From pathchar to netchar: keeping an eye on your ISP

Allen B. Downey
Colby College
Waterville, Maine

Motivation

- Why is my network connection so slow?
- At least four culprits:
  - Your local network.
  - Your ISP.
  - Your friend’s ISP.
  - Your friend’s local network.
- Round-trip times not much help.
- Need information about:
  - The path (series of IP addresses, router names).
  - The physical characteristics of the links (latency, bandwidth).
  - The dynamic behavior of the links (queue delays, drop rates).
  - Alternate paths (network topology).

Some of the tools

- Path: traceroute
  - Sends UDP probes with small *ttl* values.
  - *ttl* expiration generates ICMP error packet with router address.

- Physical link characteristics: pathchar
  - Lots of UDP probes with different sizes.
  - Estimates latency and bandwidth.

- Dynamic link characteristics: pathchar and MINC
  - pathchar reports mean queue delay and drop rate.
  - MINC, using multicast, might yield better info more efficiently.

Pathchar history

- Van Jacobson released pathchar in April 1997.

```
pathchar to mach5.sdsc.edu (132.249.22.155)

 doing 32 probes at each of 45 sizes (64 to 1500 by 32)
 0 localhost
 |   9.7 Mb/s,   328 us
 1 cb3500-02.colby.edu (137.146.194.17)
 |   101.2 Mb/s,   40 us
 2 cb3500-00.colby.edu (137.146.238.228)
 |   9.3 Mb/s,   248 us
 3 204.70.15.170 (204.70.15.170)
 |   1.4 Mb/s,   3.83 ms
 4 166.48.64.25 (166.48.64.25)
 |   59 Mb/s,    2.40 ms,  +q 1.33 ms *12,  2% drop
 5 core4.WestOrange.mci.net (204.70.4.77)
```

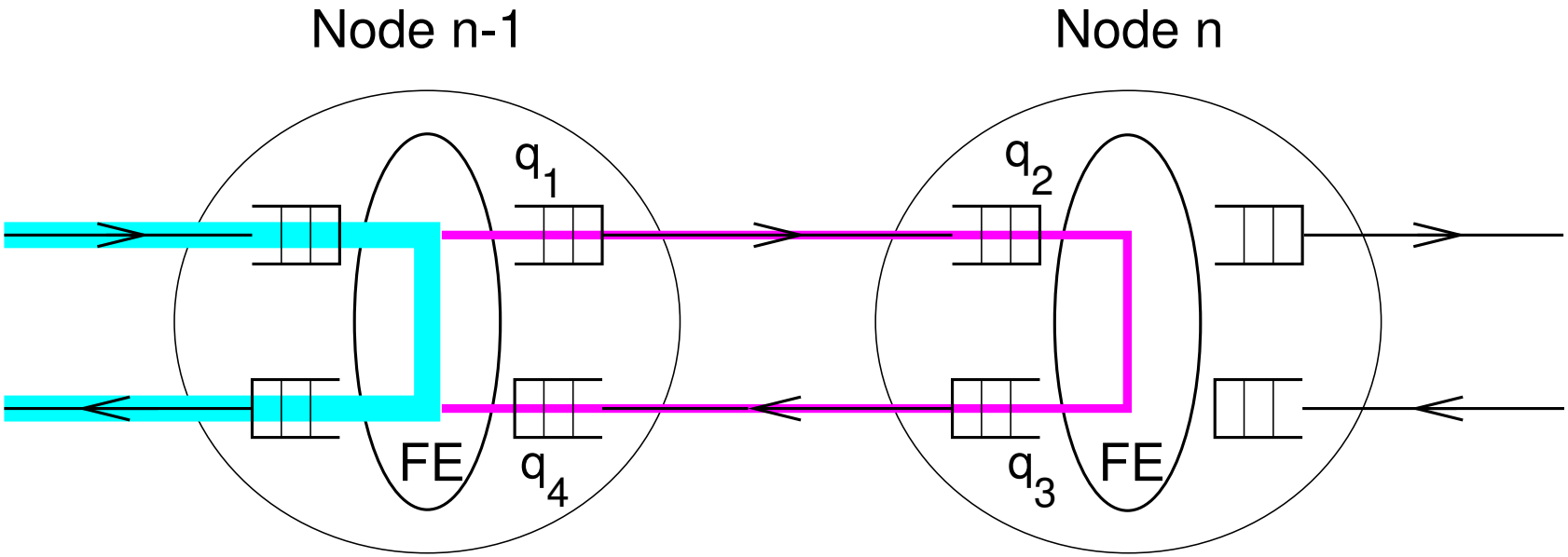
History (continued)

- No source code for alpha version, no beta version.
- Summer 1998
  - I used pathchar to collect data.
  - Reimplemented data analysis module.
- February 1999
  - Bruce Mah released pchar, an open-source reimplementation.
- Summer 1999
  - I released clink, another open-source reimplementation.
  - Additional experiments.
- January 2000
  - Bruce and I are collaborating on netchar.

Outline

- How do pathchar/pchar/clink work?
- Sources of error.
- How *well* do they work?
- Improvements:
  - Robustness
  - Accuracy
  - Efficiency
- netchar

Network model



- Similar to traceroute: use time-to-live field in IP packet.
- Measure round trip time to (*n-1)*th node and *n*th node.
- Subtract to get *rtt* from (*n-1)*th to *n*th and back.
- $rtt = q1 + (lat + ip_size / bw) + q2 + \Delta process_time$
  $+ q3 + (lat + icmp_size / bw) + q4$

Simplifying assumptions

- Start with:

$$rtt = q1 + (lat + ip_size / bw) + q2 + \Delta process_time$$
$$+ q3 + (lat + icmp_size / bw) + q4$$

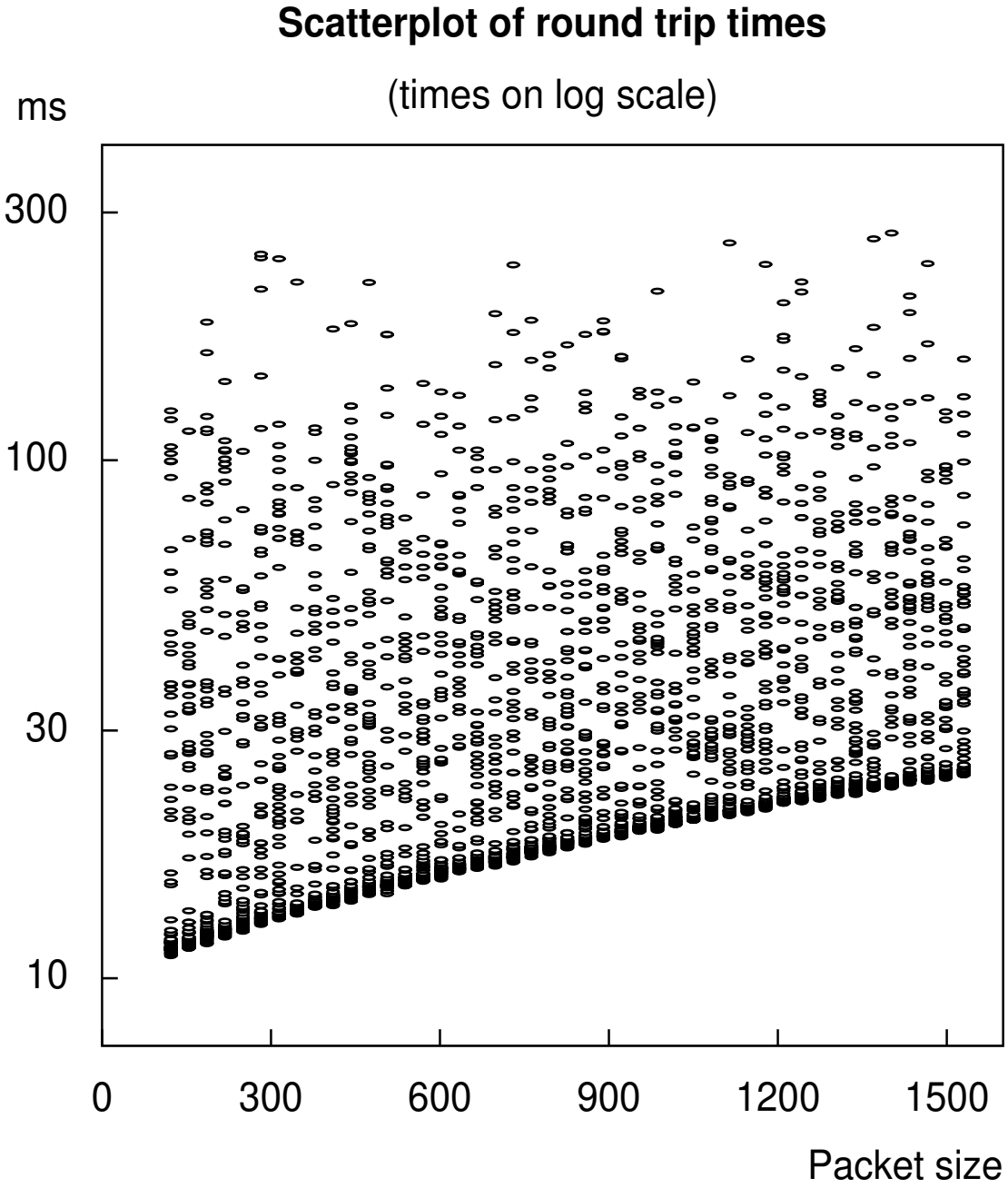
- Assume:
  - (*icmp_size / bw*) is negligible
  - Δ*process_time* is negligible
  - with some probability all queue times are near zero

- Then:

$$rtt = 2*lat + ip_size / bw$$

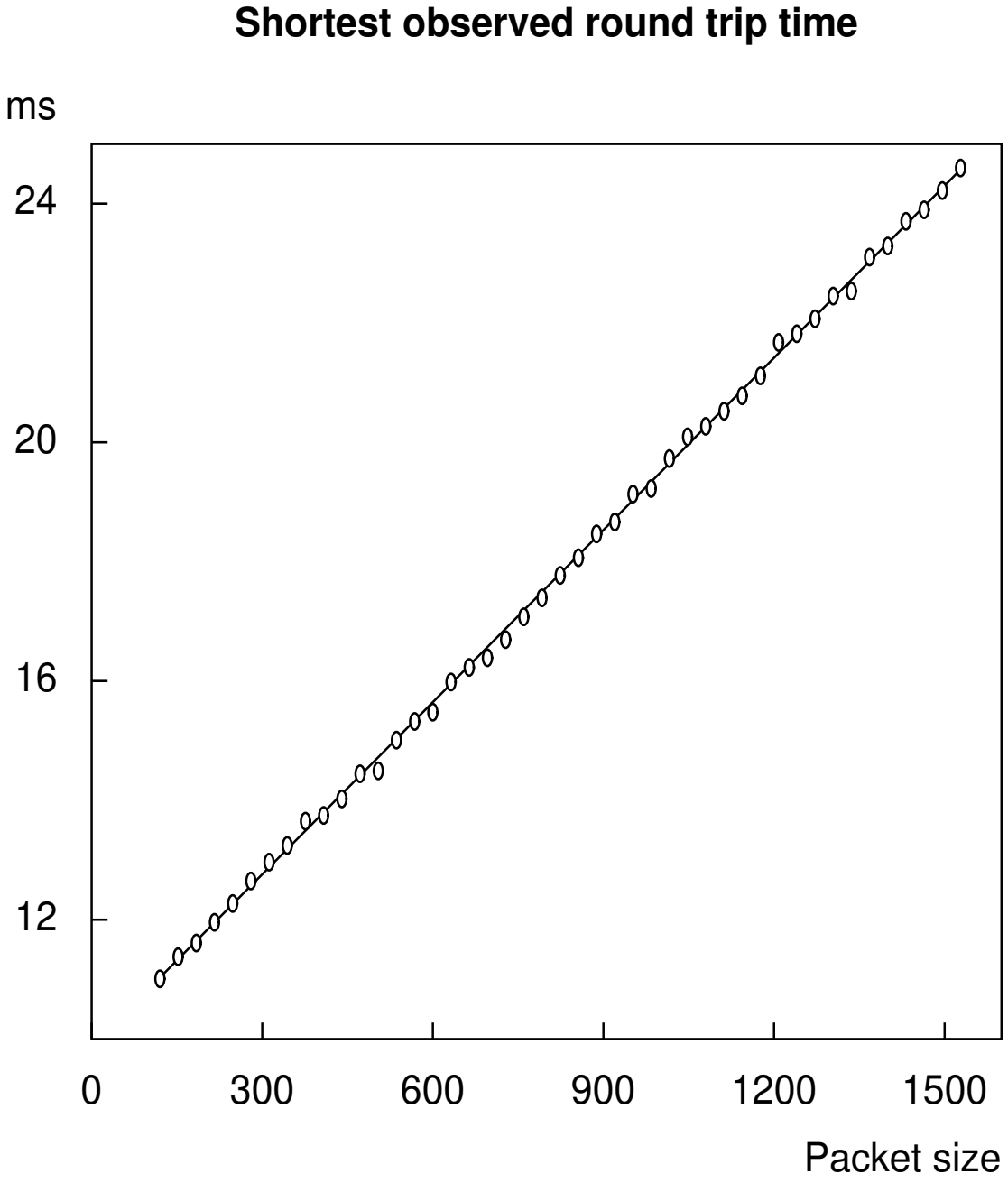
- Two parameters: 2**lat* and 1/*bw*

Minimum filtering



- Make repeated measurements over a range of packet sizes.
- Queue times vary a lot (note log scale).
- Queue time often near zero (even after six links).
- Keep lowest *rtt* in each column.

Least squares fit

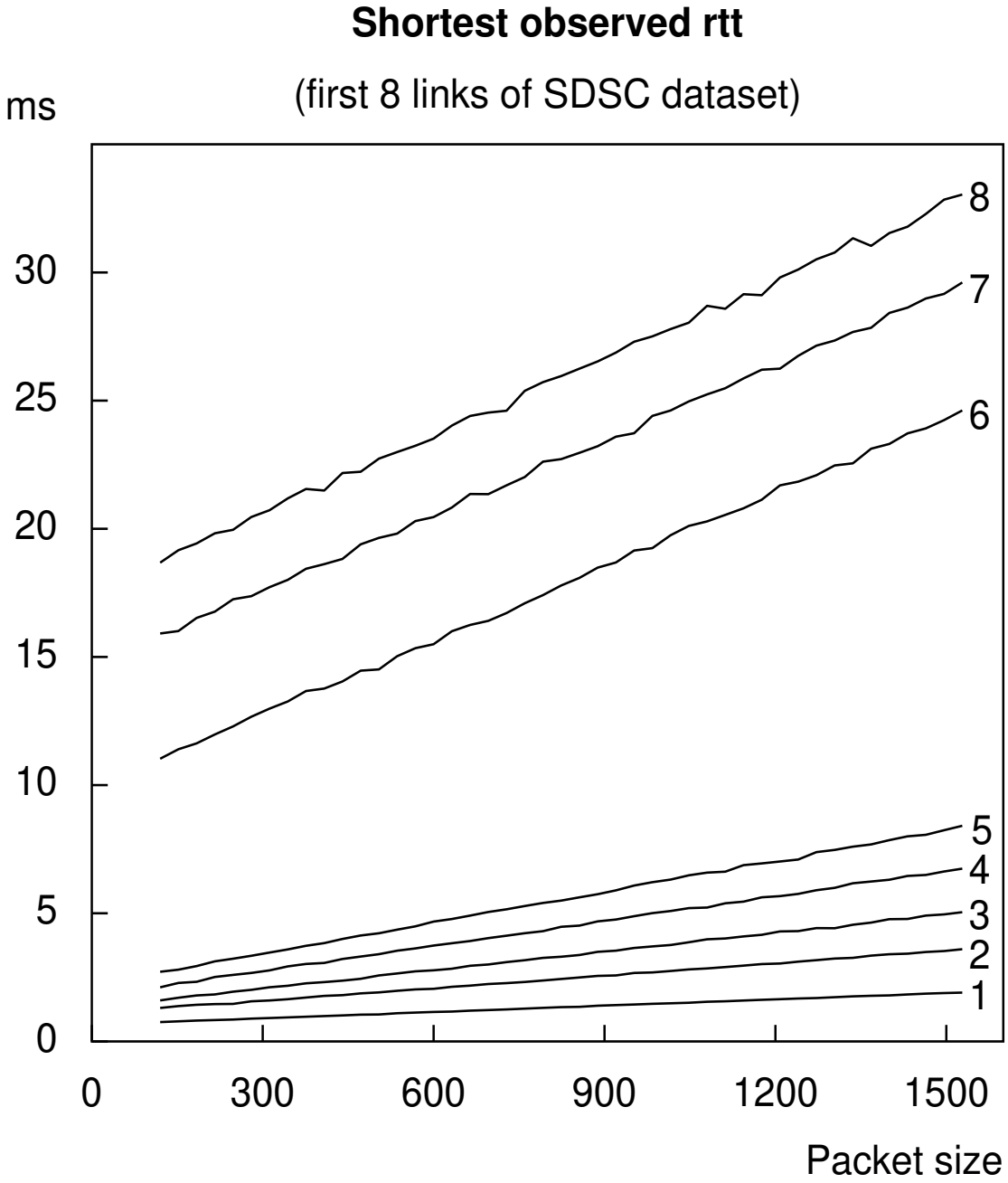


After minimum filtering, pathchar estimates:

- The intercept of the line, which is 2 * *lat*.
- The slope of the line, which is 1 / *bw*.

(*partial-path* latency and bandwidth, that is)

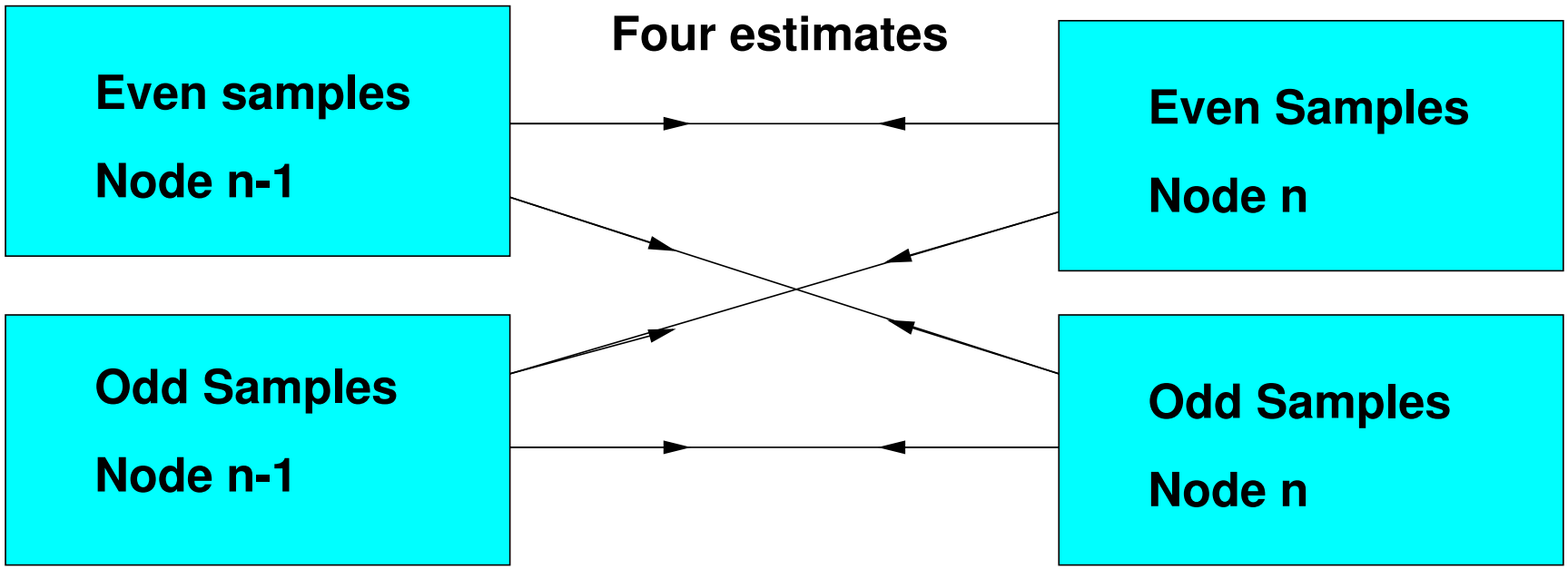
Differencing



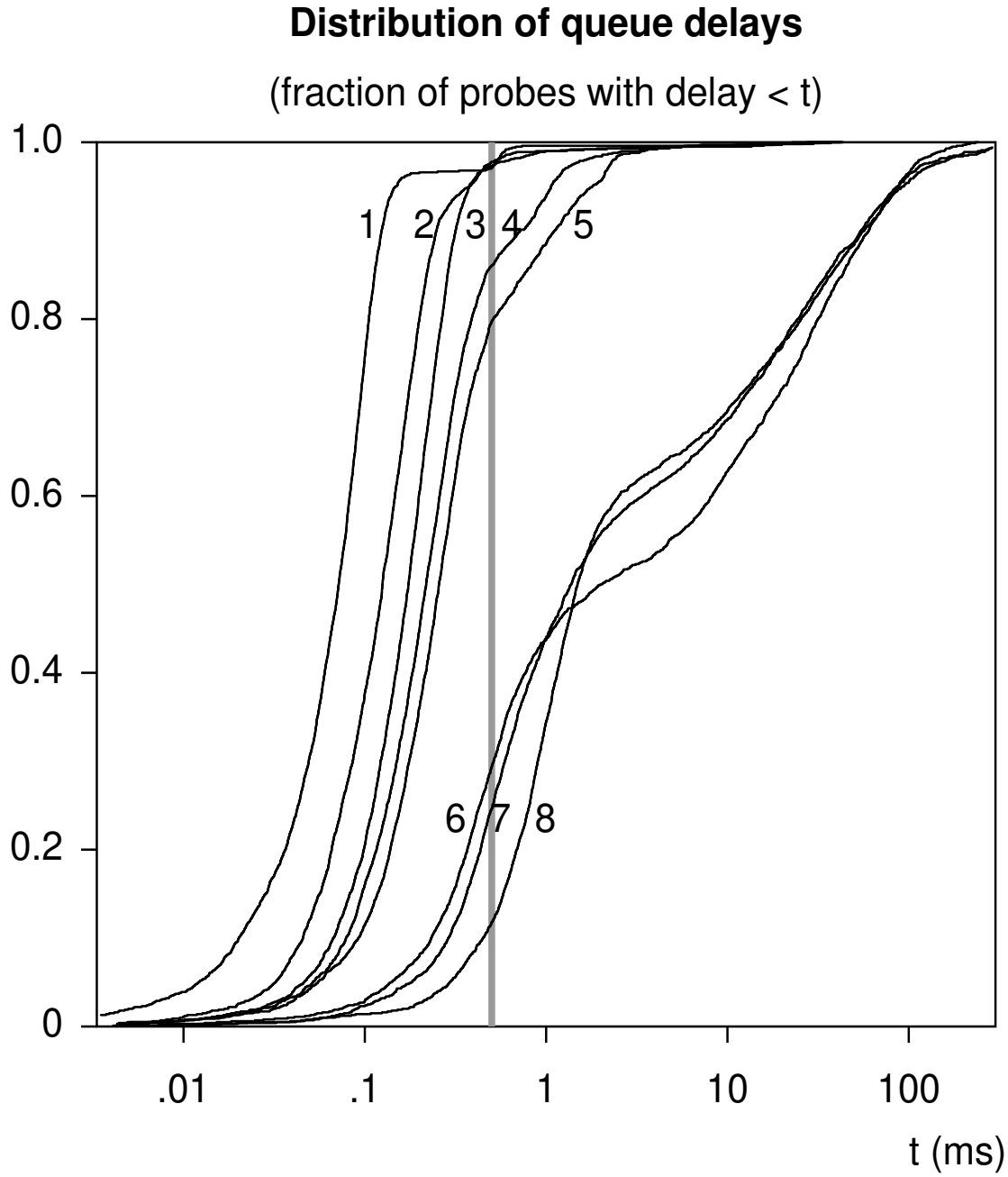
- Each line yields partial path parameters.
- By subtracting adjacent parameters, pathchar estimates the characteristics of individual links.

Intervals

- Accuracy varies from link to link.
- Need way to indicate reliability of estimate.
- Derive confidence intervals?
- Divide dataset in half, get four less accurate estimates.
- Assume: dispersion of estimates ~ error bounds.



Distribution of queue delays



- After minimum filtering, use the other measurements to estimate the distribution of queue delays.
- Each distribution is the convolution of the queue delays out to the *n*th node and back.

Modeling errors that affect latency

- *icmp_size* != 0
  - Latency estimates too high, but correctable.
- Δ*process_time* != 0
  - Intermittent delays, no problem.
  - ICMP generation time varies from router to router, bad.
- Fixed part of forwarding time added to latency.
- return path != outgoing path
  - Definitely breaks latency estimate.
  - Doesn’t hurt bandwidth estimate.
  - Hard to detect, but it’s out there [Paxson96].

Modeling errors that affect bandwidth

- Measurement errors and non-zero queue times:
  - For high *bw*, difference in *rtt* comparable to "noise."
  - Minimum filtering is not robust.
  - Differencing makes it worse.

- Parallel pipes
  - OC3 = 3 OC1; each packet goes down one of three.
  - DS3 = 28 DS1; each packet broken up and reassembled.

- Exceeding MTU breaks network model.
  - In theory, *rtt* curve levels.
  - In practice, some kind of non-linearity at MTU.
  - Either way, not good.

Route alternation

- ICMP packet has address of last router in partial path.
- Can detect alternation at second-to-last router.

- At first alternating link:
  - pathchar characterizes first alternative encountered.
  - clink keeps data for all alternatives, characterizes most frequent.

- For subsequent links:
  - pathchar and clink assume path prefix = characterized path.
  - How often do alternate paths have similar characteristics?

How well does pathchar work?

- Focus on bandwidth estimates.
  - Challenging
  - Useful
  - Verifiable
- Latency estimates.
  - Easier to do
  - Harder to test
- Dynamic behavior is hard.
  - Queue delays, drop rate.
  - On what time scale does past predict future?

Datasets

Colby to SDSC

Maine to San Diego

- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- T1
- OC-3
- DS3
- alternating next hop
- 6 more hops unknown

45 sizes, 64 probes

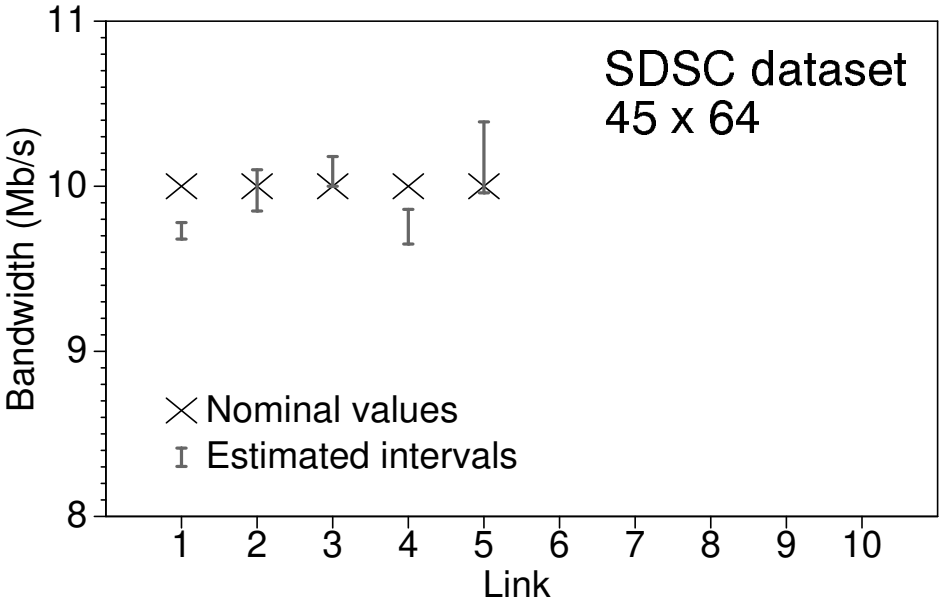
Colby to MINT

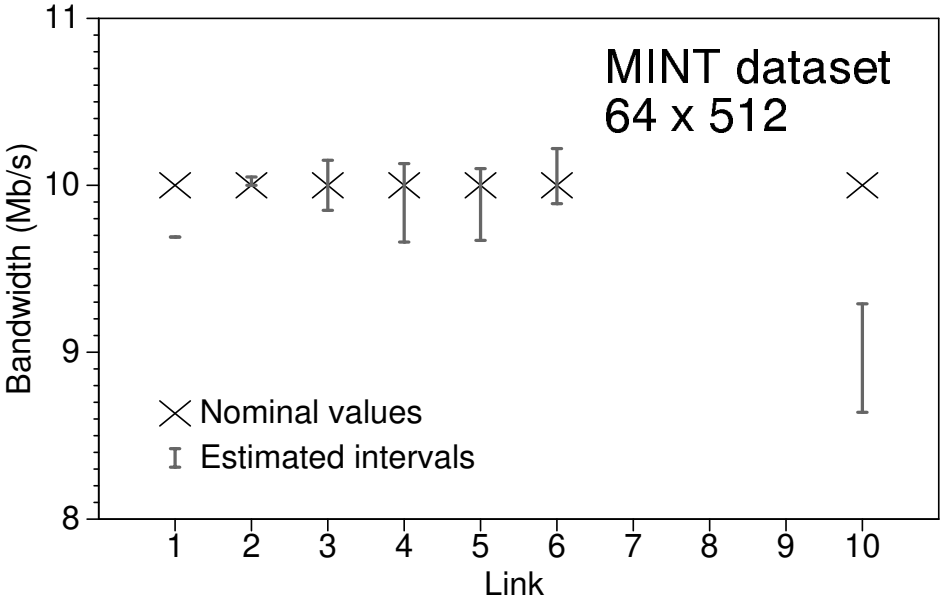
Maine to Maine (via Boston)

- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- 10 Mb/s Ethernet
- T1
- OC-3
- T1
- 10 Mb/s Ethernet

64 sizes, 512 probes

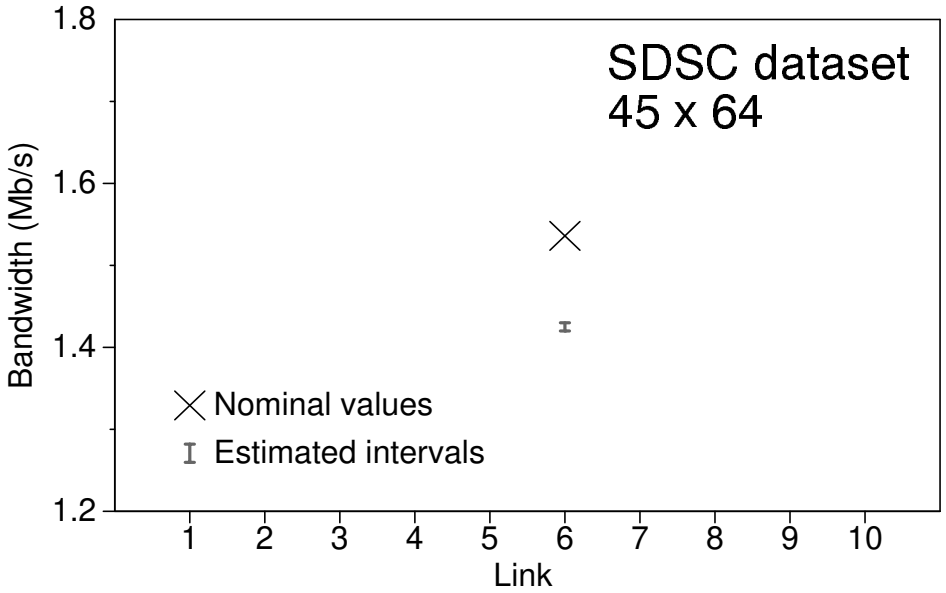
Ethernet results

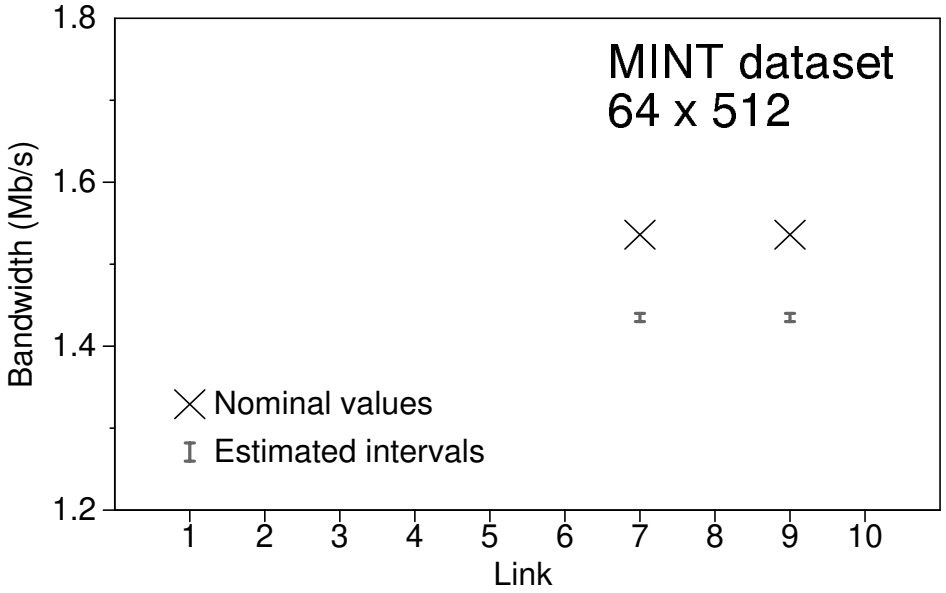




- Many intervals contain the nominal value.
- Intervals get wider as we go along.
- Extra probes help, with diminishing returns.
- First and last Ethernet link consistently underestimated or underperforming?

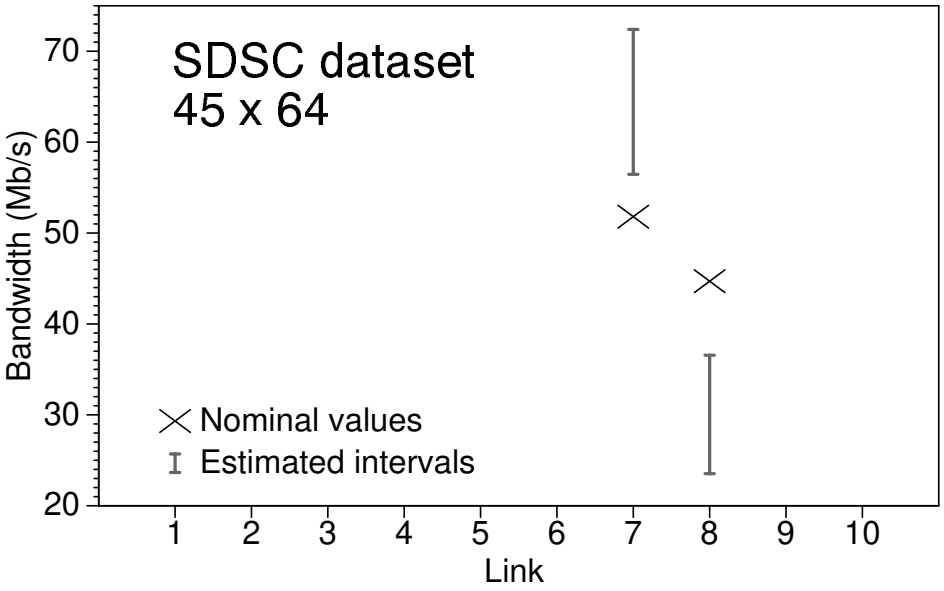
T1 results

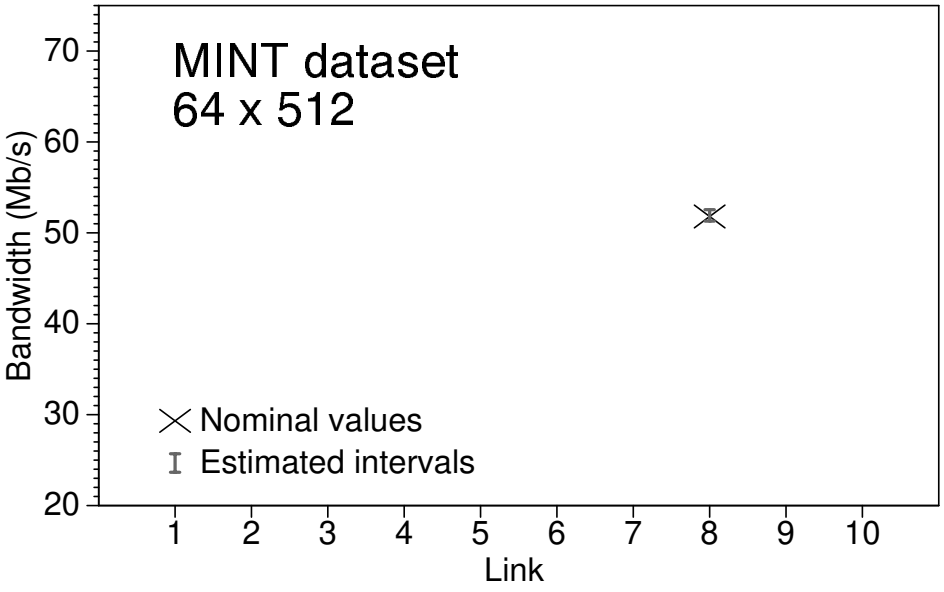




- Lower bandwidth link: consistent estimates, narrow intervals.
- 1.42 Mb/s != 1.536 Mb/s

OC-3 and DS3 results





- Parallel pipe problem:
  - DS3 = 44.7 Mb/s
  - OC-3 = 3 x 51.8 Mb/s
- As expected, fast links are less accurate.
- Intervals indicate reliability; still optimistic.
- Large dataset yields pretty good estimate. (repeatably?)

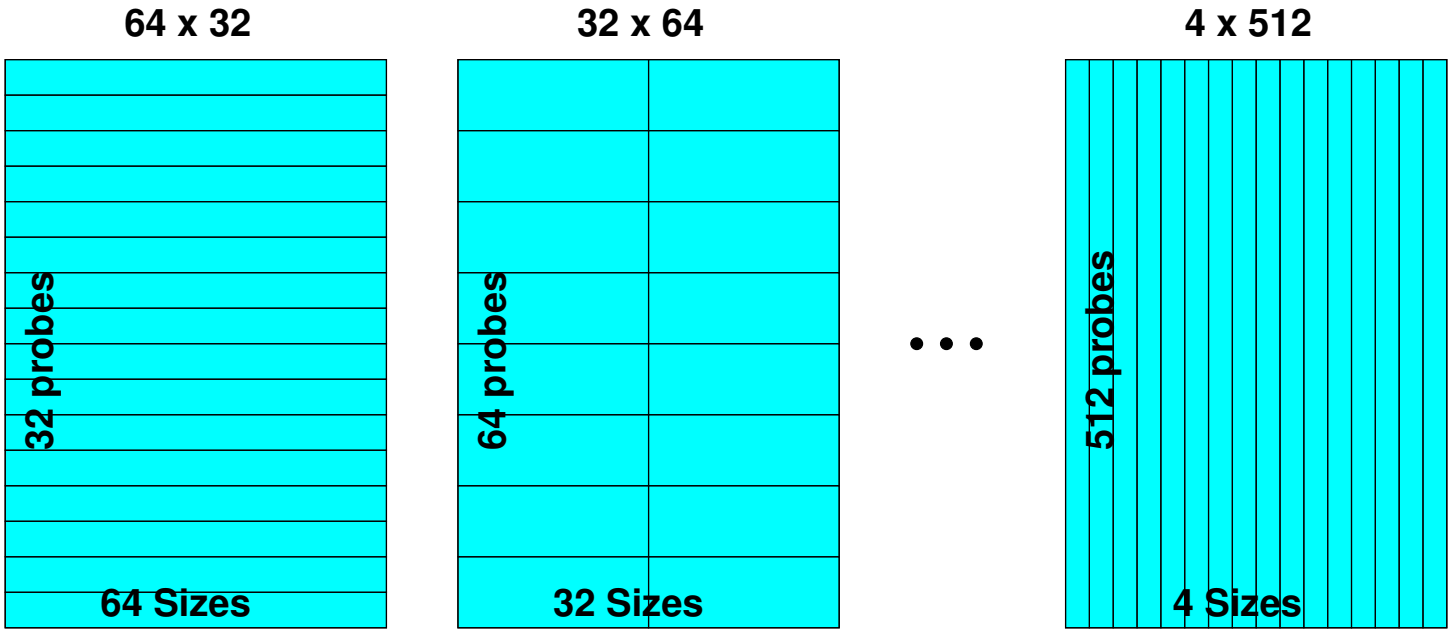
Room for improvement

- Robustness:
  - Minimum filtering is vulnerable to errors.
  - Least squares is not robust.
  - Subtraction amplifies errors.
- Accuracy: more sizes or more probes per size?
- Efficiency: some links take more data than others.
  - Use intervals to direct data collection.
  - More data for "hard" links.

Robustness

- Minimum filtering
  - 2nd percentile?
  - Modeling of distribution of queue delays?
- Least-squares fit
  - Iteratively-weighted least squares?
  - Median of pairwise fits?
- Intervals
  - Median of four estimates?

Accuracy: Sizes vs. Samples



- More sizes means more data after minimum filtering.
- More samples means better chance of zero queue time.
- Experiment:
  - Collect 64 sizes, 512 samples.
  - Divide into 16 subsamples, in various configurations.

Sizes vs. Samples

- Each configuration uses the same data.
- Each configuration yields 16 independent estimates.
- Calculate mean relative error of the 16 estimates.
- Results:
  - Most accurate with 32 x 64 or 64 x 32 (SIZES WIN).
  - Similar results with SDSC dataset.
- Need to repeat with wider dataset.
- clink default is 93 sizes.

Efficiency: adaptive data collection

- Start with small number of probes at each size (2).
- Use interval as convergence criterion:

$$\frac{\text{interval width}}{\text{median estimate}} < 0.1$$

- Collect additional data until estimates converge.
- Experiments simulate adaptive collection.
- Results:
  - Nearby, slow links require ~100 probes.
  - Distant, fast links require ~10,000 probes.
  - Adaptive collection can speed up collection by 90%.

Ongoing work

- Additional paths:
  - 93 x 4 dataset nails 100 Mb/s Ethernet (2% convergence).
  - Need more paths with known links!
- Implement and test adaptive data collection.
- Reconsider robustness techniques.
- Kernel-level timestamps.
- **`netchar`**

Kernel-level timestamps

- Hacked version of Red Hat Linux.
  - Outgoing UDP packets with "Play that funky music..."
  - Incoming ICMP packets (clobber enclosed IP header).

- How much variance in *rtt* is due to local scheduling?

- Answer: none.

netchar

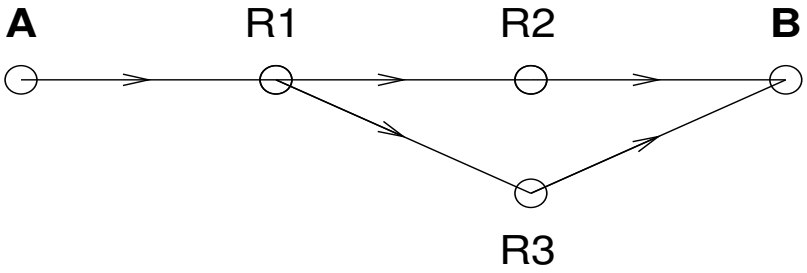
- pathchar running periodically on n hosts.
- Characterize paths to n-1 other hosts.

- Helps with alternating link problems.
  - At least the low frequency ones (period > 10 seconds).

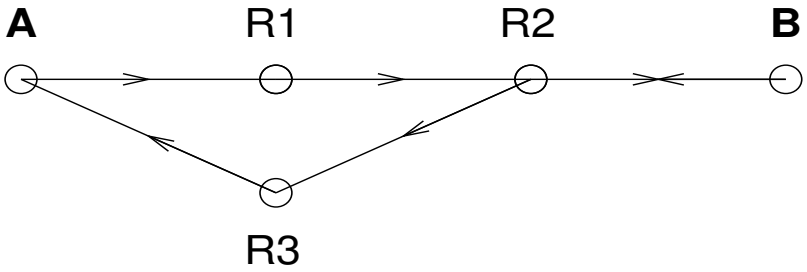
- Helps a lot with evaluating estimates:
  - Multiple views of the same link.

- Possible insight into path asymmetry.
  - Fix latency estimates.
  - Combine first half of outgoing path with second half of incoming.

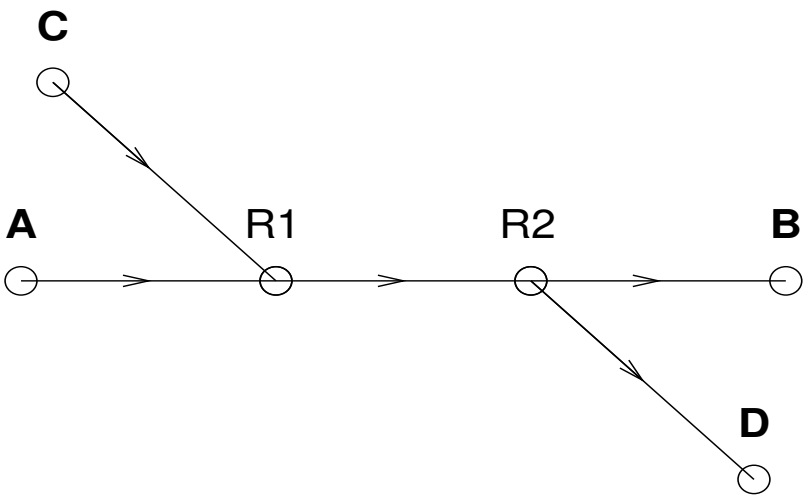
Why netchar helps



- Alternate path from A to B
- We can characterize both.



- We can detect asymmetric paths.
- And correct latency?



- Some links will appear on multiple paths.
- A low probability link on one path might be high probability on another.

Summary

- pathchar yields useful information.
  - Need a network expert to interpret it.
  - Could be more robust, maybe.
- Some problems are probably insurmountable:
  - Routers that don’t decrement ttl.
  - Parallel pipes.
  - Links that defy latency-bandwidth model (ATM).
- Some problems netchar can address:
  - Alternate paths.
  - Asymmetric paths.