Internet Programming & Protocols Lecture 21

Parallel streams

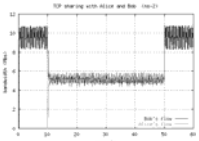
Rate-based UDP

Optical paths

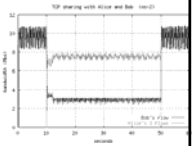
Asnmt 8

www.cs.utk.edu/~dunigan/ipp/

parallel streams

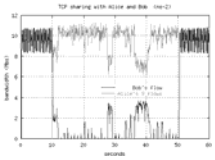


Alice and Bob sharing



Clever Alice -- 3 streams

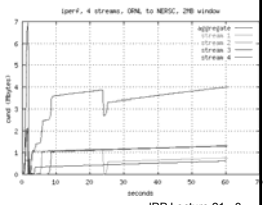
- TCP is fair
  - N flows
  - each gets 1/N of bandwidth



Bad girl ...

Psockets

- Re-write network applications to use multiple TCP sockets
  - Server and client must be re-written
  - Sender splits data across all the sockets
  - Receiver reassembles
  - Use fork() or select() or threads to concurrently feed all the sockets
- Psockets C++ class is wrapper for parallel sockets
- Examples
  - iperf –P 4 –w 1m –c hosta
  - Parallel file transfer: bbcp, bbftp, GridFTP



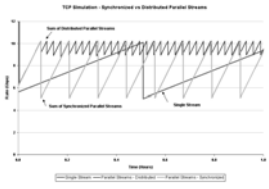
TCP behavior for N parallel streams

HSTCP vs streams	
Window	N
1	1
20	1
200	1.4
1000	3.6
10000	9.2
100000	23.0

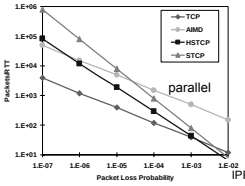
- Takes advantage of TCP's fairness (cheating!)
- Allows you to bypass OS SNDBUF/RCVBUFF limits!
  - Done at application level, no system privileges required ☺
- Faster startup
  - Each flow is doubling its rate each RTT
  - Takes $\log_2(N)$ fewer RTT's (equivalent to initial slow start of N segments)
- faster recovery
  - often only 1 stream loses a packet
    - Though droptail queues often cause losses on many streams at same time
  - MD: 1/(2N) rather than 1/2
  - AI: N times faster linear phase
    - Example: 4 streams equivalent to AIMD(4,1/8)
    - Response function: $N\frac{MSS\sqrt{3/2}}{RTT\sqrt{p}}$
- Synchronized packet loss (droptail queues) keep parallel TCP from being N times faster

Parallel TCP

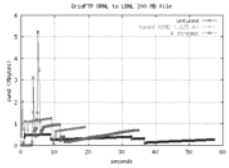
- Synchronized loss (DropTail)

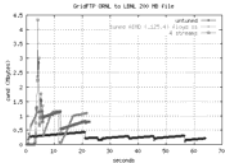


- Response functions
  - Not TCP friendly



GridFTP vs "equivalent" AIMD tuning





Data/plots from Web100 tracer

Can tuned single stream compete with parallel streams?

Mostly not with "equivalence" tuning, but sometimes….
Parallel streams have slow-start advantage.

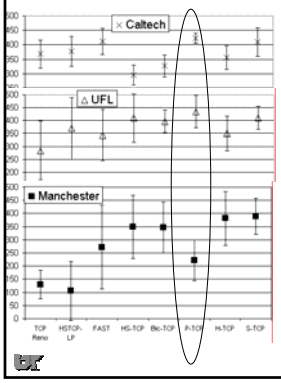
Tests inconclusive so far….
Testing on real Internet is problematic.

Is there a "congestion metric"? Per unit of time?

Flow	Mbs	congestion	re-xmits
untuned	28	4	30
tuned	74	5	295
parallel	52	30	401
untuned	25	7	25
tuned	67	2	420
parallel	88	17	440

Buffers: 64K I/O, 4MB TCP

SLAC throughput tests



Avg throughput for optimal & large window sizes from SLAC to CalTech, UFl & Manchester

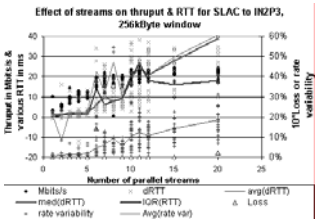
Stack more important for long RTTs

Single stream Reno & HSTCP-LP poorer on large RTTs

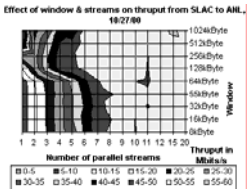
Parallel TCP (P-TCP) competitive, but how many streams? Window size?

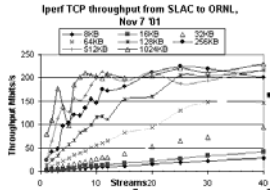
Tuning parallel TCP

- How many streams? Flavor of TCP? Buffer size?
  - Target: bandwidth-delay product
    - A few streams with big buffers?
    - A lot of streams with small buffers?
      - If OS limits you to 64K and you need 4 MB ➔ 63 streams
      - Some OS's limit open sockets to 64/process
- SLAC experiments



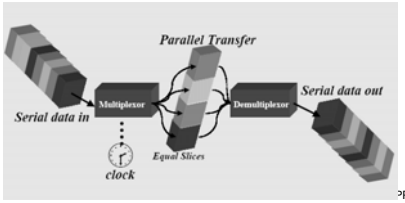
SLAC tests window size vs # streams





bbcp

- Parallel stream file transfer
- User-level code (C++), no daemon required
- Based on scp syntax – options for file block size, net block size, # of streams
- Authentication with ssh, bulk data not encrypted
- Auto-resume on failed transfer
- Options: compression, MD5, rate-limits

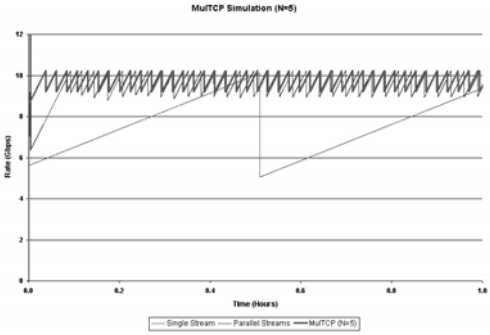


MulTCP

- Crowcroft, '98, ns evaluation of a modified TCP that behaves like N parallel streams
- Slow-start
  - Send 1 packet, then 3 for every ACK
  - After k RTT's cwnd would be 3^k where N TCP's would be $N2^k$
- Linear increase
  - N TCP's increase cwnd by N/cwnd
- Multiplicative decrease
  - Only one flow out of N experiences loss, so cwnd ← cwnd * (N – ½)/N
- MulTCP response function

$$\frac{N\sqrt{2}MSS}{RTT\sqrt{p}}$$

MulTCP



What about striping?

- Typically used in a cluster with a shared file system, but it can be a multi-homed host
- All the advantages of parallel TCP
- Also get parallelism of CPUs, Disk subsystems, buses, NICs, paths, etc..
- You can, in certain circumstances, also get parallelism of network paths
  - Example, SC02 bandwidth challenge – visipult
  - pftp
  - BitTorrent
  - Storage area networks (parallel file systems)
- This is a much more complicated implementation and beyond the scope of what we are primarily discussing here.

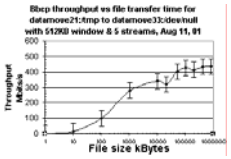
TCP land speed record: 7.7 gigabits/sec (single stream)
striping: 23 gbs

BitTorrent striping

- Peer-to-peer nets can achieve high downloads by striping across many servers
  - Same for grid applications (logistical networking)
- BitTorrent can overcome subscribers low uplink speeds (asymmetric network) by using lots of subscribers to serve requestor

When should you use parallel TCP?

- Parallel TCP is the way to get around buffer limits and slow recovery!
- Engineered, private, semi private, or very over provisioned networks are good places to use parallel TCP.
  - bbcp popular on DOE's ESnet and in high-energy physics nets (SLAC)
- Bulk data transport. It makes no sense at all to use parallel TCP for most interactive apps.
- QoS: If you are guaranteed the bandwidth, use it
- Community Agreement: You are given permission to hog the network.
- Lambda Switched Networks: You have your own circuit, go nuts.



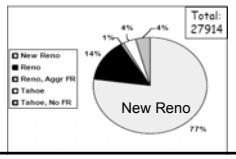
TCP choices

- Scalable, stable, fair, friendly, available

Flavor	control method	trigger	response
TCP	AIMD(1, ½)	loss	W = W – ½ W
		ACK	W = W + 1/W
HSTCP	AIMD(a(W),b(W))	loss	W = W – a(w)W
		ACK	W = W + b(W)/W
STCP	MIMD(1⅛,1/8)	loss	W = W – W/8
		ACK	W = W + 0.01
N TCPs	AIMD(N,1/(2N))	loss	W = W – W/(2N)
		ACK	W = W + N/W
BI-TCP	AIMD(b,1/8)	loss	W = W – W/8
		ACK	W = W + binary incr.
TCPW	AI?(1,FSE)	loss	ssthresh = RTTmin*FSE
		ACK	W = W +1/W
FAST/Vegas	RTT delta	RTT	W = W*minRTT/RTT + α

Feb 2004 TBIT results

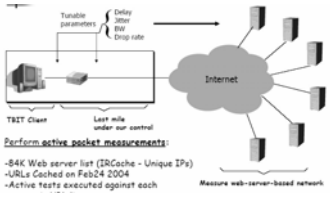
- Survey says



ECN < 1%
SACK ~ 80%
timestamp ~ 13%
window 64k ~ 73%
window scale ~ 15%

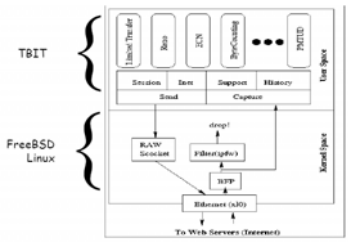
TBIT, TCP behavior inference tool

- How to determine the flavor of TCP a remote host is running?
- Passive info (tcpdump): observe SYN/SYN-ACK and window advertisements window scale, timestamps, SACK, ECN, MTU discovery
- For TCP, all the "action" is on the sender side – make remote send data
- Request web pages and induce packet loss and observe recovery
  - Emulator test bed and/or TBIT tool

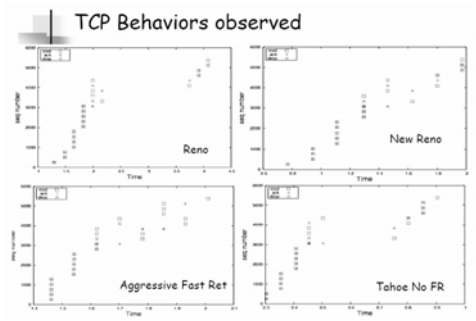


TBIT

- Modified TBIT client that "drops" packets 13 and 15 from remote
- With tcpdump observe remote's recovery
  - 3 dup retransmit or timeout, fast recovery or slow start, SACK info
  - Infer TCP flavor of remote



TBIT observations



Alternate transport protocols

- If you think TCP can't do the job …
- Need broadcast/multicast or request/reply (acceptable loss)
- Audio/video streaming
  - TCP slow-start annoying, adds delay
  - TCP loss recovery adds jitter and delay
  - It's ok to drop some packets
  - Use rate-based delivery (application level protocol management) UDP
- High bandwidth long delay networks, TCP too slow?
  - Rate-based startup, rate-based recovery, or ?
  - Use your own UDP, TCP-like protocol
    - Build flow control and congestion control into your app. (DCCP, atou)
  - Use rate-based UDP with loss feedback for rate control
    - No bursts, anecdotal evidence for higher throughput and less drops with pacing
    - UDT, SABUL, Tsunami
    - Stable, fair, friendly?

TCP-friendly protocols in UDP

- Datagram protocols can be TCP-friendly
- IETF's Datagram Congestion Control Protocol (DCCP)
  - Goal: standard way to implement *congestion control and congestion control negotiation for real-time applications*
  - Features:
    - Unreliable flow of datagrams with ACKs
    - Reliable handshake for open/close
    - Path MTU discovery
    - feedback to sender: ACK, dropped, corrupted, ECN marked
    - *TFRC* (*TCP-Friendly Rate Control*, a form of equation-based congestion control), minimizes abrupt changes in the sending rate while maintaining longer-term fairness with *TCP*
- Developing your own "protocols" in UDP is easy, no kernel mods required
  - Application-level packet processing will be slower than kernel implementation
  - Need other kernel-level transport protocols?

Almost TCP over UDP

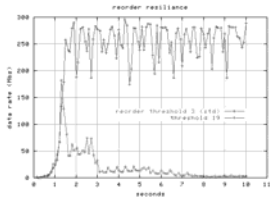
- atou (ORNL) – a user-level TCP-like transport over UDP
- A test harness for evaluating different TCP-like control options and doing experiments over real nets (test beds, emulators, Internet)
- Components: client and ACK server, written in C
- Packet header carries sequence number and timestamp
- ACK server can be configured for delayed ACK, SACK
- Client config file specifies transport operational parameters
  - Support for Reno, NewReno, Vegas, SACK, FACK, HS TCP, STCP
  - Set window size, dup ACK threshhold, initial slow-start window, AIMD, droplist, MSS
  - Optional tracing to file (invasive) based on debug level
  - Summary report of retransmits, min/max RTT, min/max cwnd ….
- Insights
  - At high data rates, user-level packet handling is CPU-intensive and ACKs are often dropped
  - OS scheduling can add further delays
  - If we were doing real file I/O, performance would suffer more

atou output

```
config: portcli $Revision: 1.29 $ port 7890 debug 4 Tue May 22 05:18:54 2001
config: initsegs 2 mss 1460 tick 1.000000 timeout 0.500000
config: rcvrwin 300 increment 1 multiplier 0.500000 thresh_init 1.000000
config: newreno 1 sack 0 delack 0 maxpkts 12000 burst_limit 0 dup_thresh 3
config: sndbuf 32768 rcvbuf 32768
swift => stingray.ccs.ornl.gov 38.835049 Mbs win 300 rxmts 0
3.609111 secs 17520000 good bytes goodput 4854.381103 KBs 38.835049 Mbs
pkts in 11542 out 11999 enobufs 0
total bytes out 17518540 loss 0.000 % 38.831813 Mbs
rxmts 0 dup3s 0 packs 0 timeouts 0 dups 0 badacks 0 maxack 24 maxburst 300
minrtt 0.065249 maxrtt 0.098987 avrgrtt 0.077747
rto 0.067194 srtt 0.066705 rttvar 0.000122
win/rtt = 45.069119 Mbs bwdelay = 377 KB 258 segs
snd_nxt 12000 snd_cwnd 300 snd_una 12000 ssthresh 300 snd_max 12000
```

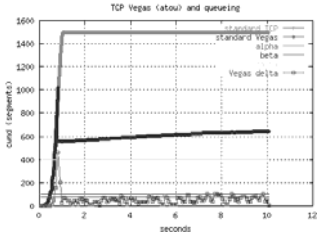

Some atou experiments

- Design originally motivated by broken AIX TCP stack
- Allows us to try different TCP flavors on a particular path
- Allowed us to try out Floyd's HS TCP, moderated slow-start, AIMD mods
- Example, Juniper router re-ordering packets at LBL, but linux on both ends was "adjusting" dup threshold. We wanted to see how a dumber TCP would react. (plot data provided by atou trace files)



atou vegas trace

- Plot vegas "diff" – increase/decrease cwnd



- Also did experiments with mss > MTU (IP frags), got better performance than 1500

UDP packet pacing

- Doing rate-based sends at the application level in UDP in UNIX
  - UNIX software timers (see table)
    - OS real-time scheduler options?
  - Spin loops/busy wait (consume CPU cycles!)
    - Calibrate or spin gettimeofday()/rdtsc()
  - PC hardware timer, 122 us resolution
    - not accessible by application

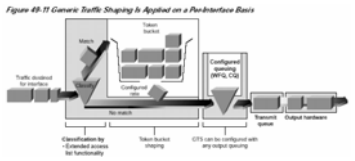
Timer	resolution
sleep	1 s
nanosleep	10 ms
select	10 ms
spin	1 us

```
tdelay(us)
{
  double secs();
  double end = secs() + us*1.e-6;
  while (secs() < end);
}
```

- OK for video/audio streaming requirements
  - Voice 64 kbs
  - MPEG 400 kbs
- Example iperf –u –b 30m –c target
  - Uses gettimeofday() busy-wait
- Trouble for high-bandwidth pacing
  - Gigabit/second rate with 1500 bytes packets → 12 us spacing
  - Bigger MTU (jumbo frames) would help (some, 72 us spacing)
  - Cluster pacing (bursts again ☹)

Other packet pacing options

- Maybe a NIC that does pacing
  - Different rates for different flows?
- ATM provides CBR virtual circuits
  - Need infrastructure for policy/scheduling/pricing
- Have routers do pacing (traffic shaping)
  - Cisco General Traffic Shaping (GTS)
  - Queue bursts, then send (token bucket) at the specified rate

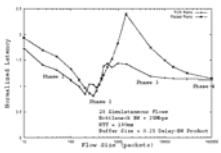


TCP packet pacing

- (anecdotal) many times we could get rate-based UDP with iperf to run at 300 to 400 mbs with no loss between ORNL and CERN, but we could rarely get TCP to run that fast because of packet loss … bursts?
- In theory, ACK clocking should smoothly pace TCP packets
- Burstiness in TCP
  - Slow-start, may send twice the bottleneck bandwidth rate. If router queue size is small, could drop lots of packets
  - Cumulative ACK for a successfully retransmitted (lost) packet could cause a packet burst from the sender
  - ACK compression from congestion on reverse path could cause bursts
  - ACK clocking only spreads data at the bottleneck rate. For high speed multiplexed paths, each flow may transmit a burst and remain idle for the remainder of the RTT
- Goal of pacing is to spread the transmission of a window (cwnd) of packets over the duration of the RTT

TCP packet pacing

- Benefits
  - Reduced loss for a flow and for the net (maybe)
  - Minimal queuing until load matches bottleneck capacity (too good?)
- Difficulties
  - Implementation requires a timer and bandwidth estimate (Vegas?)
    - ns Vegas/RBP and Reno/RBP
  - More pronounced synchronization with droptail
    - More losses sometimes than Reno
  - Late congestion signals because of minimal queuing!
    - Don't get loss til link is saturated
  - Sometimes performs worse than Reno (throughput and latency)
  - Too nice, competes poorly with other TCP flavors
- Other (simple) burst controls
  - Linux and *BSD have "max burst" parameter



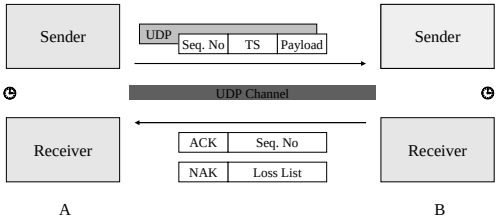
Rate-based UDP transports for high speed nets

- Since no kernel mods required, everyone can develop their own UDP-based file transfer protocol
  - Avoid TCP's annoying friendliness
  - Blast packets (Denial of Service) or with NAK's/retransmit for reliability
  - Rate-based sender with ACK/NAK receiver
    - Often separate control channel for ACK/NACK and rate info (TCP)
    - Timers, sequence numbers, flow control, congestion control
      - Rate-based slow start
      - Rate adjustments for flow control and congestion control
        - Sending rate adjusted to keep error rate below a threshold
- RBUDP, FOBS, Tsunami, SABUL, UDT ….
  - We tested several of these at ORNL a few years back (LFNs)
    - Usually slower than TCP
    - excessive retransmits (duplicates)
    - Slow to respond to congestion (unfriendly)
    - Unstable to the point of unusable
  - Continue to evolve

UDT

- UDT: UDP-based Data Transfer
  - Reliable, application level, duplex, transport protocol, over UDP with reliability, congestion, and flow control
  - Son of SABUL
  - Implementation: Open source C++ library
- Goals: fast, fair, friendly
  - High utilization of the abundant bandwidth with either single or multiplexed connections
  - Intra-protocol fairness, RTT independence
  - TCP compatibility
- Explicit loss information feedback (NAK)
- Four timers: rate control, ACK, NAK and retransmission timer
  - Rate control and ACK are triggered periodically
  - NAK timer is used to resend loss information if retransmission is not received in an increasing time interval

UDT Protocol Architecture



UDT congestion control

- Rate based congestion control (Rate Control)
  - RC tunes the packet sending period.
  - RC is triggered periodically.
  - RC period is constant of 0.01 seconds.
- Window based flow control (Flow Control)
  - FC limits the number of unacknowledged packets.
  - FC is triggered on each received ACK.
- Slow-start is controlled by FC
  - Similar to TCP, but only occurs at the session beginning.

UDT Rate Control

- AIMD:
  - **Increase parameter** is related to link capacity and current sending rate;
  - **Decrease factor** is 1/9, but does not decrease for all loss events.
- Link capacity is probed by packet pair, which is sampled UDT data packets.
  - Every 16th data packet and it successor packet are sent back to back to form a packet pair.

  - The receiver uses a median filter on the interval between the arrival times of each packet pair to estimate link capacity.

UDT Rate Control

- Number of packets to be increased in next rate control period (RCTP) time is:

$$inc = \max(10^{\lceil \log_{10}((B-C)\times MSS\times 8)\rceil} \times \beta / MSS, 1/MSS)$$

  where *B* is estimated link capacity, *C* is current sending rate. Both are in packets per second. MSS is the packet size in bytes. β = 1.5 * 10^{-6}.
- Decrease sending rate by 1/9 when a NAK is received, but only if
  1. largest lost sequence number in NAK is greater than the largest sequence number when last decrease occurred; or
  2. The number of NAKs since last decrease has exceeded a threshold, which increases exponentially and is reset when condition 1 is satisfied.

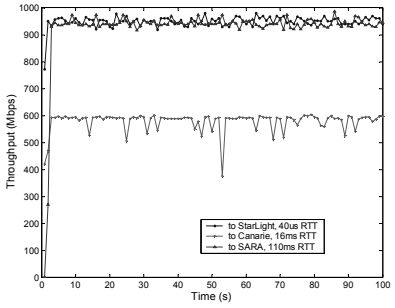
B-C mbs	increase (pkts)
1000,1000	10
100,1000	1
10,100	0.1
1,10	0.01
0.1,1	0.001
< 0.1	0.0001

UDT Flow Control

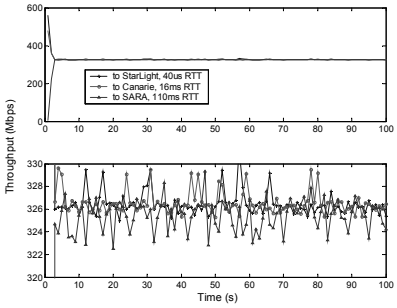
BDP

- W = W*0.875 + *AS**(RTT+ATP)*0.125
- ATP is the ACK timer period, which is a constant of 0.01 seconds.
- *AS* is the packets arrival speed at receiver side.
  - The receiver records the packet arrival intervals.
  - *AS* is calculated from the average of latest 16 intervals after a median filter.
  - It is carried back within ACK.

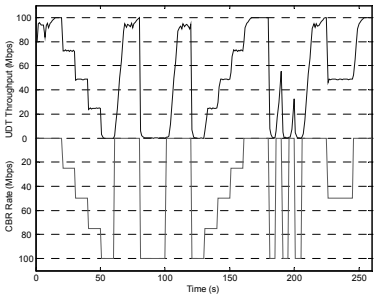
UDT performance



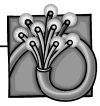
UDT intra-protocol fairness



UDT convergence/stability (simulation)



Optical network directions



- TCP over optical circuits, transparent … not a problem
  - Over OC3, OC12, OC192 …
- New optical options
  - Optical burst switching
    - Circuit only needed when a burst of data is ready to send
  - Optical packet switching
    - Issues of queuing and mapping IP/TCP onto optical packets
    - Could be queuing delays and loss
  - Lambda switching (like MPLS)
    - Circuit based
    - need infrastructure for circuit setup/takedown, scheduling
    - User application has a dedicated 10Gig path – no congestion!
    - Transport protocol
      - Do we need all the TCP baggage?
      - Low media loss rates, less than < 10^{-14}
      - UDP blasters should be good enough (UDT)

Next time …

- Slow-speed networks (compression)
- Asymmetric networks