TCP Over Satellite BOF

Briefing Two

Phased Solutions

Shawn Ostermann

Ohio University

Overview

- We need a solution now
- This is what we can do today
- This is what might be possible
- Ultimately, it would be nice if...
- What are we missing/overlooking here?

We Need An ”Environmentally Friendly” Solution Now

- Satellite links currently carry Internet traffic
- These links are quite “usable”, but have problems with standard TCP
  - Short connections (HTTP) suffer
    - Long delay relative to number of segments
  - Long connections (FTP) suffer
    - RTT interacts with receive windows to limit maximum throughput
    - RTT interacts with congestion control to limit observed throughput

XFTP

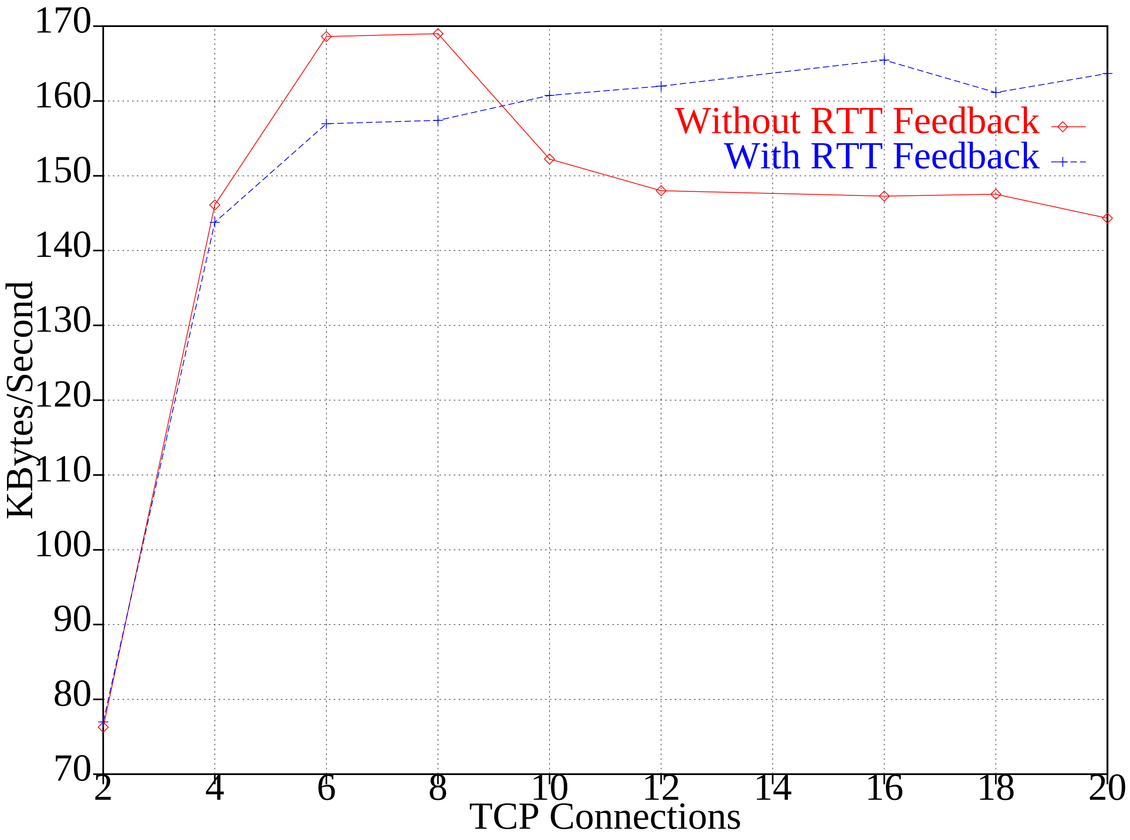
- Research funded by NASA
- Built a prototype multi-connection FTP client and server called XFTP
  - Runs under various flavors of Unix
  - Uses an extension to the FTP application protocol to request multiple connections
  - User can control the "multiplicity" of the transfer
  - Uses backward-compatible, extended version of the standard FTP protocol

XFTP Details

- Tested XFTP over T1 connection over NASA/ACTS
  - Theoretical limit $\approx$ 178,000 bytes/second
    - $\approx$ 7% framing/encapsulation overhead
- XFTP got as much as $\approx 170,000$ bytes/second (5 MByte files)
  - 96% efficiency

Graph of XFTP Results

- Transferring 5MB files over T1 NASA/ACTS



This Is What We Can Do Today

- We used the lessons from the XFTP experiments to point us toward helpful TCP extensions
  - Some only recently documented
  - Some recently standardized
  - Some recently proposed

Lesson 1
Match Receive Window to Delay

- TCP's maximum theoretical throughput limited by receive window and delay

$$throughput_{max} = \frac{receive\ buffer\ size}{round\ trip\ time}$$

- For example, we need at least 100 KByte receive window for T1/560ms RTT

- Scaled windows (RFC 1323) make this feasible

Lesson 2
Match Buffering to Receive Window and Delay

- Make sure that there's enough buffering in the router queues
  - Overcomes loss caused by standard TCP's overestimation of available bandwidth during slowstart
  - Not always under our control
- Make sure the routers are using an appropriate queuing policy
  - Perhaps Random Early Detection (RED)

Lesson 3
Detect/Correct Losses Quickly

- We can't tolerate an *empty pipe* caused by segment timeouts
  - It takes too long to build the congestion window back up
    - At 500ms RTT (512 Byte MSS)
- Some data points to think about
  - Takes ≈ 4 seconds to *slow start* up to a 100 KByte congestion window
  - Takes ≈ 50 seconds to go from a 50 KByte congestion window to a 100 KByte congestion window

Lesson 3 Possible Solutions

- Use fast retransmit/fast recovery (RFC 2001)
- Use selective ACKs (RFC 2018)
- Compute path MTU
  - Larger segments allow the congestion avoidance/control algorithms to speed up faster

This Might Be Possible

- We're still not there yet
  - Even using all these proposals, some experiments have shown a single TCP connection can't get full bandwidth
- *Extremely long* connections can do very well if they don't experience loss
  - Medium-sized transfers are more common
- We still haven't addressed the *short connection* problem

Remaining Problems With Short Connections

- Slow start working against us

Round Trip	Segments in flight
1	1
2	2
3	4
4	8

- Delayed ACKs make this even worse (grows at roughly half this rate)

Slow Start Ideas

- Possibly-helpful circulating ideas
  - Start slow start at more than 1 segment
    - Currently circulating whitepaper (Floyd)
    - Short transfers can complete in 1 RTT
  - Take delayed ACKs into consideration
    - Currently circulating email (Allman et. al.)
    - Speeds slow start by $\approx$ factor of 2
      - Set upper bound to 2 mss
- **The advisability of these "solutions" is still an open issue!**

OU Slow Start Tests

- We ran extensive tests all across the Internet using both small and large transfers
  - Tested larger initial windows
  - Tested modified slow start window increase algorithm
  - Compared loss rates
  - Looked at *goodput*
- Initial windows of 4 and 8 segments increased throughput without lowering goodput or loss significantly

Other Ideas to Test

- Additive increase after congestion detection takes a long time
  - Don't perform congestion control if segment loss due to *errors*
  - Use *explicit congestion notification*, ECN (Floyd)
- Use ssthresh estimation (Hoe)

Ultimately, It Would Be Nice If...

- Whatever we propose **SHOULD** have the following properties
  - Compete fairly with existing TCP
  - Backward compatible with existing TCP
- Needs to remain part of the TCP standard

What Are We Missing/Overlooking Here?

- If we knew this answer, we wouldn't be here!!!

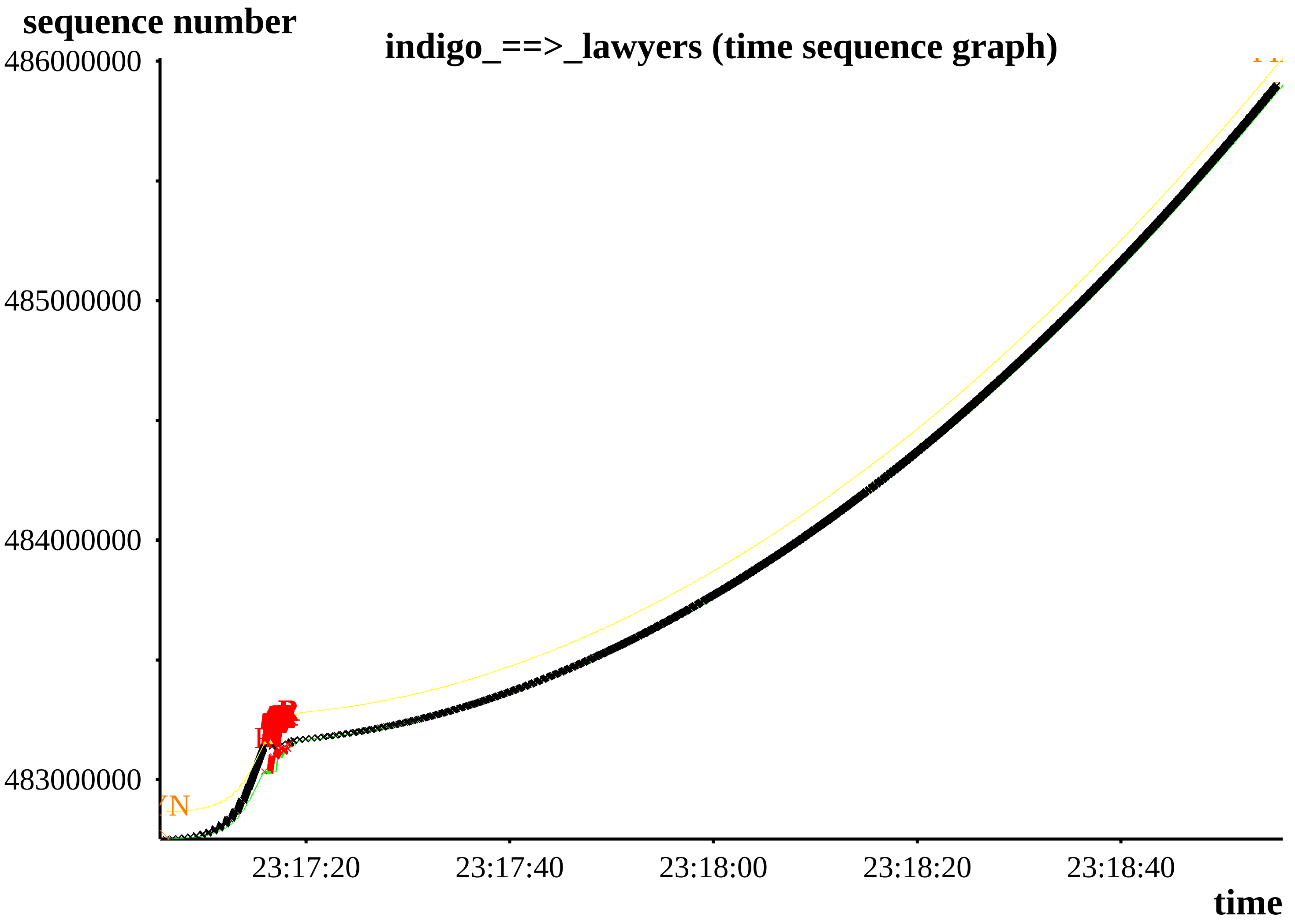
Conclusions

- Some things help
  - Fast retransmit and recovery
  - SACKs
  - Scaled windows
  - Sufficient buffering and appropriate queuing policy
- Some things might help
  - Modified slow start

For Further Information

- Shawn Ostermann
  - ostermann@cs.ohiou.edu
- Email list
  - tcp-over-satellite@listserv.trw.com
- Web information
  - http://jarok.cs.ohiou.edu/projects/satellite
  - http://www.isr.umd.edu/CCDS/IPoS

"The Big Curve"

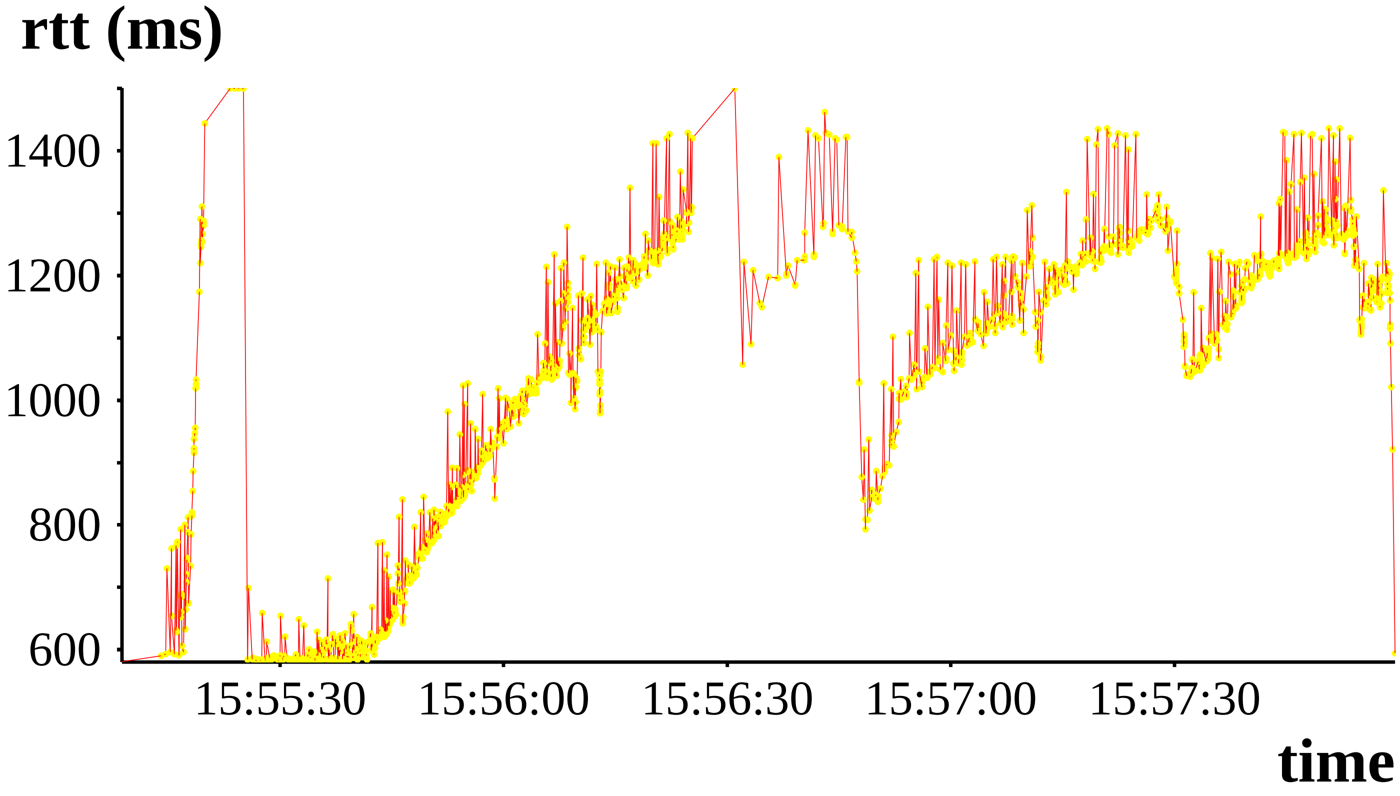


Big Curve
Text Output

```
elapsed time:  0:01:50.262477
total packets: 9763
  total packets:          2781         total packets:          6982
  ack pkts sent:          2780         ack pkts sent:          6982
  unique bytes sent:         0         unique bytes sent:   3147264
  actual data pkts:          0         actual data pkts:       6979
  actual data bytes:         0         actual data bytes:   3203811
  rexmt data pkts:           0         rexmt data pkts:         137
  rexmt data bytes:          0         rexmt data bytes:      56547
  mss requested:          1460 bytes   mss requested:           472 bytes
  max win adv:          114696 bytes   max win adv:          131070 bytes
  min win adv:           95376 bytes   min win adv:          114696 bytes
  avg win adv:          114631 bytes   avg win adv:          114698 bytes
  RTT samples:               0         RTT samples:            2439
  RTT min:                 0.0 ms      RTT min:               569.7 ms
  RTT max:                 0.0 ms      RTT max:              2159.8 ms
  RTT avg:                 0.0 ms      RTT avg:               605.8 ms
  RTT stdev:               0.0 ms      RTT stdev:             158.4 ms
  throughput:                0 Bps     throughput:            28543 Bps
```

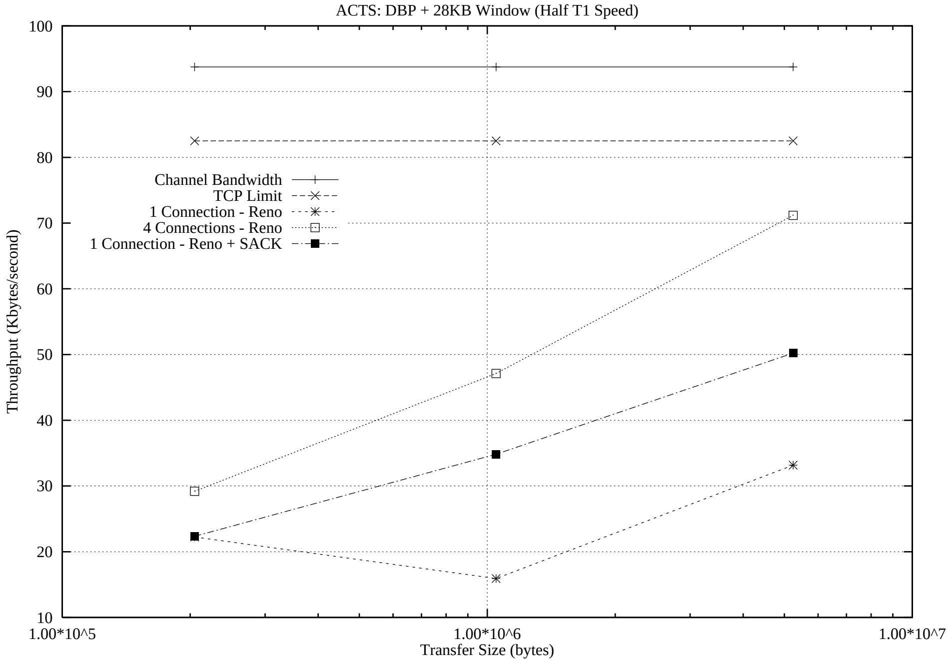
XFTP Observed RTT Example

- Improper matching of load, buffering, and queuing policy yields poor results



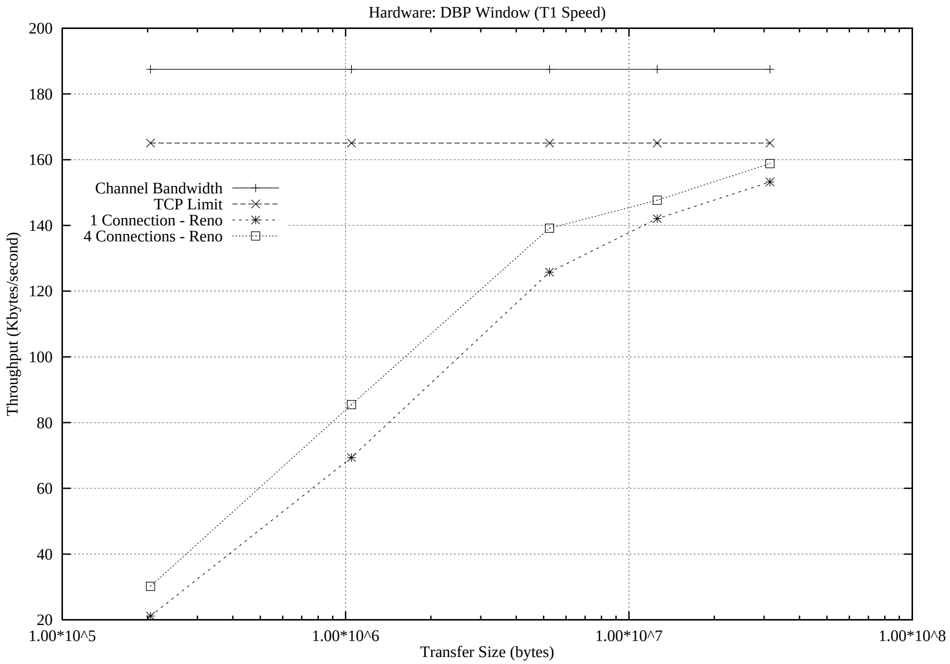
Tests over ACTS Links with Insufficient Buffering

Throughput for TCP Reno (large win), XFTP (same eff. window), and TCP SACK; $\frac{1}{2}$ T1 link. Effective window larger than router buffering.

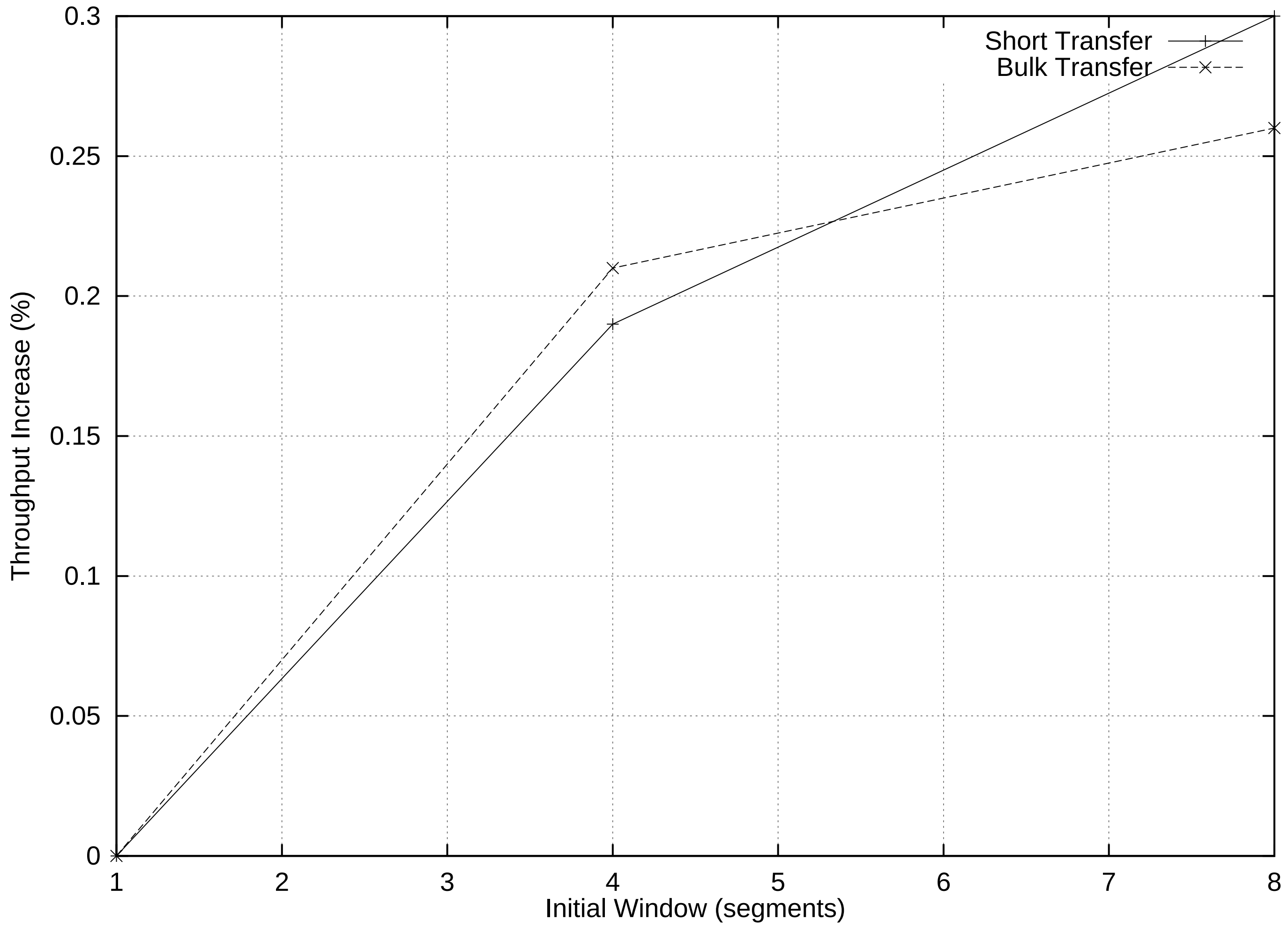


Tests over Hardware Emulator with Adequate Buffering

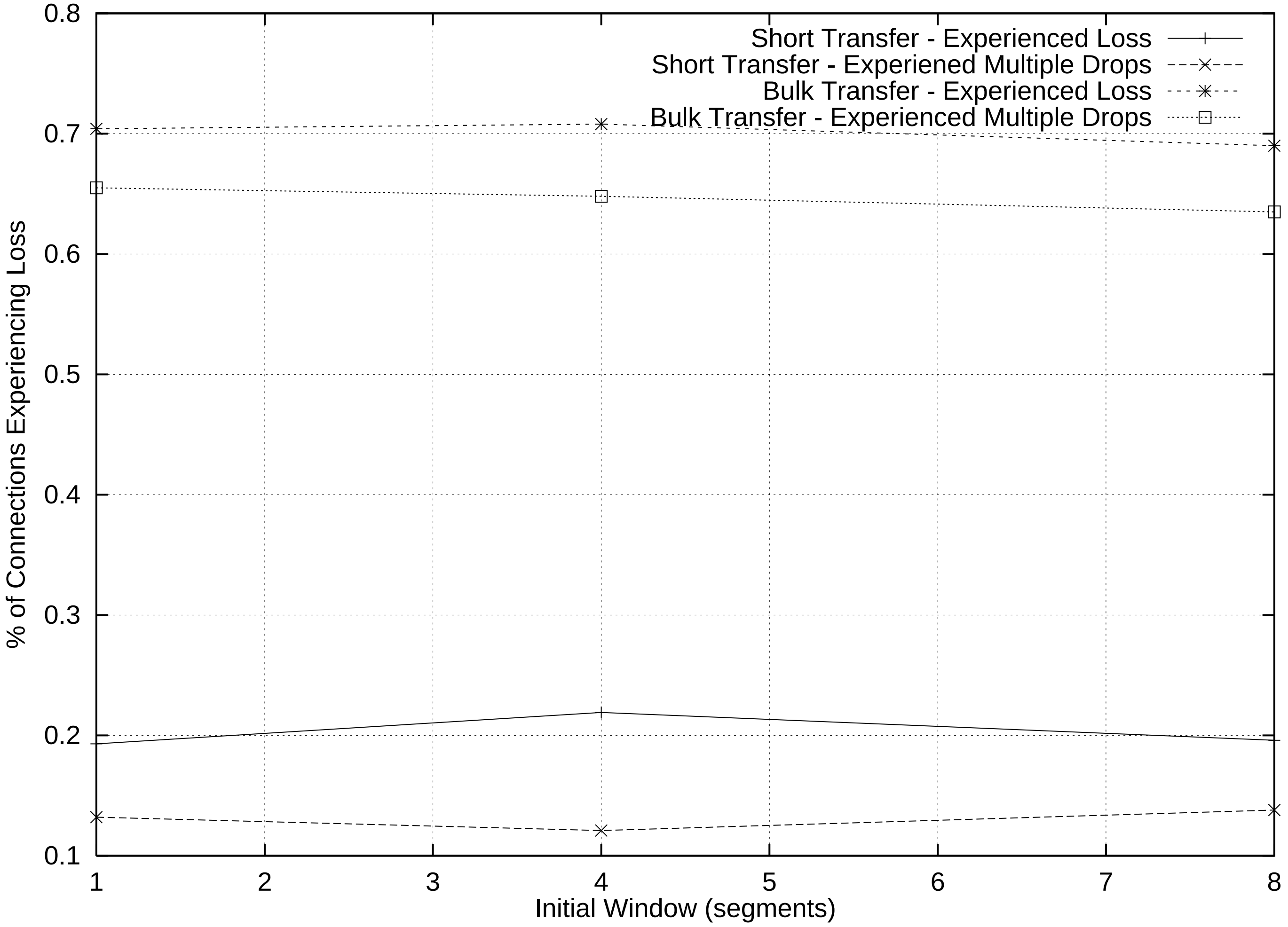
Throughput achieved with standard TCP Reno with large windows vs. XFTP with the same effective window size over T1 link.



Slow Start Experiment
Throughput vs Initial Window



Slow Start Experiment
Loss vs Initial Window



Slow Start Experiment
Goodput vs Initial Window