On Packet Reordering and TCP Performance

Ethan Blanton, Ohio U.
Mark Allman, NASA GRC/BBN

May 2002

Pointers

- Content mostly from:

  Ethan Blanton, Mark Allman. On Making TCP More Robust to Packet Reordering. ACM Computer Communication Review, 32(1), January 2002.

  http://roland.grc.nasa.gov/~mallman/papers/tcp-reorder-ccr.ps

- Slides at:

  http://roland.grc.nasa.gov/~mallman/talks/reorder-5610retreat-may02.ps

Overview

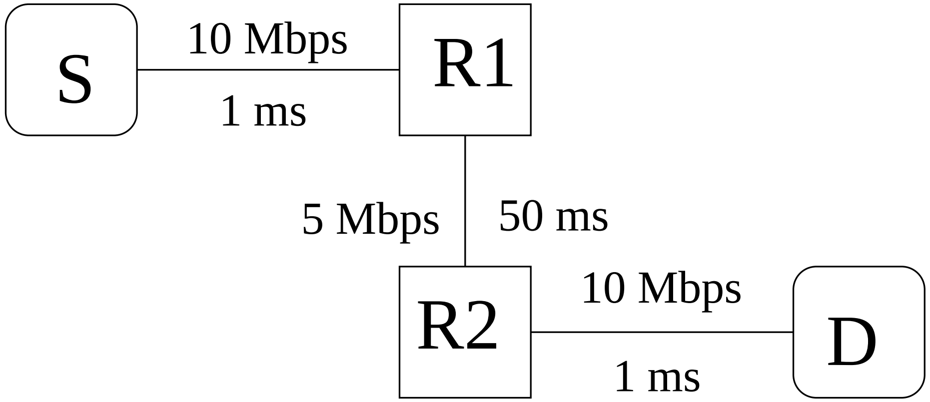
- Background
  - Why worry about packet reordering?
  - Methodology
- Problem #1: detecting bad retransmit decisions
- Problem #2: compensating for bad loss detection
- Problem #3: avoiding bad decisions
- Conclusions and future work

Background

- IP networks are free to deliver packets in arbitrary order
- Expect in-order delivery, but...
  - Bennett, et. al. illustrates that packet reordering caused by "multi-path" hops is not rare in some networks
  - Paxson shows that in the aggregate packet reordering is not common, but over certain paths it is quite prevalent
- We may see more reordering
  - Ad-hoc networks & handoffs
  - Link layer ARQ
  - Multi-path routing
  - DiffServ networks

Methodology

- Network layout:



- ns simulator
- Used a "swapping" queue
- SACK TCP (some simulations with DSACK)
- One 10 minute TCP flow
- Not realistic, but...

Impact of Reordering on TCP

- TCP uses "fast retransmit" as one method for inferring packet loss
  - Preferred to RTOs
  - The problem is that reordering causes the same pattern of ACKs as packet loss
  - Fast retransmit applies a bit of logic to trying to disambiguate packet loss from packet reordering
    - Compensation is rigid and fragile

- Packet loss is used to infer congestion
  - The congestion window (sending rate) is reduced

- So, intuitively bogus loss detection leads to sub-optimal performance

TCP Example #1

- Receiver’s view of ordered TCP stream

Arriving Packet	ACK Xmitted
101	101
102	102
103	103
104	104
105	105
106	106
107	107
108	108
109	109
110	110

TCP Example #2

- Receiver’s view of ordered TCP stream with a loss

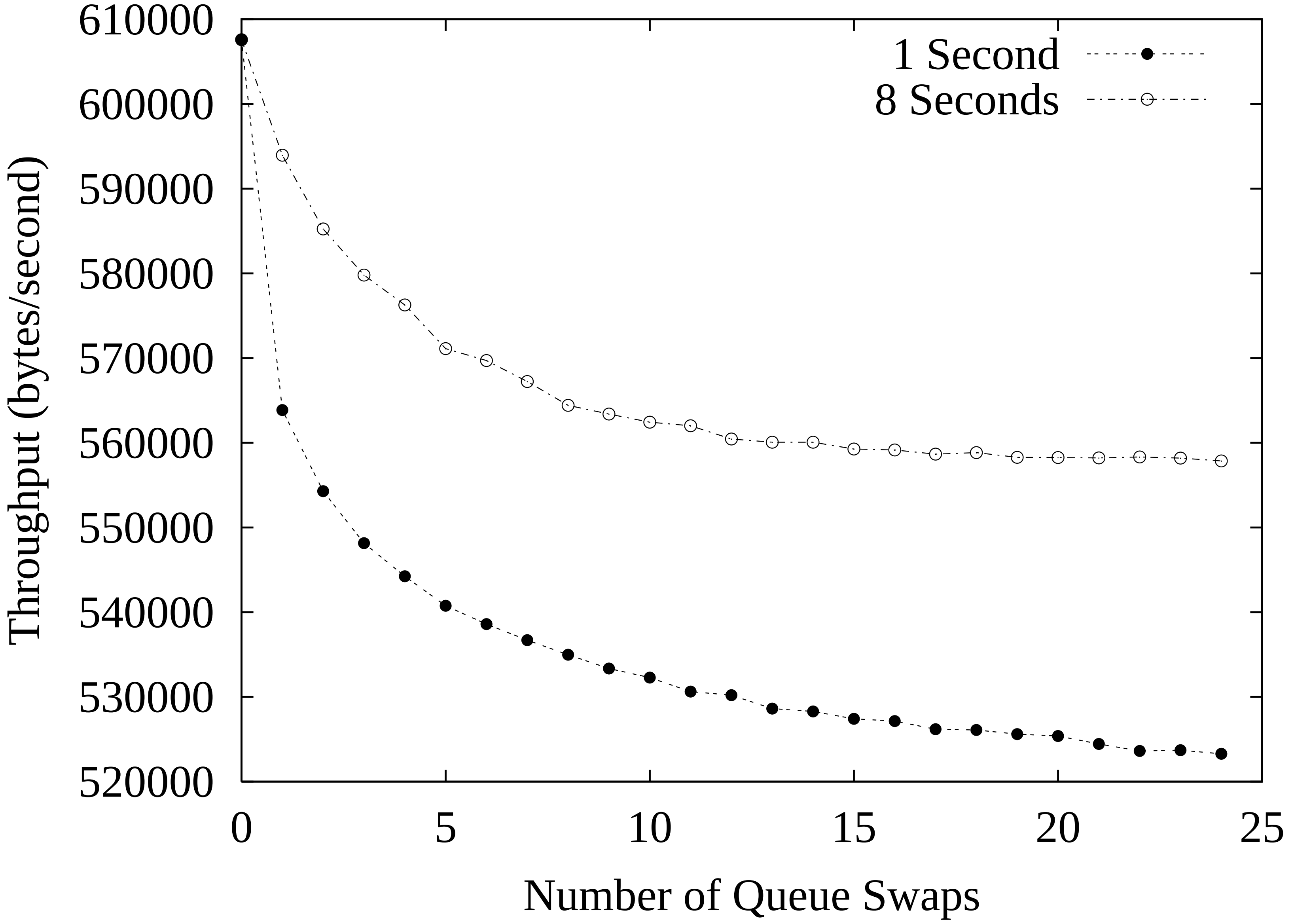
Arriving Packet	ACK Xmitted
101	101
102	102
103	103
105	103
106	103
107	103*
108	103
109	103
110	103
104	110

TCP Example #4

- Receiver’s view of TCP stream with reordering

Arriving Packet	ACK Xmitted
101	101
102	102
103	103
105	103
106	103
107	103*
108	103
109	103
110	103
104[1]	110
104[2]	110

Impact of Reordering on TCP



Detecting Spurious Retransmits

- Problem #1: How can we detect a bad retransmission decision?

- Several schemes available:
  - DSACK (Floyd, et. al.)
  - Eifel (Ludwig)
  - Timestamps (everyone)
  - Timing heuristics (Allman/Paxson, maybe others)

- While there are real-world engineering tradeoffs to which scheme to use they all would have worked about the same in our simulations

- So, we use DSACK -- as the only one of the above that is standards track

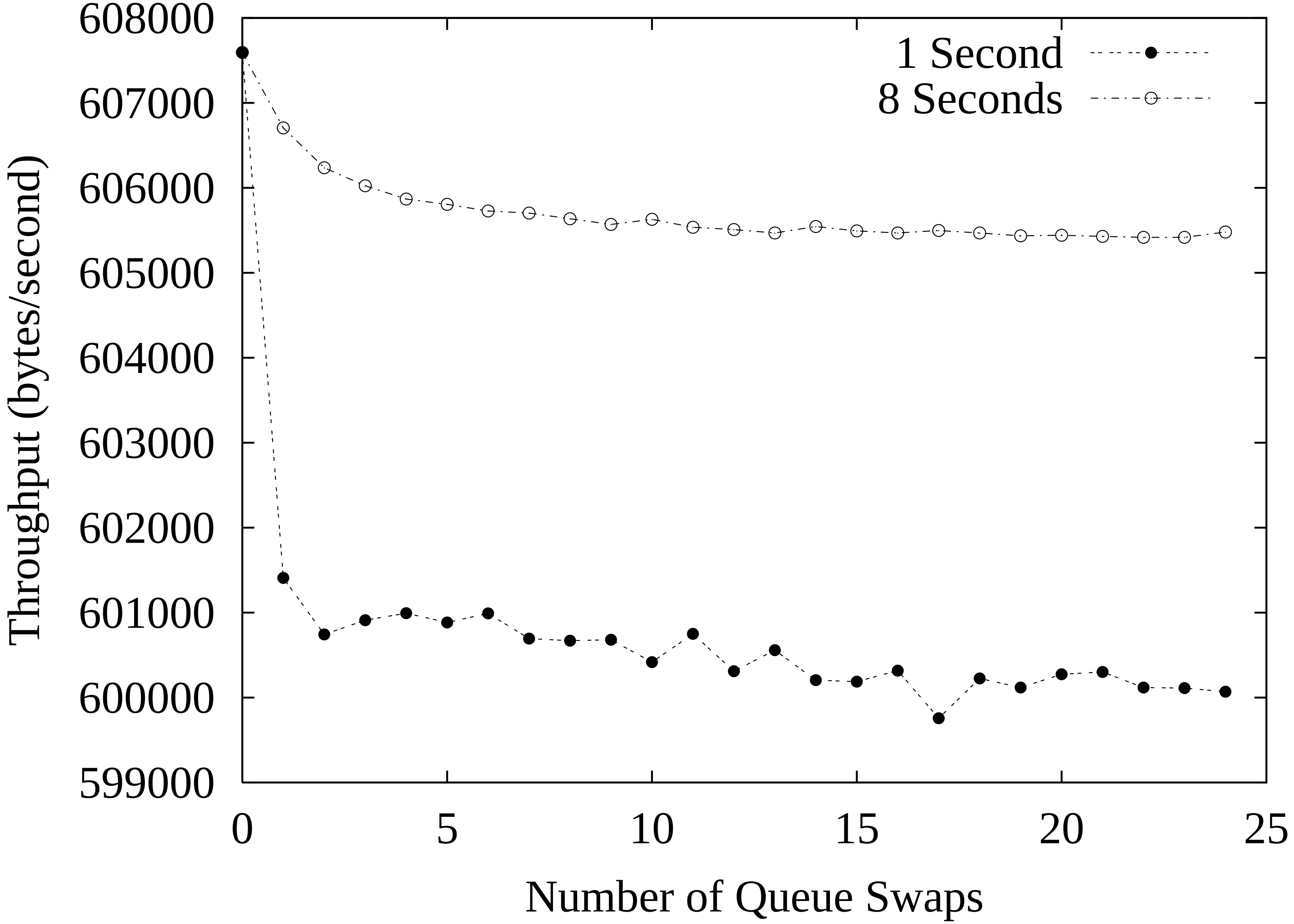
Undo-ing Bad Decisions

- Problem #2: After detecting a bad decision, what might be done?

- If TCP can detect (after the fact) that a spurious retransmission was sent we can attempt to fix some of the damage
  - Undo the reduction of cwnd (sending rate)
    - There was no congestion to react to in the first place

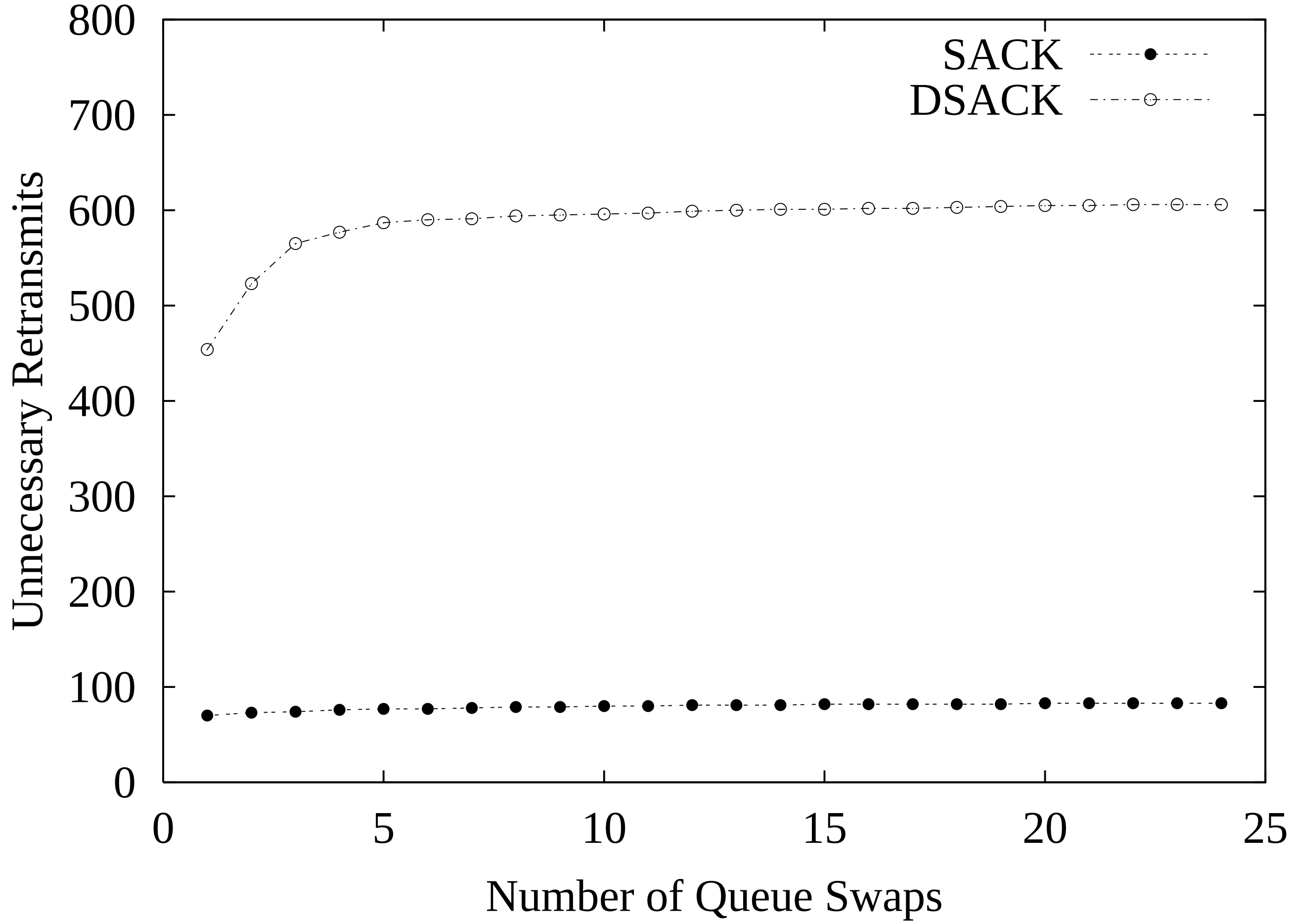
- Can’t get the network resources expended on the bogus retransmit back
  - The importance of this depends on the situation

Undo-ing Bad Decisions (cont.)



Undo-ing Bad Decisions (cont.)

- The throughput problem is largely gone
- But, a new problem:



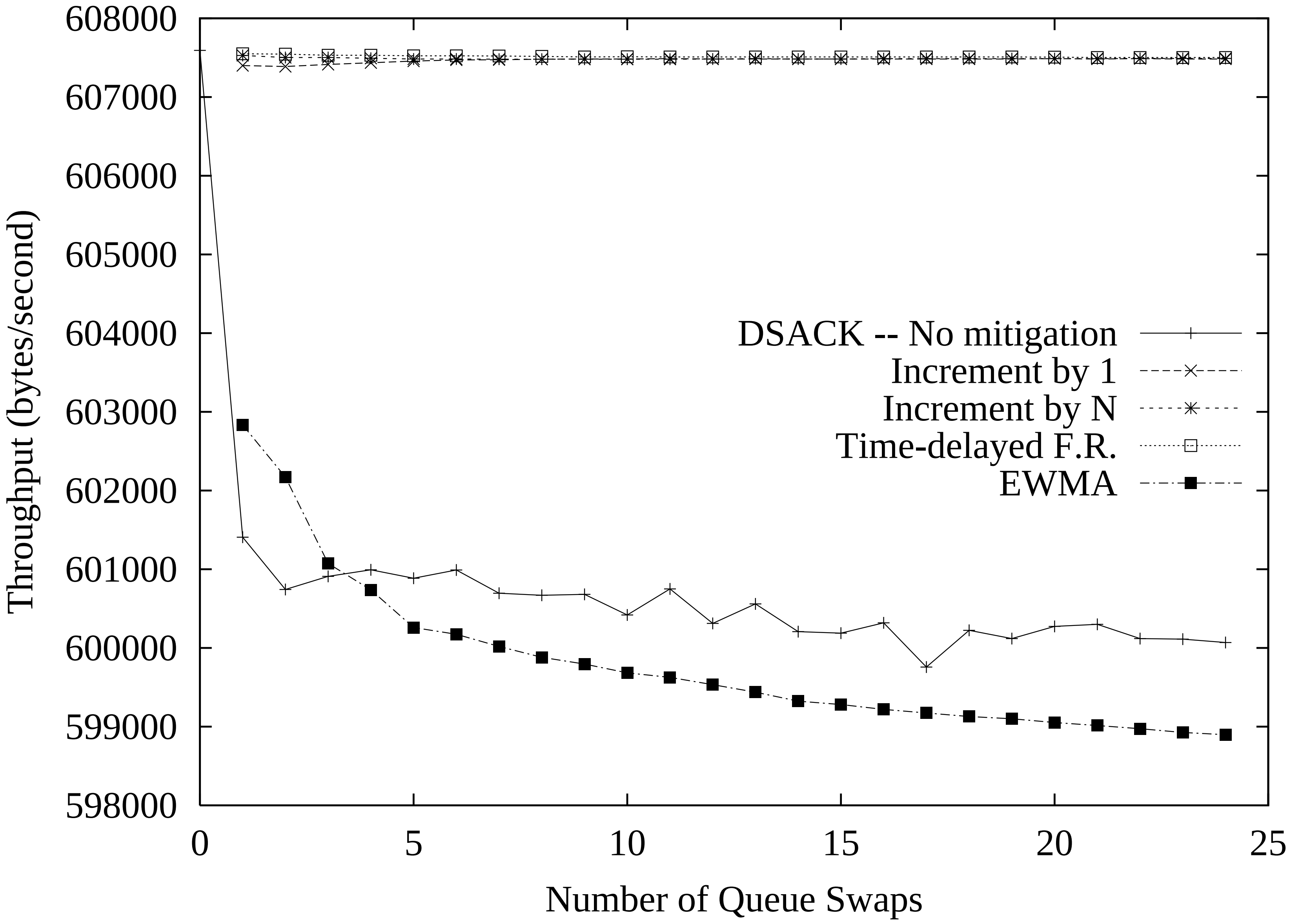
Avoiding Spurious Retransmits

- Problem #3: Can we use past performance to avoid future bad decisions?
- Lots of bogus retransmissions lead toward congestion collapse
  - We're just gunking up the network with things that don't need to be sent -- these packets accomplish no useful work
- So, we'd like to allow TCP to avoid the spurious retransmits caused by reordering
  - We want to make dupthresh a variable that changes with the observed network conditions

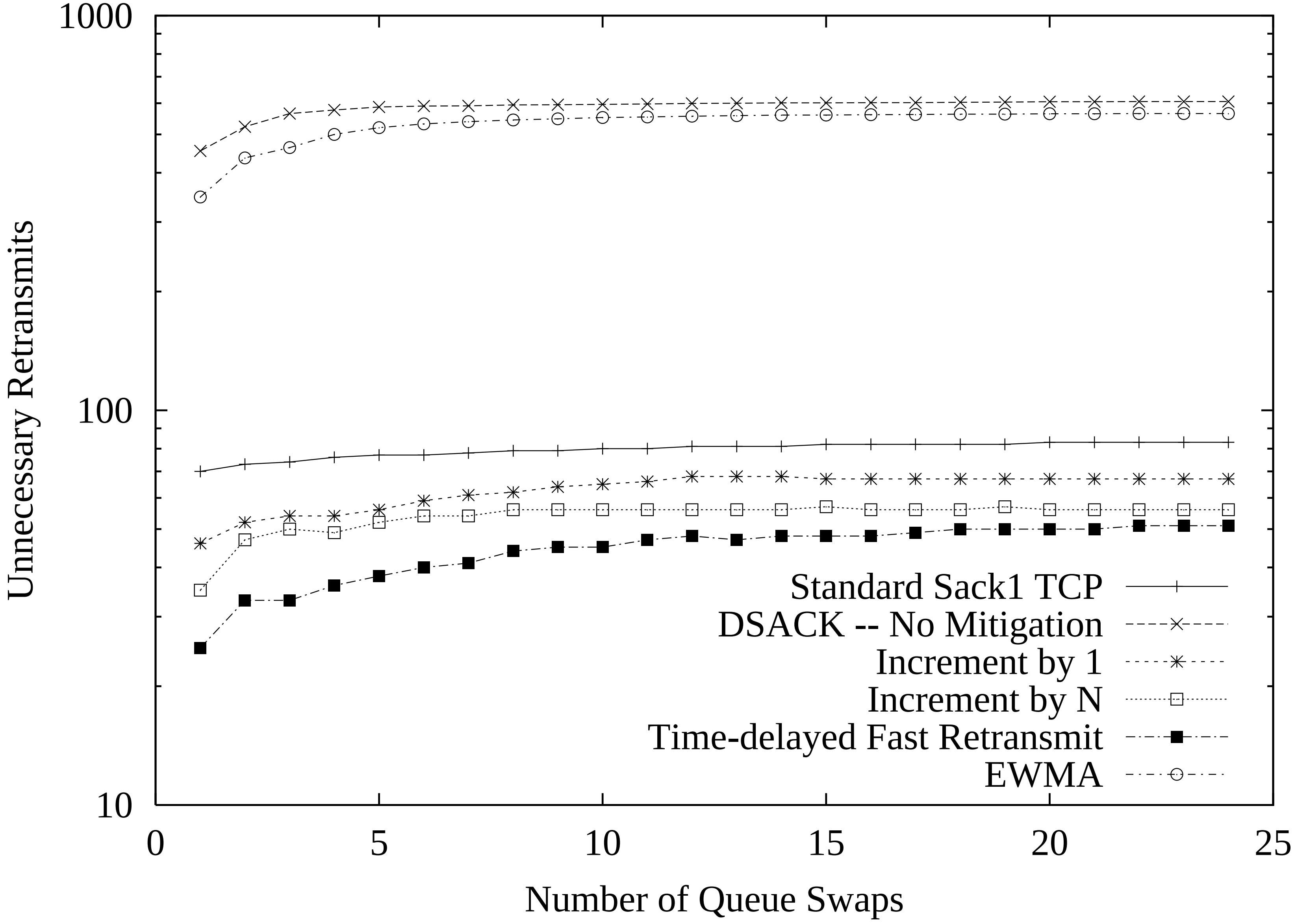
Avoiding Spurious Retransmits (cont.)

- Several schemes to adjust threshold examined:
  - Increase dupthresh by K (K=1) for each mistake
  - Increase dupthresh by length of reordering event
  - Keeping dupthresh=3, but adding a (small) timeout before retransmit
  - EWMA running average of the right (based on measurements) dupthresh
- Can't let dupthresh grow too large (> cwnd)
- When fast retransmit fails and we take an RTO we reduce dupthresh back to 3
  - The guess is that our dupthresh is no longer a useful measure of the amount of reordering in the network

Avoiding Spurious Retransmits (cont.)



Avoiding Spurious Retransmits (cont.)



Conclusions

- We can make TCP more robust to packet reordering with some fairly simple algorithms
- We can:
  - Increase the throughput of TCP connections
  - Decrease the number of spurious retransmissions sent into the network
- The cost:
  - Maybe more RTOs
    - Need more realistic simulations or real-world tests to assess
  - A bit of complexity in the TCP processing

Future Work

- Researchers from ICIR have a different scheme for making TCP robust to reordering.
  - It is a lot more complex
  - It uses a lot more state
  - It performs better
- So, useful future work in this area is to see if we can make the two schemes converge a bit into a mechanism that is simple, does not require a lot of state and performs well

Extra Slides

TCP Example #3

- Receiver's view of TCP stream with (a little) reordering

Arriving Packet	ACK Xmitted
101	101
102	102
103	103
105	103
106	103
104	106
107	107
108	108
109	109
110	110

The EWMA

$$avg = \begin{cases} \alpha \cdot N + (1 - \alpha) \cdot avg & \text{if } N > avg \\ (\alpha \cdot x) \cdot N + (1 - \alpha \cdot x) \cdot avg & \text{otherwise} \end{cases}$$

$$dupthresh = \lfloor avg + 0.5 \rfloor$$

Varying the EWMA