packets are sent from sources to sinks and 'in flight' packets are on the wire or in the network queues.

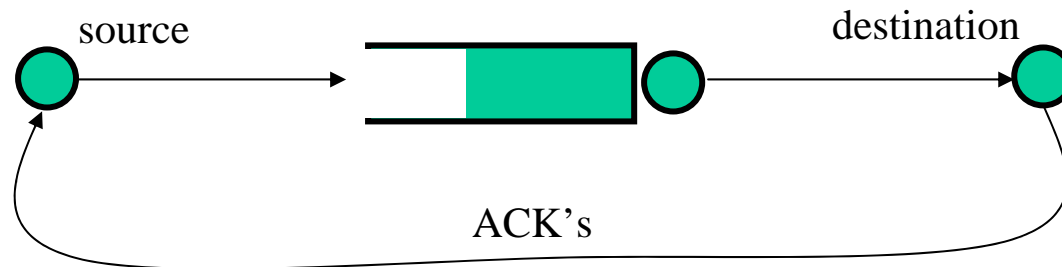


Suppose one queue is serviced at a rate that is less than the others. This is the bottleneck queue ('Westlink bridge').



The Need for Congestion Control: Reliable Delivery & Congestion Collapse

Internet data traffic (email, web, ftp etc) requires reliable delivery.



TCP is method of choice. Uses acknowledgements to ensure reliable packet delivery. Missing acknowledgements are used to infer lost packets and to initiate a resend – many versions of TCP.

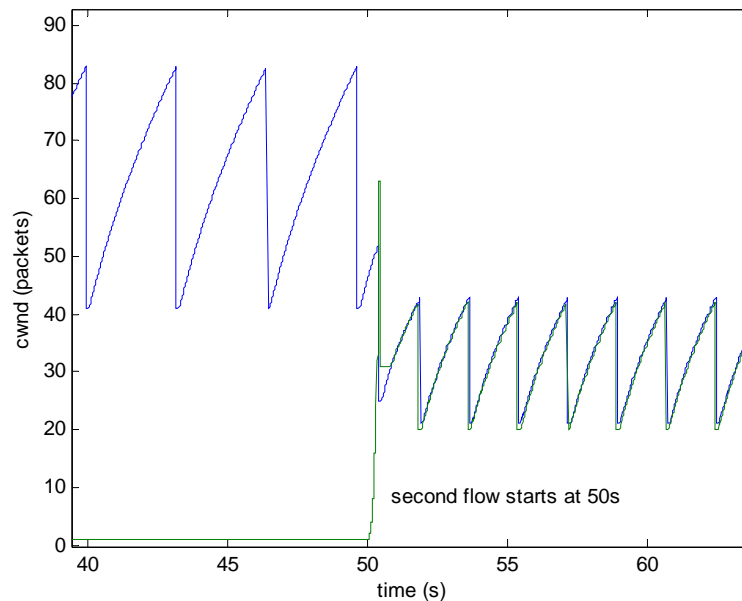
- Congestion collapse “work done by network falls as offered load is increased” Sources need to adapt to network conditions and back-off their send rate when the network becomes congested. Current TCP uses packet drop to indicate congestion.



The Need for Congestion Control: Key Objectives

- Efficiency** The requirement is for reasonable efficiency across a wide range of network conditions is required i.e. *robust efficiency*.
- Fairness** When traffic from >1 source shares the network, sources should adapt their send rate to avoid seizing an excessive share of the network bandwidth.
- Responsiveness** A congestion control algorithm should adjust its send rate rapidly in response to changing conditions (number of users changing, bandwidth changing etc).

e.g.



Drop-Tail Queues



Queue is constrained to have a lower limit of zero and upper limit of q_{\max} .

Send rate less than link bandwidth: queue sits at lower limit, no packets are dropped and packets arrive at destination at (roughly) same rate at which they were sent.

Send rate greater than link bandwidth: queue fills until upper limit is reached and packets are then dropped.

⇒ drops provide feedback when we persistently exceed link bandwidth.

⇒ transition where queue is neither empty nor overflowing – may be a complex middle ground e.g. may not have a single bottleneck queue but a varying one as cross-flows vary.



Information Constraints: Scalability

- Decentralised operation (no information sharing between sources).
- None of the sources are aware of the available bandwidth (communication between network and sources is constrained).



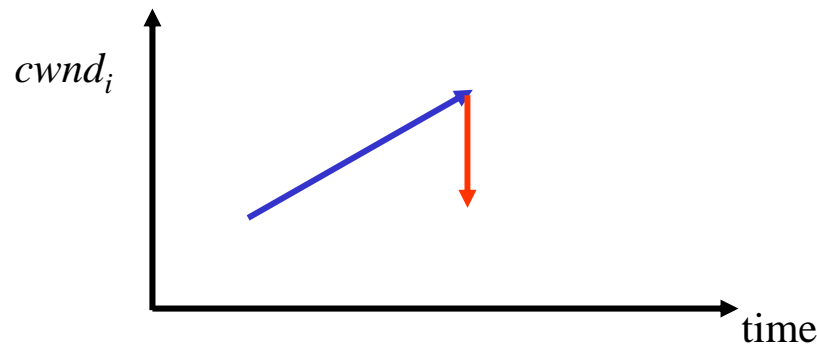
Outline of current TCP congestion control algorithm

Additive-Increase Multiplicative-Decrease:

Probing seems essential in view of queue properties (no feedback as to link bandwidth until the queue starts to fill). TCP adopts a linear increase law- when source i receives an ACK packet it increases its window size $cwnd_i$ by

$$cwnd_i \rightarrow cwnd_i + \alpha_i / cwnd_i$$

with increase parameter $\alpha_i=1$ in standard TCP.



When the window size eventually exceeds the “pipe” capacity, a packet will be dropped. When the source detects this (after a delay), the window size is reduced:

$$cwnd_i \rightarrow \beta_i cwnd_i$$

with decrease parameter $\beta_i=0.5$ in standard TCP.

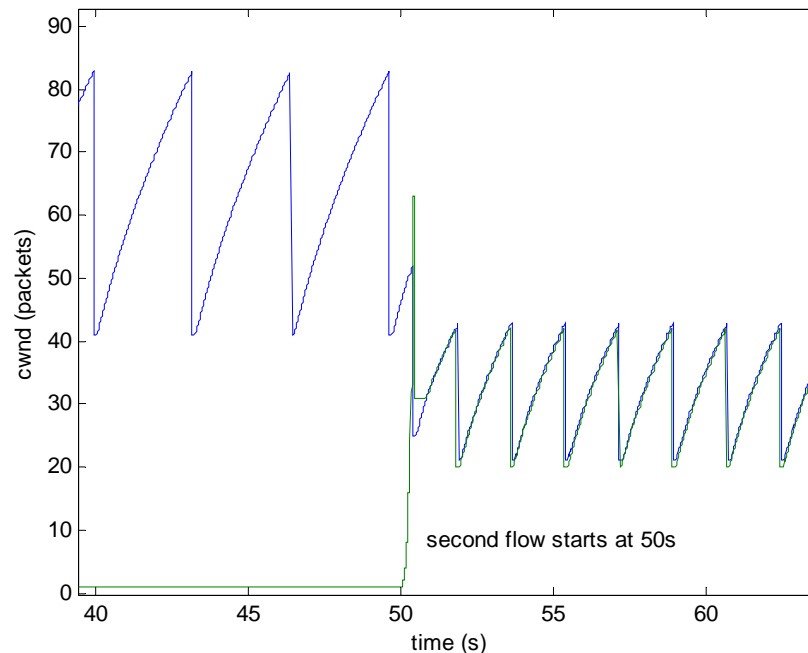


Outline of current TCP congestion control algorithm

Additive-Increase:

On each ACK, $cwnd_i \rightarrow cwnd_i + \alpha_i / cwnd_i$

- Have $cwnd_i$ packets in flight – effect of additive increase is to increase $cwnd_i$ by α_i packets each round-trip time (RTT).
- RTT is time-varying due to queues filling/emptying.



High-speed Networks

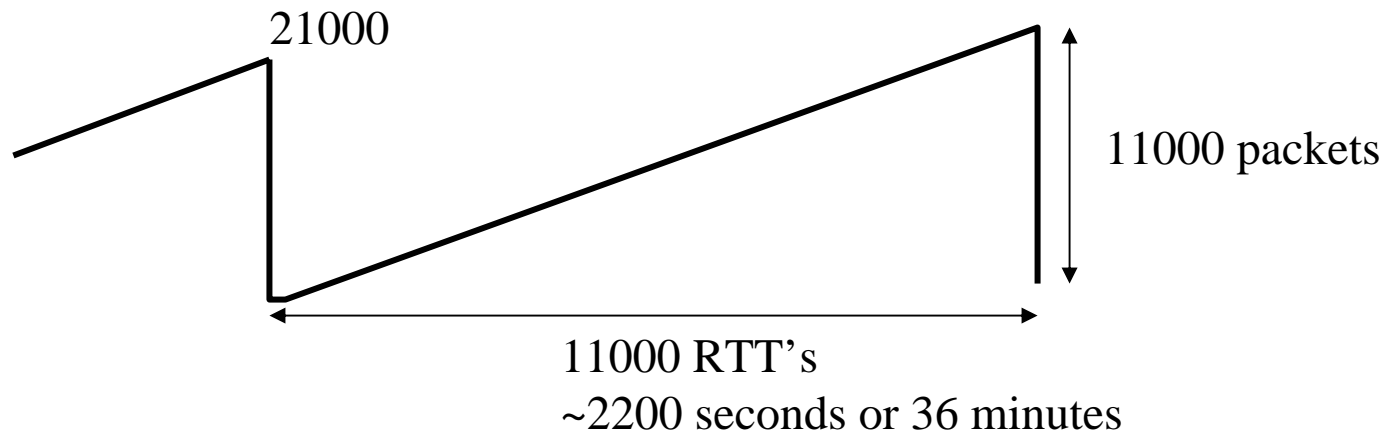
The pipe size of a link is roughly $BT+q_{\max}$

where B is the the link rate (packets/s), T is the propagation delay and q_{\max} is the queue size.

On a long distance gigabit link, $B=100,000$ packets/s, $T=200\text{ms}$, $q_{\max}=1000$ and

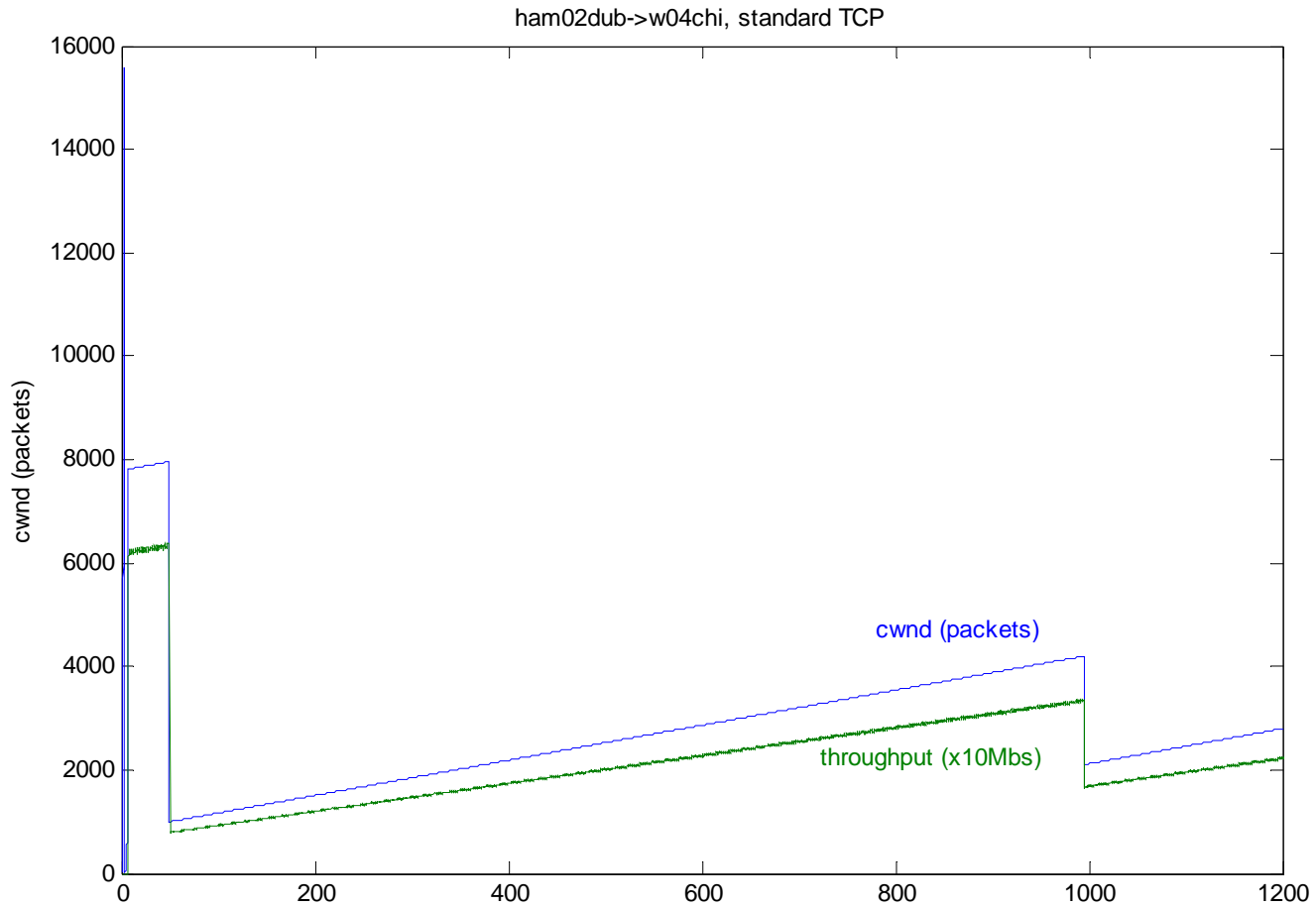
$$BT+q_{\max}=21,000$$

Note that the pipe size determines the peak window size of a TCP source.



- TCP becomes sluggish, and requires v.low drop rate to achieve reasonable throughput.





218Mbps mean throughput over 20 mins – link bandwidth is 1Gbs

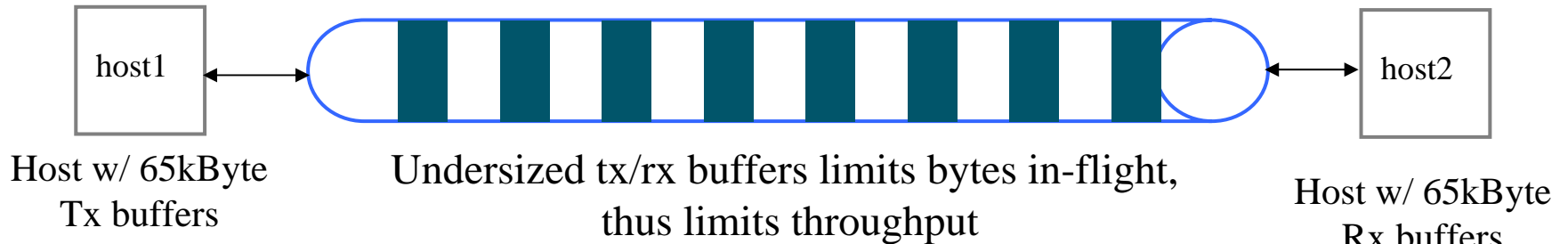


Short Digression ...

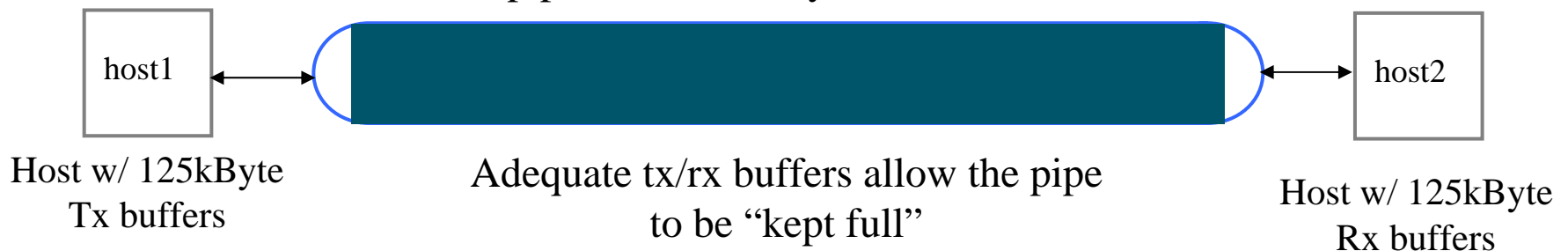
Most operating systems “out of the box” allocate send/receive buffers of 64Kb or less.

Path bandwidth*delay: 100Mbps, 1800km \rightarrow 125kBytes

“pipe” with 125kByte “volume”



“pipe” with 125kByte “volume”

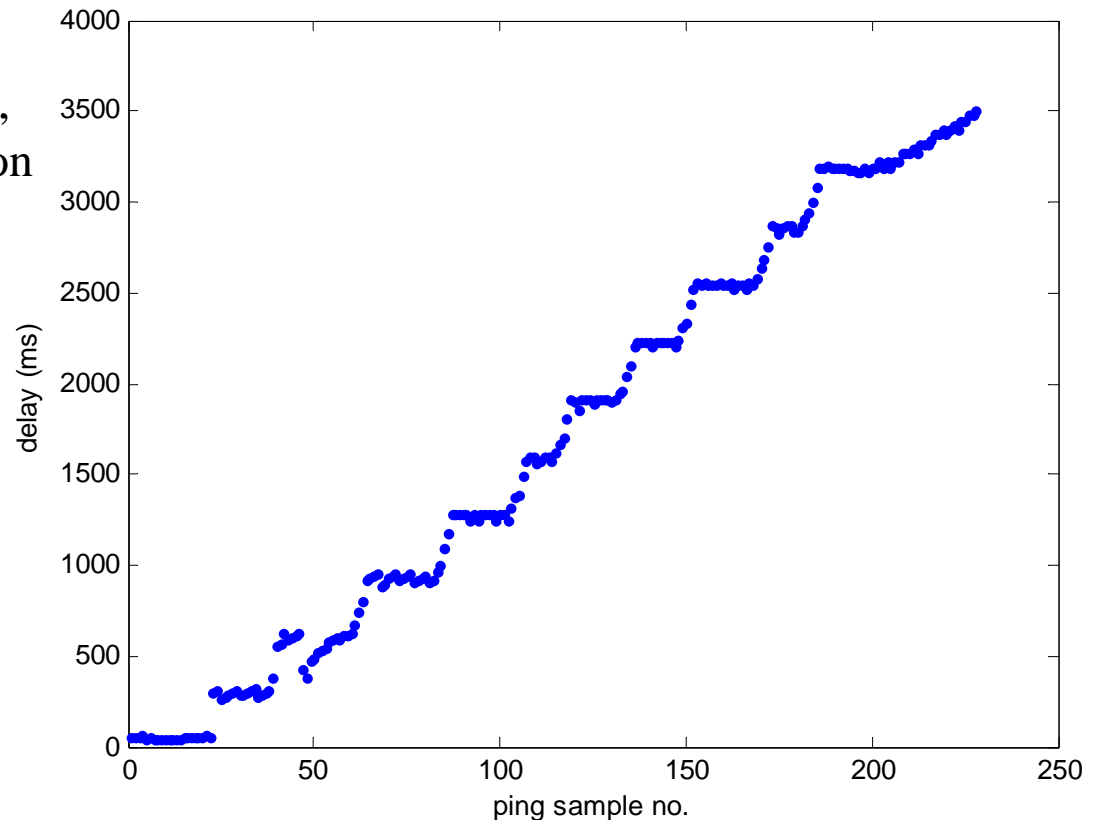


Short Digression ...

Why allocate small buffers ?

- buffers use kernel (physical) memory. A buffer is allocated for every connection and may have many connections e.g. web server.

- low-speed connections (modems, DSL) often have v. large queues on line card at exchange
⇒ Probing to fill queue is a bad idea.



High-speed Networks

Rather than a complete redesign of TCP, is it possible to devise a small modification that fixes it in high-speed networks ?

Simply making the increase parameter α larger is inadmissible – on low-speed networks we require backward compatibility with current sources.

Large α in high-speed regimes, $\alpha=1$ in low-speed regimes suggests some sort of mode switch.

One approach – Scalable TCP – use a multiplicative increase rule and smaller backoff parameter β .

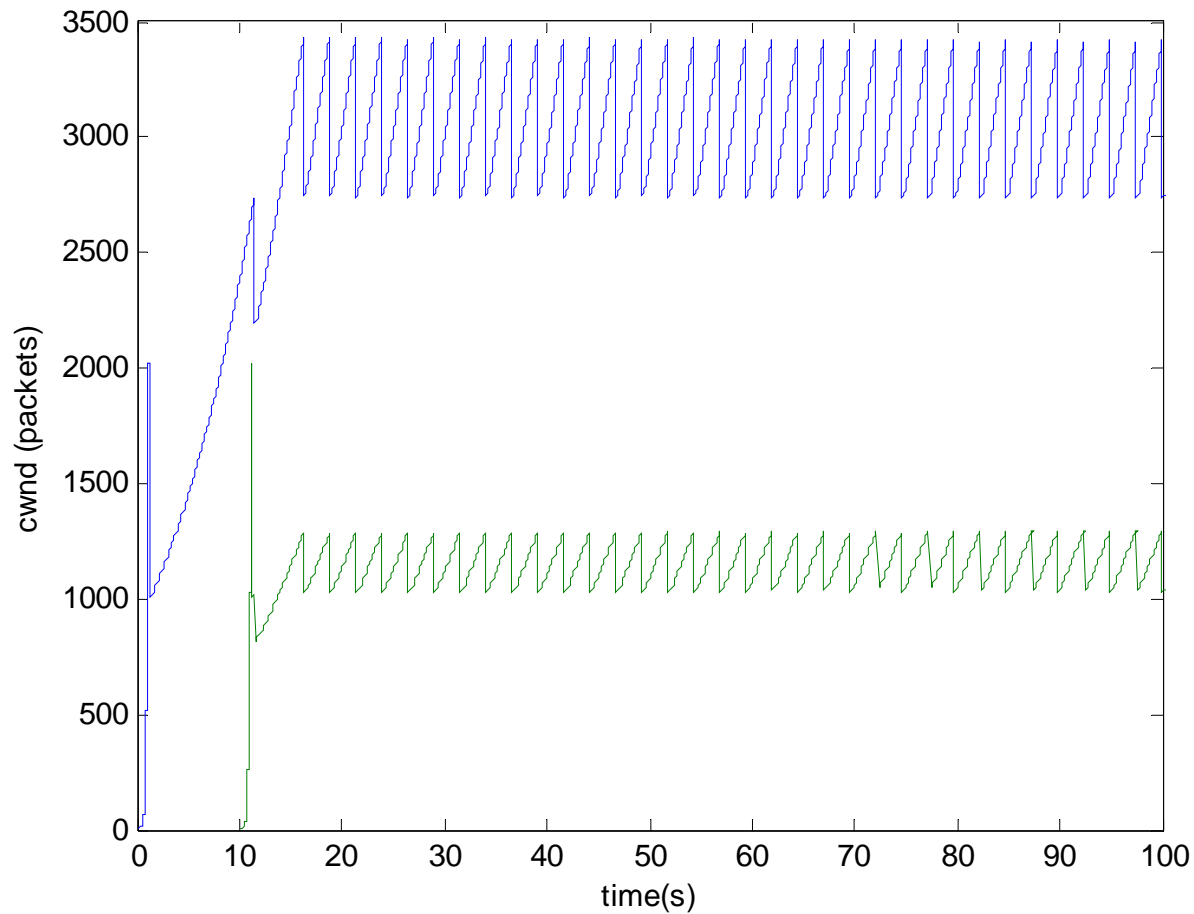
Another – HS-TCP – is to vary AIMD parameters as a function of *cwnd* ...

(increase α , decrease β as *cwnd* becomes large)

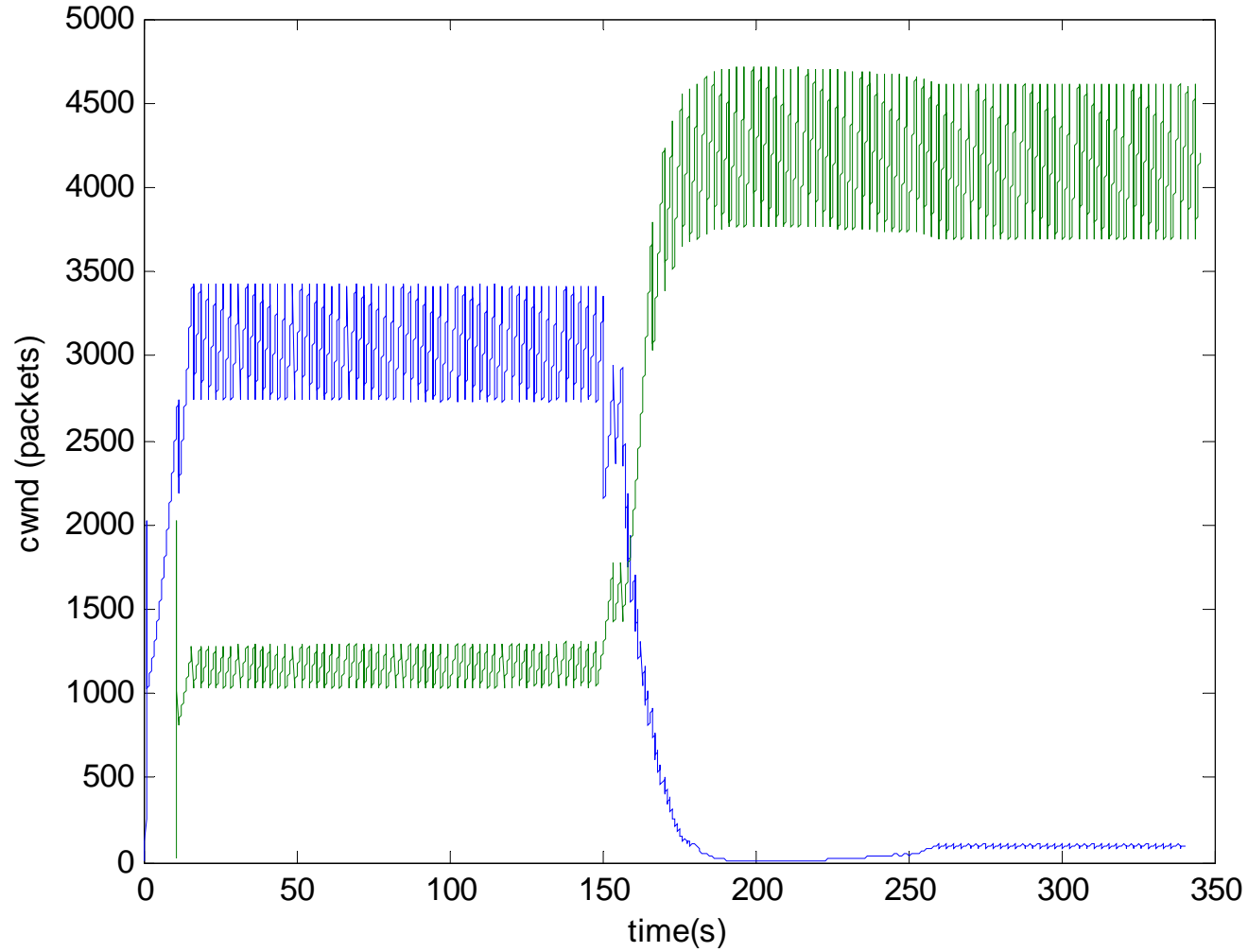


High-speed Networks **Scaleable TCP**

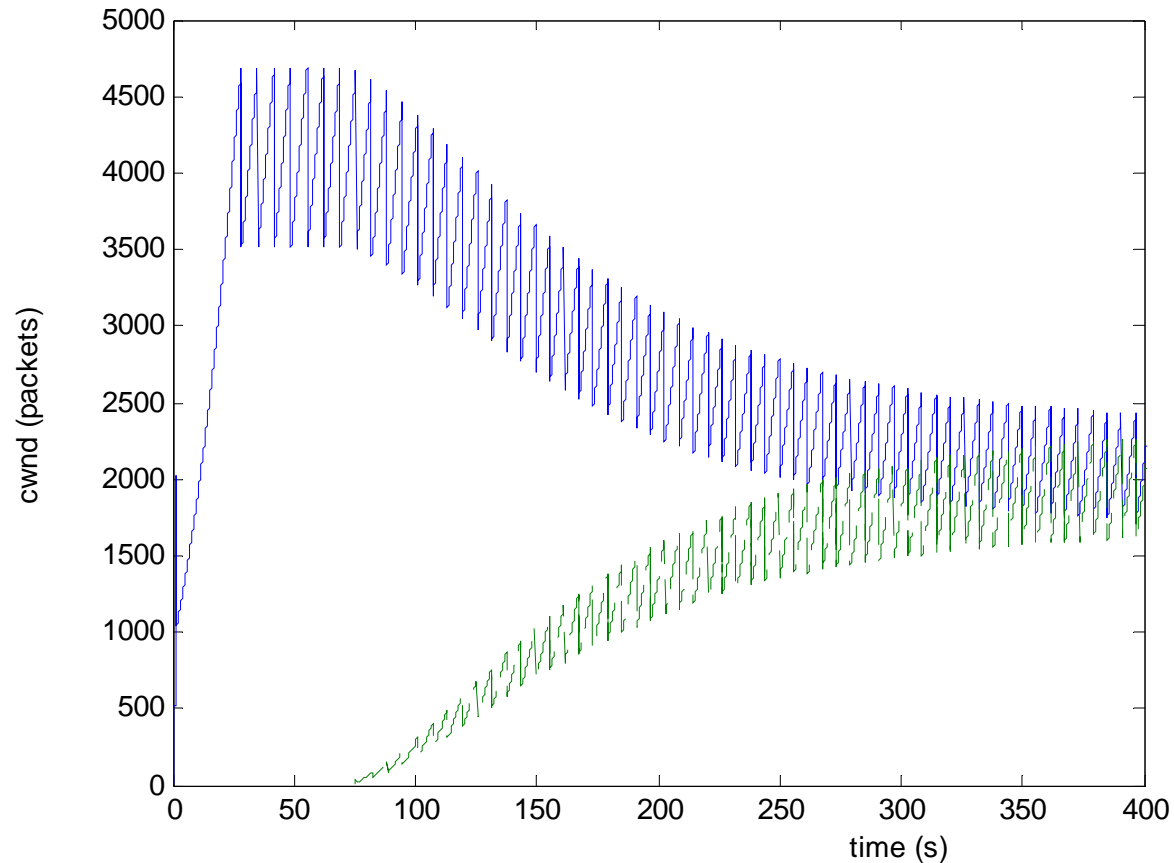
Scaleable TCP has convergence issues ...



High-speed Networks **Scaleable TCP**



High-speed Networks HS-TCP also has convergence issues

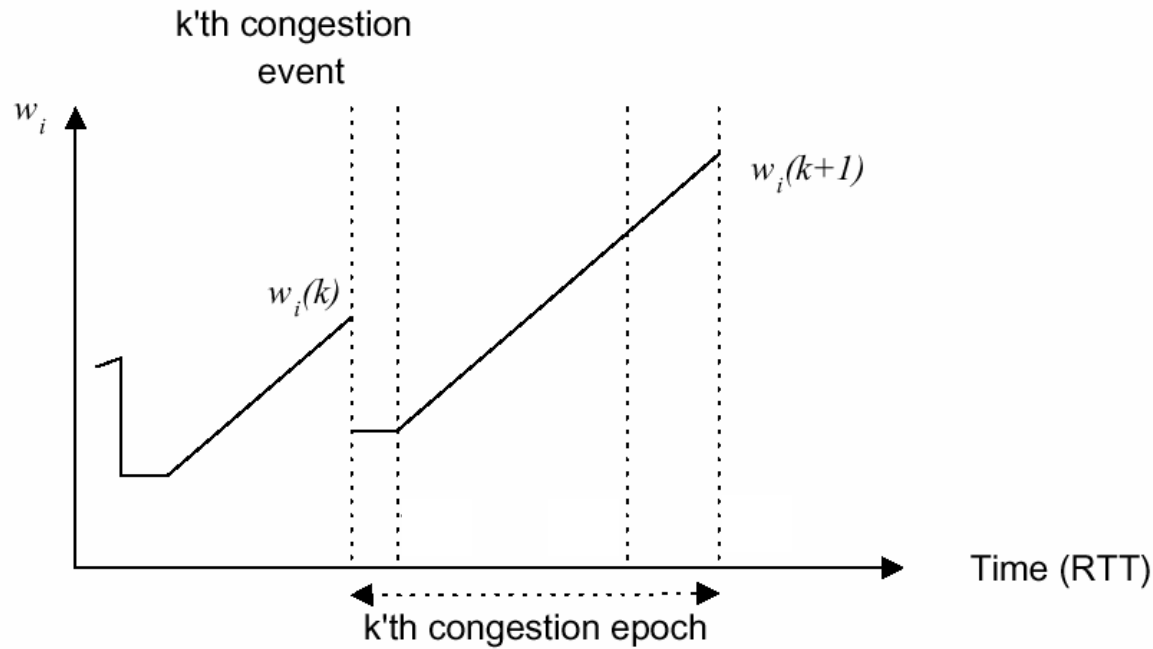


Example of two HS-TCP flows - the second flow experiences a drop early in slow-start focussing attention on the responsiveness of the congestion avoidance algorithm.

(NS simulation: 500Mb bottleneck link, 100ms delay, queue 500 packets)



Current TCP congestion control algorithm revisited.

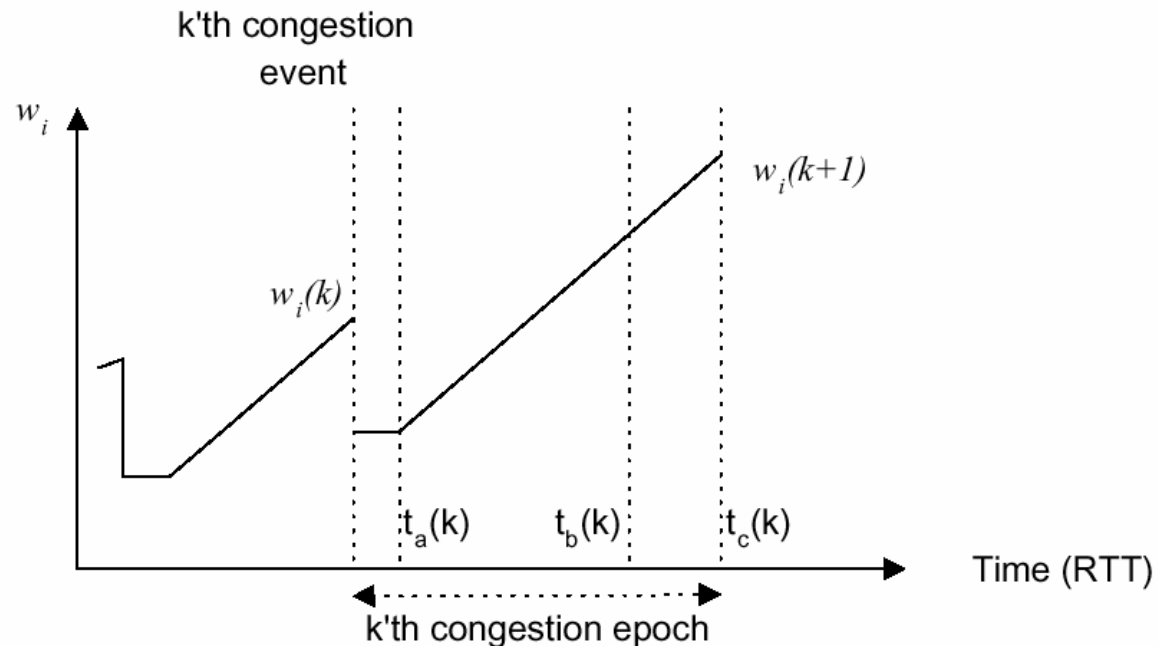


Note: $cwnd$ never converges to a steady value with this probe/back-off approach. Also, we are ignoring slow-start, timeout's etc here so as to focus on the congestion avoidance behaviour.



Synchronisation Model

Typical congestion window evolution for a TCP source in congestion avoidance:



Synchronisation assumption: t_a , t_b , t_c are the same for all sources.

e.g. when a shared bottleneck link, RTT is the same for all sources, each source transmits at least one packet every RTT ($\alpha \geq 1$)



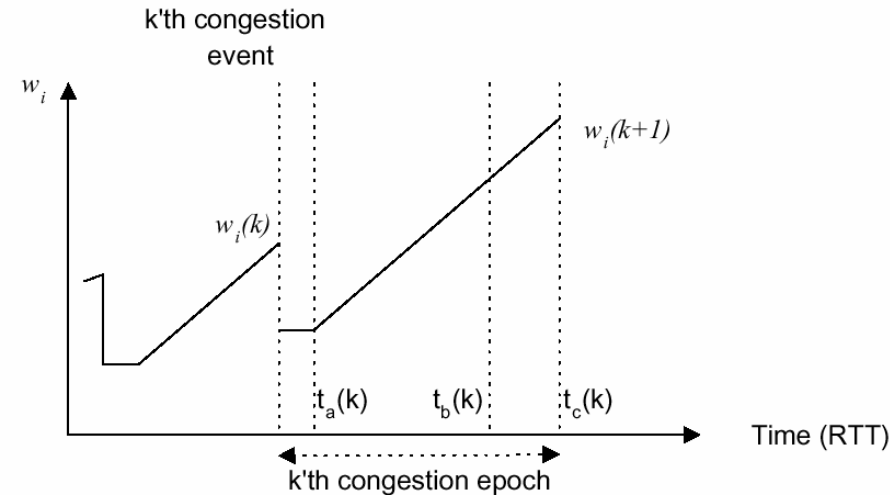
Synchronisation Model

The source congestion windows are subject to constraints:

$$w_i \geq 0, \sum_{i=1}^n w_i = P + \sum_{i=1}^n \alpha_i$$

Number of packets in pipe is non-negative

At congestion, total number of packets in pipe matches pipe size, P



For source i we have:

$$w_i(k+1) = \beta_i w_i(k) + \alpha_i [t_c(k) - t_a(k)]$$

$$t_c(k) - t_a(k) = \frac{1}{\sum_{i=1}^n \alpha_i} [P - \sum_{i=1}^n \beta_i w_i(k)] + 1$$



Synchronisation Model

Collecting the evolution equations for all n sources yields the network dynamics:

$$W(k+1) = AW(k)$$

where $W^T(k) = [w_1(k), \dots, w_n(k)]$ is the vector of window sizes at congestion and

$$A = \begin{bmatrix} \beta_1 & 0 & \cdots & 0 \\ 0 & \beta_2 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \beta_n \end{bmatrix} + \frac{1}{\sum_{j=1}^n \alpha_j} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \cdots \\ \alpha_n \end{bmatrix} [1 - \beta_1 \quad 1 - \beta_2 \quad \cdots \quad 1 - \beta_n]$$

where α_i is the AIMD increase parameter for source i , β_i the decrease parameter.

Observe that:

- The dynamics are linear
- \mathbf{A} is a positive matrix with very special structure
- This model incorporates important network features such as the hybrid nature of AIMD, time-varying delay and drop-tail queueing.



Synchronisation Model

Analysis A network of synchronised AIMD sources:

- (i) possesses a unique stationary point, $W_{ss} = \Theta x_p$ where Θ is a positive constant and
- (ii) the stationary point is globally exponentially stable. The rate of convergence depends on the second largest eigenvalue of A .

Fairness

Stationary point: $W_{ss} = \Theta x_p$ where Θ is a positive constant and $x_p^T = \gamma \left[\frac{\alpha_1}{1-\beta_1}, \dots, \frac{\alpha_n}{1-\beta_n} \right]$

$\alpha_i = \lambda(1-\beta_i) \forall i$ and for some $\lambda > 0 \Rightarrow W_{ss}^T = \Theta/n [1, 1, \dots, 1]$ i.e. $w_1 = w_2 = \dots = w_n$

For standard TCP, $\alpha=1$, $\beta=0.5$ so $\lambda=2$ and

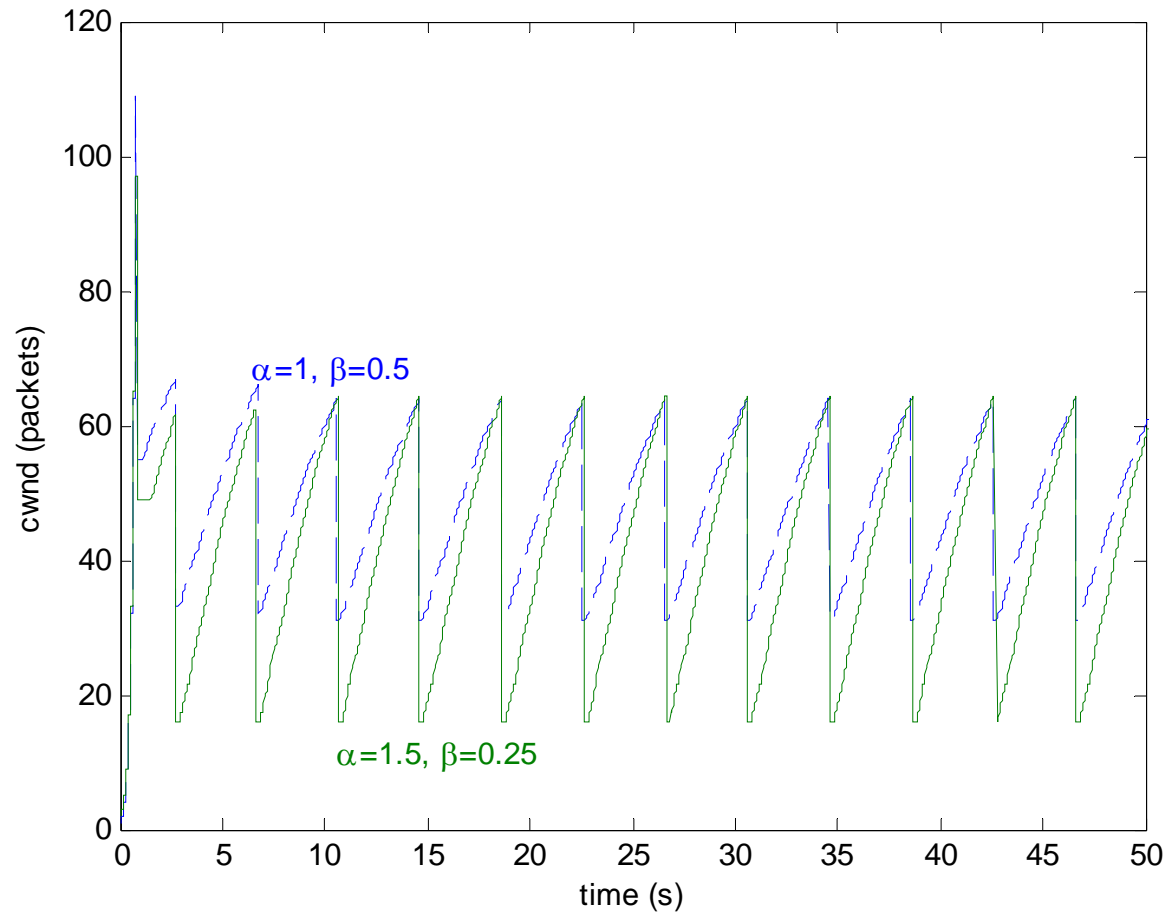
$$\alpha_i = 2(1 - \beta_i)$$

is the condition for fair co-existence of AIMD flows with TCP.



Synchronisation Model

Fairness – Example (NS simulation 10Mb link, 100ms delay, queue 40 Packets)



Synchronisation Model

Responsiveness

Special case: All of the sources have the same decrease parameter: $\beta_i = \beta \forall i$.

Then the eigenvalues of A (other than the Perron eigenvalue) are equal to β

\Rightarrow rate of convergence is β^k , where k is the congestion epoch.

95% rise time (measured in congestion epochs) is $\log(0.05)/\log \beta$

e.g. for $\beta=0.5$, rise time is 4 congestion epochs.

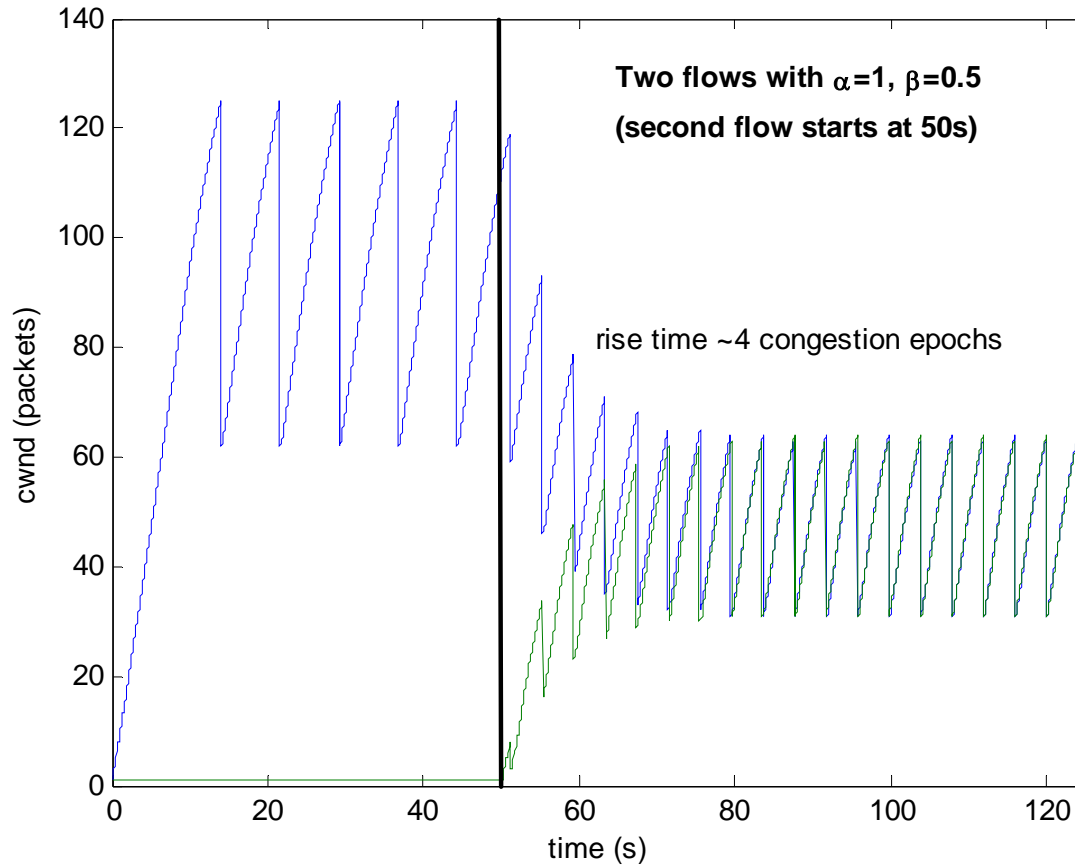
Note, *duration* of congestion epochs depends on increase parameters α_i .



Synchronisation Model

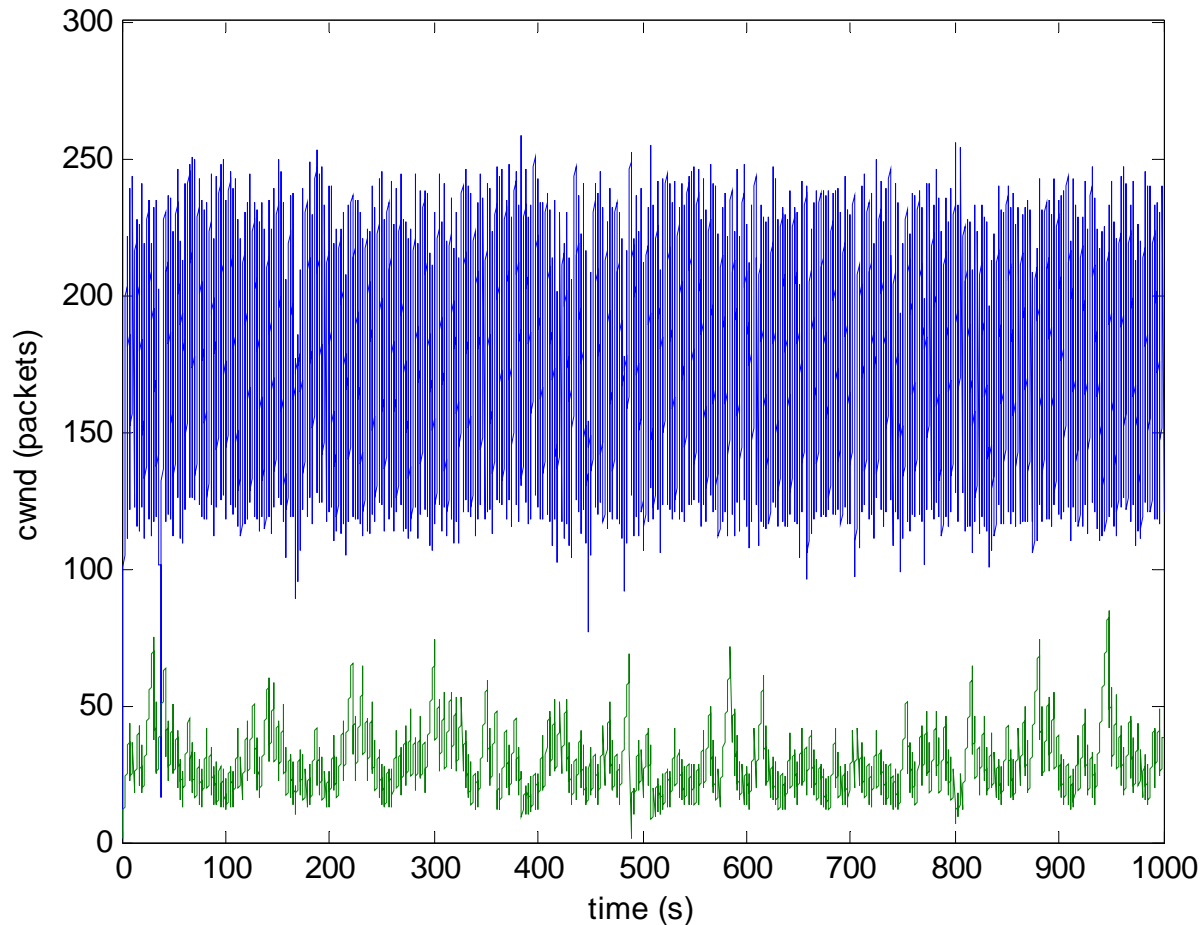
Responsiveness (cont)

e.g.



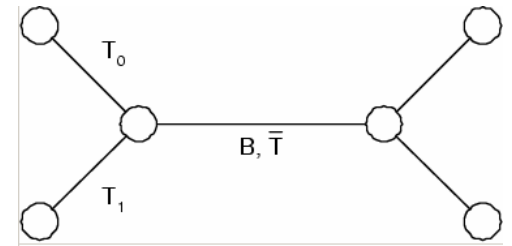
Unsynchronised TCP Flows ?

Example: Congestion window time histories ($B=100\text{Mb}$, $T_0=20\text{ms}$, $T_1=2\text{ms}$, $T_2=162\text{ms}$, queue 80 packets)

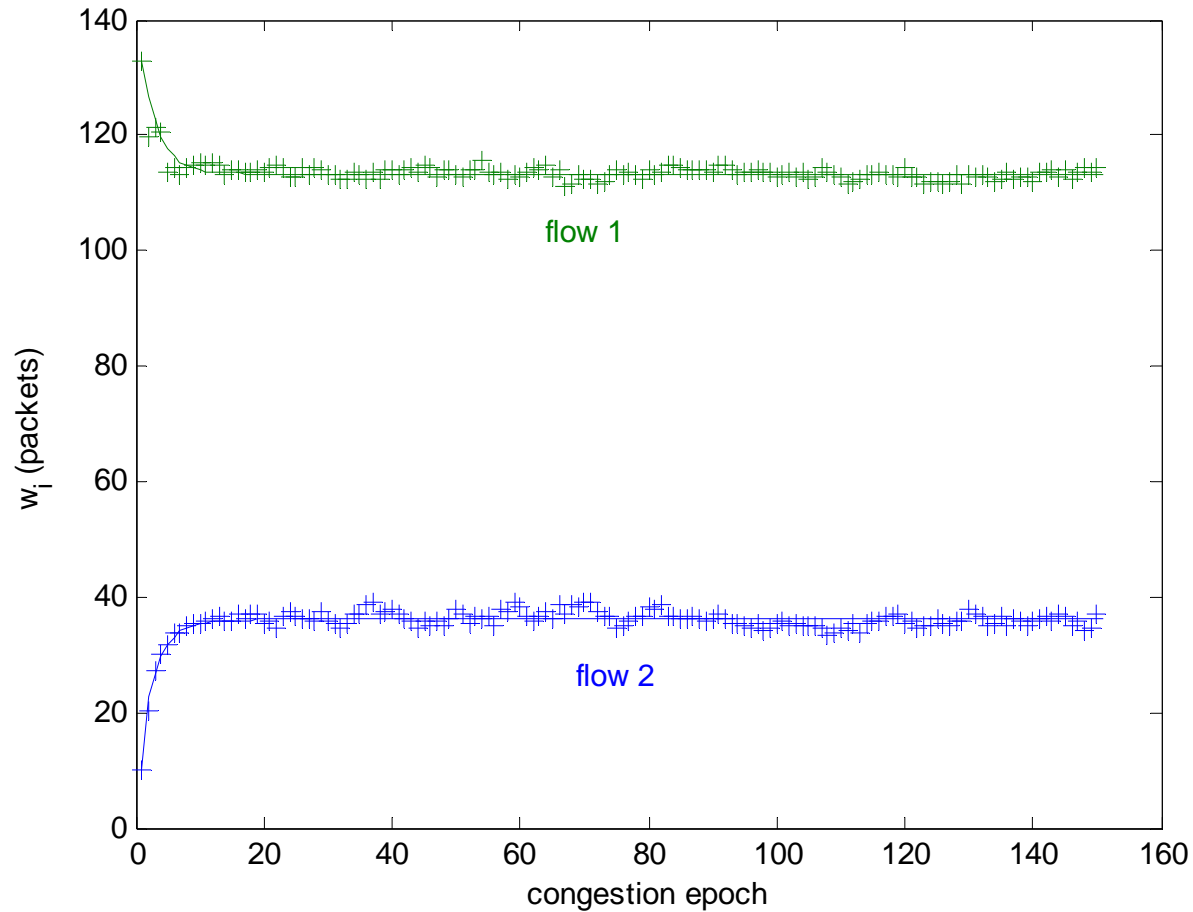


Unsynchronised TCP Flows

Previous analysis carries over provided we work in terms of average congestion windows ...

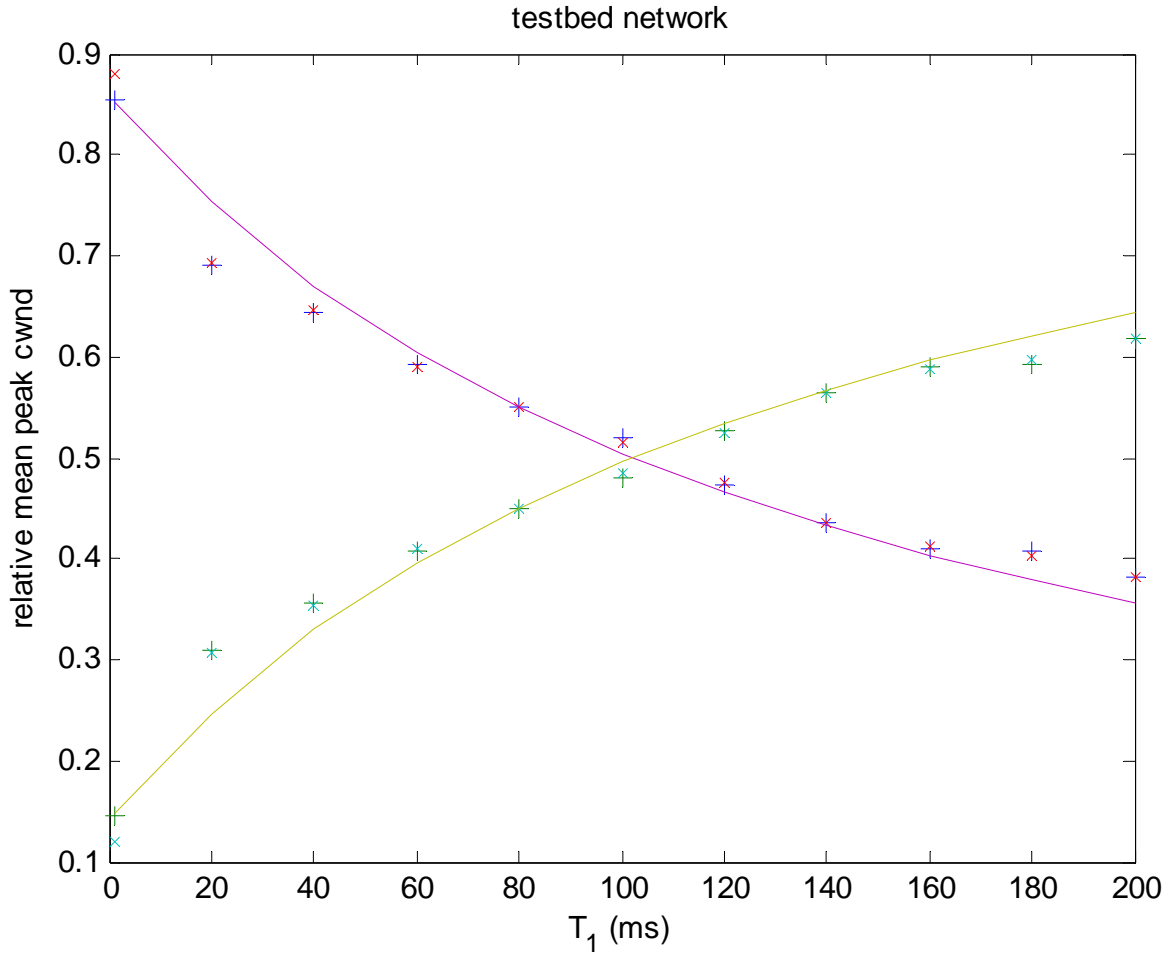
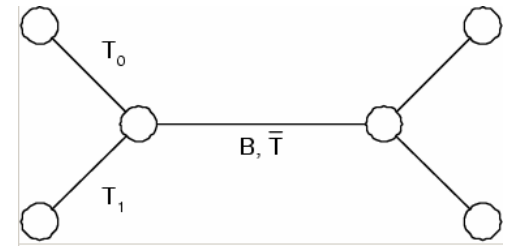


Ensemble Average



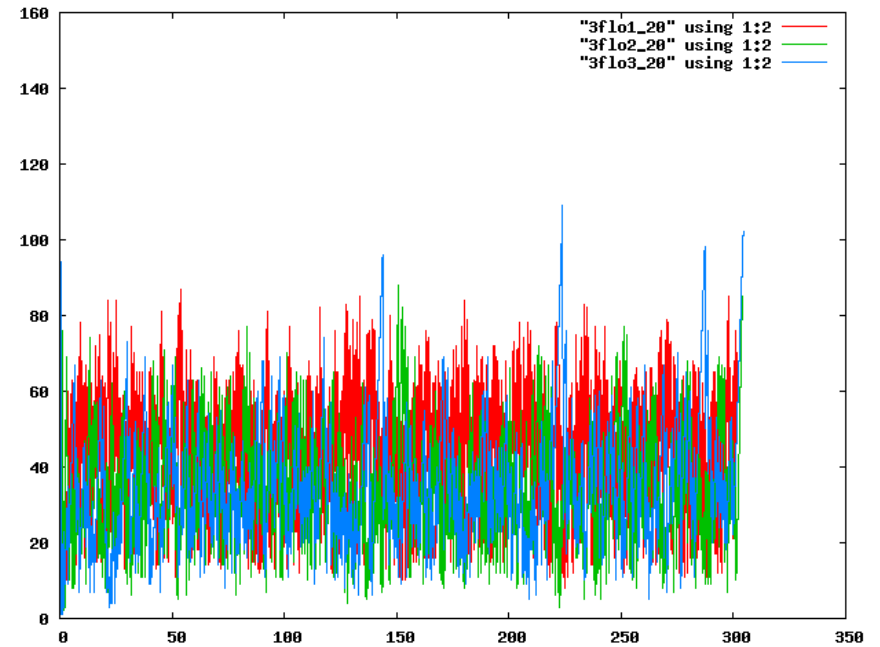
Unsynchronised TCP Flows

Time Average



Unsynchronised TCP Flows

Queue		rtt=21ms	rtt=34ms	rtt=55ms
20	Measurement	0.4055	0.3014	0.2931
	Theorem 3.3	0.4054	0.3061	0.2886
	% difference	0.0247	1.5594	1.5353
40	Measurement	0.4122	0.2849	0.3029
	Theorem 3.3	0.4121	0.2915	0.2964
	% difference	0.0243	2.3166	2.1459
60	Measurement	0.4024	0.3093	0.2882
	Theorem 3.3	0.4204	0.3087	0.2709
	% difference	4.4732	0.1940	6.0028
80	Measurement	0.416	0.3293	0.2547
	Theorem 3.3	0.3835	0.2891	0.3274
	% difference	7.8125	12.2077	28.5434

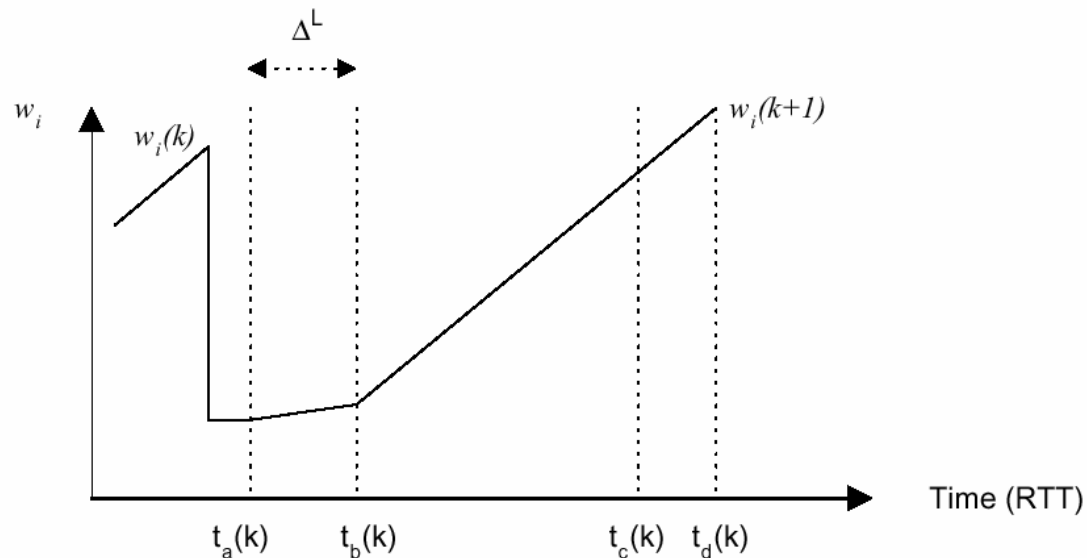


H-TCP

Simply making the increase parameter α larger is inadmissible – on low-speed networks we require backward compatibility with current sources.

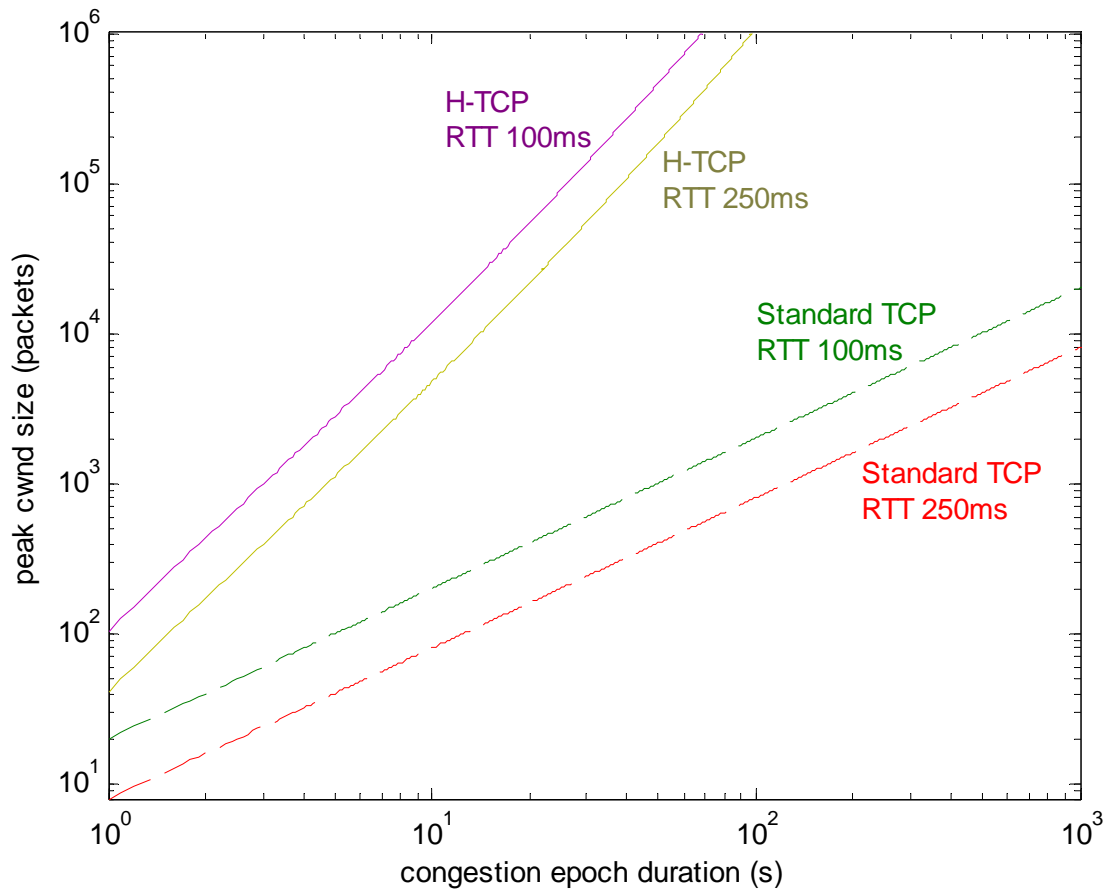
Large α in high-speed regimes, $\alpha=1$ in low-speed regimes suggests some sort of mode switch.

E.g.
$$\alpha_i = \begin{cases} \alpha_i^L & \Delta_i \leq \Delta^L \\ \alpha_i^H(\Delta_i) & \Delta_i \geq \Delta^L \end{cases}$$
 where Δ_i is the time since the last backoff.



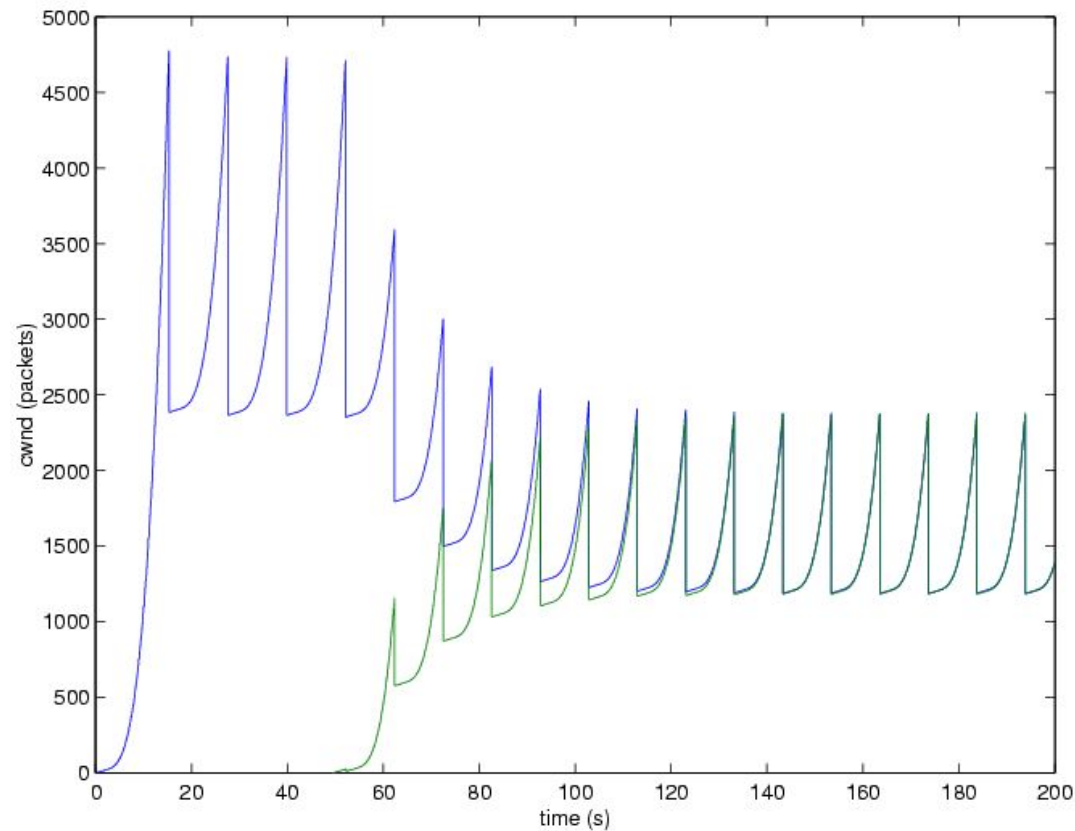
H-TCP

$$\alpha_i^H(\Delta_i) = 1 + 10(\Delta_i - \Delta^L) + \left(\frac{\Delta_i - \Delta^L}{2}\right)^2$$



H-TCP

Rate of convergence



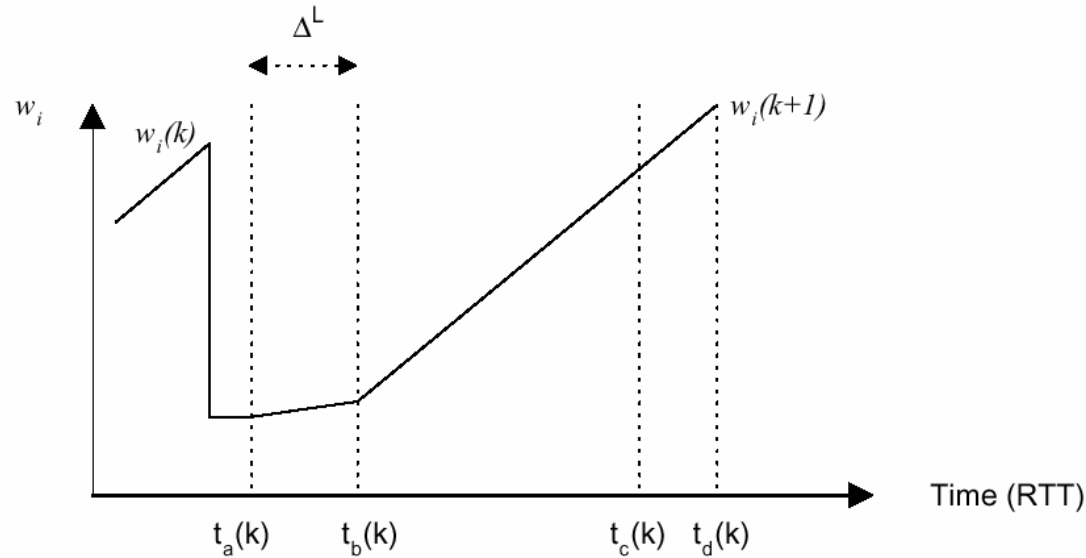
Example of two H-TCP flows illustrating rapid convergence to fairness - the second flow experiences a drop early in slow-start focusing attention on the responsiveness of the congestion avoidance algorithm.

(NS simulation: 500Mb link, 100ms delay, queue 500 packets; H-TCP parameters: $\alpha^L=1$, $\alpha^H=20$, $\beta=0.5$, $\Delta^L=19$ – corresponding to window size threshold of 38)



H-TCP

Backward compatibility

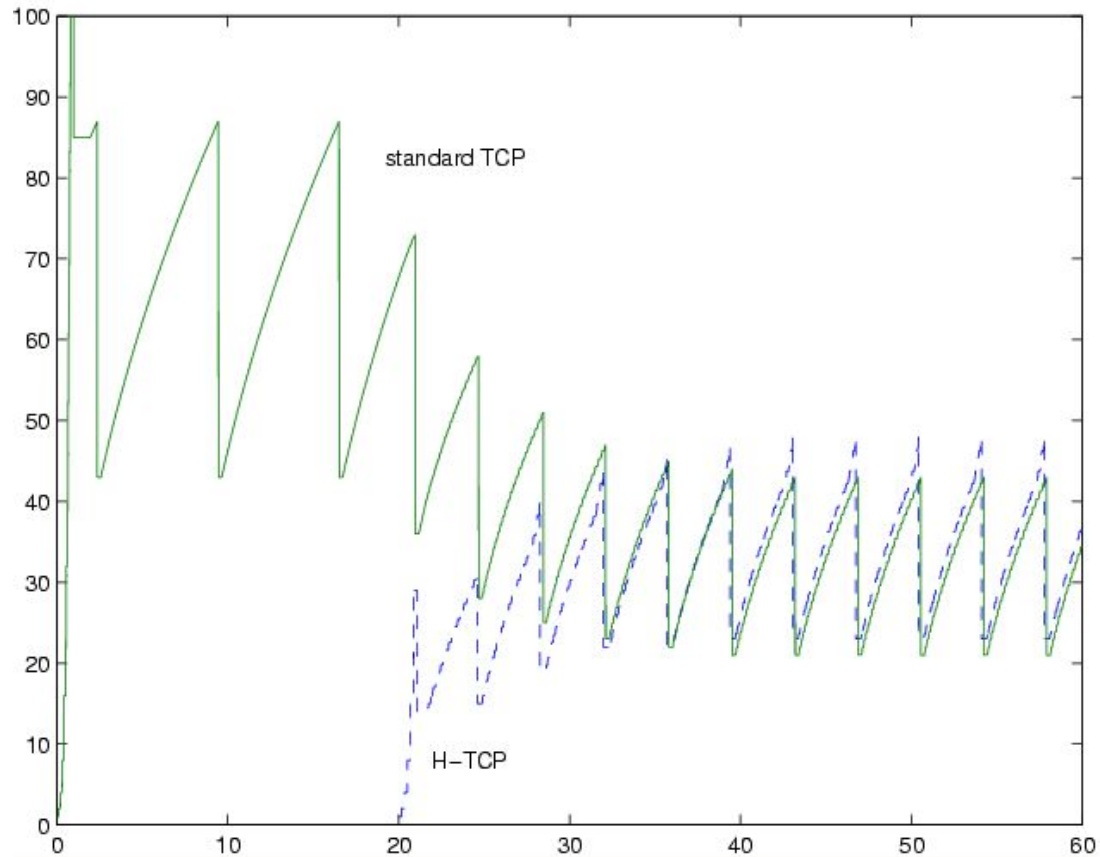


On low-speed links where duration of congestion epoch is less than Δ^L , H-TCP is identical to standard TCP.

As the duration increases above Δ^L , the effective α of H-TCP increases and so does the degree of unfairness with standard TCP.



H-TCP



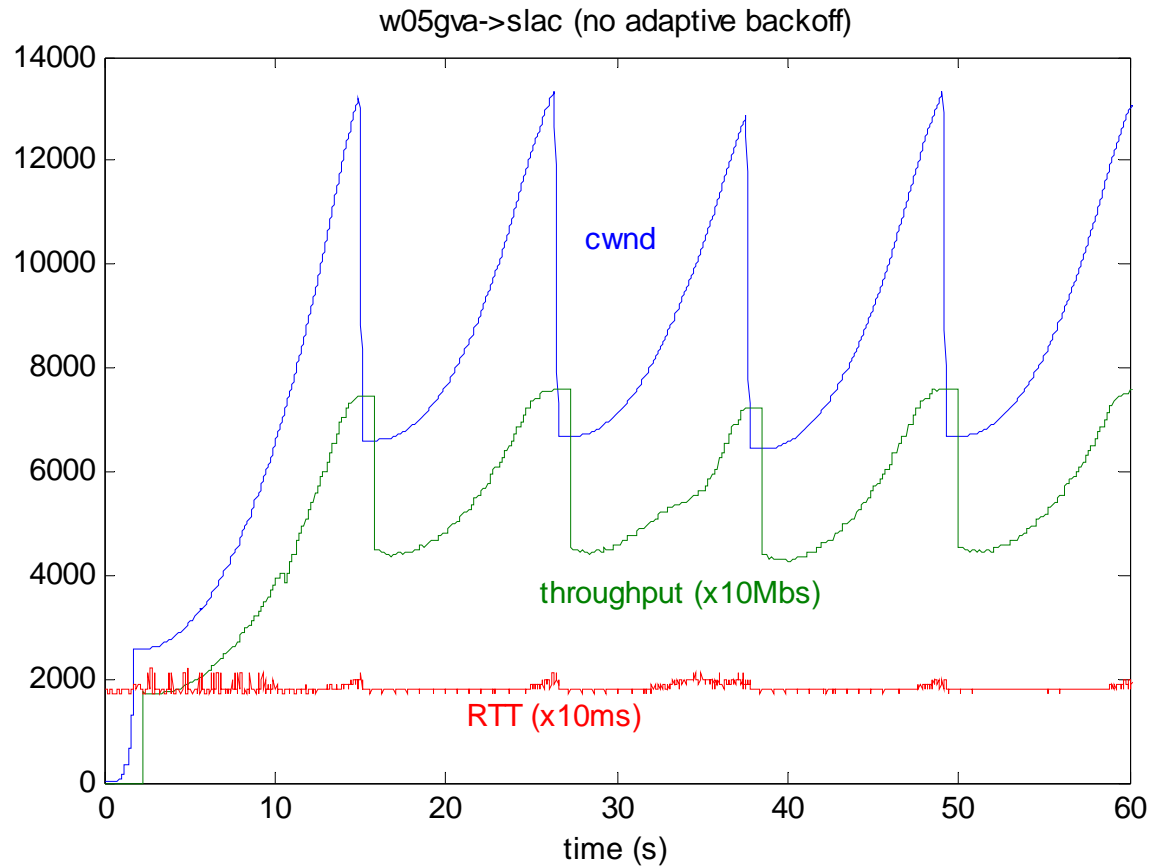
Example of standard TCP and H-TCP flows co-existing on a low speed link

(NS simulation, network parameters: 5Mb link, 100ms delay, queue 44 packets; H-TCP parameters: $\alpha^L=1$, $\alpha^H=20$, $\beta=0.5$, $\Delta^L=19$ – corresponding to window size threshold of 38)



H-TCP

Adaptation to achieve efficient bandwidth utilisation



H-TCP

Adaptation to achieve efficient bandwidth utilisation

At congestion, the bottleneck link is operating at capacity and the overall throughput is given by:

$$R(k)^- = \sum_i^n \frac{w_i(k)}{RTT_{max,i}}$$

where w_i is the window size of source i at congestion and $RTT_{max,i}$ is $BT_i + q_{max}$. After backoff, the overall throughput is

$$R(k)^+ = \sum_i^n \frac{\beta_i w_i(k)}{RTT_{min,i}}$$

where $RTT_{min,i}$ is BT_i assuming the queue empties at backoff.

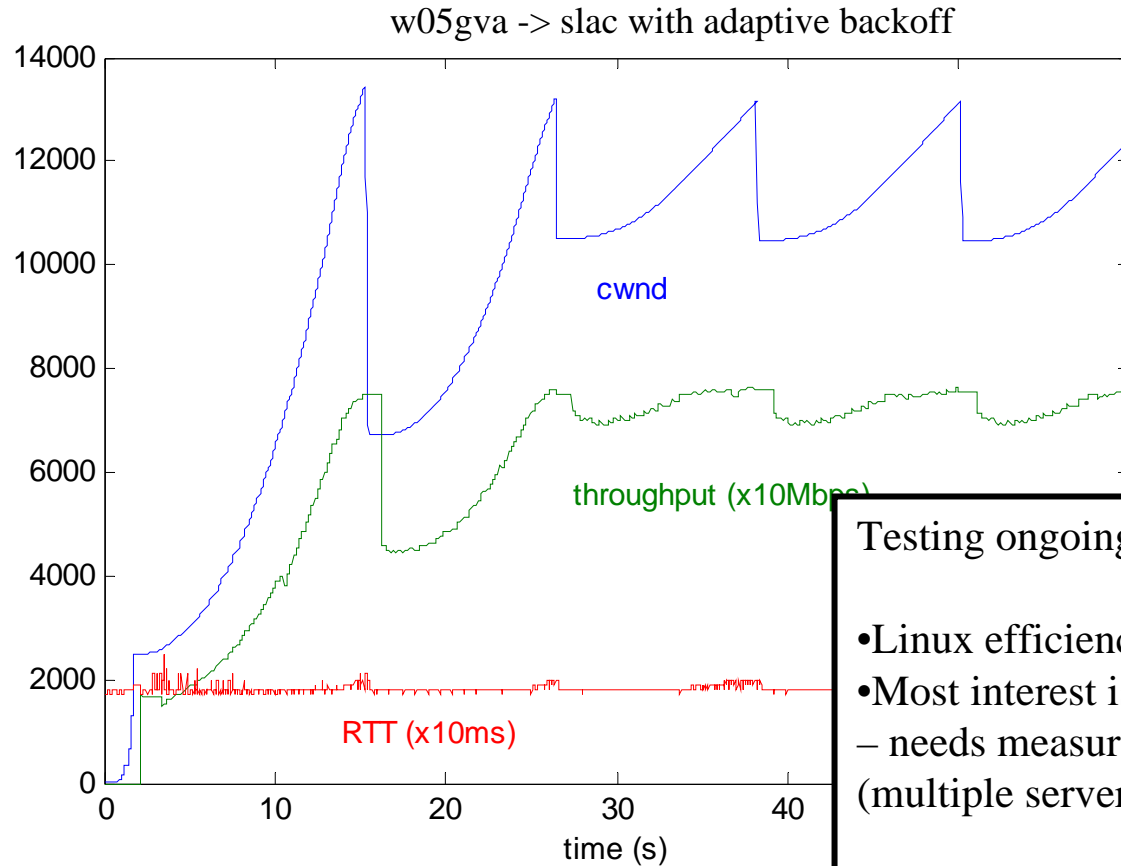
Simple approach is to equate both rates by using backoff factor

$$\beta_i = \frac{RTT_{min,i}}{RTT_{max,i}}$$



H-TCP

Adaptation to achieve efficient bandwidth utilisation



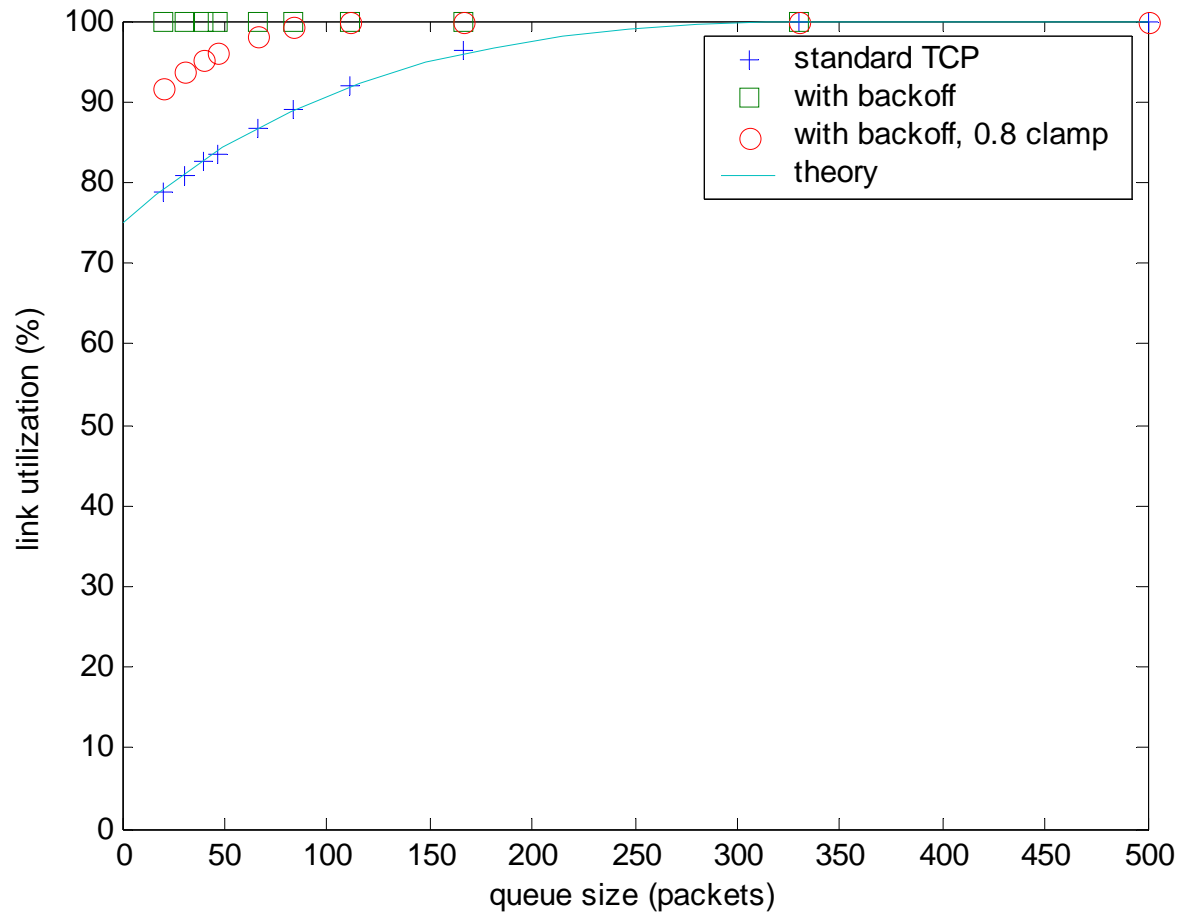
Testing ongoing ...

- Linux efficiency issues
- Most interest is in competing flows – needs measurement infrastructure (multiple servers, path diversity).

Note: for prudence we restrict the backoff factor β to lie in the example a backoff factor >0.8 is needed to completely prevent the queue emptying.



Network queue provisioning revisited



Network queue provisioning revisited

