Measuring Packet Reordering

John Bellardo
Stefan Savage

Department of Computer Science and Engineering
University of California, San Diego
November 6, 2002
Motivation

• Why is reordering important?
  – **Performance** (TCP fast retransmit)
  – Race conditions (bad protocols)

• What is hard about measuring it?
  – [Bennett et al 99]: active ICMP probing (ping)
    • Round-trip only; ICMP filtering/rate limiting bias
  – [Paxson 99]: pair-wise TCP endpoint analysis
    • Scale issues (need software at each endpoint)
  – [Jaiswal et al 02]: passive TCP analysis in net
    • Significant infrastructure requirement
Our contributions

• Unidirectional measurement techniques
  – Active approach
    • Send packet pairs and check for reordering
  – Code runs only at sender
    • Leverage TCP/IP protocol/implementation features
      • Infer if reordering is outbound or on return path
• Implementation of same
• Early experiences
First attempt:  
Single Connection Test

• Leverage TCP’s *error control* mechanisms
  – Every packet is labeled w/sequence number
  – Latest in-order sequence number acknowledged
  – Idea: Craft packets so ACKs reveal reordering

• Assumption
  – *ACK parity*: ACK generated for each packet
Single Connection Test

- Fully establish a TCP session
  - Sequence space starts at 1
Single Connection Test

- Fully establish a TCP session
  - Sequence space starts at 1
- Create a gap in sequence space
Single Connection Test

- Fully establish a TCP session
  - Sequence space starts at 1
- Create a gap in sequence space
  - Wait for remote host to ACK the gap
Single Connection Test

- Fully establish a TCP session
  - Sequence space starts at 1
- Create a gap in sequence space
  - Wait for remote host to ACK the gap
- Send two sample packets that straddle the previous packet
Single Connection Test

- Fully establish a TCP session
  - Sequence space starts at 1
- Create a gap in sequence space
  - Wait for remote host to ACK the gap
- Send two sample packets that straddle the previous packet
- If there is no reordering
  - First ACK should be for the gap
Single Connection Test

- Fully establish a TCP session
  - Sequence space starts at 1
- Create a gap in sequence space
  - Wait for remote host to ACK the gap
- Send two sample packets that straddle the previous packet
- If there is no reordering
  - First ACK should be for the gap
  - Second ACK is for the whole sequence
**Single Connection Test**

![Diagram showing data packets and acknowledgments with different reordering scenarios.](image-url)
Single Connection Test Pitfalls

- Packet loss results in unusable samples (*general limitation*)

- ACK parity assumption fails
  - Delayed acknowledgements
  - Need both ACKs to reveal order

ACK 3 gets delayed and subsequently is never sent
Dual Connection Test

- Need two samples to be reliable returned
  - Send all packets out of order (ACK not delayed)
  - ACK value useless, so infer order from other fields
    - Use two connections to differentiate samples
    - IPID – “unique” identifier for each datagram in a flow

- New assumptions
  - IPID is strictly increasing per host
    - Dominant implementations do this
  - Both connections are made to the same machine
Dual Connection Test

- Fully establish two TCP sessions (red and black)

Probing Host | Remote Host
Dual Connection Test

• Fully establish two TCP sessions (red and black)
• Send two sample packets: one in each connection
Dual Connection Test

- Fully establish two TCP sessions (red and black)
- Send two sample packets: one in each connection
- If no reordering
  - IPID of first response packet…

Probing Host  Remote Host

IPID n
Dual Connection Test

- Fully establish two TCP sessions (red and black)
- Send two sample packets: one in each connection
- If no reordering
  - IPID of first response packet, is strictly less than IPID of response packet
Dual Connection Test Pitfalls

• Connection assumption violations
  – **Load balancer** can direct two connections to different hosts

• IPID assumption violations
  – Random IPID values (e.g., OpenBSD)
  – Zero IPID after MTU discovery (e.g., Linux)
SYN Test

- Trick load balancers by starting “identical” connections
  - Appear to belong to same flow (but different seq #'s)
- Use TCP connection state machine to infer order
  - No assumptions about IPID

- Assumptions
  - Duplicate SYN’s with different seq cause ACK or RST packets
SYN Test

- Uses no pre-established sessions
**SYN Test**

- Uses no pre-established sessions
- Send two SYN packets to remote host
  - Different starting sequence number
  - Other than that, identical
SYN Test

- Uses no pre-established sessions
- Send two SYN packets to remote host
  - Different starting sequence number
  - Other than that, identical
- First received packet will generate a SYN+ACK
SYN Test

- Uses no pre-established sessions
- Send two SYN packets to remote host
  - Different starting sequence number
  - Other than that, identical
- First received packet will generate a SYN+ACK
- Other packet causes a RST or ACK
SYN Test Pitfalls

• SYN behavior assumption violations
  – Poorly understood/implemented part of spec.
  – Some TCP stacks send SYN+ACK or nothing in response to a bad duplicate SYN

• A series of SYN-based probes may be interpreted as a DoS attack
  – Implementation is good about cleaning up state
Implementation

• User-level subset of TCP stack
  – Shared origin w/ Sting, TBit, Sprobe and Alpine
  – Raw socket for sending frames
  – Packet filters (via libpcap) to capture response
  – Firewall filters to prevent host OS from seeing response
  – Detect assumption failures

• Runs on stock FreeBSD and Linux
Validation

• Controlled
  – Added reordering to FreeBSD Dummynet
  – Independently varied forward and reverse reordering
  – Match between network trace and reports from tool

• Experimental
  – Probed 50 hosts over 20 days with all tests
  – Each host probed approx. every 30 minutes
  – Probe results similar for hosts across tests (where different tests were compatible)
Observations (1)

- Significant reordering seen on some paths

www.apple.com

% reordering

hour

forward
Observations (2)

- Reordering can be highly asymmetric
Observations (3)

- Small changes in packet spacing can have large changes on reordering (on same path)
Conclusion

• We can measure unidirectional reordering from a single endpoint
• This matters
  – Reordering does happen
  – Asymmetry is common on reordered paths

• We still need a precise metric for reordering
  – Results currently not comparable between studies

• Source code will be available shortly at:
  http://ramp.ucsd.edu/reorder