HighSpeed TCP and Quick-Start for Fast Long-Distance Networks:

*

Workshop on Protocols for Fast Long-Distance Networks
CERN, Geneva, Switzerland
February 3-4, 2003

Topics:

∗

- HighSpeed TCP.

URL: http://www.icir.org/floyd/hstcp.html

- Quick-Start.

URL: http://www.icir.org/floyd/quickstart.html

The Problem: TCP for High-Bandwidth-Delay-Product Networks

*

- Sustaining high congestion windows:

A Standard TCP connection with:

– 1500-byte packets;

– a 100 ms round-trip time;

– a steady-state throughput of 10 Gbps;

would require:

– an average congestion window of 83,333 segments;

– and at most one drop (or mark) every 5,000,000,000 packets (or equivalently, at most one drop every 1 2/3 hours).

This is not realistic.

Is this a pressing problem in practice?

$*$

- Nope. In practice, users do one of the following:
  – Open up N parallel TCP connections; or
  – Use MulTCP (roughly like an aggregate of N virtual TCP connections).

- However, we can do better:
  – Better flexibility (no N to configure);
  – Better scaling (with a range of bandwidths, numbers of flows);
  – Better slow-start behavior;
  – Competing more fairly with current TCP (for environments where TCP is able to use the available bandwidth).

The Solution Space:

∗

- At one end of the spectrum:

Simpler, more incremental, and more-easily-deployable changes to the current protocols:

  – HighSpeed TCP (TCP with modified parameters);

  – QuickStart (an IP option to allow high initial congestion windows.)

- At the other end of the spectrum:

More powerful changes with a new transport protocol, and more explicit feedback from the routers?

- And other proposals along the simplicity/deployability/power spectrums.

Standard TCP:

$*$

- Additive Increase Multiplicative Decrease (AIMD):

Increase by one packet per RTT.
Decrease by half in response to congestion.

- But let's separate the TCP response function from the mechanisms used to achieve that response function.

- The response function: the average sending rate S in packets per RTT, expressed as a function of the packet drop rate p.

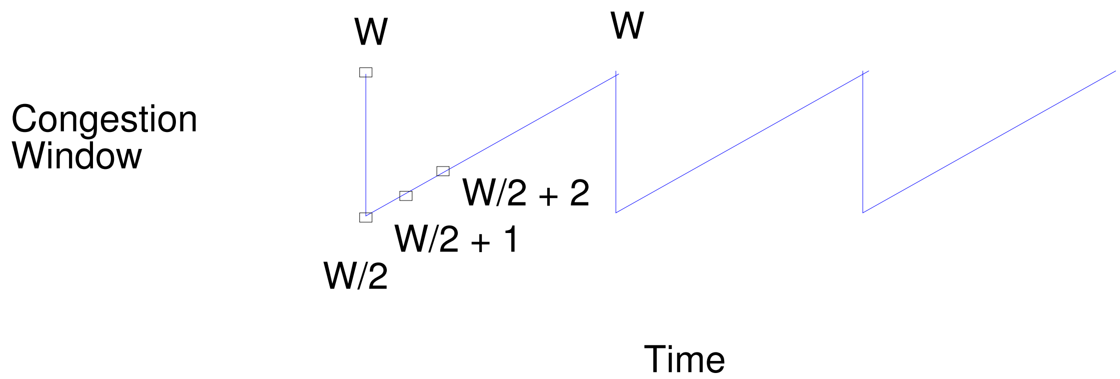
- There are many possible mechanisms for a specific response function.

E.g., equation-based congestion control.

The TCP response function:

$*$

- The steady-state model:



- The average sending rate S is $\frac{3}{4}W$ packets per RTT.

- Each cycle takes $\frac{W}{2}$ RTTs, with one drop in $\approx \frac{3}{8}W^2$ packets.

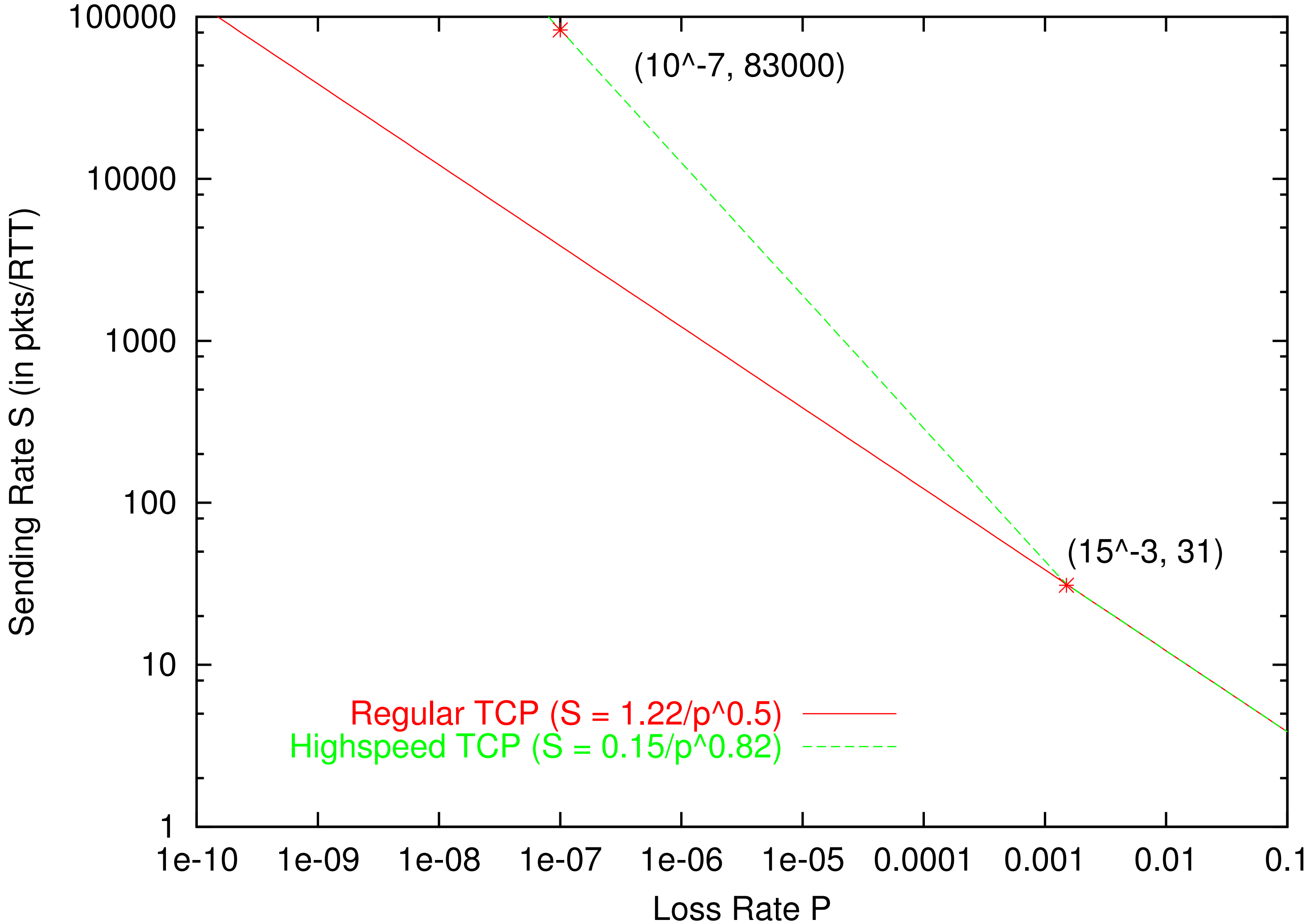
- Therefore, $p \approx \frac{1}{\frac{3}{8}W^2}$, or $S \approx \frac{\sqrt{1.5}}{\sqrt{p}}$, for packet drop rate p.

What is HighSpeed TCP:

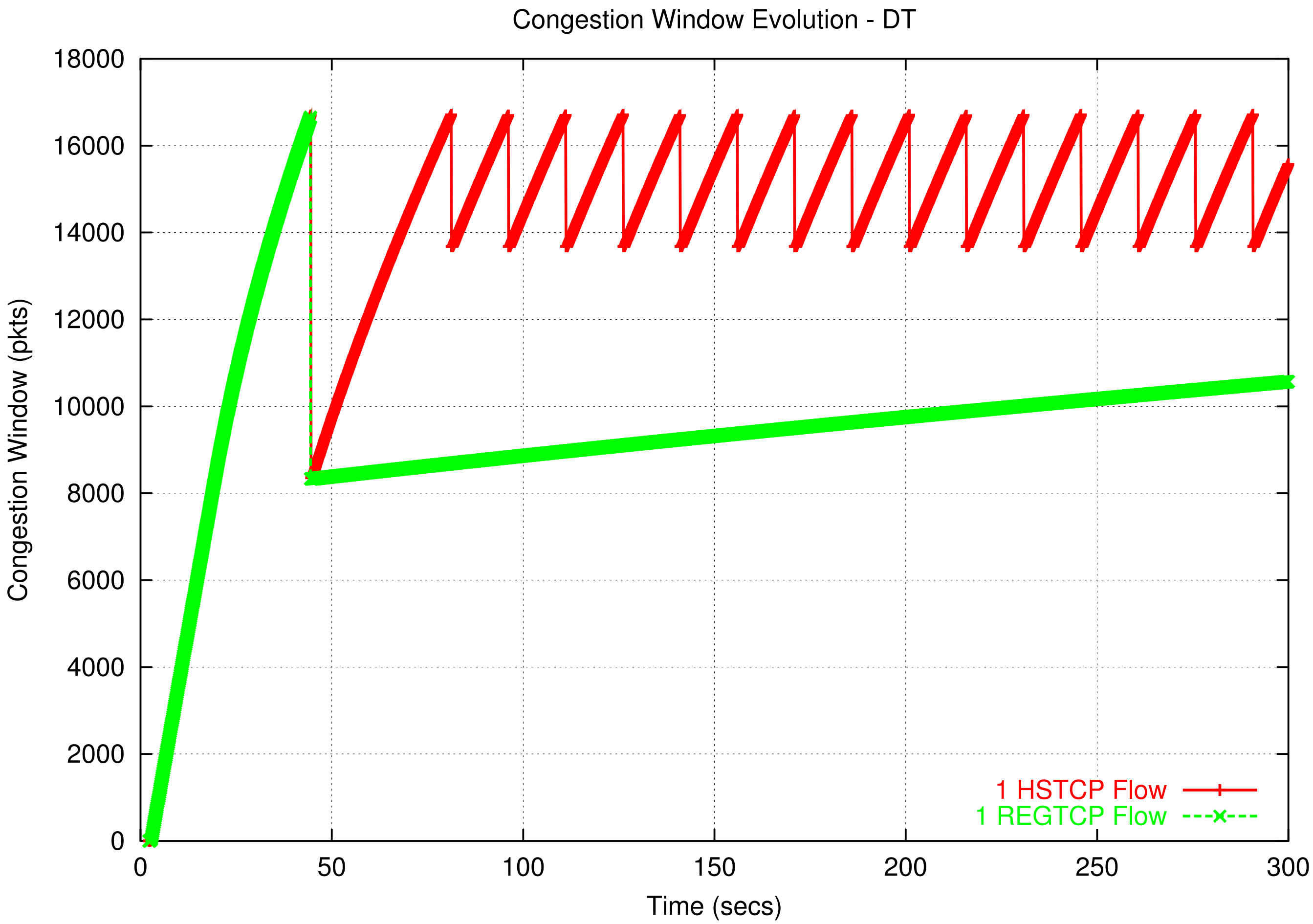
$*$

- Just like Standard TCP when cwnd is low.
- More aggressive than Standard TCP when cwnd is high.
  – Uses a modified TCP response function.
- HighSpeed TCP can be thought of as behaving as an aggregate of N TCP connections at higher congestion windows.
- Joint work with Sylvia Ratnasamy and Scott Shenker, additional contributions from Evandro de Souza, Deb Agarwal, Tom Dunigan.

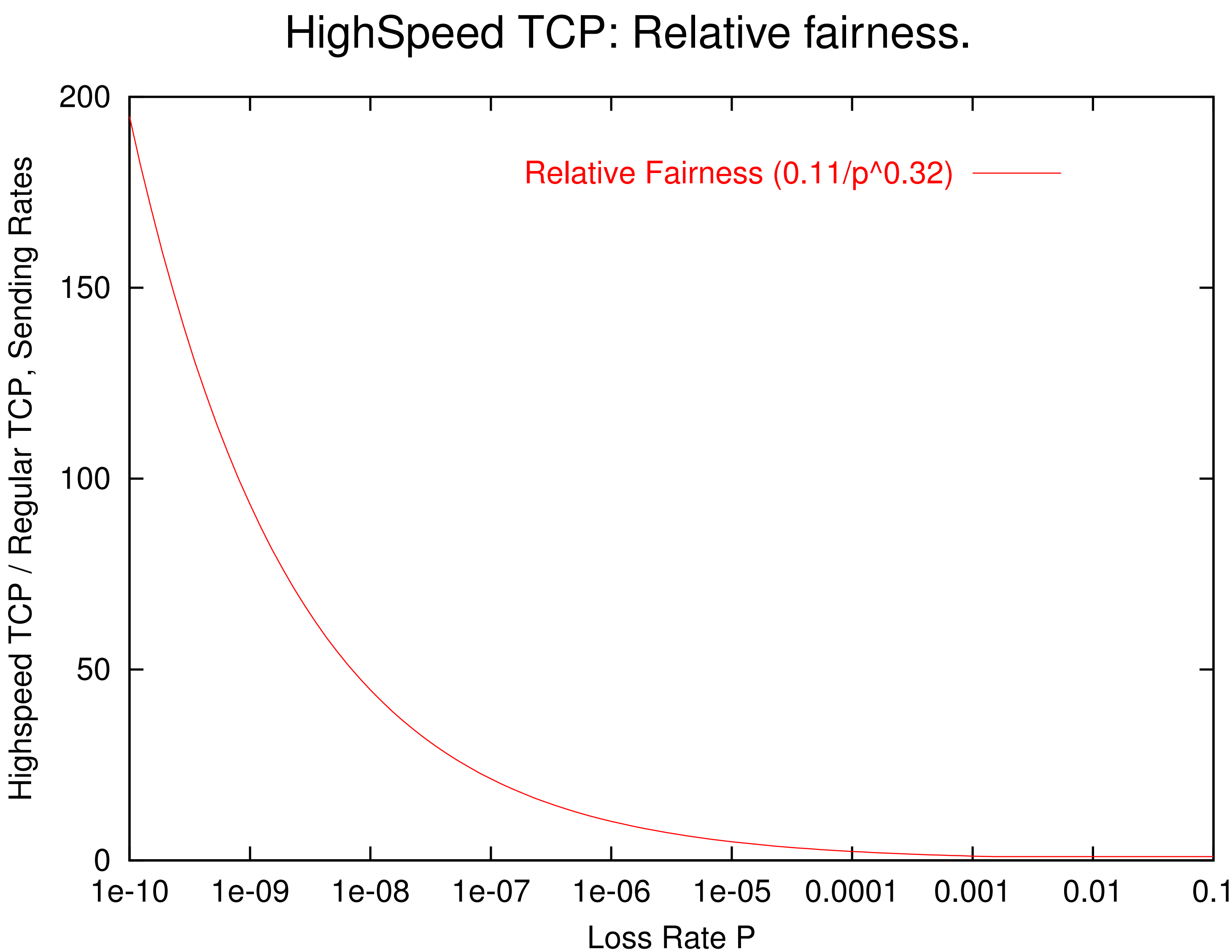
HighSpeed TCP: the modified response function.



Simulations from Evandro de Souza:



HighSpeed TCP (red) compared to Standard TCP (green).


HighSpeed TCP: Relative fairness.
Relative Fairness (0.11/p^0.32)
200
150
100
50
0
Highspeed TCP / Regular TCP, Sending Rates
1e-10
1e-09
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
0.1
Loss Rate P

HighSpeed TCP in a Drop-Tail Environment?

$*$

- Drop-Tail queues: a packet is dropped when the (fixed) buffer overflows.

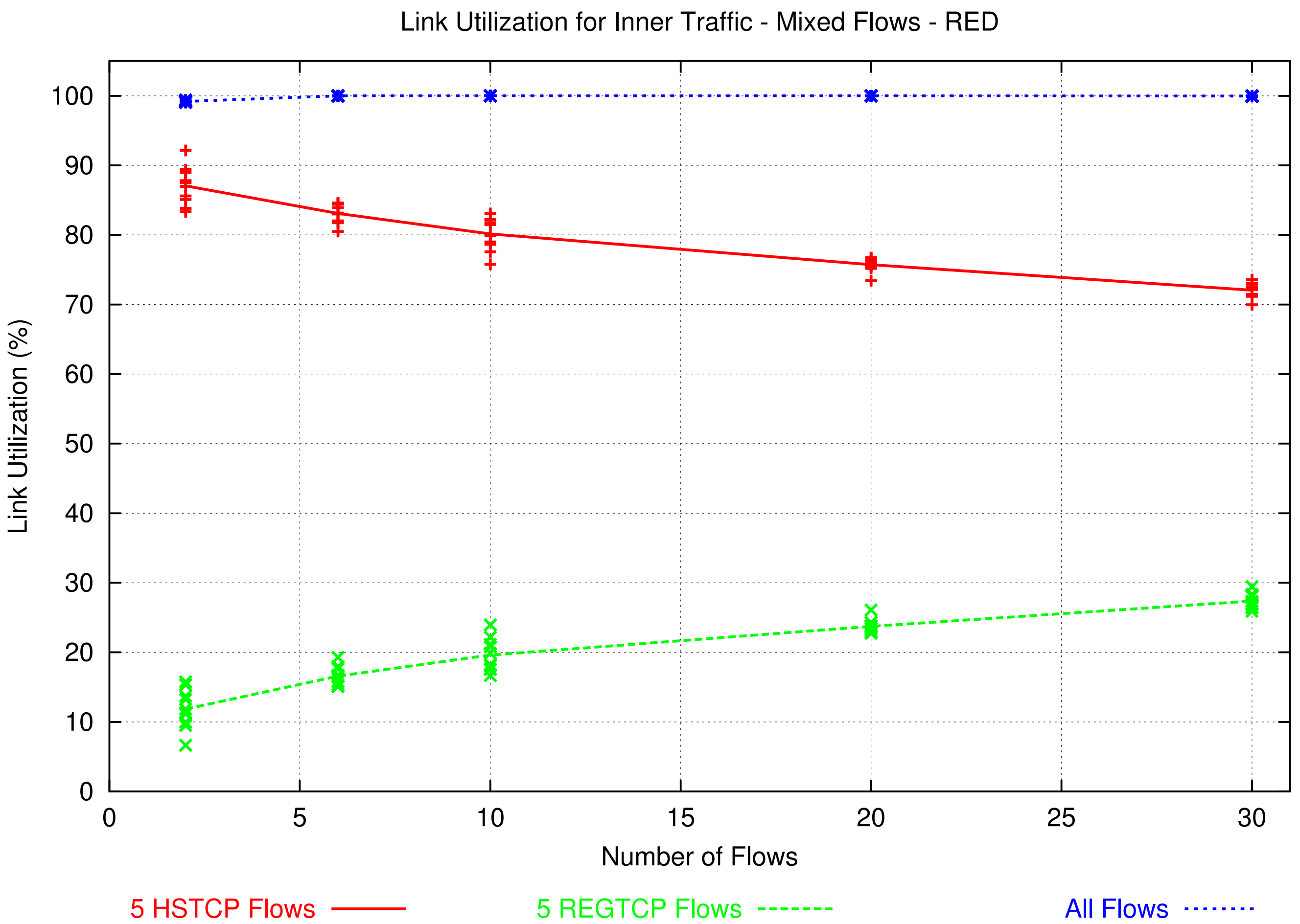
- Active Queue Management: a packet is dropped before buffer overflow.

E.g. RED, where the average queue size is monitored.

- In a Drop-Tail environment:

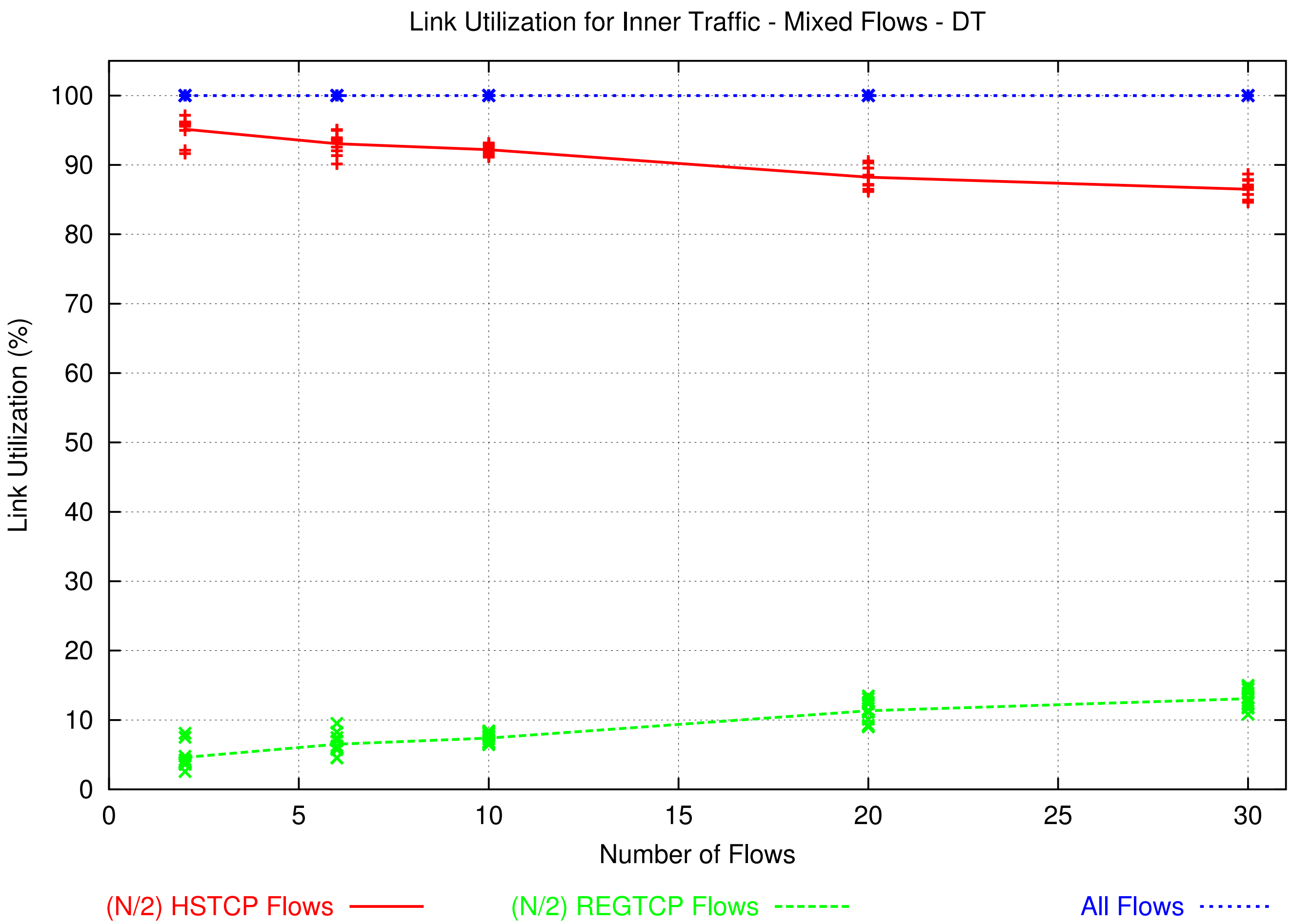
Assume that TCP increases its sending rate by P packets per RTT.
Then P packets are likely to be dropped for each congestion event for that connection.

Relative Fairness with RED queue management:



Simulations from Evandro de Souza.

Relative Fairness with Drop-Tail queue management:



Simulations from Evandro de Souza.

HighSpeed TCP: Simulations in NS.

∗

- ./test-all-tcpHighspeed in tcl/test.
- The parameters specifying the response function:
  - Agent/TCP set low_window_ 38
  - Agent/TCP set high_window_ 83000
  - Agent/TCP set high_p_ 0.0000001
- The parameter specifying the decrease function at high_p_:
  - Agent/TCP set high_decrease_ 0.1

HighSpeed TCP: The Gory Details:

w	a(w)	b(w)
38	1	0.50
118	2	0.44
221	3	0.41
347	4	0.38
495	5	0.37
663	6	0.35
851	7	0.34
1058	8	0.33
1284	9	0.32
1529	10	0.31
1793	11	0.30
2076	12	0.29
2378	13	0.28
...		
84035	71	0.10

Conclusions:

*

- This proposal needs feedback from more experiments.

- My own view is that this approach is the fundamentally correct path:
  – given backwards compatibility and incremental deployment.

- More results are on the HighSpeed TCP web page.
  – http://www.icir.org/floyd/hstcp.html
  – Simulations from Evandro de Souza and Deb Agarwal.
  – Experimental results from Tom Dunigan.
  – Experimental results from Brian Tierney.

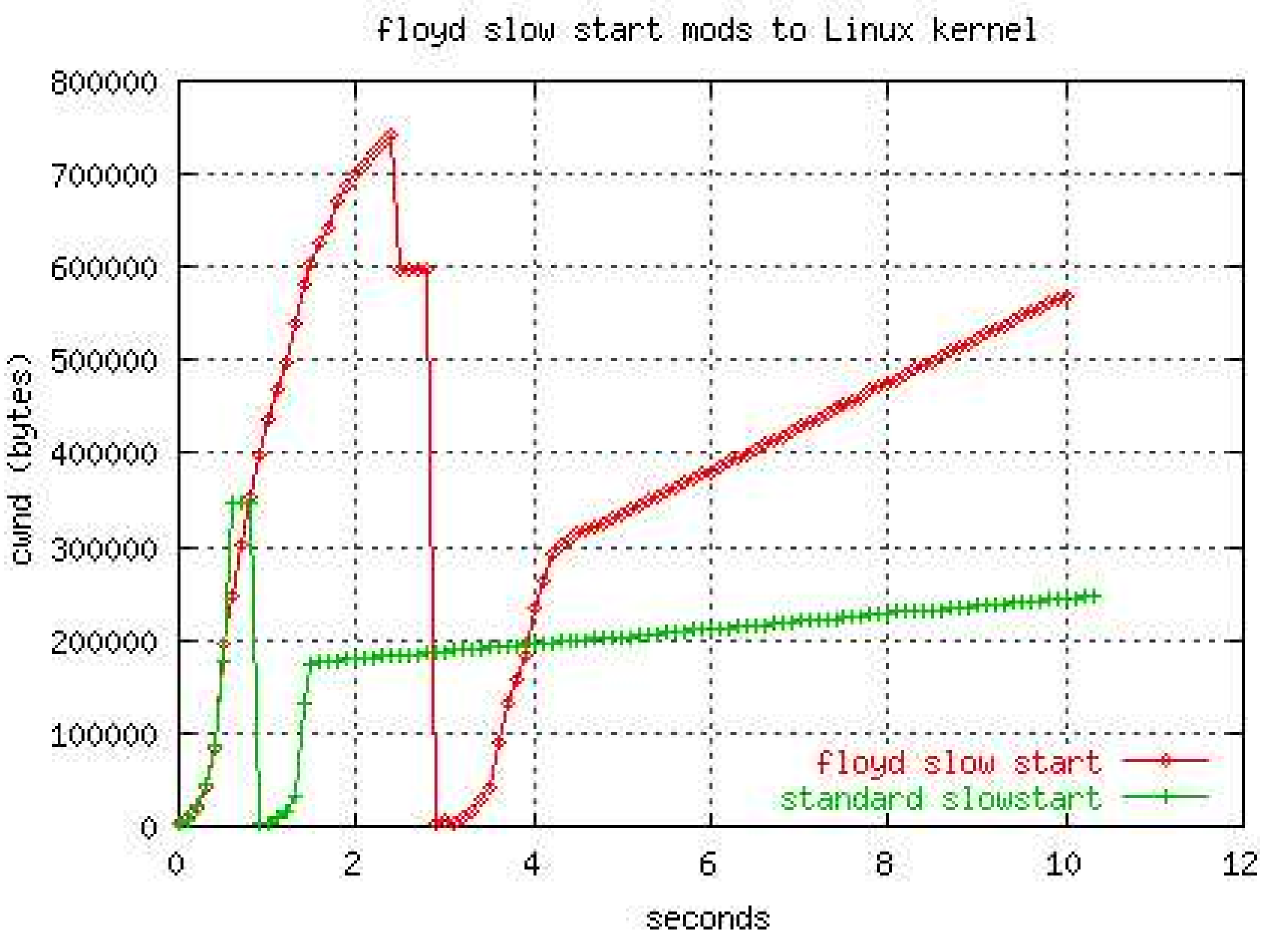
HighSpeed TCP requires Limited Slow-Start:

$*$

- Slow-starting up to a window of 83,000 packets doesn't work well.
  - Tens of thousands of packets dropped from one window of data.
  - Slow recovery for the TCP connection.

- The answer: Limited Slow-Start
  - Agent/TCP set max_ssthresh_ N
  - During the initial slow-start, increase the congestion window by at most N packets in one RTT.

Tests from Tom Dunigan:



This shows Limited Slow-Start, but not HighSpeed TCP.

The pseudocode:

*

```
For each arriving ACK in slow-start:
  If (cwnd <= max_ssthresh)
     cwnd += MSS;
  else
     K = 2 * cwnd/max_ssthresh ;
     cwnd += MSS/K ;
```

Other small changes for high congestion windows:

*

- More robust performance in paths with reordering:

Wait for more than three duplicate acknowledments before retransmitting a packet.

- Recover more smoothly when a retransmitted packet is dropped.

Additional Problems:

*

- Starting up with high congestion windows?
- Making prompt use of newly-available bandwidth?

What is QuickStart?

*

- In an IP option in the SYN packet, the sender's desired sending rate:
  - Routers on the path decrement a TTL counter,
  - and decrease the allowed initial sending rate, if necessary.
- The receiver sends feedback to the sender in the SYN/ACK packet:
  - The sender knows if all routers on the path participated.
  - The sender has an RTT measurement.
  - The sender can set the initial congestion window.
  - The TCP sender continues with AIMD using normal methods.
- From an initial proposal by Amit Jain

The Quick-Start Request Option for IPv4

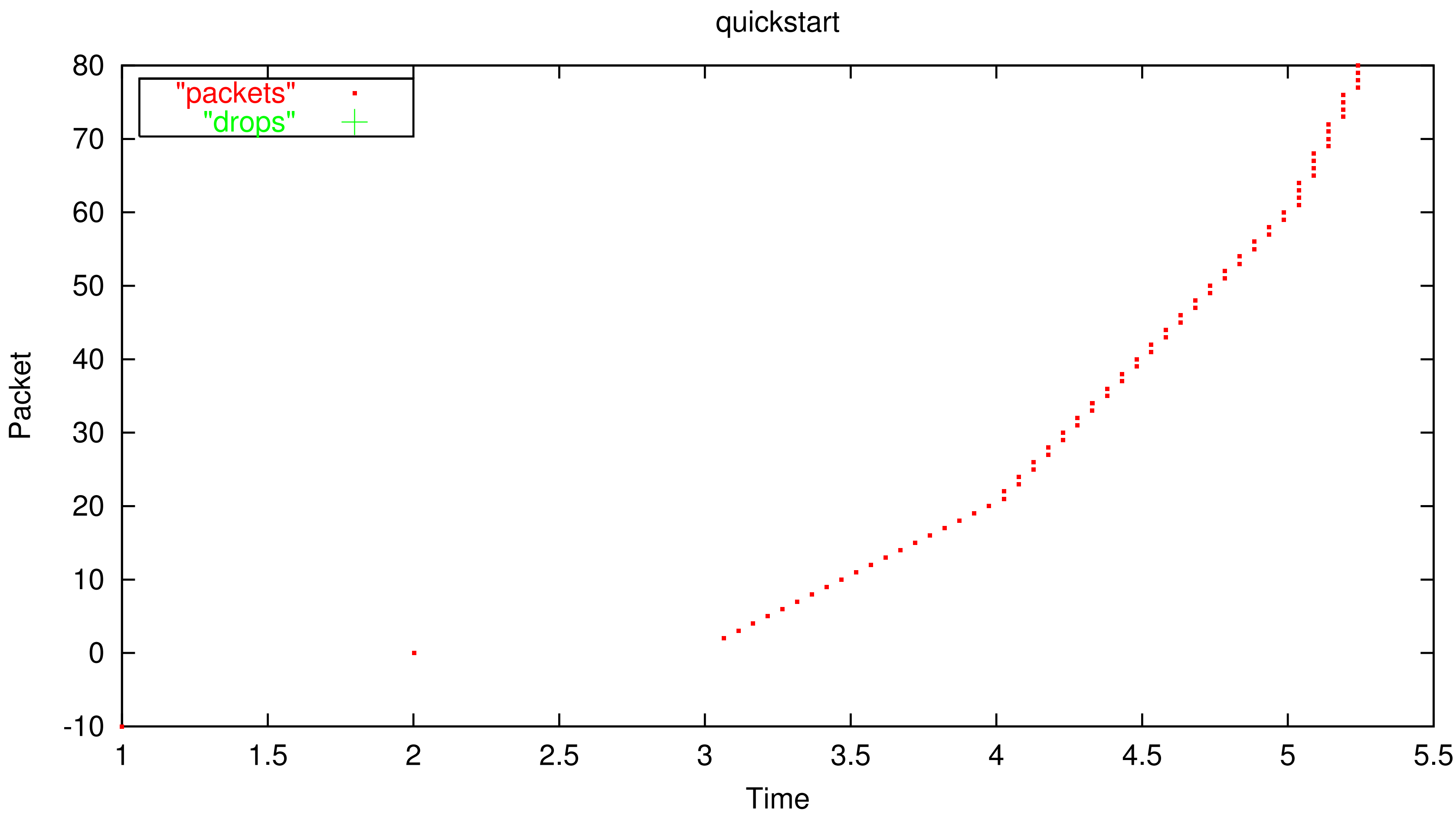
*

```
     0          1          2          3
+----------+----------+----------+----------+
| Option   | Length=4 |  QS TTL  | Initial  |
|          |          |          | Rate     |
+----------+----------+----------+----------+
```

- Explicit feedback from all of the routers along the path would be required.
- This option will only be approved by routers that are significantly underutilized.
- No per-flow state is kept at the router.

Quick-Start in the NS Simulator:

- Added to NS by Srikanth Sundarrajan.



Questions:

∗

- Would the benefits of Quick-Start be worth the added complexity?
  – SYN and SYN/ACK packets would not take the fast path in routers.
- Is there a compelling need to add some form of congestion-related feedback from routers such as this (in addition to ECN)?
- Is there a compelling need for more fine-grained or more frequent feedback than Quick-Start?
- Are there other mechanisms that would be preferable to Quick-Start?

Architectural sub-themes favoring incremental deployment:

*

- A goal of incremental deployment in the current Internet.
- Steps must go in the fundamantally correct, long-term direction, not be short-term hacks.
- Robustness in heterogeneous environments valued over efficiency of performance in well-defined environments.
- A preference for simple mechanisms, but a skepticism towards simple traffic and topology models.
- Learning from actual deployment is an invaluable step.
- The Internet will continue to be decentralized and fast-changing.

DCCP: Datagram Congestion Control Protocol

*

Requirements:

*

- Unreliable data delivery, but with congestion control.
- ECN-capable.
- A choice of TCP-friendly congestion control mechanisms.

Constraints:

∗

- Low overhead, for applications that send small packets.
- Traversing firewalls?
- Ability to negotiate congestion control parameters:
  - ECN.
  - type of congestion control.