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, $p \approx \frac{1}{3W^2}$, 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.

- Regular TCP (S = 1.22/p^{0.5})
- Highspeed TCP (S = 0.15/p^{0.82})

Sending Rate S (in pkts/RTT) vs. Loss Rate P

- (10^{-7}, 83000)
- (15^{-3}, 31)
Simulations from Evandro de Souza:

HighSpeed TCP (red) compared to Standard TCP (green).
HighSpeed TCP: Relative fairness.

Relative Fairness \((0.11/p^{0.32})\)
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:

Link Utilization for Inner Traffic - Mixed Flows - DT

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

- 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 Sundararajan.
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 fundamentally 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.