Modeling TCP Performance

Allen B. Downey

Boston University

TCP Performance

Goal: model and predict transfer times.

- Interactive applications.
    - Predict duration of downloads.
    - Select mirror site.
- Distributed applications.
    - Resource selection.
    - Scheduling.

Goals

Complementary questions:

- What do we need to know about a network path?
- What can we measure about a network path?

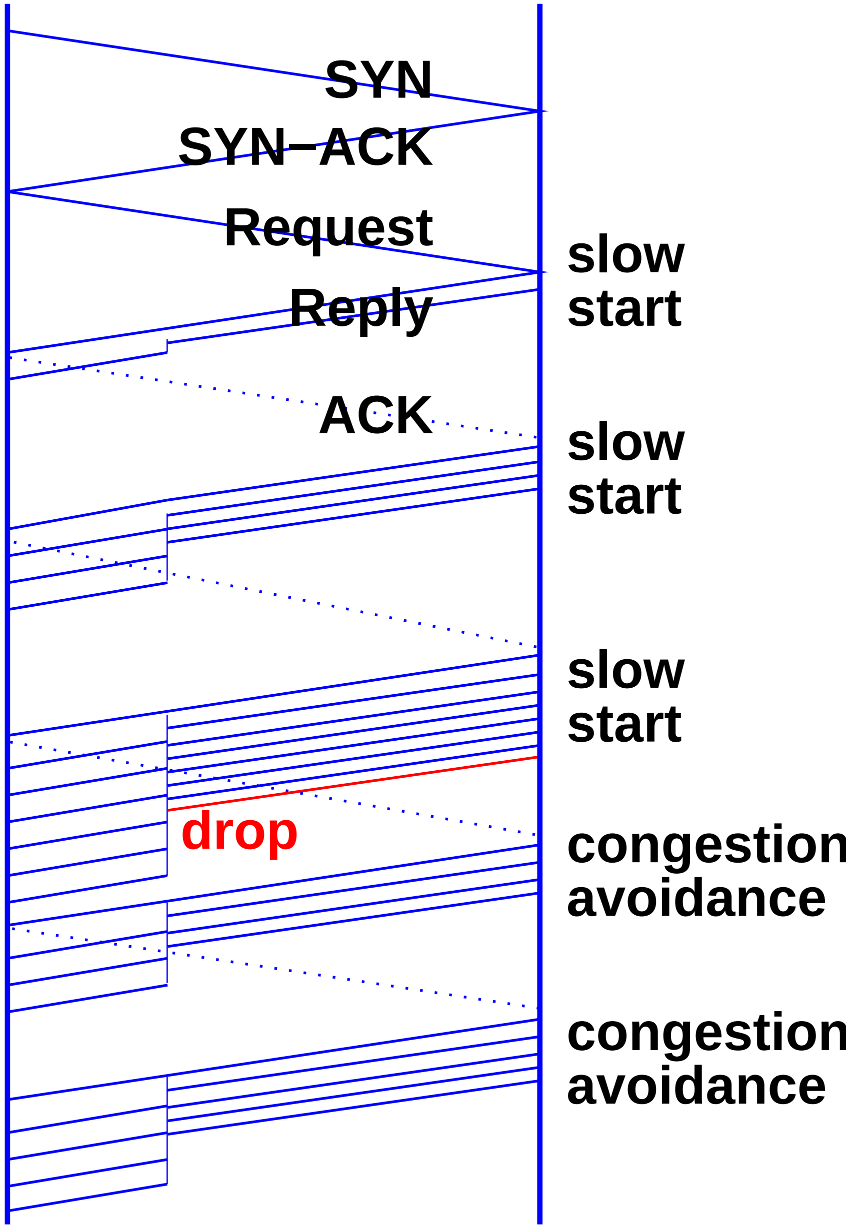
Complementary goals:

- Maximize accuracy.
- Minimize measurement.

TCP Basics

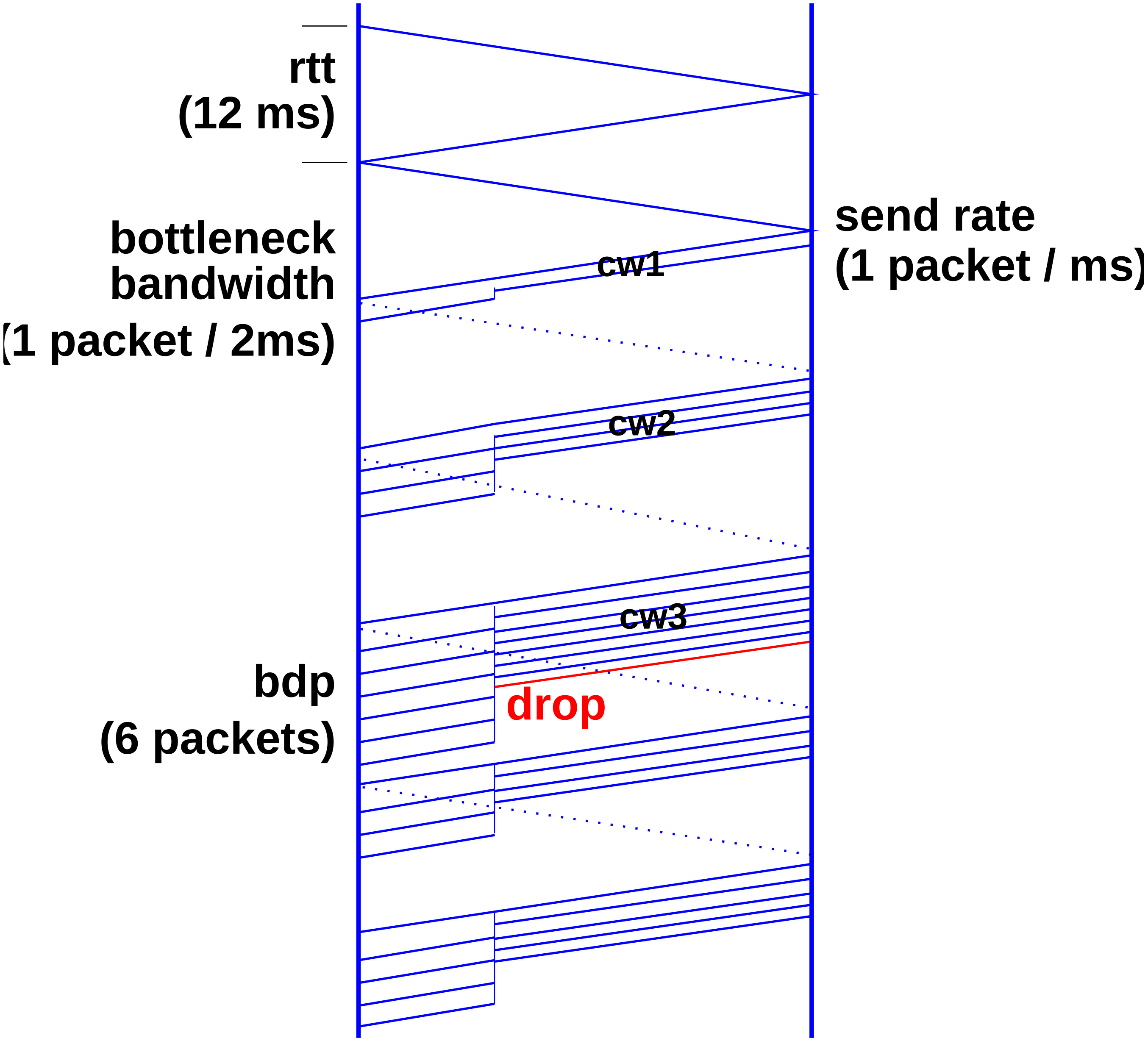
- Most network transfers managed by TCP.
- Reliable delivery by ACK-retransmit.
- Window-based: sender limits data “in flight”.
- Flow control: receiver advertises available buffer space.
- Congestion control:
  - Slow start to discover capacity.
  - Congestion avoidance for stability.

TCP transfer



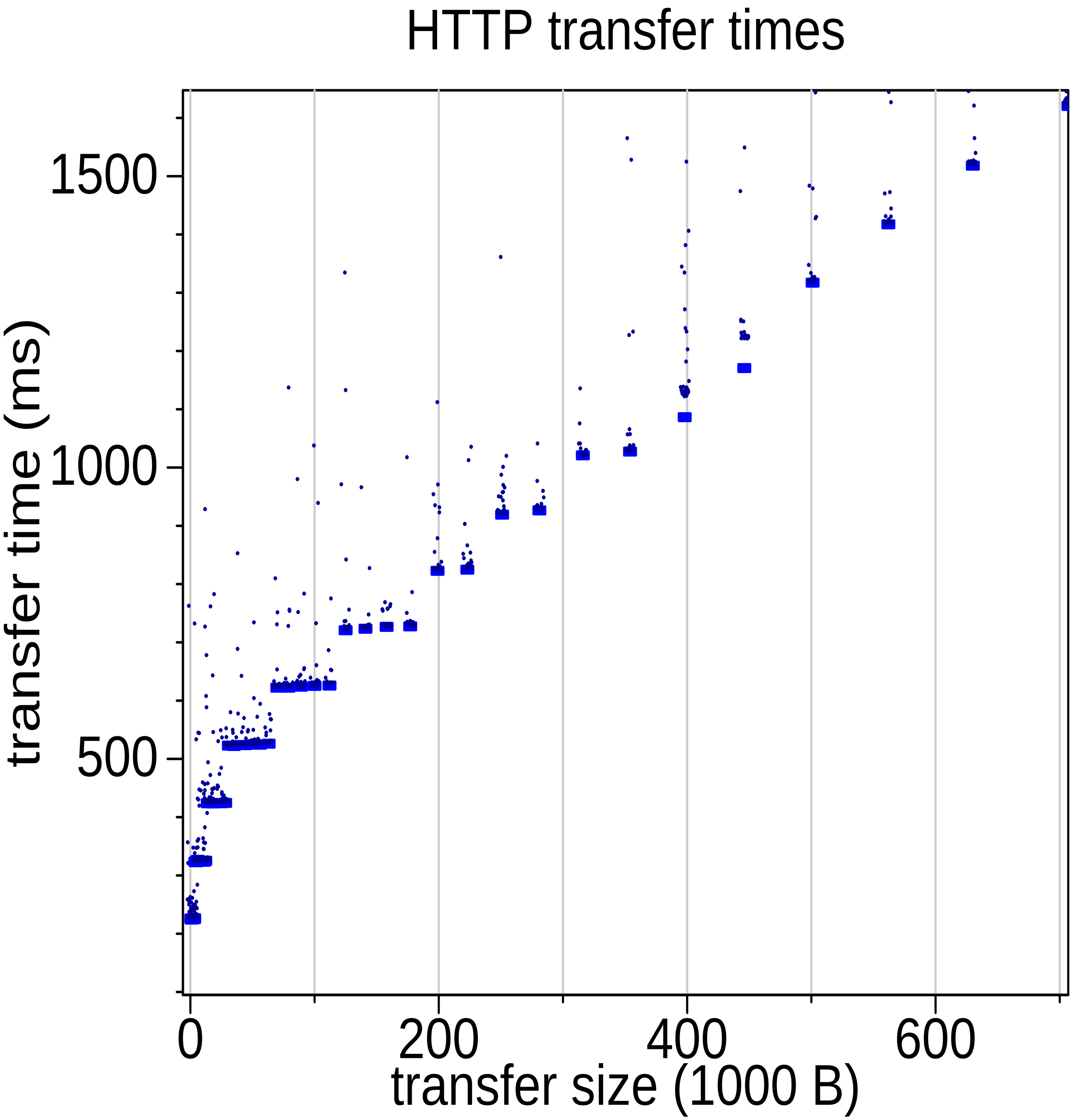
- Packet-level view of an HTTP request.

TCP transfer



- Send rate often faster than bw.
- Packets queue, maybe drop, at bottleneck.

Basic performance model



- Short transfers depend on rtt and cw.
- Long transfers depend on duration of slow start and throughput.

Path parameters

So what do we need to know?

- Distribution of rtt.
- cw_1, cw_2, cw_3 ...
- Bottleneck bandwidth (and bdp).
- Effective throughput (distribution?)

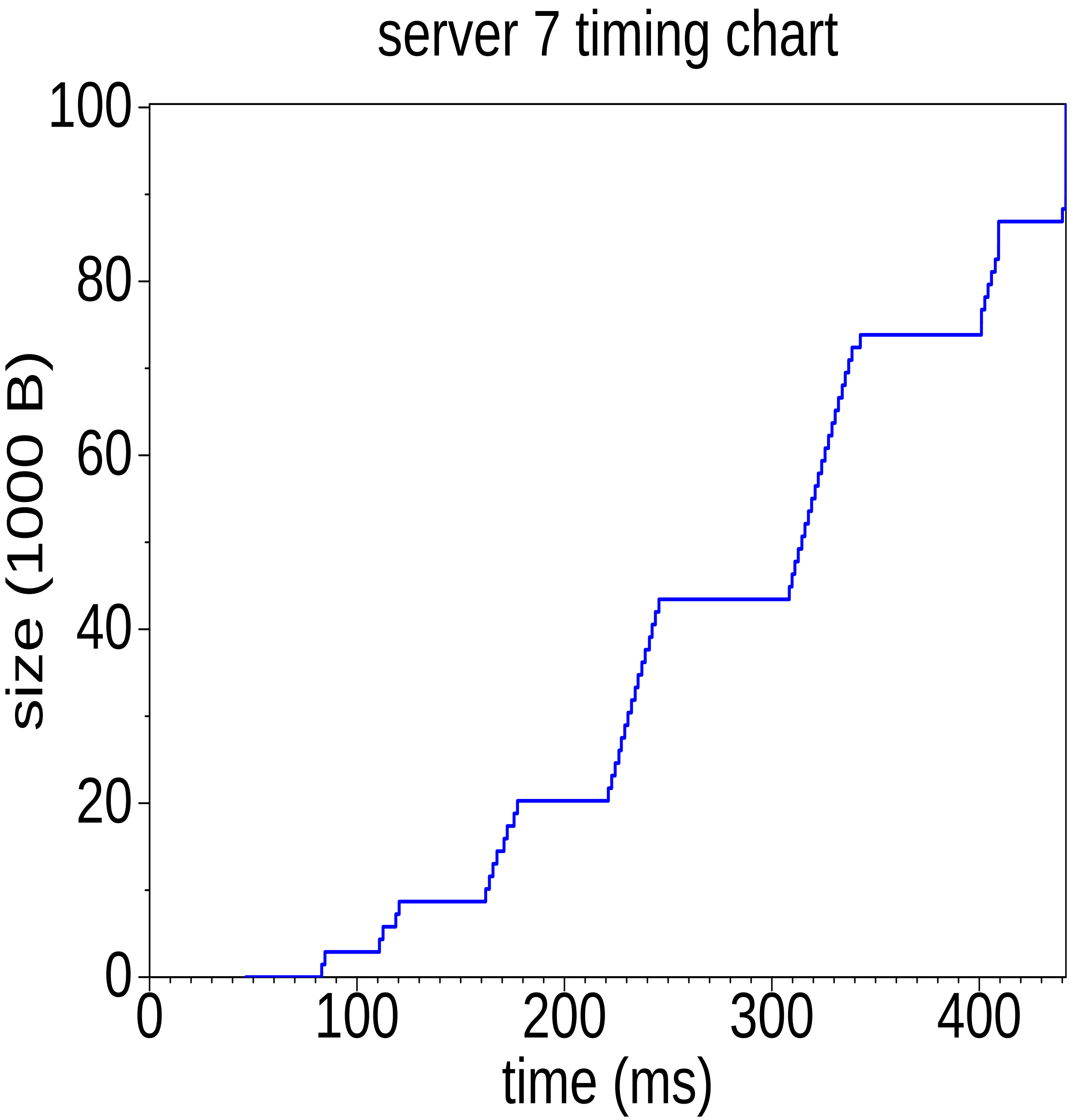
Can we measure these parameters?

Measurement

- Application-level HTTP timing (instrumented `wget`).

```
set the timer
connect (socket)
record elapsed time
write (request)
while (more data) {
    select (socket)
    record elapsed time
    read (buffer)
    record amount of data
}
```

Timing chart

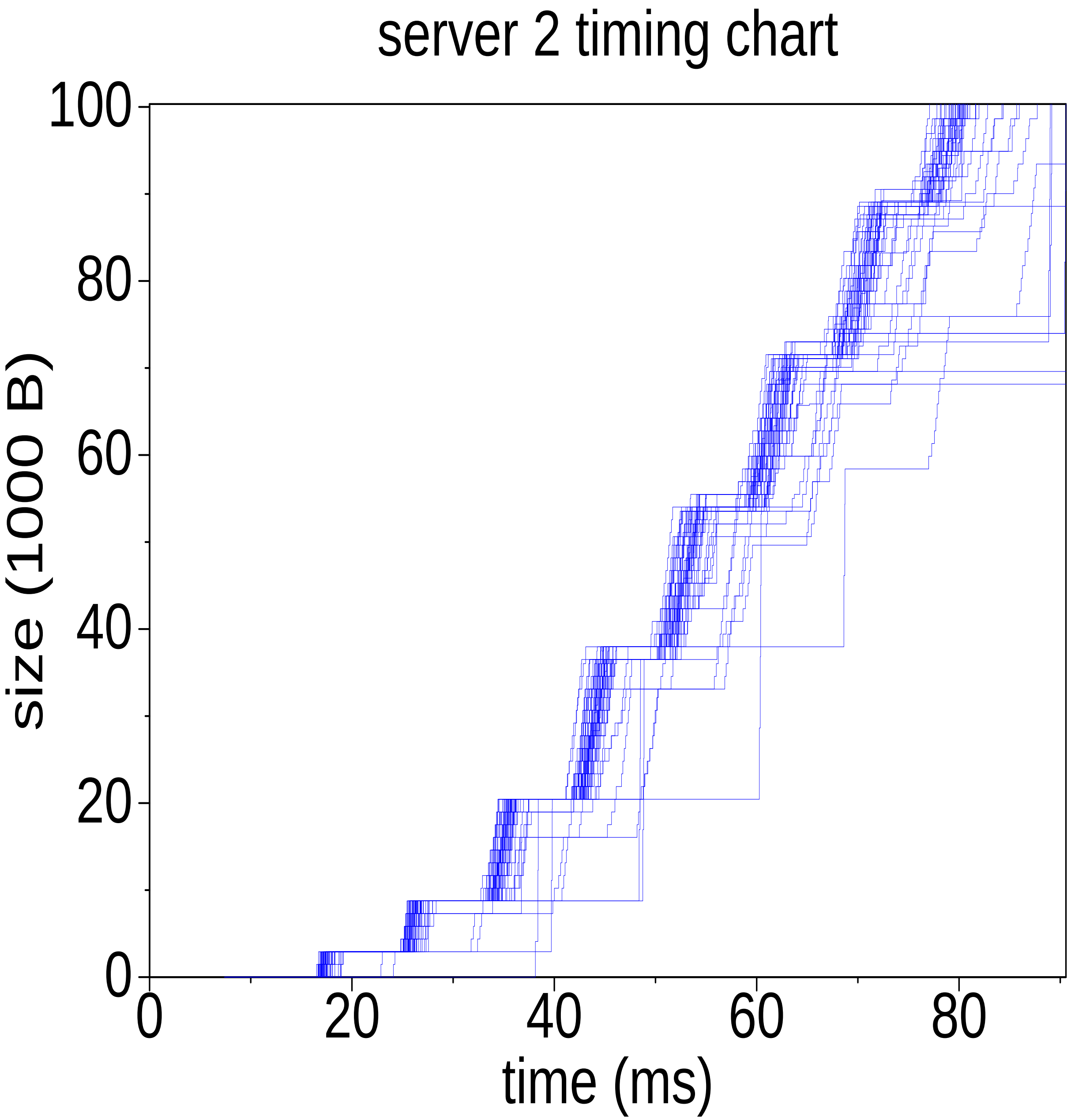


- Plot bytes read vs. time.
- Immediately, we can estimate rtt, cw, bw and bdp!

Measurement

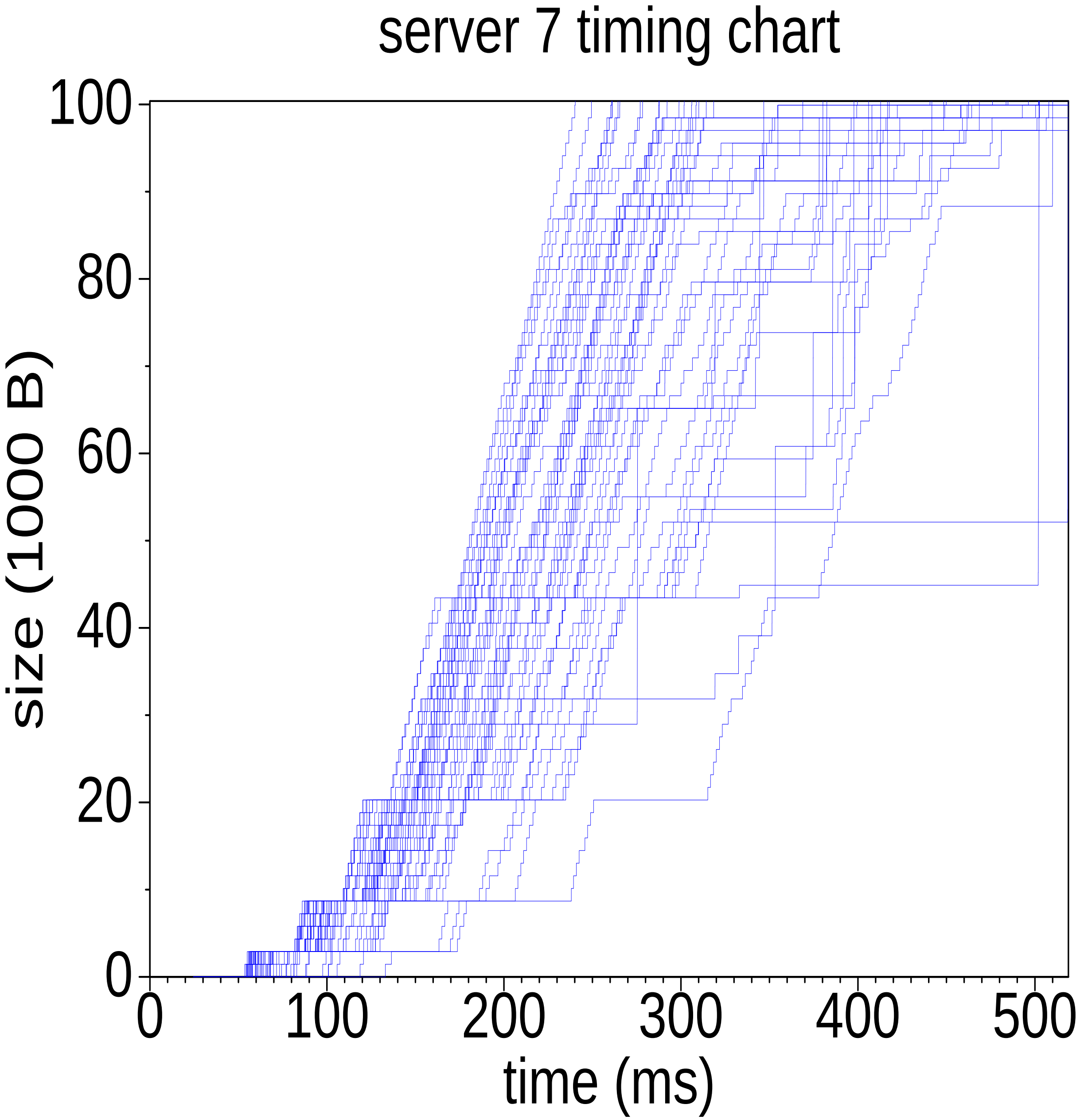
- Measurements:
  - 100,000 byte transfers.
  - 100 transfers, with 100s between.
- HTTP downloads:
  - 2 URLs provided by collaborators.
  - 11 URLs culled from proxy cache logs.
- Diverse network paths:
  - rtt from 7 to 270 ms.
  - bw from 0.350 to 100 Mbps.

Timing chart



- $rtt \approx 7$ ms, with some variability.
- $cw = 2, 4, 8, 12, 12...$
- Some dropped packets.

Timing chart



- $rtt \approx 25$ ms.
- characteristic slope ≈ 7 Mbps.
- $bdp \approx 11$ packets.
- $cw = 2, 4, 8, \infty$
- How can cw exceed bdp?

Endogenous drops

Conventional wisdom: if $cw > bdp$, TCP induces endogenous drops:

- Brakmo and Peterson, 1995: “[TCP] *needs* to create losses to find the available bandwidth...”
  “... if the threshold window is set too large, the congestion window will grow until the available bandwidth is exceeded, resulting in losses...”
- Hoe, 1996: “... the sender usually ends up outputting too many packets too quickly and thus losing multiple packets in the same window.”

Endogenous drops

- Allman and Paxson, 1999: “For TCP, this estimate is currently made by exponentially increasing the sending rate until experiencing packet loss.”
- Barakat and Altman, 2000: “Due to the fast window increase, [slow start] overloads the network and causes many losses.”

Fortunately, this is not true.

Self-clocking

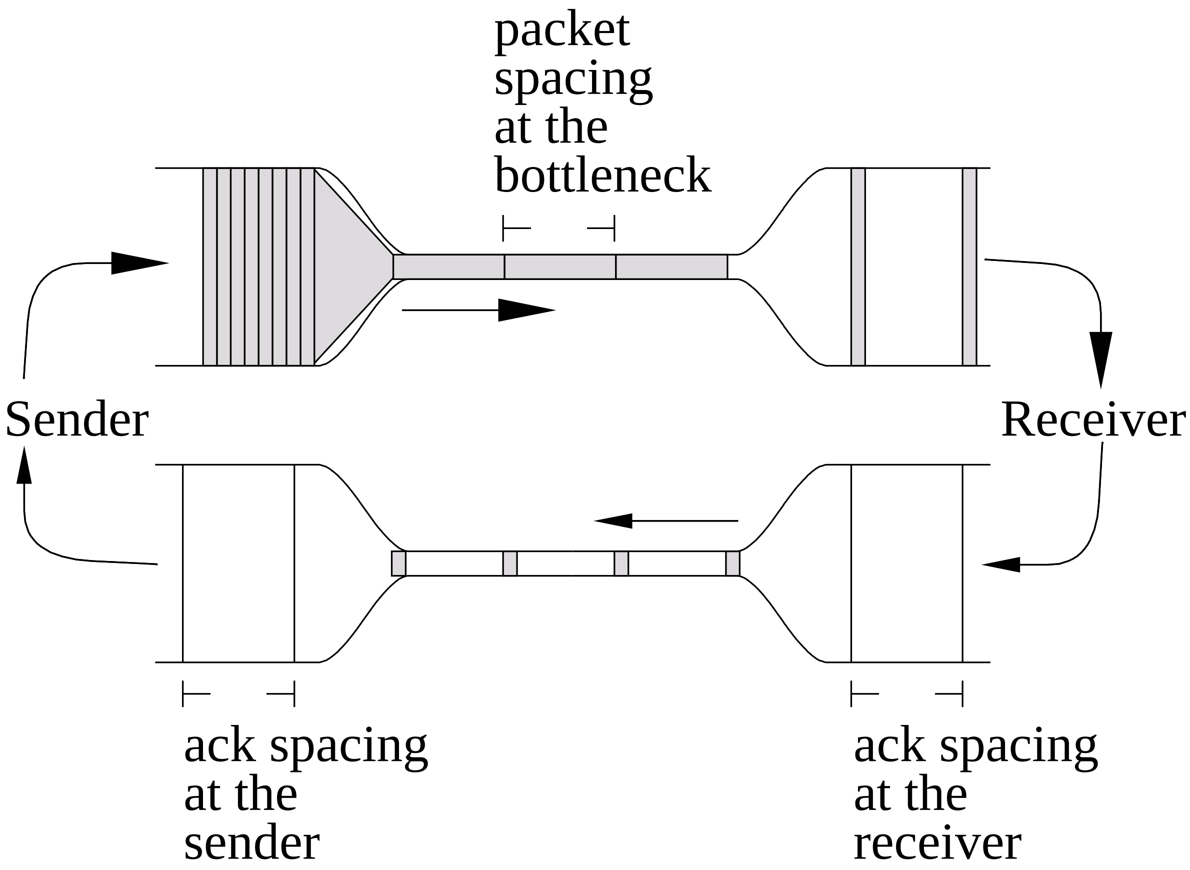
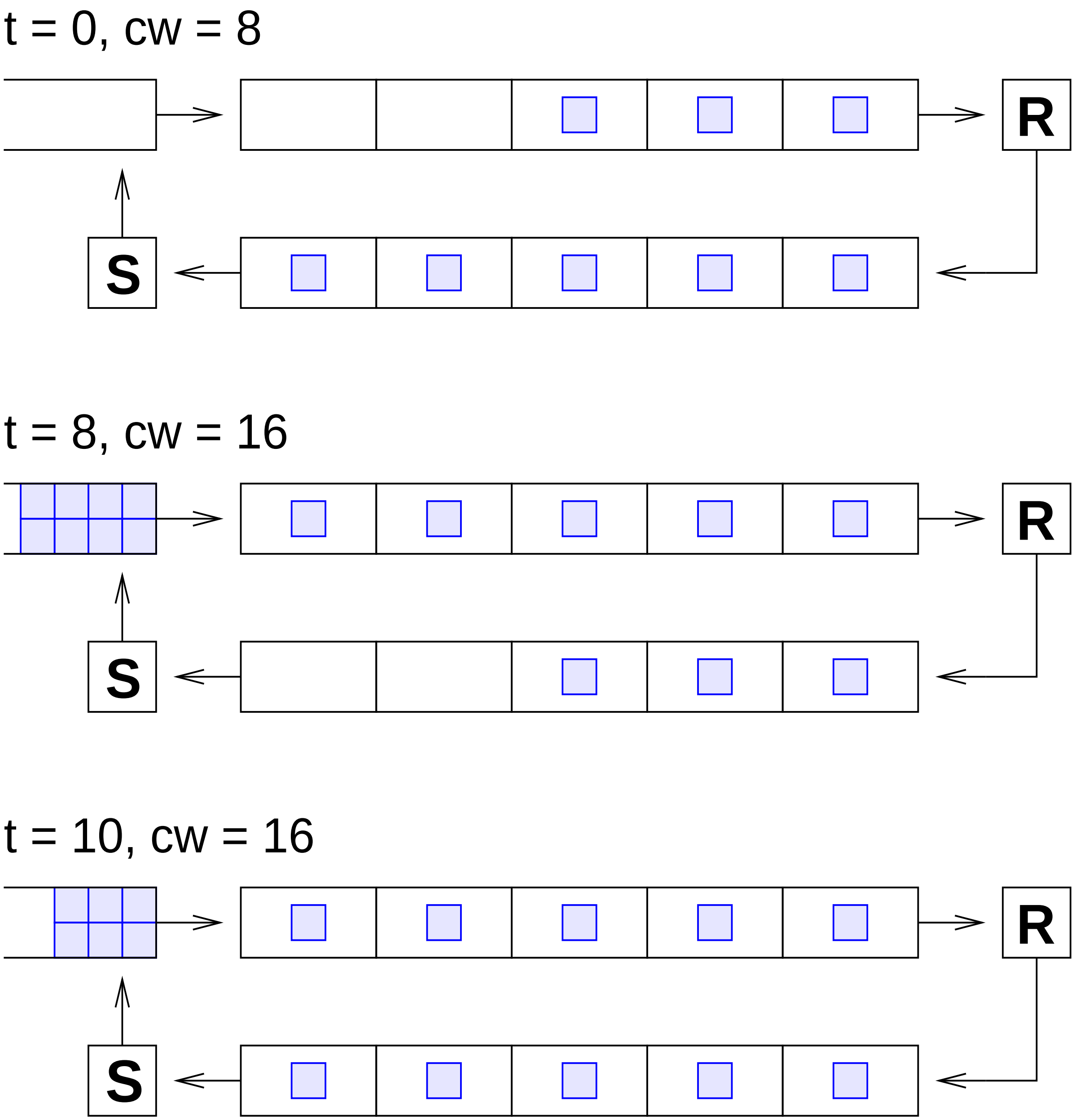


Figure 1 from Jacobson,
"Congestion Avoidance and Control," 1988.

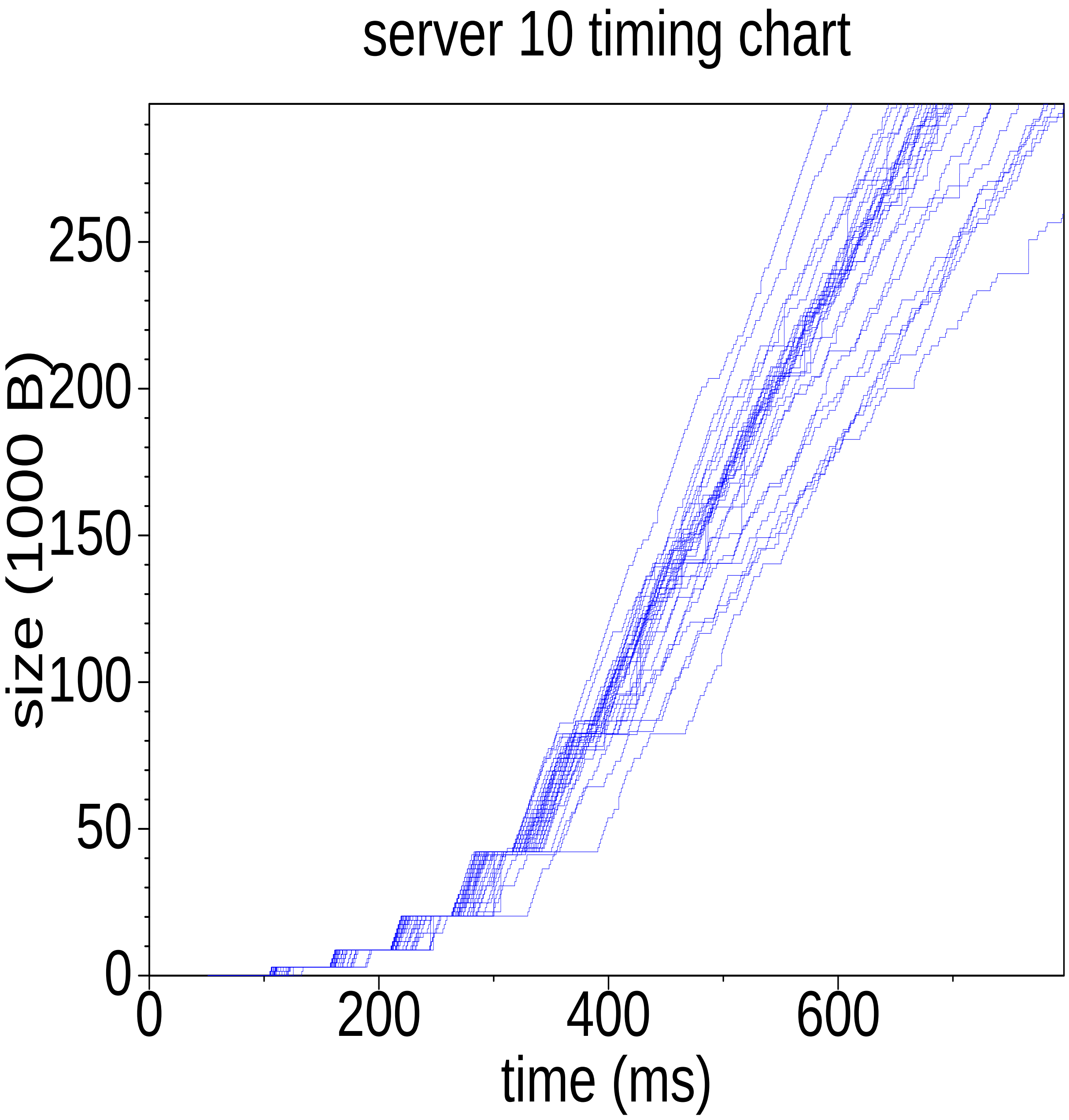
- bottleneck bw $\Rightarrow$ receive rate $\Rightarrow$ ACK rate $\Rightarrow$ send rate
- Endogenous drops not inevitable.
- If no exogenous drops, cw grows arbitrarily.

Conditions for self-clocking



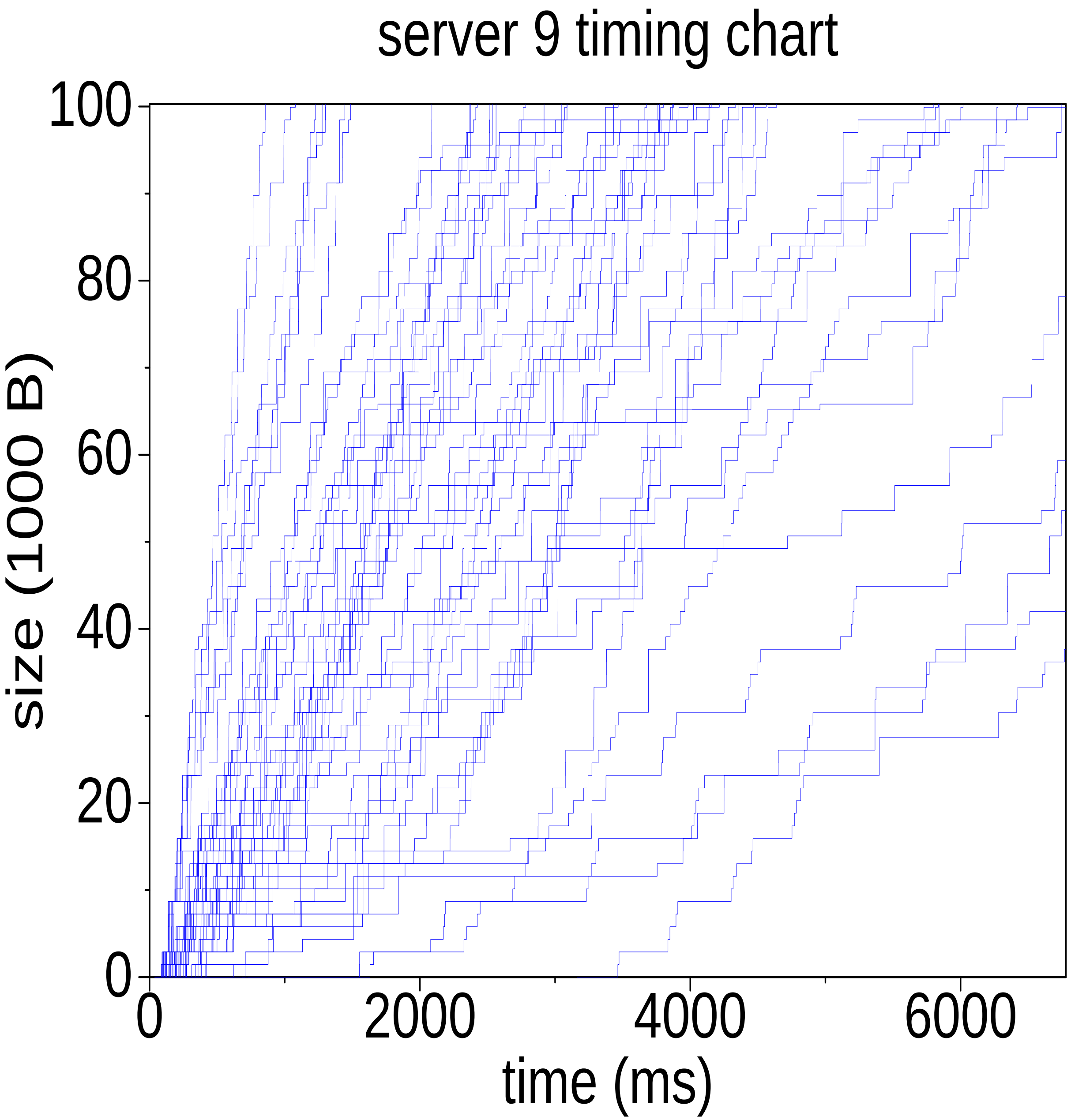
- cw^* is the last window smaller than bdp.
- During transition, packets accumulate in queue.
- Need queue capacity $ssthresh - bdp$ or $bdp - cw^*$.

Self-clocking



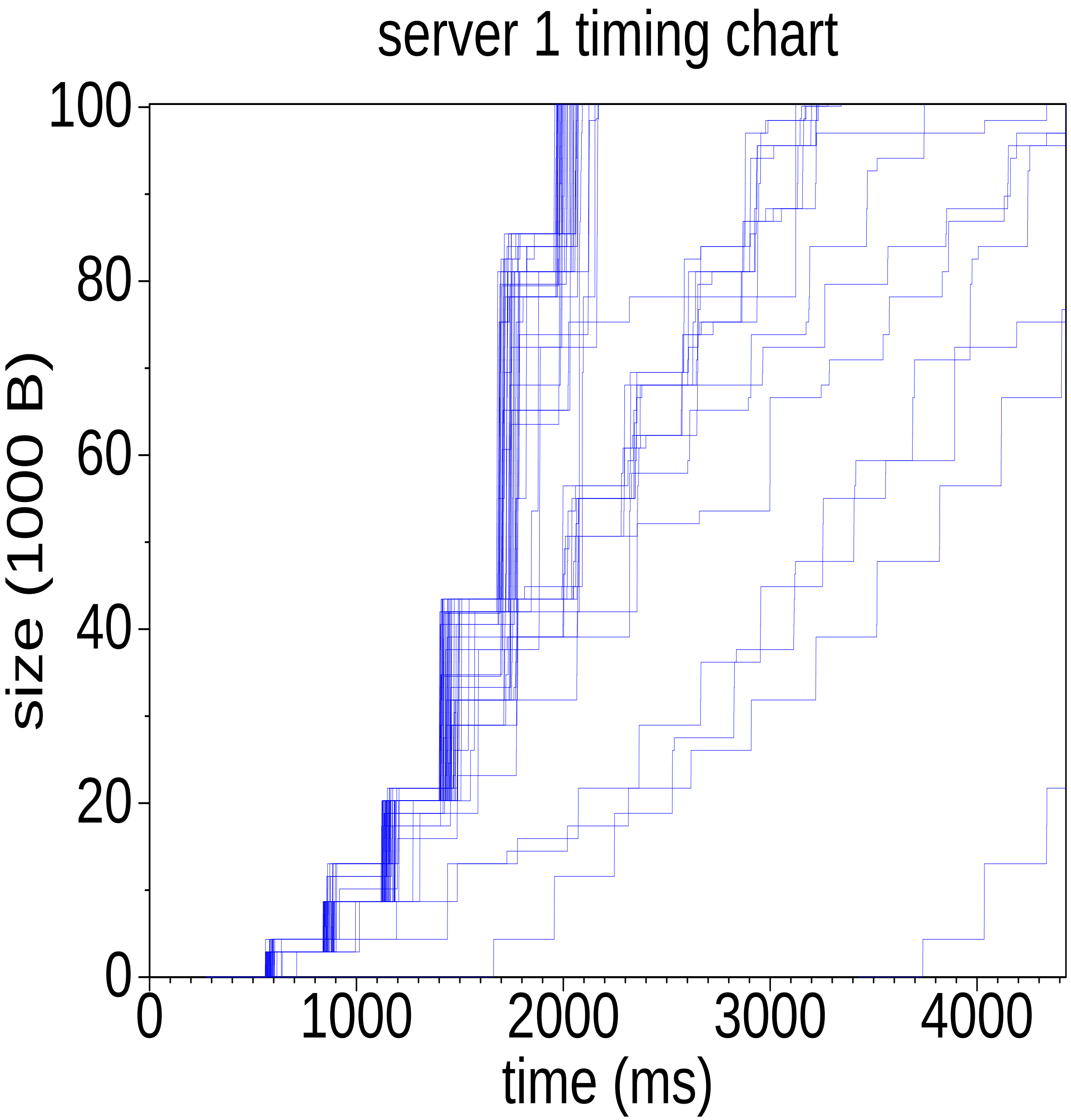
- 10 of 13 either buffer-limited or end in slow start.
- Other 3 show self-clocking.
- Here, $bdp \approx 41$ packets, $cw > 100$.

Self-clocking



- Some self-clocking, some congestion avoidance.
- Large variability in steady-state throughput.

The future?



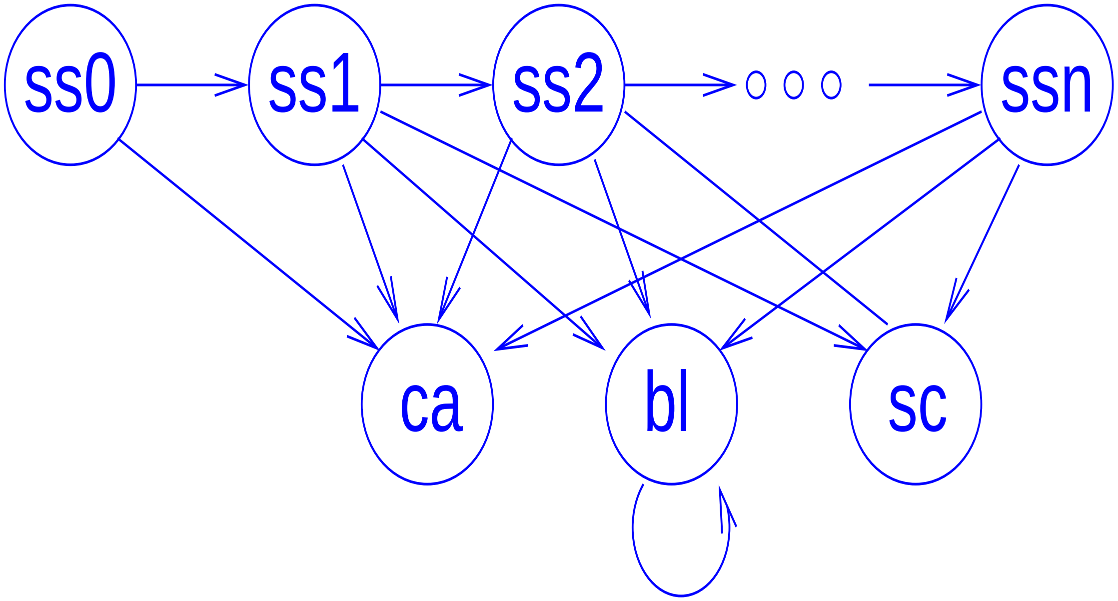
- High bw, high rtt, $bdp = 620$ packets.
- Most transfers never leave slow start.
- The ones that do never catch up.
- Variability in rtt $<$ variability due to congestion avoidance.

Steady-state behavior

So what do we need to know?

- Transition from slow start:
  - Exogenous drop rate, p.
  - Endogenous drop rate $= f(cw)$
  - Slow start theshhold, $ssthresh$.
  - Buffer size at sender.
- Three kinds of steady state:
  - Congestion avoidance, buffer-limited and self-clocking.

State transition model



- Drop rates, $ssthresh$, and buffer size implicit as state transition probabilities.
- How to estimate probabilities?

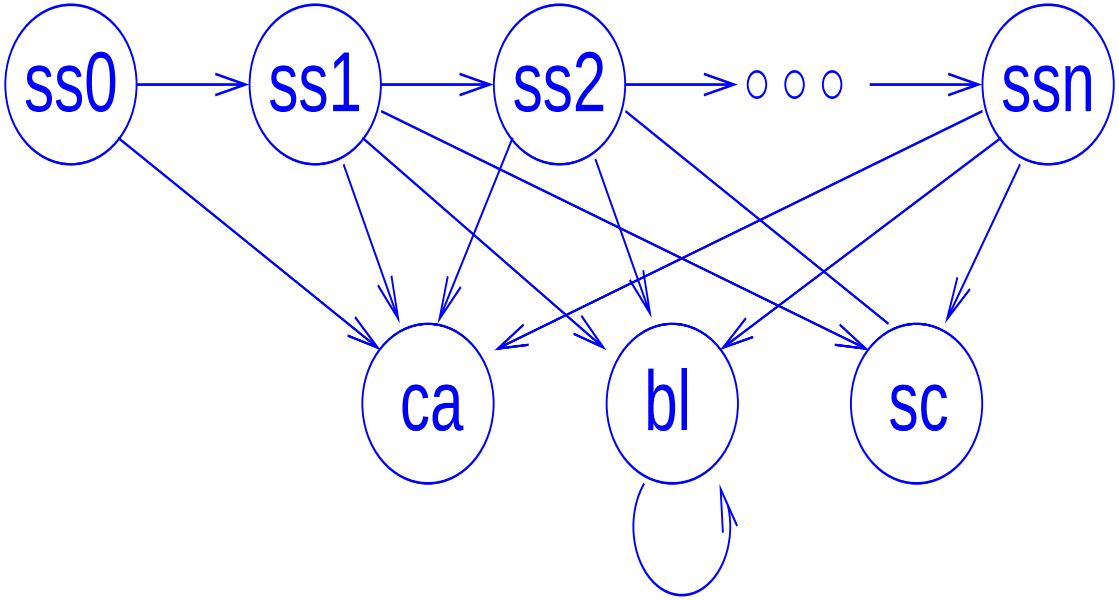
Estimating parameters

Just do statistically what we’ve been doing visually.

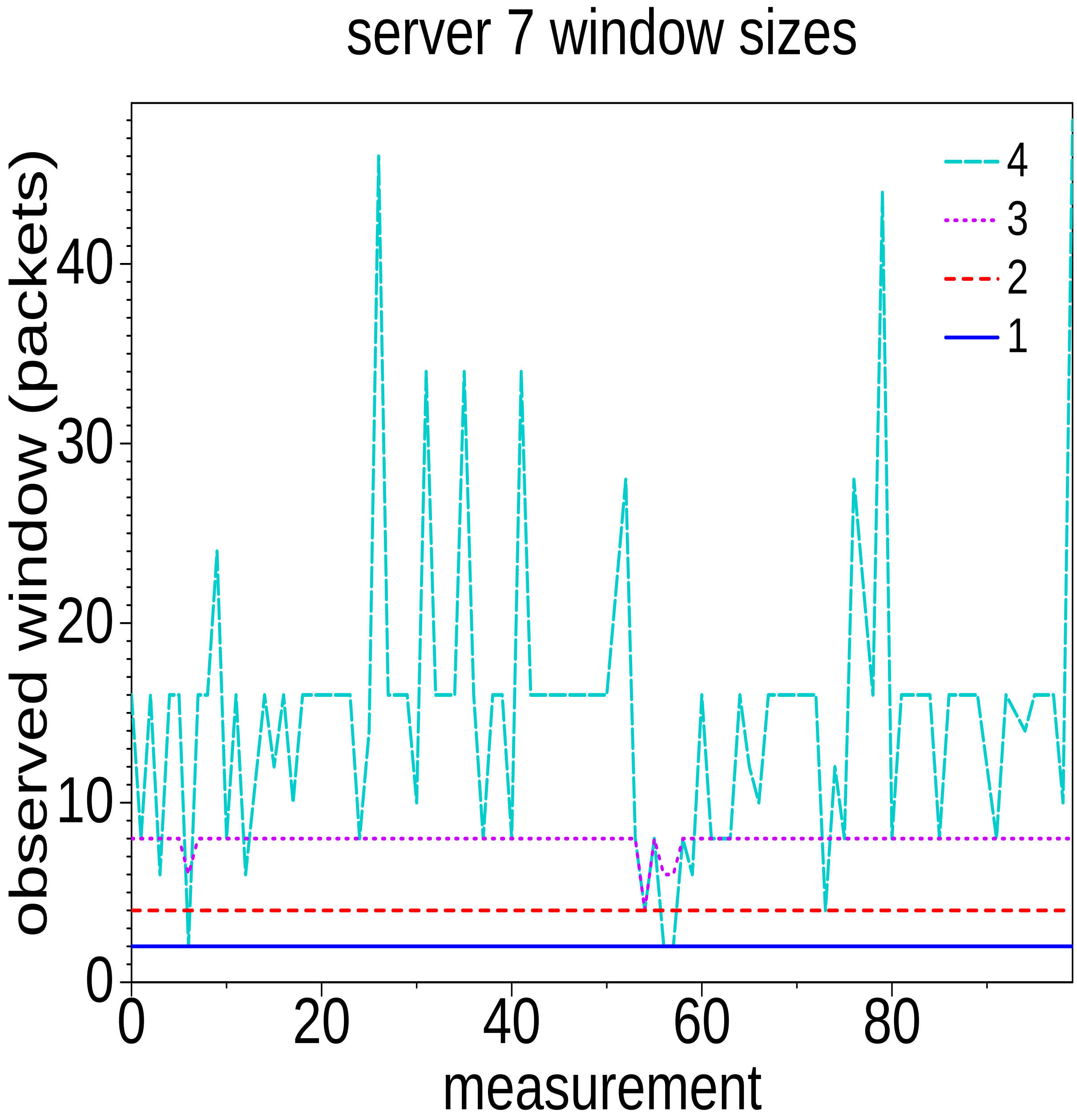
- Divide timing chart into rounds.
- Measure window size for each round.
- Pattern match on window sizes:
  - 2, 4, 8, 16, 32 ...
  - 2, 4, 6, 4, 5, 6 ...
  - 2, 4, 6, (long pause) 11, 6 ...
  - 3, 6, 12, 15, 15, 15 ...
  - 3, 6, 12, 51, 17, 63 ...

Estimating Parameters

- Distribution of rtt.
- Window sizes for ss_i and bl.
- Distribution of throughputs for ca and sc.
- State transition probabilities.

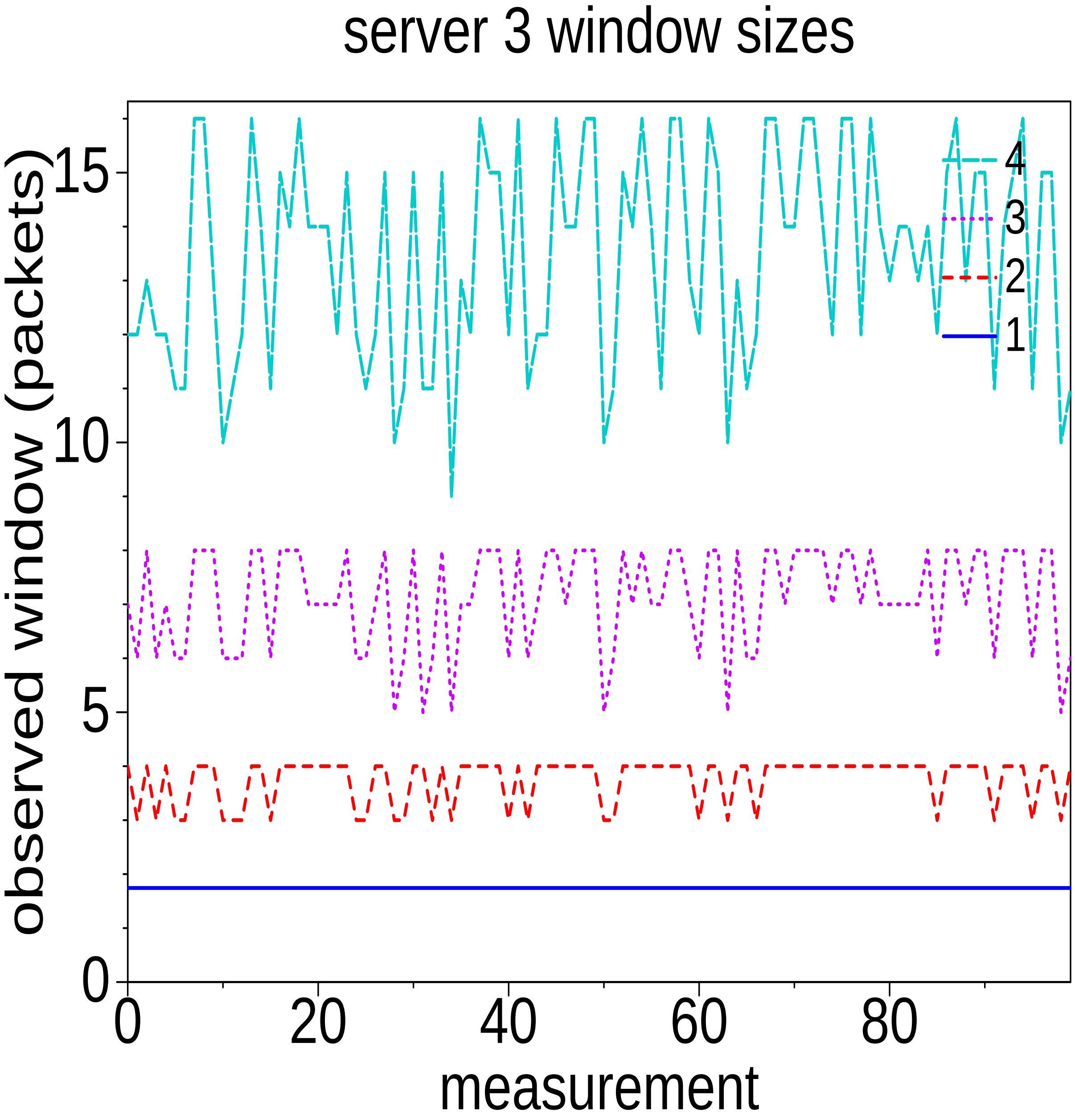


Window sizes



- This is the sort of thing we expect.
- Too bad it’s the exception.

Window sizes



- cw_2 is sometimes 3, sometimes 4.
- cw_{n+1} is $2 \cdot cw_n - m$, where m is $0, 1, 2, \ldots$
- 10 out of 13 are similar.

Non-deterministic slow start

- Sender increases cw by one packet each new ACK.
- Receiver usually sends one ACK per two packets.

$$cw_{n+1} = 1.5 \cdot cw_n$$

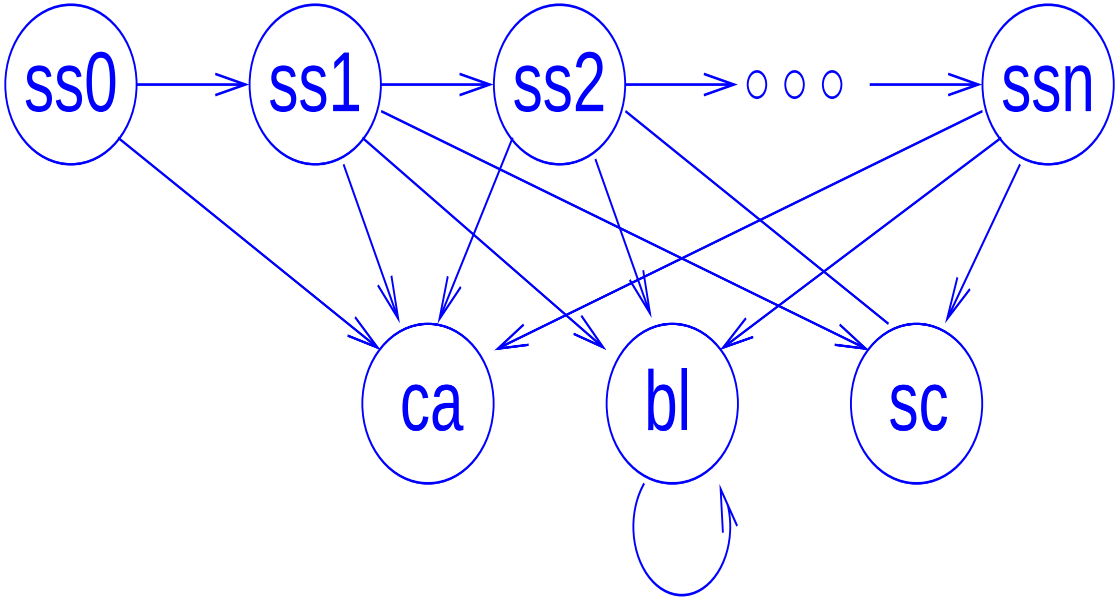
- So, receivers have heuristics to ACK every packet during slow start.

$$1.5 \cdot cw_n \leq cw_{n+1} \leq 2.0 \cdot cw_n$$

- Timer bounds ACK delay. Introduces nondeterminism?
- Implementation dependent. (Linux kernel version 2.4.18-3)

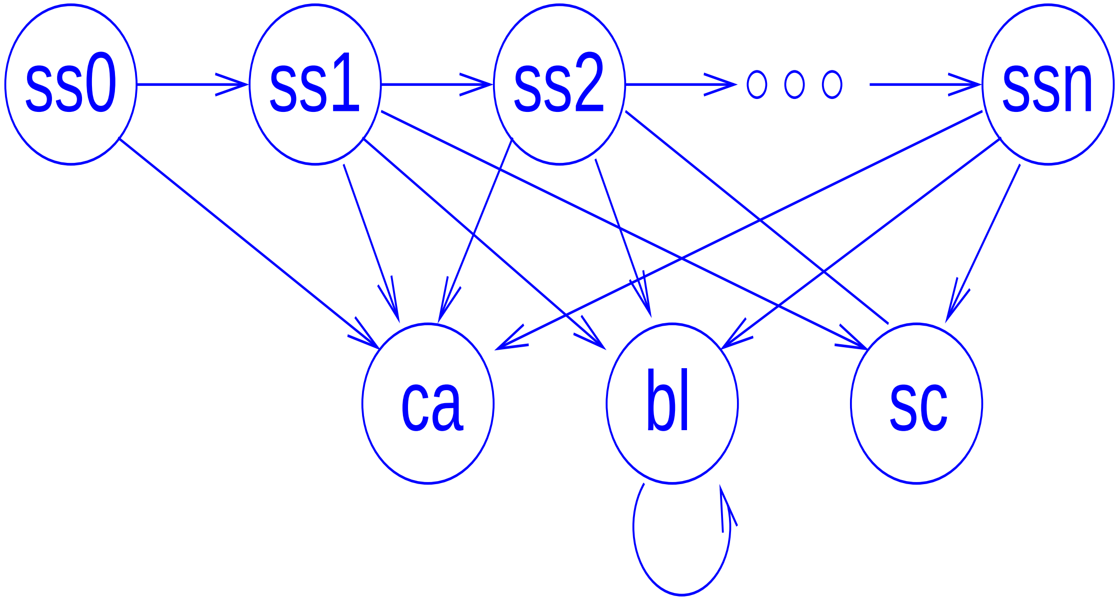
Estimating Parameters

- Distribution of rtt.
- Window sizes for ss_i and bl.
- Distribution of throughputs for ca and sc.
- State transition probabilities.



Estimating Parameters

- Distribution of rtt.
- Window size distributions for ss_i and bl.
- Distribution of throughputs for ca and sc.
- State transition probabilities.



Generating predictions

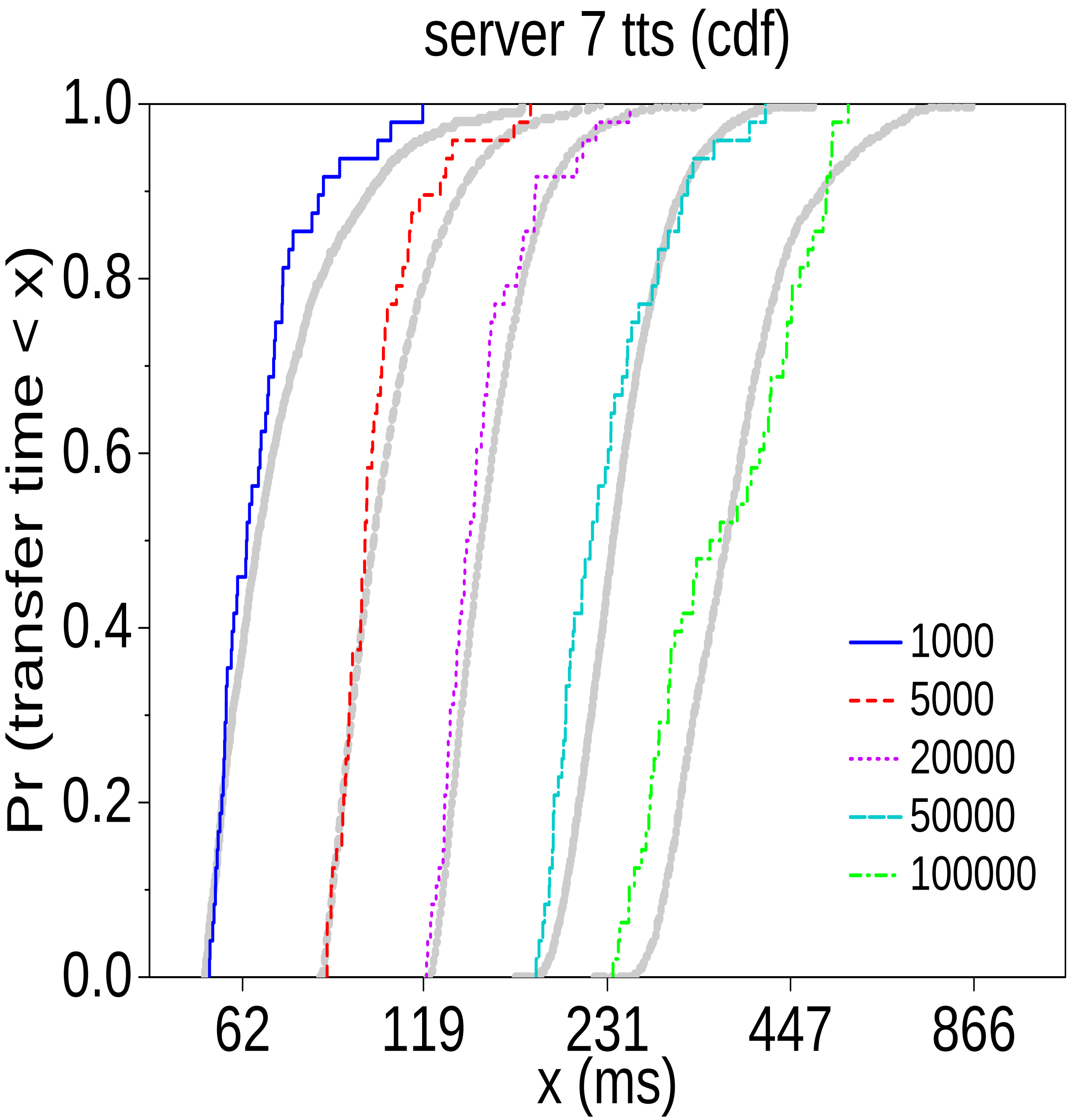
Given size s...

1. Initialize $state = ss_0$, $s_{total} = 0$, $t_{total} = rtt_0 + rtt_1$.
2. Choose a state transition, $state = S_{state}$.
3. If $state = ca$ or sc, $throughput = T_{state}$, $t_{rem} = (s - s_{total})/throughput$, return $t_{total} + t_{rem}$.
4. $win = W_{state}$, $s_{total} = s_{total} + win$.
5. If $s_{total} > s$, return t_{total}.
6. $rtt = R_{state}$, $t_{total} = t_{total} + rtt$.
7. Go to step 2.

Validation

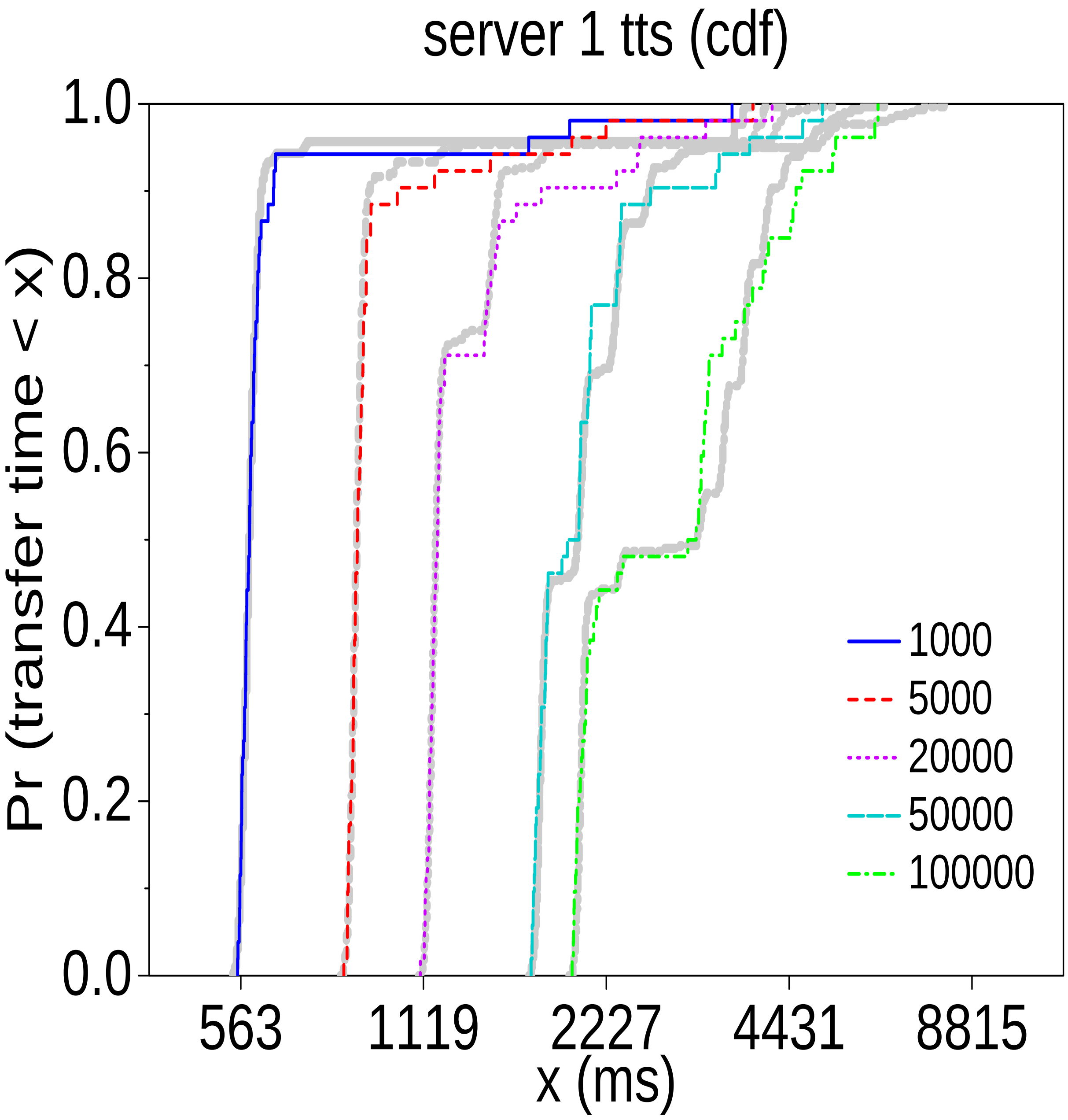
- Randomly partition 2 datasets of 50 measurements.
- Estimate parameters and generate distributions from one subset.
- Compare to actual times from other subset.
  $t(s) =$time until receive sth byte.
- Agreement indicates that the model is sufficiently detailed, and that the estimated parameters are consistent.

Example #1



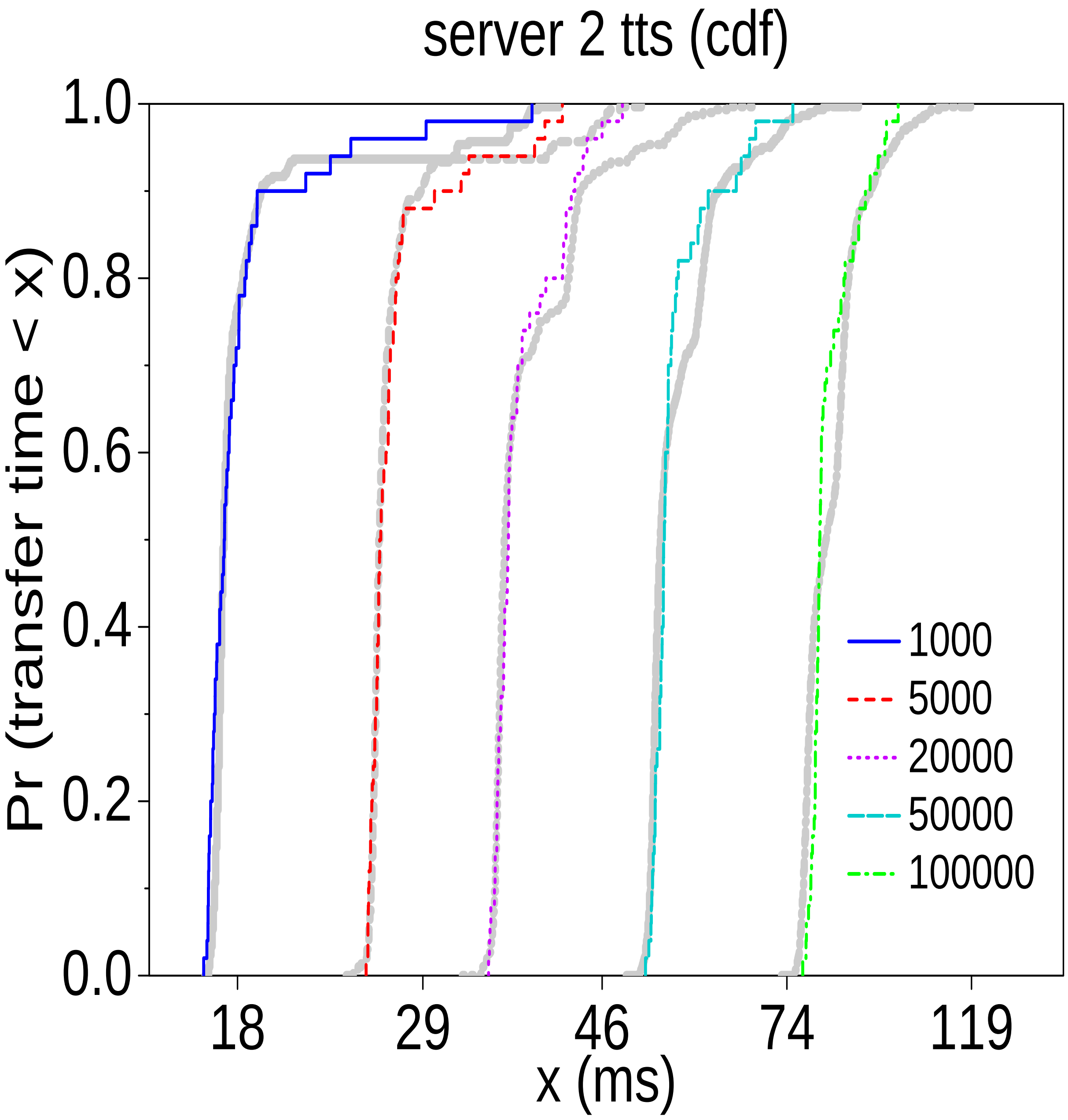
- Deterministic slow start.
- Most transfers self-clocking.

Example #2



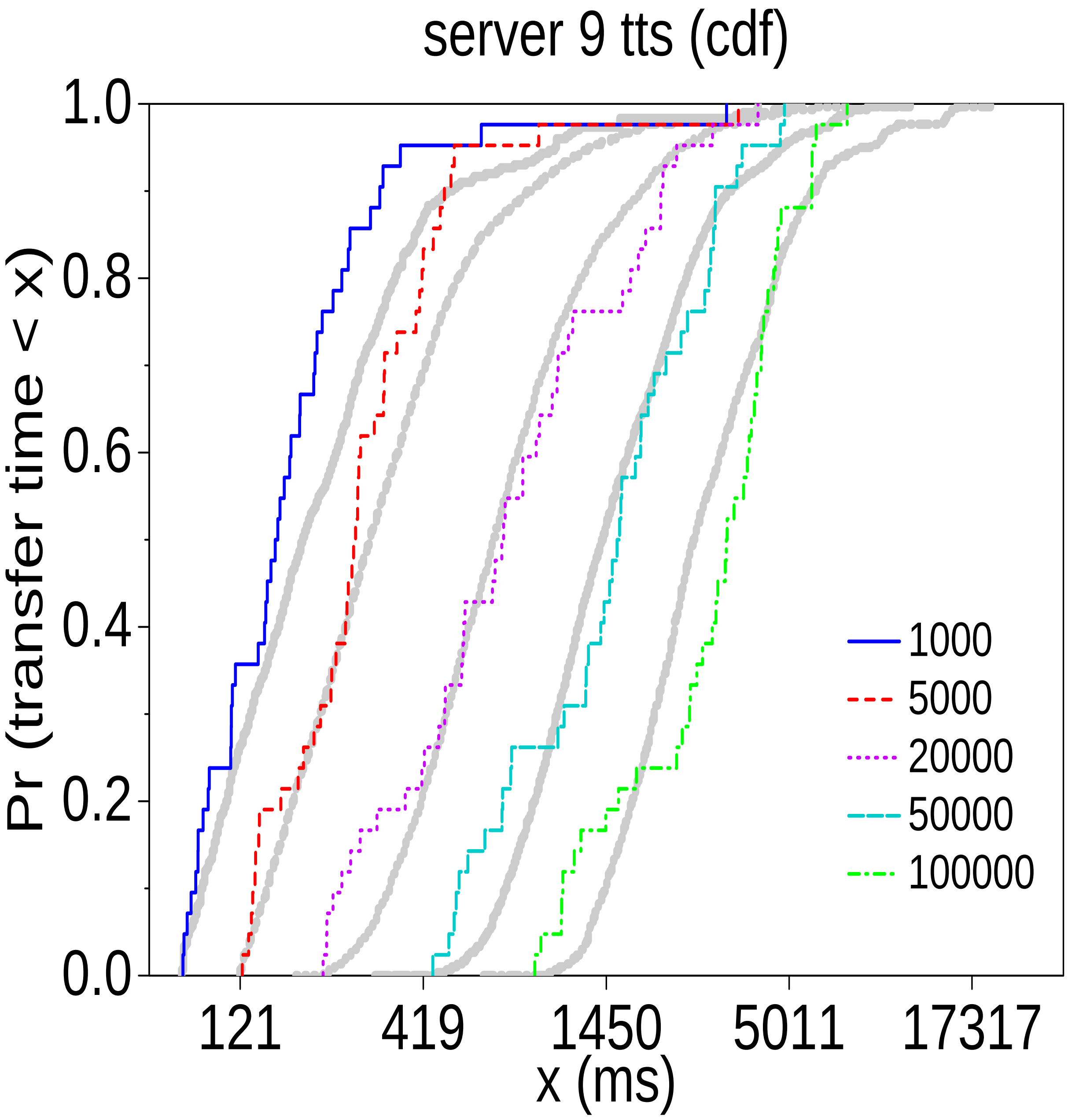
- Nondeterministic slow start $\Rightarrow$ multimodal distributions.
- Modes at multiples of rtt.

Example #3



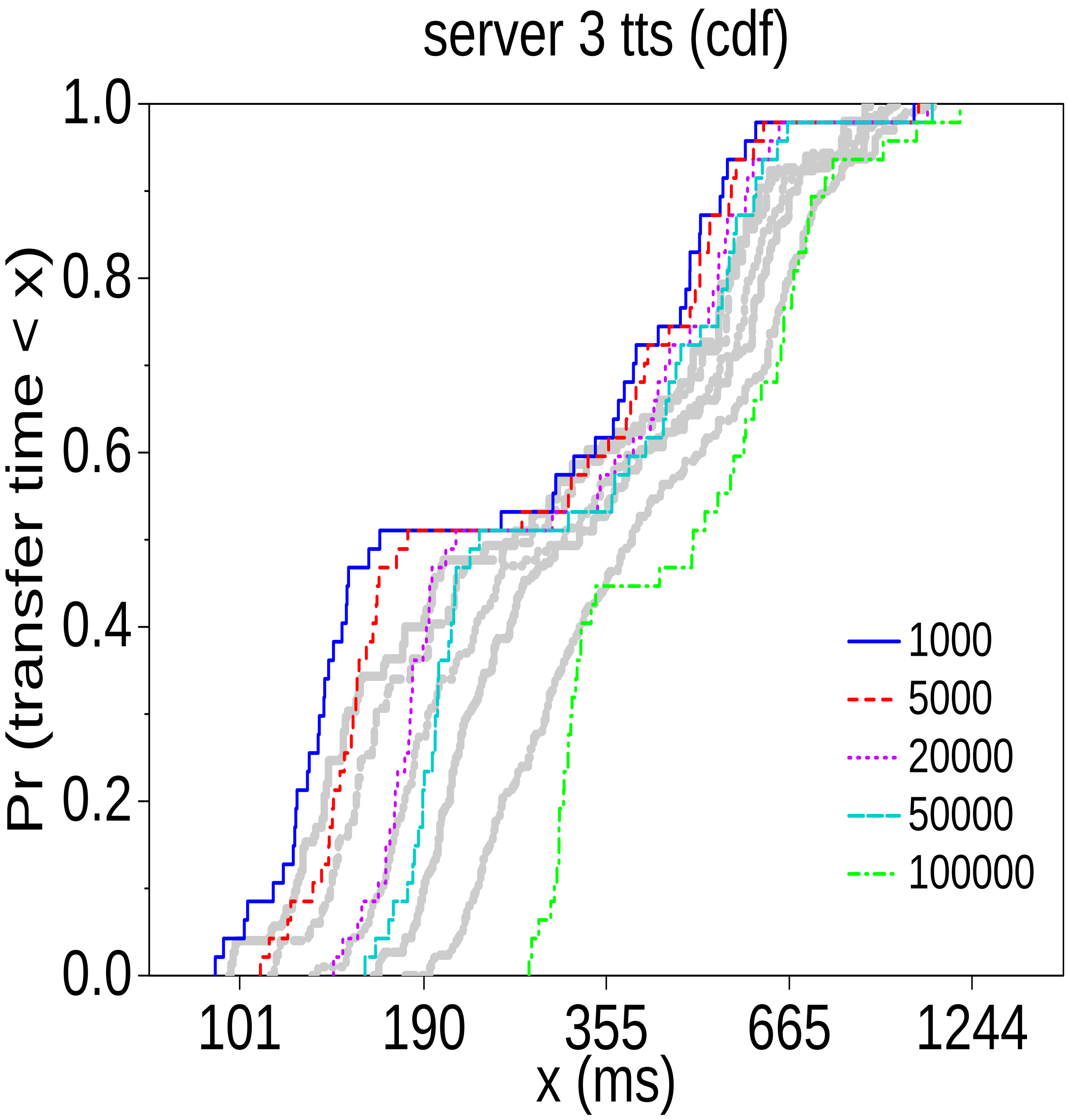
- Buffer-limited.

Example #4



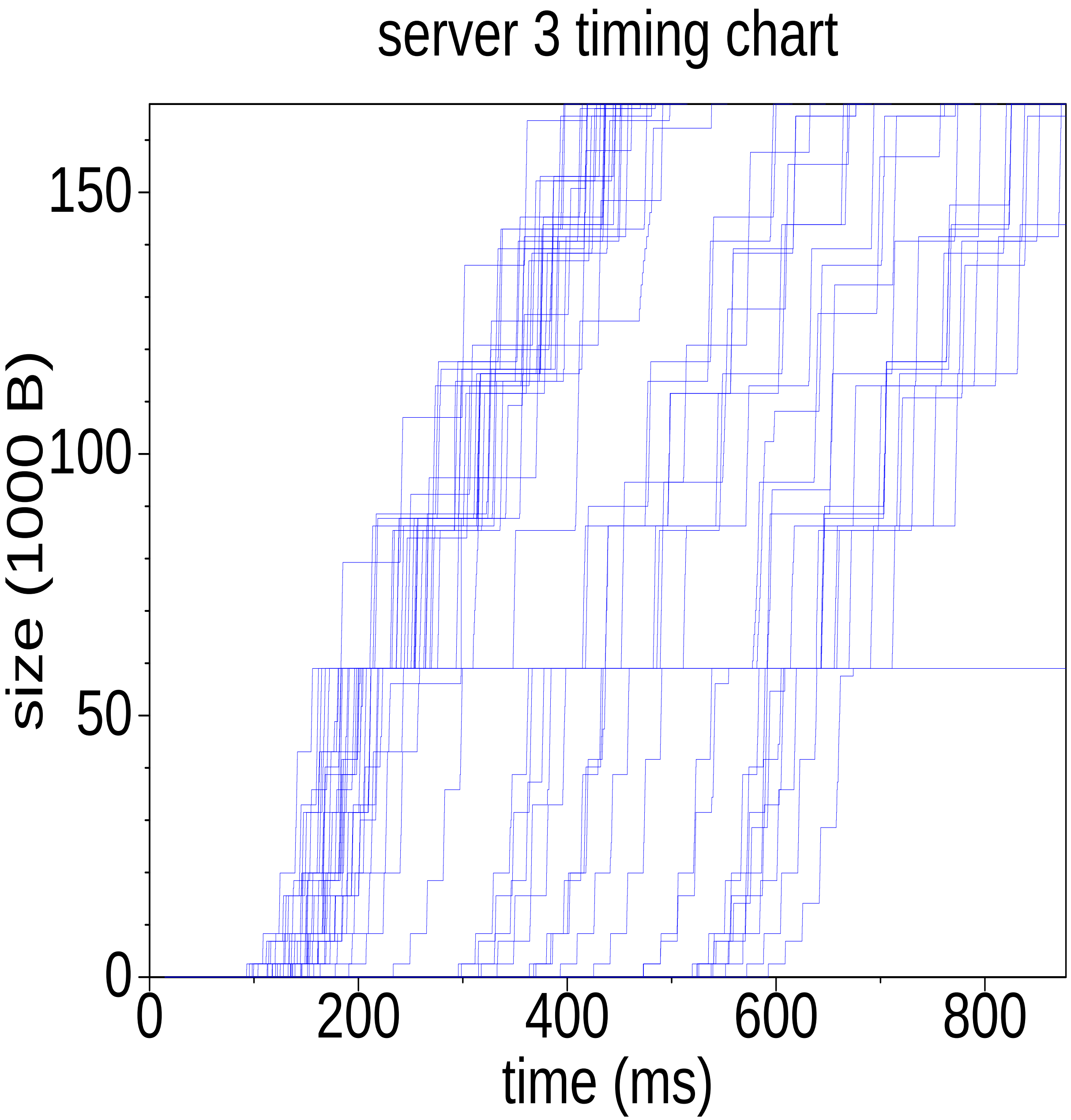
- Mostly congestion avoidance, some self-clocking.
- Underestimating variability?

Evil case #1



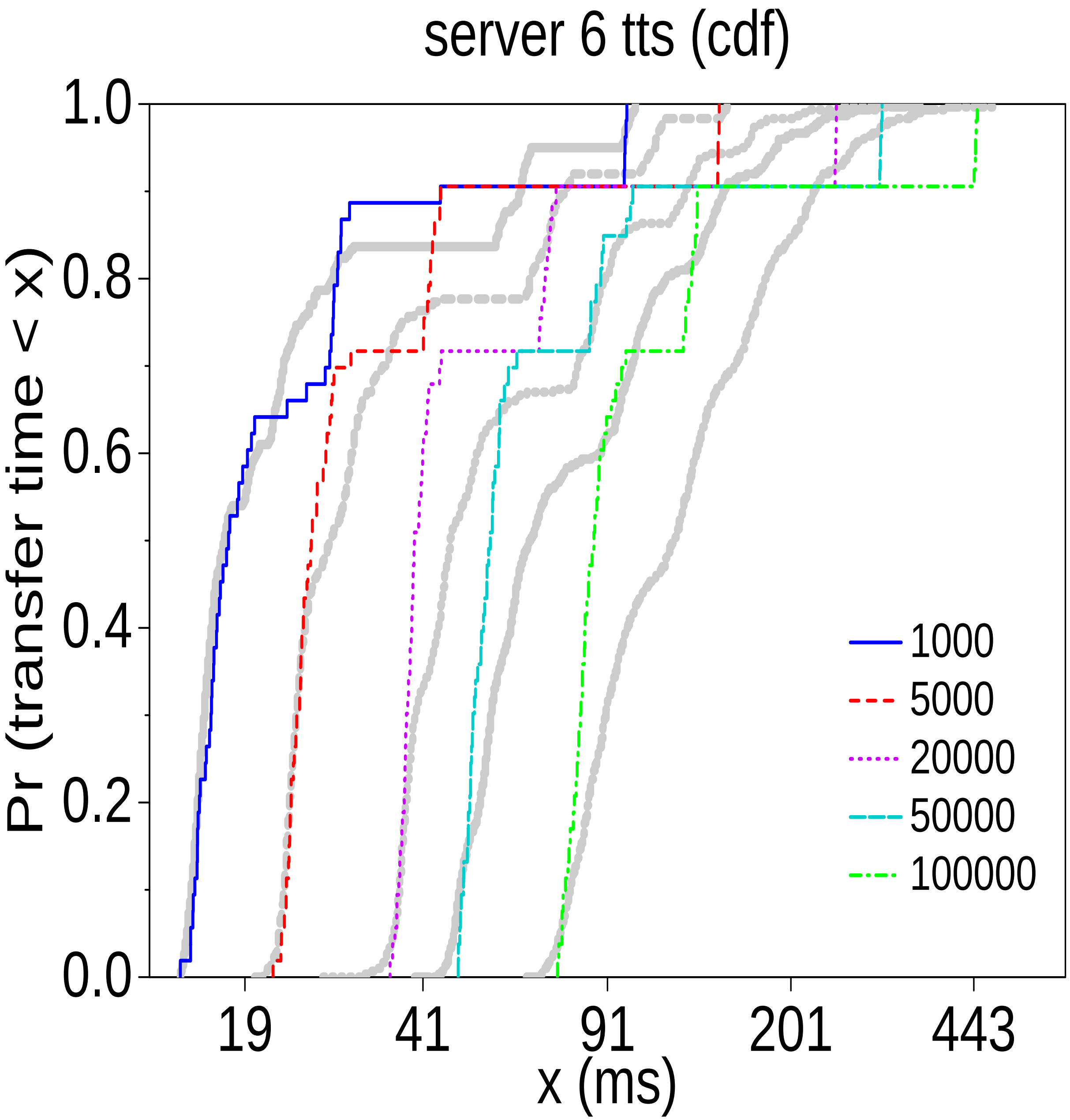
- Up to 50,000 bytes, not bad.
- Anything over 60,000, way off!

Server performance



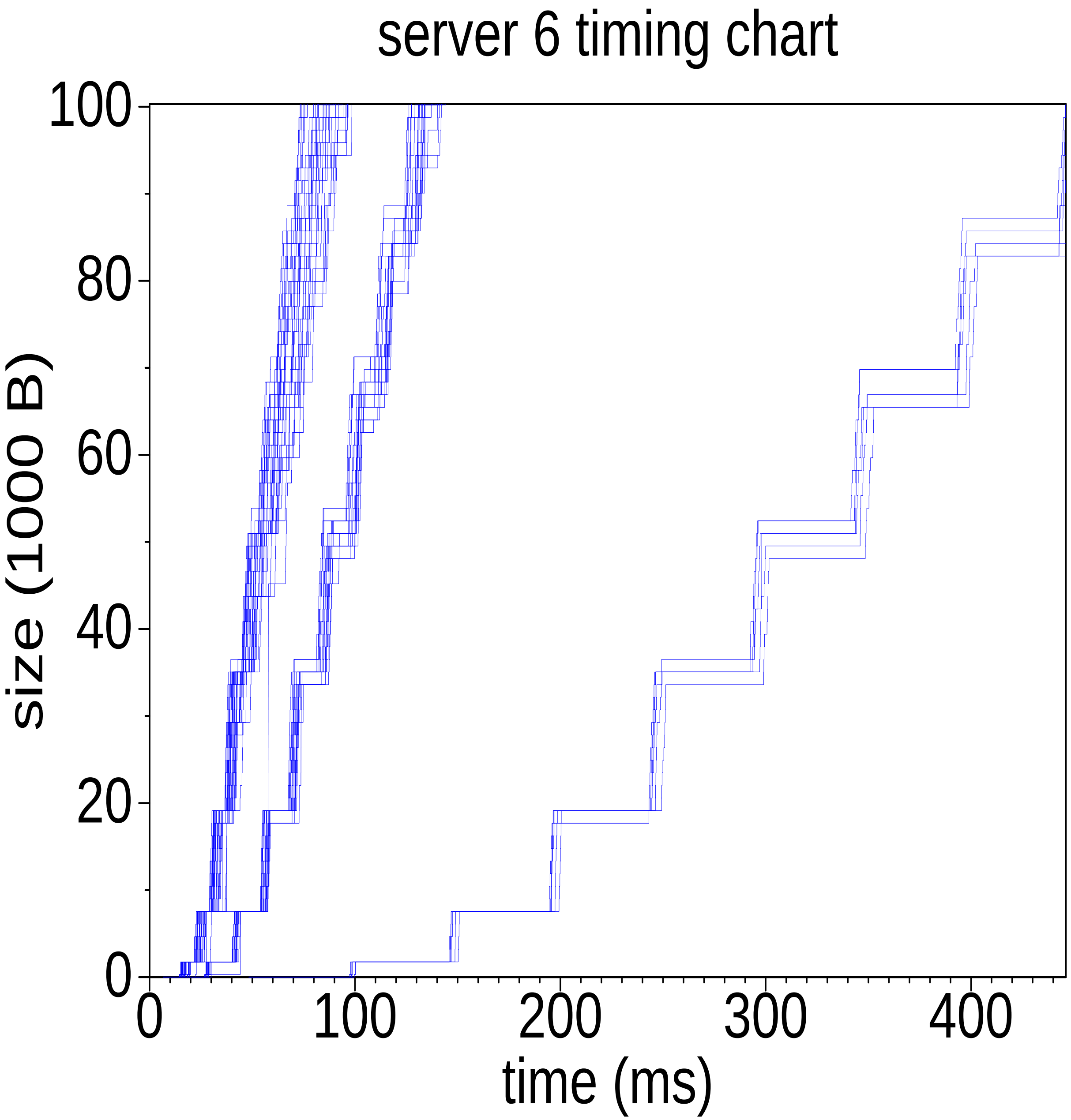
- 50 ms delay after 40 packets.
- Model includes initial processing at server.
- After that, assumes that servers keep up.

Evil case #2



- Actual times are sharply multimodal.
- Model smoothes the modes.

Multiple paths



- Akamai-style content delivery $\Rightarrow$ multimodal rtt.
- Model includes correlation, but not the right correlation structure.

Headlines

- Model includes three steady-state behaviors.
  - Author claims good agreement with measurements.
- Endogenous drop risk exaggerated.
  - With enough queue capacity, self-clocking works!
- Non-deterministic slow start sighted.
  - TCP characteristic or Linux bug?

Future work

- Do short transfers predict long transfer performance?
- Put model into predictive structure.
- TCP as bandwidth estimation tool?
- Application-level server tuning.
- Application-level TCP pacing.

Had enough?

- Full paper and additional data available from

  `http://allendowney.com`

- Contact me at

  `downey@allendowney.com`

Future work

- Do short transfers predict long transfer performance?
- Put model into predictive structure.
- TCP as bandwidth estimation tool?
- Application-level server tuning.
- Application-level TCP pacing.

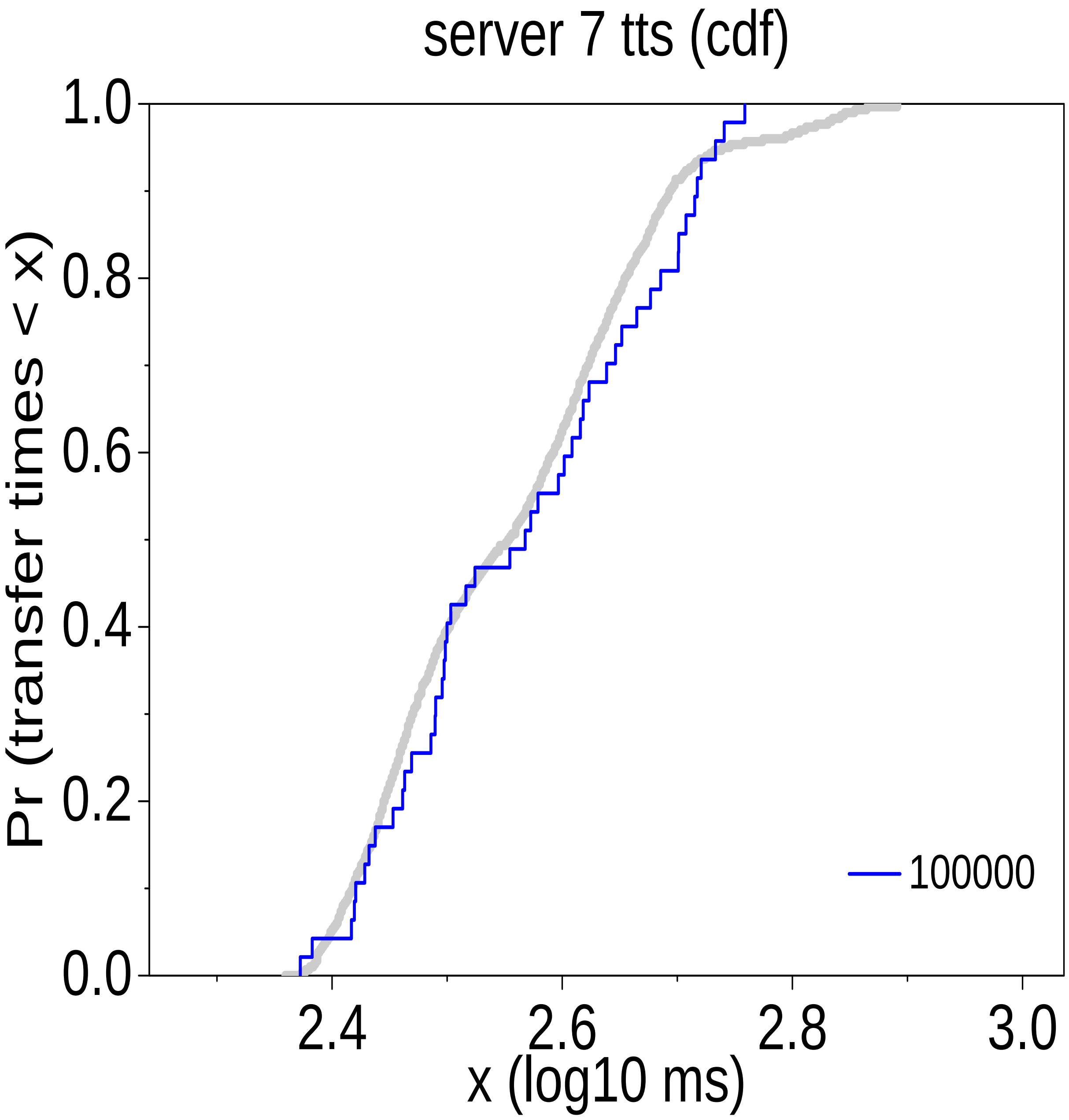
So far, mostly good

First test: 100K measurements predict 100K transfers.

- The right location.
- The right variability.
- Usually the right shape.
- Tail behavior?

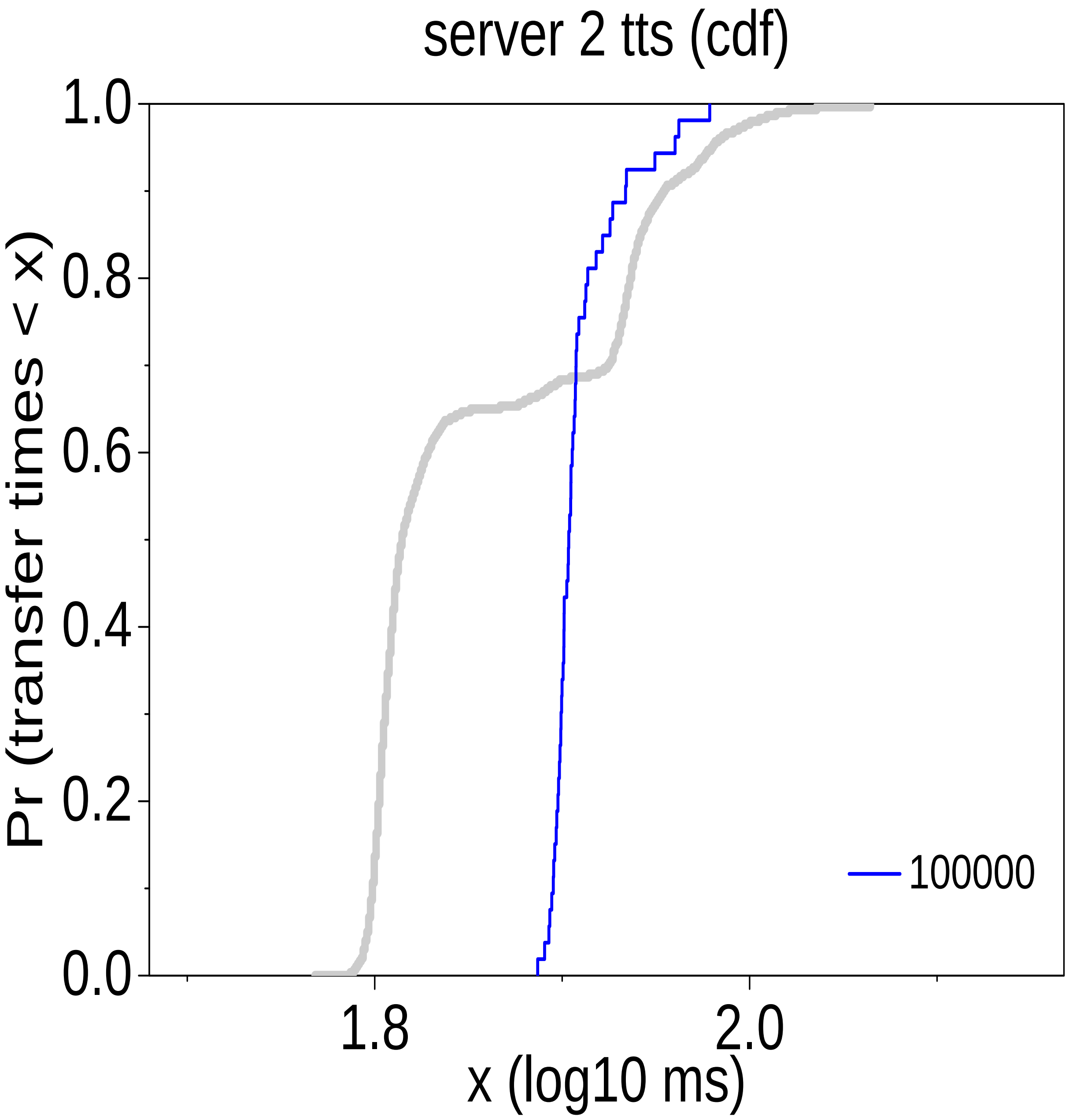
Stronger test: do 50K measurements predict 100K transfers?

More validation



- Censor data at 50,000 bytes, predict 100,000 bytes.
- Results good, if we see the end of slow start.

More validation

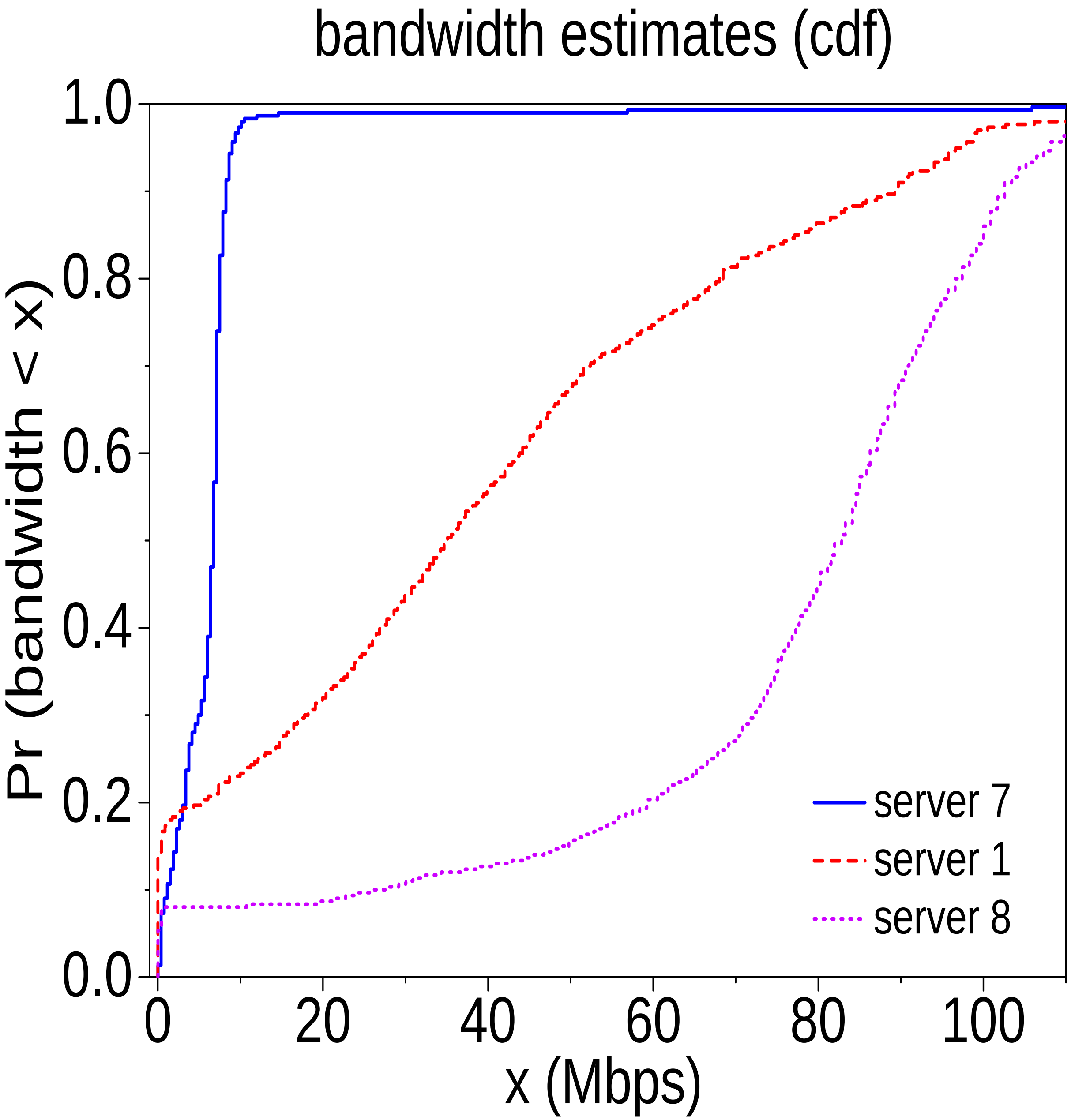


- Pretty bad, if we don’t get a clear view of steady-state behavior.
- In this case, bl looks like ss_4.

Prediction

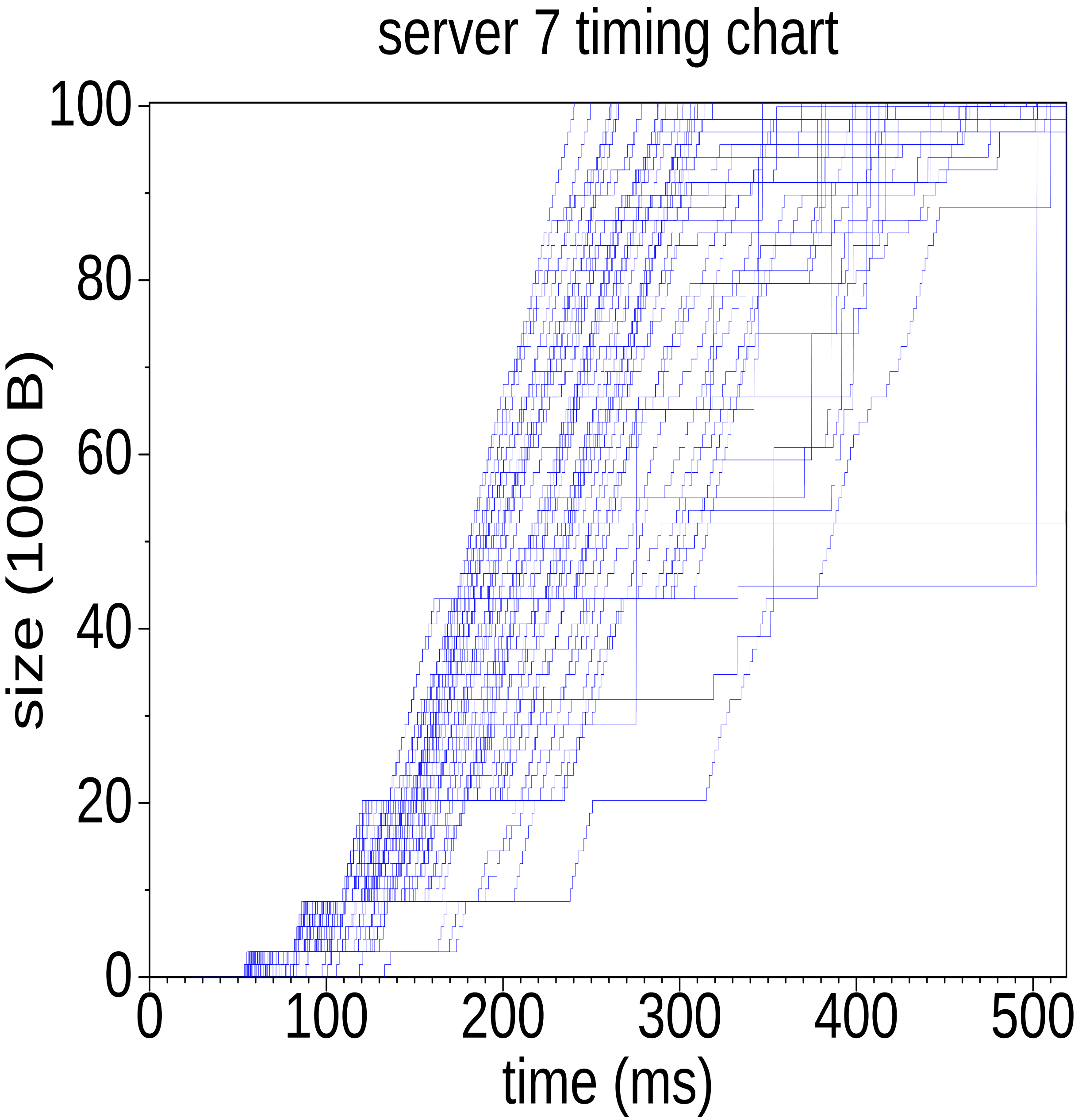
- Need to collect data “long enough”.
- Capture short-term variation.
- Identify long-term shifts (path changes).
- Embed model in NWS-like structure.

Bandwidth estimation



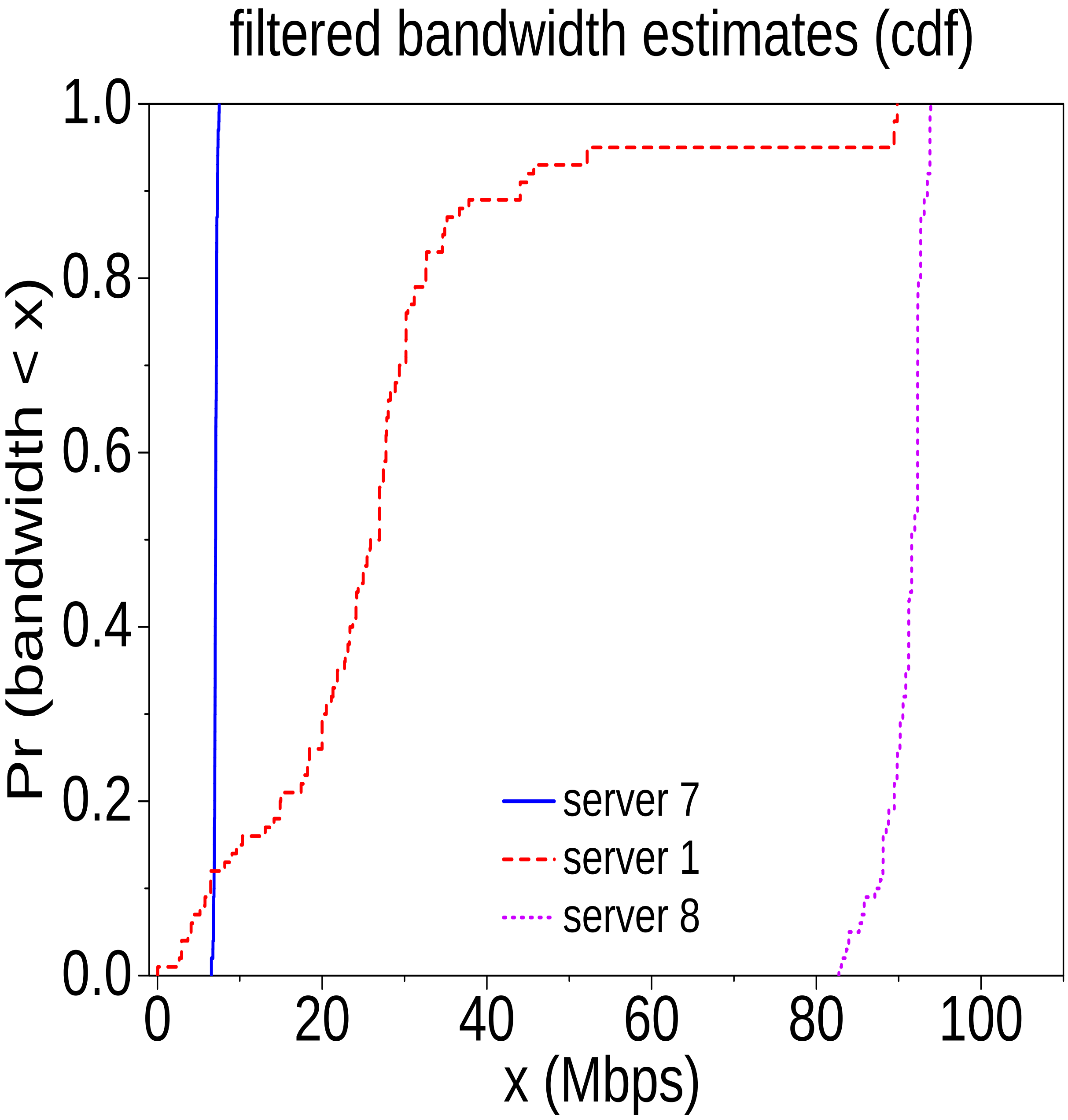
- Lots of prior work on packet pair bw estimation.
- Assumption: bottleneck bw is at least a local mode in distribution of pair-wise estimates.

Bandwidth estimation



- Visually, the characteristic slope seems obvious.
- Statistical filter: look for the straightest k-packet sequences.
- Keep n sequences with lowest variability.

Bandwidth estimation



- Estimates much more repeatable.
- More accurate?

Server tuning

- Bigger bdp increases demand for send buffers.
- Vast majority of connections either ss or bl.
- Use performance model for:
  - Acceleration: faster slow start, dynamic $ssthresh$.
  - Allocation: bigger buffers for connections that can use them.
  - Scheduling: bigger-shorter vs. smaller-longer.

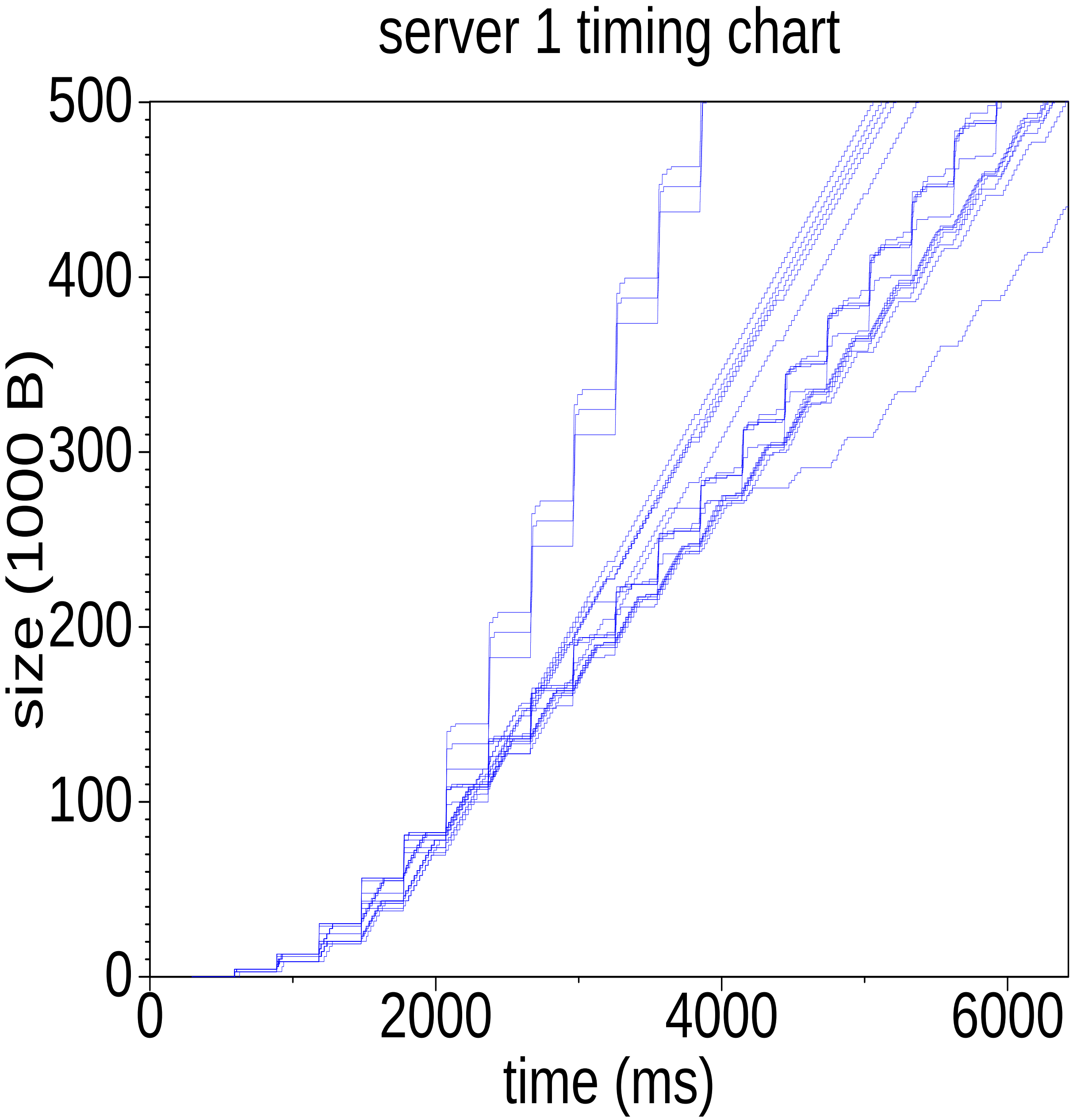
TCP Pacing

- Good: self-clocking achieves rate-based transmission in a window-based mechanism.
- Bad: vast majority of connections either ss or bl.
- Packets are sent faster than bw.
- Unnecessary burstiness, queueing at bottleneck.
- TCP pacing: good for you, good for the network. [a]

---

[a] Your mileage may vary. See Aggarwal, Savage and Anderson, “Understanding the Performance of TCP Pacing.”

TCP Pacing



- Application rate $\Rightarrow$ advertised window $\Rightarrow$ send rate.
- Here, rate is static.
- tt should be no worse.
- Better queueing/drop behavior.

Prior work

- Lots of work on congestion avoidance.

$$E[throughput] = f(rtt, p)$$

  - Assume throughput is congestion-limited.
  - Various drop models: usually exogenous.

- Some work on slow start.

$$cdf_{tt} = f(rtt, p)$$

  - Known cw_1, cw_2 ...
  - Again, drop rate is exogenous.