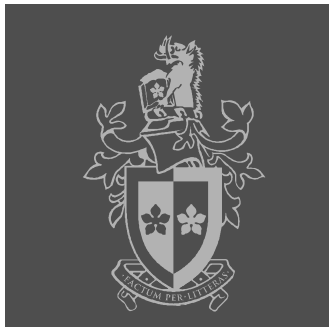
SWINBURNE

CENTRE FOR ADVANCED INTERNET ARCHITECTURES

Passive TCP Stream Estimation of RTT and Jitter Parameters

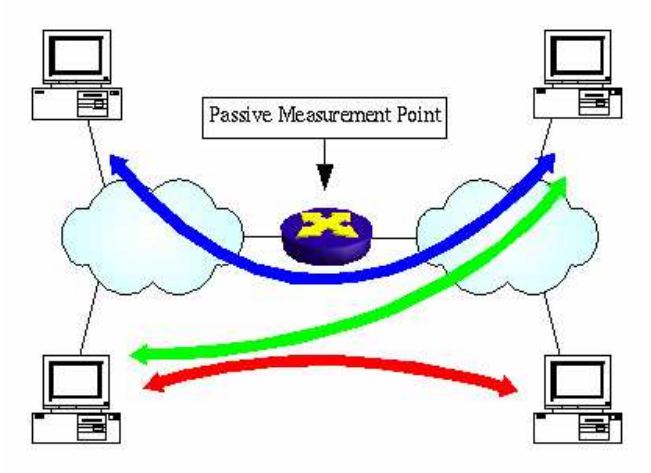
Jason But
Urs Keller
Grenville Armitage
David Kennedy



Outline

- Passive Monitoring of TCP
- RTT and Jitter Estimation algorithm
- Accuracy of Algorithm
  - Estimates under configured network conditions
  - Estimates in a congested network
  - Comparison against the end host TCP RTT estimate

Passive Monitoring of TCP



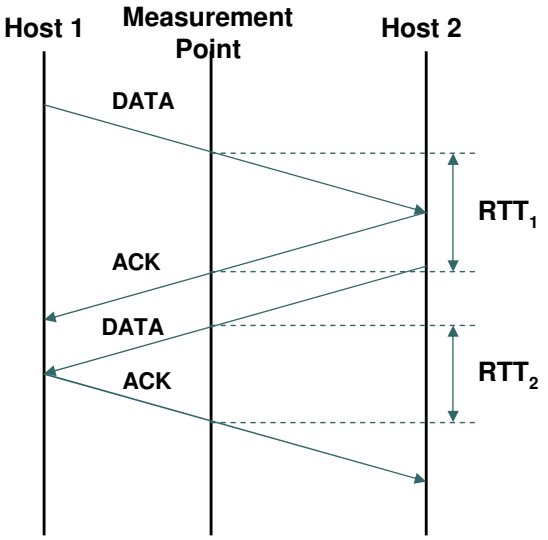
- Allows us to monitor more streams
- Can measure in the network core
- No requirements on end-clients

Passive Monitoring of TCP

- Issues with trying to estimate TCP RTT and Jitter when not at an endhost
  - Packets can be dropped/lost
  - Packets can be retransmitted
  - Witnessed packets may not be causal
- Existing approaches
  - SYN-ACK[1] Estimation, Slow Start[1] Estimation, TStat[2] – Not applicable for stream duration
  - Jaiswal et al[3]. – Can over estimate RTT, no Jitter estimates

1. H. Jiang and C. Dovrolis *"Passive Estimation of TCP Round-Trip Times"*, ACM SIGCOMM Computer Communications Review, vol. 32 no. 3, 2002, pp. 75-88
2. TSTAT – TCP Statistic and Analysis Software Tool, Politecnico di Torino, http://tstat.tlc.polito.it
3. S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley *"Inferring TCP Connection Characteristics Through Passive Measurements"*, Proceedings of IEEE InfoCom2004, March 2004

RTT and Jitter Estimation




```
ProcessDataPacket:
  input:
     timestamp, seqno

  if seen_before seqno calcRTT = false
  store(timestamp, packet.seqno)

ProcessAcknowledgement:
  input:
     timestamp, ackno

  if no stored values return
  if no stored sequence numbers < ackno return

  ts = max_stored_timestamp(stored.seqno < ackno)
  remove_stored_info(stored.seqno < ackno)

  RTTsample = timestamp - ts;

  if calcRTT = false
     if no_retransmitted_packets_stored
       calcRTT = true
     return

  if RTT > 0
     RTT = (7 * RTT + RTTsample) / 8
     Jittersample = Abs(RTT - RTTsample)
     if Jitter > 0
       Jitter = (7 * Jitter + Jittersample) / 8
     else
       Jitter = Jittersample
  else
     RTT = RTTsample
```


RTT and Jitter Estimation

- Define TCP Recovery state
  - Enter state if we witness out of order data packets or retransmitted packets
  - Leave state when all re-transmitted data packets have been acknowledged
  - No estimates made while in TCP Recovery
- RTT samples smoothed using Jacobsons algorithm
- Jitter Sample – Difference between the current RTT sample and the current (smoothed) RTT estimate
- Jitter samples smoothed using Jacobsons algorithm

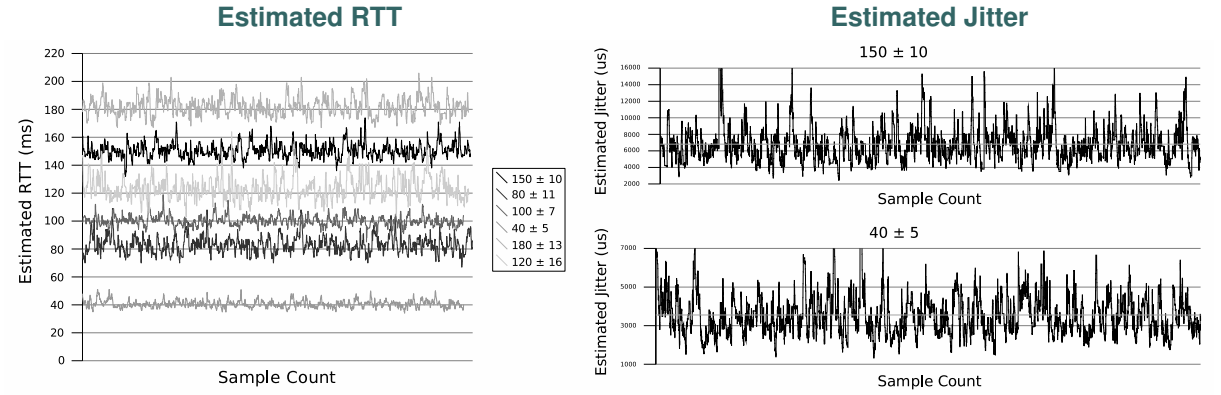
Algorithm Accuracy

- Do we generate enough samples to obtain a suitable running estimate of RTT and Jitter?
- How well do RTT and Jitter estimates match configured network delay and network delay variation?
- How well do RTT estimates match the TCP RTT estimations in the end hosts?

Comparison with End Host RTT

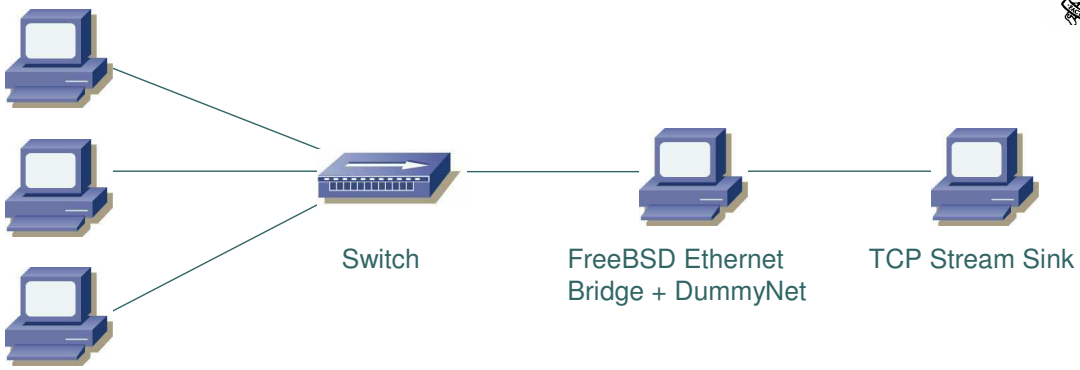
- Traffic captured on bridge
  - TCPdump on Linux does not store packets in chronological order
- NISTNet Router
  - Programmed with configured mean and standard deviation delay
  - Correlation factor used to minimise packet re-ordering

Configured Network Delays



- Estimated RTT and Jitter mean value accurate when compared against configured values
- Degree of fluctuation relative to configured jitter

Comparison with End Host RTT

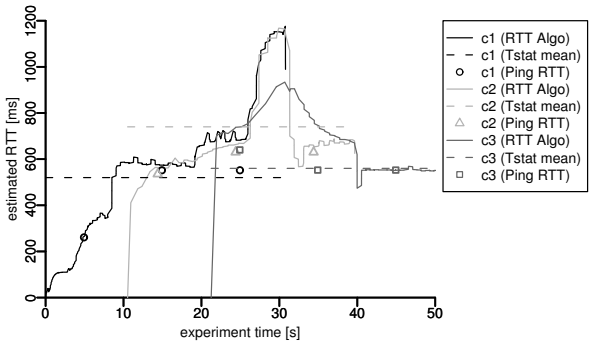


TCP Stream Sources

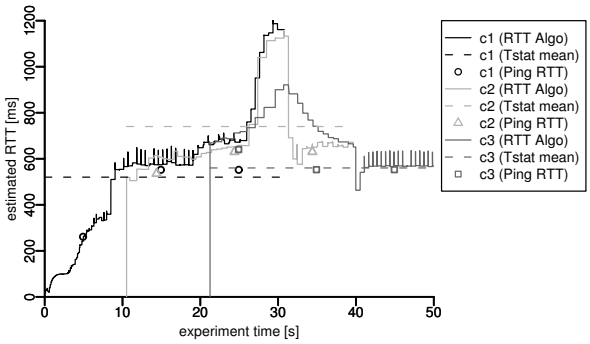
- Three TCP Streams
  - Stream duration ~30s
  - Each stream started at 10s intervals
- DummyNet Bridge
  - Configured delay 75ms
  - Configured throughput 500kbps

Comparison with End Host RTT

Algorithm estimated RTT

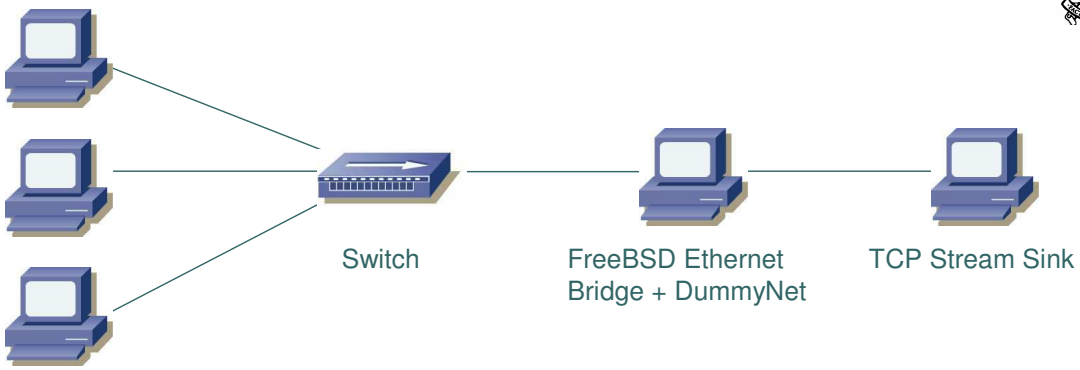


RTT as estimated by end host



- For further comparison
  - TSTAT calculated mean for each flow
  - RTT as measured by ping
  - Same values on both graphs

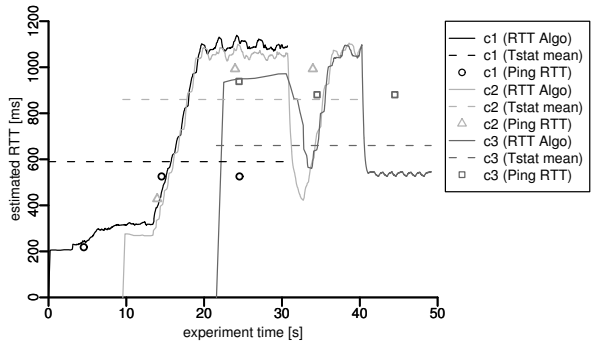
Comparison with End Host RTT



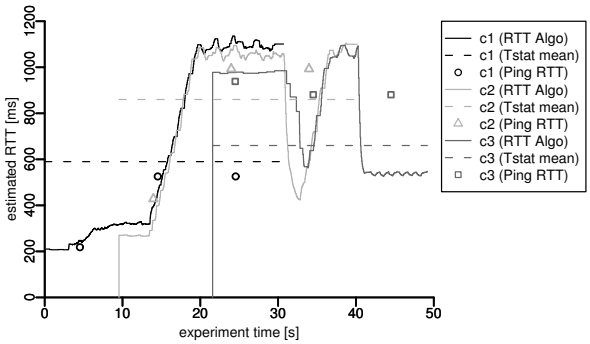
- Same experimental setup
- DummyNet configured with a different delay for each TCP flow – 100ms, 150ms, 200ms

Comparison with End Host RTT

Algorithm estimated RTT



RTT as estimated by end host



- Again the estimate matches the TCP stack estimate
- Single value calculated by TSTAT not representative for the duration of flow

Conclusions

- Passive RTT estimation is an inaccurate process
- We present a new continuous RTT and Jitter estimation algorithm
  - Jitter estimation algorithm can be used on any sequence of RTT estimates generated by another algorithm
- Accurate RTT and Jitter estimation under test conditions of artificially induced jitter
- Under congested network conditions
  - Algorithm tracks the RTT estimate in the TCP endhost stack accurately
- Running estimates superior to output generated by existing tools such as TSTAT (single mean and standard deviation)
  - More accurately reflect instantaneous network conditions
- Algorithm implemented in netsniff
  - http://caia.swin.edu.au/ice/tools/netsniff

Questions