An Application-Level Solution to
TCP’s Satellite Inefficiencies

Shawn Ostermann
Mark Allman
Hans Kruse
Ohio University

Sponsored, in part, by a grant from
NASA Lewis Research Center
Why Does TCP Have Problems?

- TCP was designed to be a general-purpose, reliable stream protocol
  - Works well in many different types of network environments
  - Used for many widely-used types of applications
    - file transfer (FTP)
    - remote login (TELNET, rlogin)
    - email (SMTP)
    - news (NNTP)
    - WWW (HTTP)
Does TCP Work Correctly Over Satellite Links?

- **YES!**
  - No errors introduced
  - No file corruption
  - No “tragic flaws”
Why Does TCP Have Problems?

- Although TCP works “correctly”, it is unable to use the entire bandwidth of the satellite link in this environment
  - Satellite links introduce a large delay
    - \( \approx 560 \) milliseconds over the NASA ACTS satellite
    - Limits TCP’s maximum throughput
  - Satellite link error characteristics
    - TCP assumes that when data is lost, it’s because intervening routers are overloaded
      - Called router congestion
      - TCP responds by slowing down
• In a *sliding window* protocol like TCP, there can only be a fixed amount of data on the link at any one time
  - Called the “receiver’s window” in TCP
  - In standard TCP, the maximum size of the receiver’s window is 64 KBytes

• Assuming a TCP connection with a receive window that holds $N$ packets (segments)
  - Segment $S$ is sent as soon as the ACK for segment $(S - N)$ is received
  - This means that there can only be $N$ segments outstanding at once

• In the expected case, $\frac{N}{2}$ are in the form of data segments traveling toward the receiver, $\frac{N}{2}$ are returning as acknowledgments (ACKs)
As a simple example, consider this TCP connection

- 8 Kbyte receive window
- 1 Kbyte data packets (segments)
- 10ms round trip time
Maximum throughput is limited by the Round Trip Time (RTT)

- One “window” of data can be transmitted per RTT

\[
\text{throughput}_{\text{max}} = \frac{\text{receive buffer size}}{\text{round trip time}}
\]

- In the previous figure (with an RTT of 10ms), this yields

\[
\text{tput}_{\text{max}} = \frac{8\text{KBytes}}{10\text{ms}} \approx 800,000 \frac{\text{bytes}}{\text{second}}
\]

- Using the NASA ACTS Satellite, this yields:

\[
\text{tput}_{\text{max}} = \frac{24\text{KBytes}}{560\text{ms}} \approx 44,000 \frac{\text{bytes}}{\text{second}}
\]
“Obvious” Solutions to the RTT problem

- RTT limits TCP’s maximum throughput
- There are (at least) 3 obvious solutions to this problem
  - Reduce the RTT
    - Move the satellites lower!
  - Increase the window size
    - Recently-proposed version of TCP (RFC 1323) allows window sizes of up to $2^{30}$ bytes ($\approx 1$ GByte)
  - Use multiple TCP connections
    - That’s the approach taken for the research in this presentation
• We built a prototype multi-connection FTP client and server
  - Called XFTP
  - Prototype 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
  - Source code is available
    • See the URL at the end of the talk
How FTP Transfers Files

- The standard FTP client application starts the TCP connection
  - FTP client opens *Passive* (listening) TCP connection on random local port
  - FTP client uses `PORT` command to tell the FTP server which port to use
  - FTP server makes an active TCP connection to that port on the client
  - The file is sent across that connection
  - TCP connection is closed when entire file has been sent
Changes to the FTP Application Protocol

- We added a command MULT to the FTP protocol
  - Sent from the client to the server
  - If supported, the server will respond with the maximum number of parallel TCP connections that it supports

- We use a modified version of the PORT command
  - Allows the client application to specify a list of ports to connect to
  - Uses modified version of the FTP PORT command
    - Called MPRT
    - Better solution is extended EPRT command (see IETF Draft)
Changes to the FTP user interface

- There needs to be a way for the user to request multiple-connection transfers
  - Details depend on user interface

- Our prototype adds a new user-level command `MULT`
  - With no arguments, requests that the client and server applications negotiate a default number of connections (currently 4)
  - With an argument, requests that number of connections

- Actual number of connections used will depend on configuration limits imposed by both the client and the server application
Dividing a File across Multiple Connections

- This idea can be thought of as "file striping"

- Divides a file across multiple connections

- Naive design can result in poor performance
  - Not all of the TCP connections will progress at the same rate
    - Don’t all start at the same time
    - All see different loss and congestion
  - Static division yields poor performance

- XFTP divides a file into 8k records
  - Number of records assumed to be much greater than the number of connections
  - Each record includes offset value
    - Allows reassembly
  - Records sent over next free connection
Initial Experiments

- We tested the original XFTP client/server using the NASA ACTS satellite

- Maximum theoretical throughput
  - T1 channel - 1.536 Mbits/second
  - 192,000 bytes/second
  - TCP/IP packet overhead
    - 20 bytes of IP header
    - 20 bytes of TCP header
    - 512 bytes of TCP data
    - $\approx 7\%$ overhead
  - Best possible throughput should be about 178,000 bytes/second
• In our initial experiments, we saw \( \approx 170,000 \text{ bytes/second} \) (5 MByte files)
  
  - 96% efficiency

• Unfortunately, throughput was sensitive to the number of connections
  
  - Best results for 6 to 8 connections
Overhead of Managing Multiple Connections

- Could multiple connections overhead be hurting application throughput?
- Ran experiment which varied the number of connections and the receive window size
  - Plotted the “effective” window size vs. throughput

![Graph showing effective window size vs. throughput for different TCP window sizes (12K, 24K, 48K).]
The reason that too many connections hurts performance is the interaction of TCP's congestion control and Slow Start algorithms and router queuing:

- Large router queues allow RTT's to grow
- When router queues overflow, many segments from many connections are discarded
- Loss causes connections to slow down

rtt (ms)

139.88.90.92 ==> pongo.lerc.nasa.gov:2131
Dynamic Multiplicity Control

- XFTP monitors RTT to control the number of TCP connections in use over time
  - RTT gathered using UDP “echo” packets
  - Try to keep RTT between $\alpha$ and $\beta$
    - $\alpha$ is the expected RTT if each connection kept one “extra” segment in the network
    - $\beta$ is the expected RTT if each connection kept three “extra” segments in the network
    - The concept is similar to TCP Vegas
  - $\alpha$ and $\beta$ roughly correspond to too little and too much data in the network
  - If the observed RTT falls below $\alpha$, a connection is added (up to the maximum number of connections)
  - If the observed RTT exceeds $\beta$, half the connections currently in use are “turned off”
By monitoring the RTT, a version of XFTP that adapts the number of connections in use achieves improved average throughput.
Satellite Error Characteristics and Satellite Links

- Damaged segments can be more likely over satellite links than over terrestrial links
  - Spreading segment losses across more connections improves throughput
- Fewer connections slow down

- Selective ACKs will likely turn out to be a better solution to the problem
  - RFC 2018 recently released as a proposed standard
Future Work

- We’re currently testing experimental TCP versions
  - Selective Acknowledgments
  - Large windows
  - Modified slow-start
  - FACK

- Experimental environments
  - Software simulator
  - Software emulator
  - Hardware emulator
  - NASA ACTS satellite

- Still working to fine tune the \( \alpha \) and \( \beta \) mechanism in XFTP
For Further Information

- Author’s email addresses:
  - Shawn Ostermann
    - ostermann@cs.ohiou.edu
  - Mark Allman
    - mallman@cs.ohiou.edu
  - Hans Kruse
    - hkruse1@ohiou.edu

- Web information
  http://jarok.cs.ohiou.edu/projects/satellite/