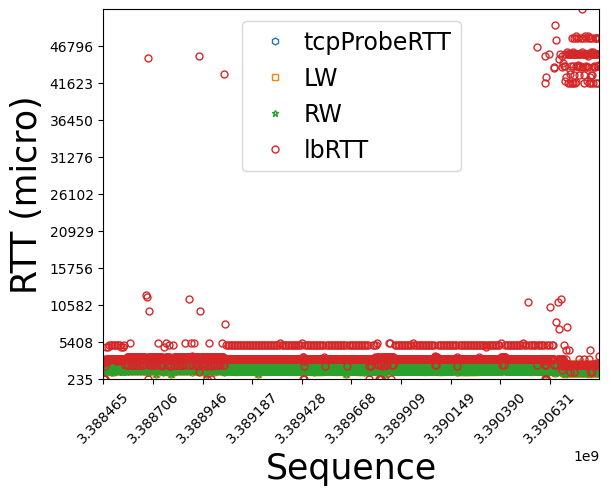
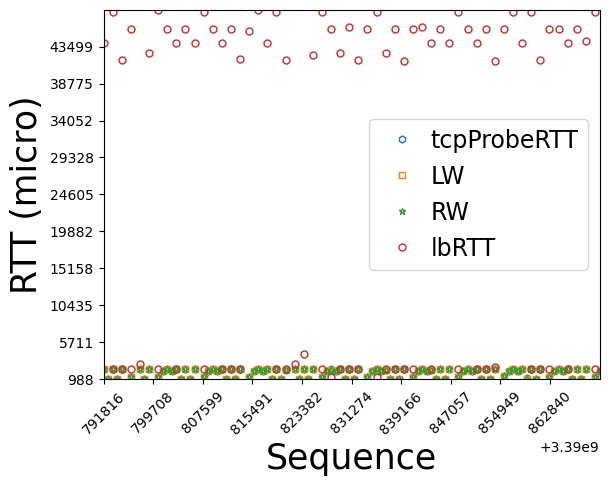
The project is focused on the evaluation of [] on cases like reorder, comparison of the algorithm with RTT measurements collected using linux tracepoint subsystem, and two other methods of determining RTT based on earliest and latest ACK on client side. The comparison is based on number of ssamples, mean RTT values. The short comings observed are also presented with reasoning.

**Case 1 Reorder**

The following graph shows the comparison of RTT samples collected in the case of packet reordering. Reordering of packets is done by using reorder feature available with qdisc. The setting is to reorder every 5th packet i.e every 5th packet will be transmitted immediately and the remaining 4 packets are delayed by queueing it for 1 ms. This is done on an intermediate box in the client-server path. The command used to achieve the same is

tc qdisc add dev enp9s4f1 root netem reorder 100 gap 5 delay 1ms





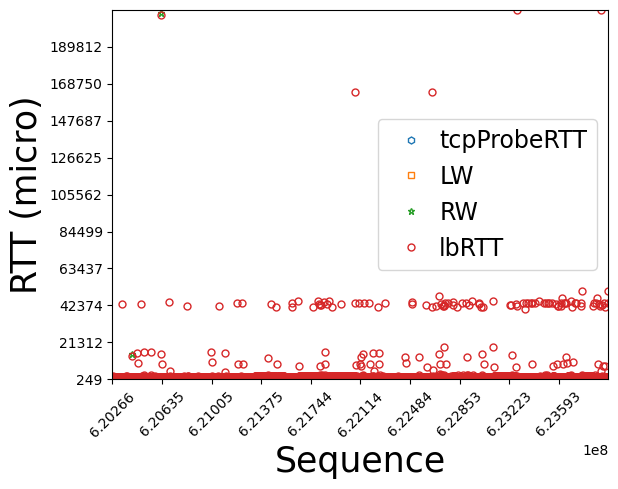
Observations

1. ~2ms difference from RTT. 1-2 ms application latency on server. 300-500 micro client think time.
2. High threshold causes accumulation - due to lack of samples in an epoch
3. Nagle + delayed ack causes bands around 45 ms

**Case 2 Retransmission**

The following graph shows the comparison of RTT samples collected in the case of packet loss. Packet loss is achieved by dropping the response packets from server in an intermediate node. We use the loss property of Qdisc to achieve the same. The setting is to drop .1% of packet in the client to server direction of traffic. The command used to achieve this is

tc qdisc add dev enp9s4f0 root netem loss 0.1%



Chart

Description automatically generated

Observations

1. Scatter above the lowest band is due to loss (retransmission timeout around 12-16ms). 2 segment sent. After 14-16ms stack retransmit last segment. Sometimes further dup ACK leads to immediate retransmit of 1st segment, otherwise after another timeout of 200ms (frame 1053).
2. Nagle + delayed ACK leads to interpkt gaps of more than 40ms - 2 layer
3. When 2. and retransmission timeout happens together (40ms + 160 ms)

**Case 3 Caida**

The following graph shows the comparison of RTT samples collected from Caida traces. The total number of connections recorded in the caida trace where <>, but in order to evaluate the algorith we need to have a ground truth measurement. For finding the ground truth measurement we need to filter connections which has packets in both direction of traffic and also data transmission on both sides. The number of such connections available where <>.

PS: The requirement of data transmission is to avoid the ambiguity in RTT measurement based on sequence number and ACK. Data transmission on one side makes it difficult to determine the caulsality of transmission on that side just based on ACK from the other side.

Steps

Make a topology with 3 boxes (B1, B2, B3). B2 is connected to both B1 and B3. B1 and B3 traffic for each other flows through B2.

Case 1

1. Configure B2 to reorder packets from B3 to B1

tc qdisc add dev enp9s4f1 root netem reorder 100 gap 5 delay 1 ms

< enp9s4f1> is interface connecting B2 to B1

1. Configure tcpProbe tracepoint on client to get the dump for traffic

echo 1 > /sys/kernel/debug/tracing/events/tcp/tcp\_probe/enable

1. Start the server script server.py on B3
2. Start packet captures on all 3 boxes. Note: On middle box add an extra filter to capture only packets destined to server. This capture will be fed to LB algorithm
3. Start the client script client.py on B1
4. Get tcpProbe output for client
   1. cat /sys/kernel/debug/tracing/trace | grep dest=ip:port > file
5. Run the parseInput.py on file obtained on step 6
6. Run the parseClient.py with client side packet capture
   1. python3 parseClient.py --inputFile /users/bvs17/ testc.pcap
7. Run the parseLB.py with middle box packet capture
   1. python3 parseLB.py --fwdFile /users/bvs17/forward.pcap --srcIP <clientIP> --srcPort <clientPort> --dstIP <serverIP> --dstPort <serverPort>
8. Compare the results

Case 2

1. Configure B2 to drop packets from B1 to B3

tc qdisc add dev <interface> root netem loss 0.1%

<interface> connecting B3 to B2

1. Configure tcpProbe tracepoint on client to get the dump for traffic

echo 1 > /sys/kernel/debug/tracing/events/tcp/tcp\_probe/enable

1. Start the server script server.py on B3
2. Start packet captures on all 3 boxes. Note: On middle box add an extra filter to capture only packets destined to server. This capture will be fed to LB algorithm
3. Start the client script client.py on B1
4. Get tcpProbe output for client
   1. cat /sys/kernel/debug/tracing/trace | grep dest=ip:port > file
5. Run the parseInput.py on file obtained on step 6
6. Run the parseClient.py with client side packet capture
   1. python3 parseClient.py --inputFile /users/bvs17/ testc.pcap
7. Run the parseLB.py with middle box packet capture
   1. python3 parseLB.py --fwdFile /users/bvs17/forward.pcap --srcIP <clientIP> --srcPort <clientPort> --dstIP <serverIP> --dstPort <serverPort>
8. Compare the results

Case 3:

1. Run the parseCaida.py script on forward and reverse direction packet trace

python3 parseCaida.py --inputFile equinix-chicago.dirA.20160121-125911.UTC.anon.pcap --suffix A --fwd 1

python3 parseCaida.py --inputFile equinix-chicago.dirB.20160121-125911.UTC.anon.pcap --suffix B --fwd 0

1. Run generateGroundTruth.py. Generate caidaGround file.

python3 findConnections.py

python3 generateGroundTruth.py (input the parameters needed as prompted and directed)

1. Run parseLbCaida.py file on forward traffic as in Case 2 Step 9. Generates lbRTT file

python3 parseLBCaida.py --fwdFile sample2\_A.pcap --srcIP 153.193.136.80 --srcPort 443 --dstIP 43.147.252.137 --dstPort 44609

< sample2\_A.pcap> is filtered pcap of mentioned connection ID from caida trace of forward direction

1. Compare the results