Is Going Faster Getting Easier?
Some thoughts on the state of high speed IP networking

John Wroclawski
MIT Lab for Computer Science
jtw@lcs.mit.edu
A moment of historical perspective

- Network technology is advancing extraordinarily fast.
- How far have we come?
Connecting Computers: 1994
2000

- The first appearance of consumer-level gigabit networking…
  - GB/E interface standard
  - > 2 x GB I/O subsystem bandwidth
  - >> Sufficient CPU
  - Fast SIMD math processing
  - Mainstream pricing

Photo © 2000, Apple Computer
Price per port:
- 1994 - $20000
- 1998 - $2000
- 2002 - $225

Consumer good
Wide Area Networks: 1994

Aurora Gigabit Testbed Map
2002

The Abilene Network

completed connections:
221 participants
58 connectors + 3 NGIXs + STAR TAP
50 connections to 30 peer networks
Are we done yet?
You are here

You want to be here

Making it possible
Making it easy

(Making it invisible...)

Usefulness

Time
What does “easy” imply?

Not cutting edge.

- Price sensitive commercial environment
  - Multiple competing vendors
  - Beginnings of commercial availability at next level

- Guru knowledge not required
  - Auto-tuning, adaptation, etc.
  - “High Speed Networking for Dummies”
The Wizard Gap

(ratio has gone from 3:1 to 300:1 in last decade)
Rest of the talk: A little bit about...

- How the Internet works
- Why it sometimes doesn’t
- Some things you can do..
- Some things we are doing..
The Internet Hourglass

Four Main Layers

Applications

Transport Services

IP Bearer Service

Link Layers

(Network Technologies)

Email, Web, File

FTP, SMTP, HTTP, ...

TCP RTP

UDP

IP

Ethernet, PPP
SONET, ...

Fiber Radio
Copper ...

Four Main Layers
The IP Protocol

- Inter-network connectivity
- Bearer service

“Spanning Layer”

- Interconnects lower-layer technologies (networks) to form an Internet
  - Addressing - where the other end is
  - Routing - how to get there

- Provides a (single, common) “bearer service” to transport data

- Different networks
- Different technical capabilities
The IP Bearer Service: “Best Effort Packet Delivery”

- **Goals**
  - Support broad range of technologies
    - Designed before LAN’s existed!
  - Support heterogeneity
    - Many technologies at once...
  - Support evolution below

- **Very simple**
  - Can be implemented almost anywhere

- **Limiting**
  - Promises very little
Contract

- You put data packets into the net
- The net will deliver them if it can

- Why it might not
  - No path
  - No capacity
  - Policy restriction

- What it does not promise
  - How fast
  - How reliably
  - What path it will use
Transport Services

- Convert IP Bearer Service to things applications need.

- Goals:
  - Provide useful service to application
  - Adapt to wide variation in the quality of the bearer service

IP Best-Effort Service
Classes of Transport Service

- **Unreliable best effort delivery -> UDP**
  - (User Datagram Protocol)
- **Quasi-synchronous realtime traffic -> RTP**
  - (Realtime Transport Protocol)
- **Reliable, ordered byte-stream -> TCP**
  - (Transmission Control Protocol)
  - Basis of most current Internet applications:
    - FTP - file transfer protocol
    - HTTP - the web
    - SSH - secure remote login
TCP

- Reliable ordered bytestream:
  App -> ]]>]]]]] -> Ugly Stuff -> ]]>]]]]] -> App

- Application puts data into a pipe.

- Comes out the other side:
  - Reliably
  - In order
  - With indication of nonrecoverable failure

- As fast as possible, but no faster
Making Things Work
Two Things to Do

- Make the best use of the bearer service you’ve got.

- Make your best effort better. (more predictable, “good enough”, etc.)

- The first leverages the second.

- The second can be done in several ways.
A Closer Look at TCP

- TCP algorithms implement:
  - Reliability and ordering of data
  - Rate control
    - So transmitter won’t overrun receiver
    - So transmitter won’t overrun network
- Algorithms execute entirely at end nodes
How TCP achieves reliability

- Bytes in data stream are assigned a sequence number (SN)
  - Each packet records the SN of its data bytes
- At receiver:
  - SN is used to put arriving bytes in the right order.
    - Data “reassembled” in a buffer; given to application when no missing pieces.
  - Received bytes are acknowledged to transmitter by SN.
    - “got everything through SN 20936, thanks”
- Transmitter resends unacknowledged data after a while (assumed to be lost)
How TCP Achieves Rate Control

Not overrunning the receiver is easy:

- Receiver advertises to sender how much data it can handle.
  - a “window” in sequence number space.
- Transmitter won’t send more than that
- Receiver updates window once application has removed some data from the buffer.
TCP’s Network Rate Control

- Not overrunning the network is hard
  - Must estimate the capacity of the network
  - Many competing flows
  - Flows come and go quickly
  - Complicated topology - many possible bottlenecks.

- Goals of the algorithm - in order:
  1. Avoid congestion collapse
  2. Use resources effectively - provide good performance to application
TCP Congestion Control - Basics

- Conservation of packets:
  - Use acknowledgement of old packets leaving net to clock new ones in.
    - (Ideally) ensures against congestion collapse
    - Self-clocking algorithm adapts to wide variation in network performance

- Dynamics (how to start a self-clocking system?)
  - Start sending slowly (carefully)
  - Probe for available bandwidth
    - Speed up a little if things are going well
    - Slow down if congestion is detected
Detecting Congestion..

- Networks lose packets for two reasons:
  - Transmission failure
    - Error - turned into loss by CRC check
    - Real loss - cable struck by lightning, etc..
  - Congestion - not enough capacity to carry

- The internet assumption:
  - Real loss is rare and/or controllable
  - Lost packets indicate congestion
A few details

- Transmitter maintains “congestion window” CWND to control rate:
  - No more than \( \min(\text{CWND}, \text{RWND}) \) packets outstanding at once

- Uses round trip time “RTT” as a sampling window

- Increases CWND by 1 packet each RTT without congestion

- Cuts CWND to \( \text{CWND}/2 \) if a packet is lost in the interval
  - “AIMD” (additive increase, multiplicative decrease)
  - Known to be stable
“Slow Start”

- Algorithm I described is very slow to get started (linear increase)
- Modification:
  - Increase CWND exponentially up to a threshold, then switch to AIMD to explore for congestion
  - Adjust threshold down if needed
- Easy to implement (<10 instructions..)
- Good startup performance for most situations
FIGURE 12a

Cross country transfer, showing timeout and fast retransmit
Tip of the Iceberg

- Many enhancements to the basic algorithm
  - Fast retransmit
  - Fast recovery
  - Selective acknowledgement
  - Explicit congestion notification
  - …

- Particularly important in high-speed, high delay environments
Takeaway Points

- TCP’s slow-start and probing behaviors mean that high-bandwidth wide area transfers will not get up to speed quickly.
  - Applications must be designed accordingly

- TCP’s ultimate throughput rate in any situation is limited by a number of factors
  - After overoptimistic approximations:

\[
Rate = \frac{MSS}{RTT} \times \frac{0.7}{\sqrt{p}}
\]
TCP data rates

\[
Rate = \frac{MSS}{RTT} \times 0.7 \sqrt{p}
\]

- **MSS** -> Maximum Segment (packet) Size
- **RTT** -> Round Trip Time
- **p** -> Packet loss probability

Assuming reasonable routing
- MSS ranges from 512 - 8K bytes
- RTT is bound by topology
  - Nearly all paths within a small constant of ideal
- \( p \), the loss probability, can span many orders of magnitude

For bulk transport \( p \) is the primary control
Model: Cross-US 10mb Ethernet

Given / Estimate
- MSS: 1448 bytes
- RTT: 0.08 seconds
- Rate: 1.25MB/ytes / Sec

Calculate
- Wb (window in bytes) 200Kbytes
- Wp (window in packets) 138.122 pkts
- Lp 7154.09 pkts between losses
- Lb 10.3591 Mbytes between losses
- Lt 8.8.29 seconds between losses
- p = 0.00013978 (loss probability)
Model: Cross-US OC12 Carrying Gigabit Ethernet

- **Given / Estimate**
  - MSS: 1448 bytes
  - RTT: 0.08 seconds
  - Rate: 75MB/Sec (600 mbits/sec)

- **Calculate**
  - Wb - 12000 Kbytes (12 megabytes)
  - Wp - 8287.29 pkts
  - Lp - 2.58e+07 pkts between losses
  - Lb - 37292.8 Mbytes between losses
  - Lt - 497.2 seconds between losses
  - \( p = 3.88e-08 \) (loss probability)
Model: Cross-US OC48

Given / Estimate
- MSS: 8196 bytes
- RTT: 0.08 seconds
- Rate: 300MB/Sec (2400 mbits/sec)

Calculate
- Wb - 48000 Kbytes (48 megabytes)
- Wp - 5856.5 pkts
- Lp - 3215508 pkts between losses
- Lb - 26354 Mbytes between losses
- Lt - 351.4 seconds between losses
- \( p = 3.1e-7 \) (loss probability)
What to do?

- A few possibilities:
  - Workaround: multiple TCP connections..
  - Find and fix problems..
    - Host implementations, window mgmt, etc
    - Network packet loss, engineering
  - Modify the control algorithms

- I’ll talk a little about the first two
- Very interesting recent work on the third
Multiple TCP Streams

Why might this work?
- Each stream is individually loss-limited; will go faster if $L_p$ is independent
- Will reach steady-state faster ($N_x$ slow-starts)
- Hides some common operating system problems.
Why Might This Not Work?

- Problem of non-independent paths
  - If traditional queues, and queue is almost full:
    - TCP N sends a window of packets
      - All are lost
  - What should have happened
    - TCP 1 - N slow up a little bit each
  - What does happen?
    - TCP N slows up a lot
  - Application is gated on slowest TCP?
    - Oh well
Multistream design hints

- Strategy works well if application can:
  - Use received data independently
    - (no blocking of one TCP connection on another)
  - Find the right number of streams
    - Too few, poor TCP performance
    - Too many, triggers several potential operating system problems
- Your neighbors may hate you
- Parts of the Internet may eventually take defensive action
Web100: Automating the Network Wizard

- TCP knows why it is not going fast.
  - Otherwise, it would be going fast :)

- Ask it.

http://www.web100.org
Network performance debugging is hard

◆ **Many problems limit performance**
  - IP routing, long round trip times
  - Improper MSS negotiations / MTU determination
  - IP packet reordering
  - Packet losses, congestion, faulty hardware
  - TCP sender or receiver window limits (operating system problems)
  - Inefficient applications
  - ...

◆ **Any one problem can mask all other problems**
  - Confounding all but the best experts

◆ **How to close the wizard gap?**
The Web100 Idea

- When there is a problem, ask TCP why
  - TCP has an ideal vantage point
  - TCP can identify the bottleneck subsystem
  - TCP already measures key network properties
  - TCP can measure the application

- Results of asking TCP can be used
  - To tune system behavior
  - To report “untunable” problems
Web100 Technology

Applications

User Mode Debugger

User Mode Autotuner

Re-Instrumented
TCP Implementation

Operating System Functions

User

User Level Code

Operating System Kernel

April 8, 2002

MIT Haystack e-VLBI Workshop
Project and Status

- **Software**
  - In-kernel TCP instruments
  - Example user-mode tools
    - Diagnostics
    - Autotuning
- **Outreach**
  - User support
  - Vendor liason
- **Standards as appropriate**
  - TCP-MIB update
  - Etc.

- **Status**
  - alpha 1.1 software released Feb 28th, 2002
  - Still targeted at experienced network programmers
  - Early community pickup
The Other Problem: Host Performance

- Networked computers must
  - Do their job
  - Run the network
  - Run the rest of the operating system

- Two types of network protocol overhead
  - Per packet (control, etc)
  - Per byte (data handling)
Per Packet Overheads: Protocol Processing

- TCP: ~200 instructions [Clark et al, 1989]
- TCP with “fast path”: < 100? [Jacobson]
  - Assume that arriving packet is the expected one (next in sequence, etc.) - optimize this path at expense of others
  - Much faster, but more vulnerable to poor network or peer implementation
- Other overheads - memory mgmt, etc.
- But basically not a problem.
Per Packet Overheads: Interrupt Processing

- Machine must stop what it’s doing to handle packet
- Overhead of context switch
- “Livelock” problem
- Can be addressed with careful implementation and wise choice of hardware
Per Byte Overheads

- Checksumming
- Simple copying
- Etc..

![Diagram showing Speed (log scale) over time from 1980 to 2000 with trends for Host Memory and Network Technology.]
Traditional Implementation (arriving packet)

1. Network interface puts packet in operating system memory
2. CPU reads packet to compute checksum
3. CPU copies packet into reassembly buffer in operating system memory
4. CPU copies reassembled data to application memory on demand

◆ What’s good?
  - Application can read as much or as little as it wants, at any time
  - Application is decoupled from network performance

◆ What’s bad?
  - __Four__ trips through memory!
Optimizations (tricks..)

- Integrated Layer Processing
  - Do several protocol steps at once
- Protocol offloading
  - Network interface hardware does more work
- Page flipping
  - “Move” data using virtual memory tricks
- Direct-to-user device buffers
  - Network interface works directly with application memory

What’s good?
- Down to two (easier) or one (hard) trip through memory

What’s bad?
- Application much more tightly coupled to network and OS
- Network hardware must be chosen with care

.. back to wizard gap
State of the art

- **Knee:**
  - 1gb/s requires work. >2gb/s requires magic.

- **Latest versions of many operating systems do a lot right**
  - SGI, Linux 2.4, FreeBSD 4.5, Windows XP.

- **Hardware must be carefully chosen**

- **Application programmer must be aware of several performance issues**
Conclusions

- Going fast is getting easier, but it is not quite ‘easy’ yet
  - Some hardware and software vendors are increasing emphasis on network performance with arrival of cheap gigabit ethernet
  - Fast wide area networking remains tricky. Wizards still helpful.
  - Folks in the network community are developing some tools to help you. They love customers!
  - Interesting algorithms for control of 10-40gb/sec wide area applications are in research stage today