Internet Programming & Protocols Lecture 18

Bandwidth estimation

Auto-tuning (not tuning cars)

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

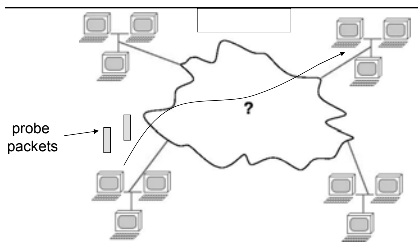
Accelerating TCP

- Tuning configuration parameters
  - SNDBUF/RCVBUF – bandwidth-delay product
  - Txquelen
  - RFC1323 (window scaling, timestamps)
  - Nagle, delayed ACK
  - Initial slow-start
- Speeding recovery after packet loss
  - Fast retransmit, fast recovery
  - SACK/FACK
  - AIMD, STCP, HSTCP, BI-TCP, Westwood
- Avoiding packet loss
  - Dup threshold (out of order resilience)
  - Slow-start and congestion avoidance (reduces losses)
  - Vegas/FAST

Sizing send/receive buffers

- Tools and techniques for estimating the bandwidth-delay of a path
  - Path bandwidth capacity (narrow link)
    - pathrate pathchar capprobe
  - Available path bandwidth (tight link)
    - Pathload pathchirp pathvar
- Incorporating path buffer estimations into TCP or applications
  - Auto-tuning
  - TCP AIMD (cwnd adjustments) is already doing dynamic buffer tuning up to the SNDBUF/RCVBUF limit

The Internet black box



- End-systems can infer network state through end-to-end (e2e) measurements
  - Without info from routers
  - Objectives: accuracy, speed, non-intrusiveness
- Many of these powerpoint slides courtesy of Dovrolis at GaTech

Bandwidth estimation in packet networks

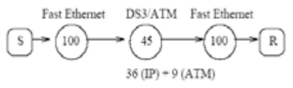
- Bandwidth estimation (bwest):
  - Infer various throughput-related metrics with end-to-end network measurements
- Early works:
  - Keshav's packet pair method ('89)
  - Bolot's capacity measurements ('93)
  - Carter's bprobe and cprobe tools ('96)
  - Jacobson's pathchar for per-hop capacities ('97)
  - Melander's TOPP method for avail-bw estimation ('00)
- **In the last 3-4 years:**
  - Several fundamentally new estimation techniques
  - Many research papers at prestigious conferences
  - More than a dozen of new measurement tools
  - Several applications of bwest methods

Applications of bandwidth estimation

- Large TCP transfers and congestion control
  - Bandwidth-delay product estimation
  - Socket buffer sizing
- Streaming multimedia
  - Adjust encoding rate based on avail-bw
- Intelligent routing systems
  - Overlay networks and multihoming
  - Select best path based on capacity or avail-bw
- Content Distribution Networks (CDNs)
  - Choose server based on least-loaded path
- SLA and QoS verification
  - Monitor path load and allocated capacity
- End-to-end admission control
- Network "spectroscopy"
- TCP-based estimators employed in Vegas, FAST, Westwood

Path capacity

- Maximum possible end-to-end throughput at IP layer
  - In the absence of any cross traffic
  - Achievable with maximum-sized packets



- If C_i is capacity of link i, end-to-end capacity C defined as:

$$C = \min_{i=1,\dots,H} C_i$$

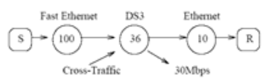
- Capacity determined by narrow link

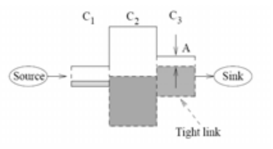
Available bandwidth definition

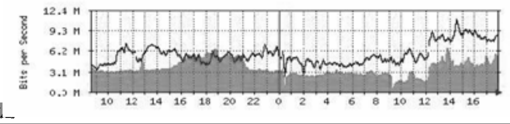
- Per-hop average avail-bw:
  - $A_i = C_i\,(1-u_i)$
  - u_i: average utilization
  - A.k.a. residual capacity
- End-to-end avg avail-bw A:

$$A = \min_{i=1,\dots,H} A_i$$

- Determined by tight link
- ISPs measure per-hop avail-bw passively (router counters, MRTG graphs)

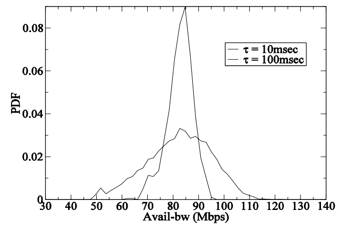






Available bandwidth distribution

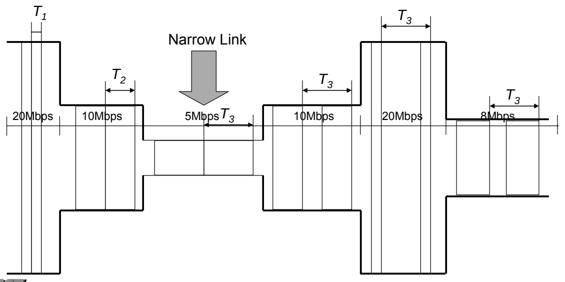
- Avail-bw has significant variability
  - Need to estimate second-order moments, or even better, the avail-bw distribution
- Variability depends on averaging timescale τ
  - Larger timescale, lower variance
- Distribution is Gaussian-like, if τ >100-200 msec and with sufficient flow multiplexing



Capacity estimation -- Packet Pair Dispersion

(slide by Ling-Jyh Chen, UCLA)

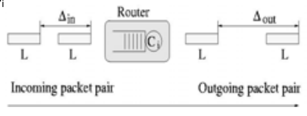
Like the question on assignment 6, use arrival times of ACK's



Packet pair dispersion

- Packet Pair (P-P) technique
  - Originally, due to Jacobson & Keshav
- Send two equal-sized packets back-to-back
  - Packet size: L
  - Packet trx time at link i: L/C_i
- P-P dispersion: time interval between last bit of two packets
- Without any cross traffic, the dispersion at receiver is determined by narrow link:

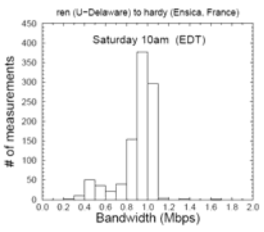
$$\Delta_{out} = \max\left(\Delta_{in}, \frac{L}{C_i}\right)$$

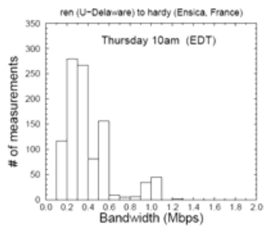


$$\Delta_R = \max_{i=1,\dots,H}\left(\frac{L}{C_i}\right) = \frac{L}{C}$$

Cross traffic interference

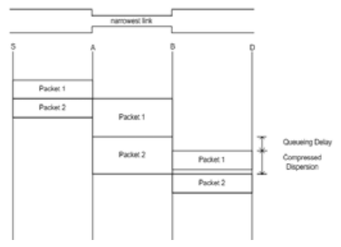
- Cross traffic packets can affect P-P dispersion
  - P-P expansion: capacity underestimation
  - P-P compression: capacity overestimation
- Noise in P-P distribution depends on cross traffic load
- Example: Internet path with 1Mbps capacity





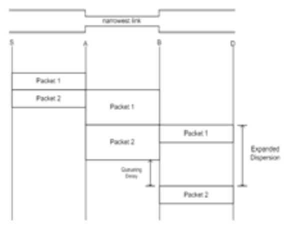
Compression of p-p dispersion

- First packet queueing => compressed dispersion
  - E.g., ACK compression
    - ACK's queued up in router on fast link, they pop out "fast"
  - Result: capacity overestimation



Expansion of p-p dispersion

- Second packet queueing => expansion of dispersion
  - Another packet gets between your probe packets
  - Result: capacity underestimation



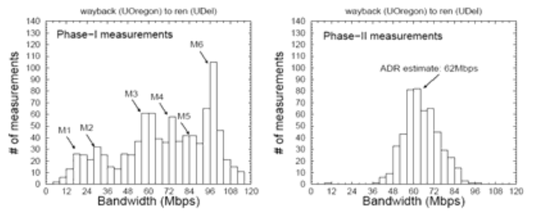
Estimation tools

- Various tools have been developed to send
  - Lots of packet pairs
  - Or packet trains (N back-to-back packets)
- Statistical analysis of timings of returning packets
- These tools can be invasive, and high confidence often requires long runs (minutes … hours)
- The trick is to develop efficient and accurate tools
  - Typically one-sided (no remote server), use ping or traceroute like probes

pathchar
pchar
pathprobe
capprobe
iperf pathrate pathload (2-sided)

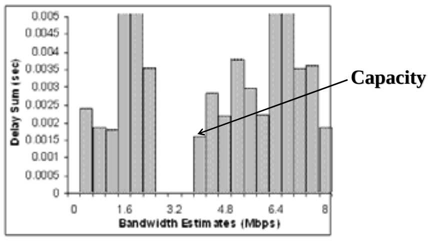
pathrate (capacity estimator)

- Perform many P-P measurements & detect local modes
  Estimate Average Dispersion Rate (ADR) with long packet trains
  - ADR is provably a lower bound for path capacity
  - Reject local modes < ADR
- Select capacity from remaining modes (kurtosis-density statistic)
- See www.pathrate.org for more details and estimation tool



CapProbe

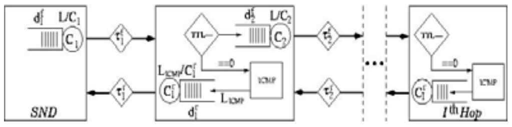
- Both *expansion* and *compression* are result of probing packets experiencing queuing
- Key insight: packet pair that sees zero queueing delay would yield exact estimate



Per-hop capacity estimation

- traceroute on steroids
  - Tools that try to calculate capacity of each link in path!
  - Slow and invasive
- pathchar
- clink
- pchar

Variable Packet Size (VPS) probing



- Sender transmits probes with
  - Different packet sizes L
  - Time-to-live (TTL) set to expire at I^{th} router
- Receive ICMP time-exceeded messages from I^{th} router
- RTT up to I^{th} router sum of
  - Serialization delays L/C_i, L_{ICMP}/C_i^r
  - Propagation delays τ_i^f, τ_i^r
  - Queuing delays d_i^f, d_i^r

Capacity Estimation with VPS

- Assume minimum RTT for each packet size does not include any queuing delay

$$T_I(L) = \sum_{i=1}^{I} \left(\frac{L}{C_i} + \tau_i^f + \frac{L_{ICMP}}{C_i^r} + \tau_i^r \right)$$

- Linear relation between minimum RTT and and probe packet length L

$$T_I(L) = \alpha_I + \beta_I L \qquad \beta_I = \sum_{i=1}^{I} \frac{1}{C_i}$$

- Capacity of I th link

$$C_I = \frac{1}{\beta_I - \beta_{I-1}}$$

pathchar

- Traceoute-like probes with different packet sizes
- Repeated tests til "confident" … or doesn't converge ☹ … slow

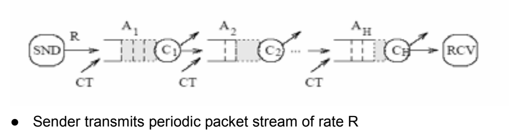
```
pathchar ka9q.ampr.org
pathchar to ka9q.ampr.org (129.46.90.35)
mtu limitted to 1500 bytes at local host
doing 32 probes at each of 64 to 1500 by 44
0 192.172.226.24 (192.172.226.24)
| 9.3 Mb/s, 269 us (1.83 ms)
1 pinot (192.172.226.1)
| 85 Mb/s, 245 us (2.46 ms), 1% dropped
2 sdscdmz-fddi.cerf.net (198.17.46.153)
| 45 Mb/s, -13 us (2.70 ms)
3 qualcomm-sdsc-ds3.cerf.net (134.24.47.200)
| 8.8 Mb/s, 1 us (4.07 ms)
4 krypton-e2.qualcomm.com (192.35.156.2)
| 5.2 Mb/s, 1.02 ms (8.42 ms)
5 ascend-max.qualcomm.com (129.46.54.31)
| 53.2 Kb/s, 4.20 ms (243 ms)
6 karnp50.qualcomm.com (129.46.90.33)
| 12 Mb/s, -172 us (243 ms), +q 8.96 ms (13.0 KB) *3, 6% dropped
7 unix.ka9q.ampr.org (129.46.90.35)
7 hops, rtt 11.1 ms (243 ms), bottleneck 53.2 Kb/s, pipe 4627 bytes
```

Measurements - Internet, Internet2

- CapProbe implemented using PING packets, sent in pairs

To	UCLA-2		UCLA-3		UA		NTNU	
	Time	Capacity	Time	Capacity	Time	Capacity	Time	Capacity
Cap Probe	0'03	5.5	0'01	96	0'02	98	0'07	97
	0'03	5.6	0'01	97	0'04	79	0'07	97
	0'03	5.5	0'02	97	0'17	83	0'22	97
Pathrate	6'10	5.6	0'16	98	5'19	86	0'29	97
	6'14	5.4	0'16	98	5'20	88	0'25	97
	6'5	5.7	0'16	98	5'18	133	0'25	97
Pathchar	21'12	4.0	22'49	18	3 hr	34	3 hr	34
	20'43	3.9	27'41	18	3 hr	34	3 hr	35
	21.18	4.0	29'47	18	3 hr	30	3 hr	35

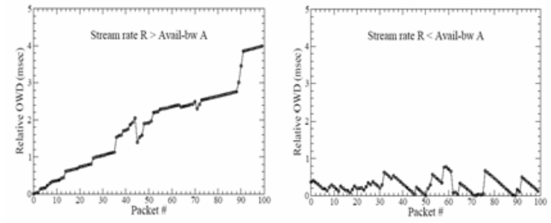
Bandwidth estimation (pathload)



- Sender transmits periodic packet stream of rate R
  - K packets, packet size L, interarrival T = L/R
- Receiver measures One-Way Delay (OWD) for each packet
  - $D(k) = t_{arv}(k) - t_{snd}(k)$
  - OWD variations: $\Delta(k) = D(k+1) - D(k)$
    - Independent of clock offset between sender/receiver

Example of OWD variations

- 12-hop path from U-Delaware to U-Oregon
  - K=100 packets, A=74Mbps, T=100μsec
  - R_{left} = 97Mbps, R_{right}=34Mbps
- Increasing OWDs means R>A
- Almost constant OWDs means R<A

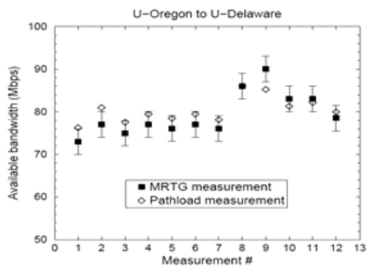


Pathload's iterative algorithm

- Source: send n-th stream with rate R(n)
- Receiver:
  - Measure OWDs of n-stream
  - Check for presence of increasing trend
  - Notify sender
- Source:
  - If "increasing OWDs", i.e., R(n)>A,
    - $R_{max} = R(n)$
  - Else, R(n) < A,
    - $R_{min} = R(n)$
  - $R(n+1) = (R_{max} + R_{min}) / 2$
- Exit when $R_{max} - R_{min} < \omega$

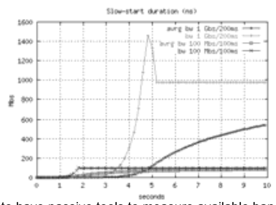
Pathload tool

- Pathload: also available at www.pathrate.org
  - About 10-20 seconds per measurement
  - Accurate within 10% as long as path does not include multiple bottlenecks



Bandwidth estimation with iperf

- You could manually do rate-based estimation with UDP and iperf –u –b NNm –c target
- Could do estimation with TCP and iperf but due to slow-start, final datarate not accurate so use iperf –i 1

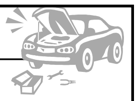


- NOTE: would be nice to have passive tools to measure available bandwidth. Active probes affect the thing you are trying to measure, e.g. other TCP flows may backoff in response to your probe packets. Although if you are trying to figure out how your TCP application is going to perform, using TCP to measure the available bandwidth may be the right thing…

Problems with bandwidth/capacity estimation

- Invasive and often slow
- Snapshot of path … could change
  - Burstiness makes it harder
- Layer-2 devices (switches) add to delay but not to TTL, so mess up link-by-link estimators
- Multiple tight links cause underestimation in all avail-bw estimation tools
- Probes affect available bandwidth estimation
- Not very good at gigabit and higher links
- Sender-side estimation techniques are used in Vegas, FAST, Westwood

Auto-tuning TCP



- Objective of auto-tuning
  - Applied bandwidth estimation
  - Selecting the perfect SNDBUF/RCVBUF size
- Application examples
  - NLANR's ftp
  - DRS ftp
- Out-of-band auto-tuning
  - Net100
- Kernel auto-tuning
  - Dynamic Right Sizing
  - Web100
  - Linux 2.6

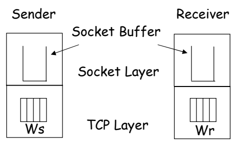
Too small, too big, just right

- Window size too small, limit bandwidth
- Window size too big
  - consume OS memory resources
  - Doesn't improve bandwidth
    - If bigger than router queue space, can cause loss ➔ worse performance
  - Add to delay (RTT increases because of q'ing)
    - Consume router memory resources
- OS limits to setsockopt() or default SNDBUF/RCVBUF sizes
- If you knew bandwidth-delay product of path, could set it just right, BUT available bandwidth will vary with load …
- Example, 100ms RTT, 1.6 mbs path, 1000 byte packets, Newreno
  - Bandwidth delay product ➔ window = 20 packets (pkt transmit time: 5 ms)

```
window    datarate    RTT     maxq
  20        1481      105      10
  10         721      105      10
  40        1481      205     200
  40         961      155      10     20 drops
```

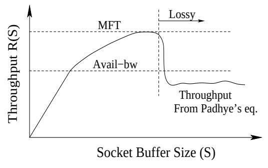
TCP flow-control

- Socket-layer buffers
  - Send buffer: B_s
  - Receive buffer: B_r
- TCP windows
  - Send: $W_s <= B_s$
  - Congestion: W_c
  - Receiver (advertised): $W_r <= B_r$
- $W_s = \min \{W_c, W_r\}$ either sender or receiver can control window
- Ideally, TCP send window W_s should be set to connection's bandwidth-delay product
  - "Bandwidth": connection's fair share
  - "Delay": connection's RTT
  - Achieve efficiency, fairness, no congestion
- Recall, TCP probes for path capacity, by increasing cwnd til PACKET LOSS – let's try to avoid this by judicious choice of W
  - Too small, slow
  - Too big, loss and/or added delay (queuing)



Socket buffer size

- $W_s = \min \{W_c, W_r\}$
- But $W_r <= S$
  - S: rcv socket buffer size
- We can limit S so that connection is limited by receive window: $W_s = S$
- Non-congested path:
  - Throughput R(S) = S/T(S)
  - R(S) increases with S until it reaches MFT
  - MFT: Maximum Feasible Throughput (onset of congestion)
  - Objective: set S close to MFT, to avoid any losses and stabilize TCP send window to a safe value
- Congested path:
  - Path with losses before start of target transfer
  - Limiting S can only reduce throughput



How to dynamically estimate capacity?

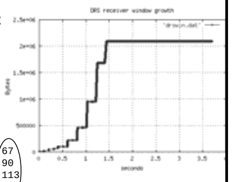
- **Out-of-band probes or historical data base?**
  - Use bw-est tools: pathchar etc.
  - Network Weather Service
    - Grid scheduling
    - Periodic probes: network capacity, CPU load
  - Net100 (tuning daemon)
- **Network-aware ftp (no kernel mods, out of band)**
  - NLANR ftp (pre-transfer probes)
    - Packet train of ICMP or UDP packets to estimate RTT and bandwidth
    - Calculate window size for FTP from bandwidth-delay product
  - DRS ftp
    - Modified GridFTP
    - Continuously monitor RTT and data rate and adjust buffer size
- **TCP stack estimates (kernel mods, in band)**
  - Sender side
  - Receiver side
    - When receive data rate becomes constant, clamp advertised window
  - Worry about initial window-scale value … can't grow bigger than that ☹

Sender-side tuning

- Mathis, '98, mods to NetBSD kernel
- Receiver must advertise a big window (big RCVBUF)
- Window-scale must be "adequate"
- If application has not explicitly set SNDBUF, kernel will "grow" SNDBUF as cwnd grows
  - Memory resource will match bandwidth-delay product as measured by cwnd
  - Won't be too small, won't be too big … just right
  - Still can experience packet loss
- Helps legacy applications with no options for buffer tuning
  - 10x improvement (over system default SNDBUF size)
  - No changes to application
- Estimation could be adversely affected by reverse path load

Receiver side tuning

- Receiver adjusts advertised window based on observed bandwidth-delay product
  - Use RTT estimation (DRS) or TCP timestamps for delay estimation
  - Grow advertised window in anticipation of slow-start doubling
    - Still will slightly over-estimate and thus still experience packet loss
    - But won't consume unnecessary buffer space at receiver
  - Not affected by reverse path traffic
- LANL's Dynamic Right Sizing, mods to Linux
- PSC Web100 auto-tuning
- Linux 2.6 auto-tuning
  - Sender side
  - And receiver side



Observe with tcpdump
```
192.91.245.29.5001 > 160.91.212.75.34287: . ack 1449 win 67
192.91.245.29.5001 > 160.91.212.75.34287: . ack 2897 win 90
192.91.245.29.5001 > 160.91.212.75.34287: . ack 4345 win 113
192.91.245.29.5001 > 160.91.212.75.34287: . ack 5793 win 135
```

PSC receiver side autotuning

- Modified linux kernels with emulated path delays
- Auto-tuning vs manually set buffer sizes
- Window-scale must be big …

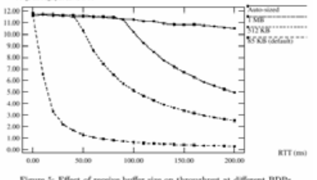


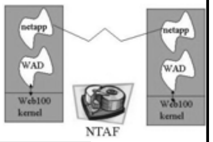
Figure 5: Effect of receive buffer size on throughput at different BDPs

Net100 TCP Tuning Daemon

- **Work-around Daemon (WAD)** … a hack
  - tune unknowing sender/receiver at startup and/or during flow
  - Web100 kernel extensions
    - pre-set *windowscale* to allow dynamic tuning
    - uses *netlink* to alert daemon of socket open/close (or poll)
    - besides existing Web100 buffer tuning, new tuning parameters and algorithms
    - knobs to disable Linux 2.4 caching, burst mgt., and sendstall
  - config file with static tuning data
    - mode specifies dynamic tuning (AIMD options, NTAF buffer size, concurrent streams)
  - daemon periodically polls NTAF for tuning data
  - can do out-of-kernel tuning (e.g., Floyd)
  - written in C (also Python version)

WAD config file

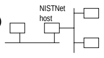
```
[bob]
 src_addr: 0.0.0.0
 src_port: 0
 dst_addr: 10.5.128.74
 dst_port: 0
 mode:  1
 sndbuf: 2000000
 rcvbuf: 100000
 wadai:  6
 wadmd:  0.3
 maxssth: 100
 divide:  1
 reorder:  9
 sendstall: 0
 delack: 0
 floyd: 1
 kellyai: 0
```



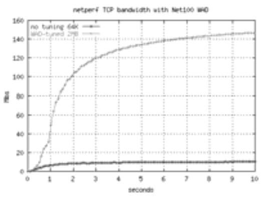
NTAF is out-of-band path prober

Tuning daemon experimental results

- Evaluating the tuning daemon in the wild
  - emphasis: bulk transfers over high delay/bandwidth nets (Internet2, ESnet)
  - tests over: 10GigE/OC192,OC48, OC12, OC3, ATM/VBR, GigE,FDDI,100/10T,cable, ISDN,wireless (802.11b),dialup
  - tests over NISTNet testbed (speed, loss, delay)



- Example
  - ORNL to PSC, OC12, 80ms RTT
  - Network-challenged app. gets 10 mbs (defaualt buffers)
  - Same app. tuned by the daemon gets 143 mbs



Next time …

- Using bandwidth estimation in the TCP kernel …
  - Delay-based congestion avoidance
    - TCP Vegas
    - TCP FAST
  - TCP Westwood