TCP Retransmission Timeout Algorithm Using Weighted Medians

CSE322 NS3 Project Report

by

Ridwanul Hasan Tanvir

1705016

Supervised by: Syed Md. Mukit Rashid

Department of Computer Science and Engineering

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, BANGLADESH

February 2022

Table of Contents

1 Network Topology

In the cited paper a mobile ad hoc network (MANET) composed of 20 nodes is used. In our simulation we can vary the number of nodes [1].

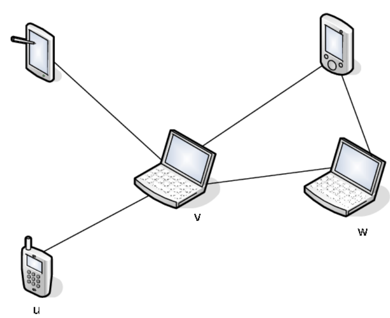


Figure 1.1: TaskB Mobile Ad hoc Network (MANET)

To compare the effect of RTT and RTO a wired topology is used (to be specific dumbbell-topology).

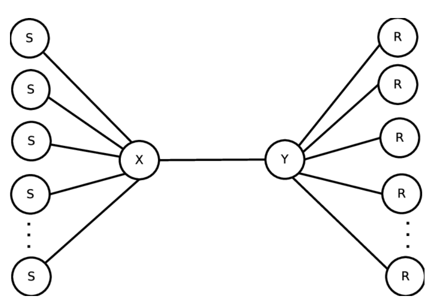


Figure 1.2: TaskB dumbbell-topology

For TaskA mobility model the same Manet topology is used.

2 Variation of Parameters

- number of nodes
- number of flow (a flow is represented by tuple (srcIp, srcPort, destIp, destPort))
- number of packets per second
- Speed of nodes (in case of having mobility)
- Coverage area (static nodes)

3 Overview Proposed Algorithm

Estimation of RTT

RTT - a TCP sender's estimation of the round-trip-time from Sender to Receiver

Let the RTTs of all the packets up to packet $i-1$ are observed, then the RTT estimate for packet i is given by

$$a_i = Median(A_1.a_{i-1}, B_1.m_{i-1}, B_2.m_{i-2}, ..., B_M.m_{i-M}) \tag{3.1}$$

where

a_i = RTT estimate for packet i = Calculated RTT

m_i = RTT observation for packet i = Actual RTT

M = number of previous RTT observations considered

In the paper for simulation purpose M = 5 has been used.

The correlation between RTT samples drops rapidly with lag, we utilize weights that emphasize the relative importance of samples with small lag. An exponential function has been used for this purpose.

$$A_1 = \beta = \frac{1}{2} \tag{3.2}$$

$$B_j = \alpha^{j-1} \quad where \quad \alpha = \frac{7}{8} \tag{3.3}$$

Experimentation shows that $\alpha = \frac{7}{8}$ and $\beta = \frac{1}{2}$ values and provide good performance for a wide range of network conditions.

Determination of RTO

The estimated RTT can be scaled, where the scale factor reflects the RTT variability. In this paper this approach is utilized in conjunction with the recursive WM RTT estimates which produces the best results.

To be proportional to the variability in the RTT, the scale factor λ can be expressed as

$$\lambda = 1 + \mu\zeta \tag{3.4}$$

where ζ = Mean Absolute Deviation About the Mean

$$\zeta = \frac{1}{M}\sum_{n=1}^{M} |m_i - \overline{m}| \tag{3.5}$$

where m_i = RTT observation for packet = Actual RTT

μ varies in the range [4.3,4.7]. This small range indicates that a fixed μ (for this paper 4.5) gives optimal results.

We multiply the RTT estimate by λ and set the product as the RTO,

$$RTO = \lambda a_i \tag{3.6}$$

4 Modifications in Simulator to implement Algorithm

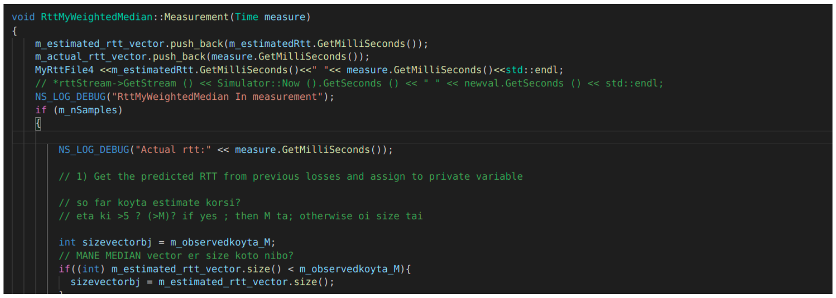
We have declared *RttMyWeightedMedian* Class to inherit the *RttEstimator* Class (Base class for all RTT Estimators). We implemented our own *GetTypeId* function as *RttEstimator* Class inherits *Object*. We included three additional attributes Alpha, Beta and m_observed. The main calculation is done in the *RttMyWeightedMedian::Measurement()* function.

In *TcpSocketBase* Class we need to edit RTO in three functions:

- EstimateRtt : this function is called by DoForwardUp() which is used to get a packet from L3. This is the real function to handle the incoming packet from lower layers.

- SendEmptyPacket: this is used to send an empty packet that carries a flag, e.g., ACK.

- NewAck: this is used to Update buffers w.r.t. ACK. Called by the ReceivedAck() when new ACK received and by ProcessSynRcvd() when the three-way handshake completed. This cancels retransmission timer and advances Tx window



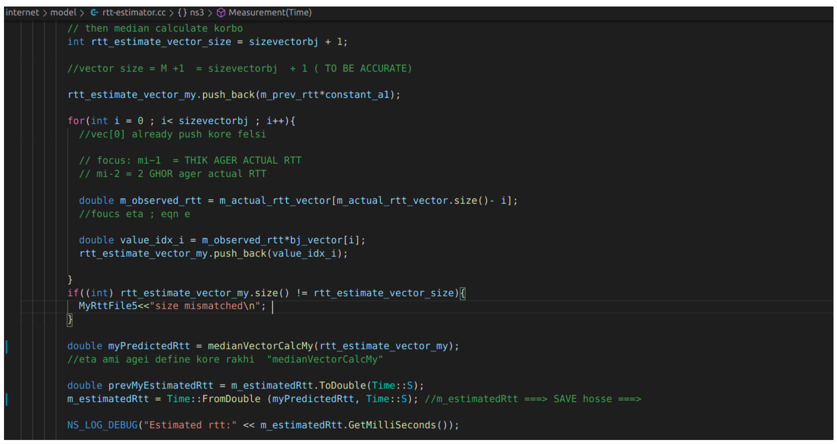
```
void RttMyWeightedMedian::Measurement(Time measure)
{
    m_estimated_rtt_vector.push_back(m_estimatedRtt.GetMilliSeconds());
    m_actual_rtt_vector.push_back(measure.GetMilliSeconds());
    MyRttFile4 <<m_estimatedRtt.GetMilliSeconds()<<" "<< measure.GetMilliSeconds()<<std::endl;
    // *rttStream->GetStream () << Simulator::Now ().GetSeconds () << " " << newval.GetSeconds () << std::endl;
    NS_LOG_DEBUG("RttMyWeightedMedian In measurement");
    if (m_nSamples)
    {

        NS_LOG_DEBUG("Actual rtt:" << measure.GetMilliSeconds());

        // 1) Get the predicted RTT from previous losses and assign to private variable

        // so far koyta estimate korsi?
        // eta ki >5 ? (>M)? if yes ; then M ta; otherwise oi size tai

        int sizevectorbj = m_observedkoyta_M;
        // MANE MEDIAN vector er size koto nibo?
        if((int) m_estimated_rtt_vector.size() < m_observedkoyta_M){
          sizevectorbj = m_estimated_rtt_vector.size();
```



```
internet > model > rtt-estimator.cc > {} ns3 > Measurement(Time)
          // then median calculate korbo
          int rtt_estimate_vector_size = sizevectorbj + 1;

          //vector size = M +1  = sizevectorbj  + 1 ( TO BE ACCURATE)

          rtt_estimate_vector_my.push_back(m_prev_rtt*constant_a1);

          for(int i = 0 ; i< sizevectorbj ; i++){
            //vec[0] already push kore felsi

            // focus: mi-1  = THIK AGER ACTUAL RTT
            // mi-2 = 2 GHOR ager actual RTT

            double m_observed_rtt = m_actual_rtt_vector[m_actual_rtt_vector.size()- i];
            //foucs eta ; eqn e

            double value_idx_i = m_observed_rtt*bj_vector[i];
            rtt_estimate_vector_my.push_back(value_idx_i);

          }
          if((int) rtt_estimate_vector_my.size() != rtt_estimate_vector_size){
            MyRttFile5<<"size mismatched\n";
          }

          double myPredictedRtt = medianVectorCalcMy(rtt_estimate_vector_my);
          //eta ami agei define kore rakhi  "medianVectorCalcMy"

          double prevMyEstimatedRtt = m_estimatedRtt.ToDouble(Time::S);
          m_estimatedRtt = Time::FromDouble (myPredictedRtt, Time::S); //m_estimatedRtt ===> SAVE hosse ===>

          NS_LOG_DEBUG("Estimated rtt:" << m_estimatedRtt.GetMilliSeconds());
```

Figure 4.1: RttMyWeightedMedian::Measurement()

Files in ns3 to update:

- rtt-estimator.h
- rtt-estimator.cc
- tcp-socket-base.h
- tcp-socket-base.cc

Included two header files for easier calculation:

- rtt-estimator1.h
- rto-estimator1.h

```cpp
void
TcpSocketBase::EstimateRtt (const TcpHeader& tcpHeader)
{
  SequenceNumber32 ackSeq = tcpHeader.GetAckNumber ();
  Time m = Time (0.0);

  // An ack has been received, calculate rtt and log this measurement
  // Note we use a linear search (O(n)) for this since for the common
  // case the ack'ed packet will be at the head of the list

  if (!m.IsZero ())
    {
      m_rtt->Measurement (m);                // Log the measurement
      TcpSocketFile1<<"CheckIfRttWeightedMedian="<<CheckIfRttWeightedMedian<<"\n";
      // why 5?
      // for now 5 = weighted median dhore nisi; dhoro je RTO ber korar 5 ta method ase; eta 5th method
      if(CheckIfRttWeightedMedian==5){
        // std::cout<<"CheckIfRttWeightedMedian==5 found\n";
        std::vector<double> m_actual_rtt_vect1 = m_rtt->GetVectorRttActual();
        // std::vector<double> m_estimated_rtt_vect1 = m_rtt->GetVectorRttEstimate();

        double zeta1_paper = calculateZeta(m_actual_rtt_vect1);
        double lambda_paper = 1.0 + 4.5* zeta1_paper;

        // double oldRttVar = m_estimatedVariation.ToDouble(Time::S);
        double last_estimated_rtt1 = m_rtt->GetEstimate ().ToDouble(Time::S);

        // //basically last RTT JETA AMI ESTIMATE korsi
        double paper_rto = lambda_paper*last_estimated_rtt1;

        m_rto = Max(Time::FromDouble (paper_rto, Time::S), m_minRto);
```

```cpp
> internet > model > tcp-socket-base.cc > {} ns3 > NewAck(SequenceNumber32 const &, bool)
void
TcpSocketBase::NewAck (SequenceNumber32 const& ack, bool resetRTO)
{
  NS_LOG_FUNCTION (this << ack);

  // Reset the data retransmission count. We got a new ACK!
  m_dataRetrCount = m_dataRetries;

  if (m_state != SYN_RCVD && resetRTO)
    { // Set RTO unless the ACK is received in SYN_RCVD state
      NS_LOG_LOGIC (this << " Cancelled ReTxTimeout event which was set to expire at " <<
                    (Simulator::Now () + Simulator::GetDelayLeft (m_retxEvent)).GetSeconds ());
      m_retxEvent.Cancel ();
      // On receiving a "New" ack we restart retransmission timer .. RFC 6298
      // RFC 6298, clause 2.4
      TcpSocketFile1<<"CheckIfRttWeightedMedian="<<CheckIfRttWeightedMedian<<"\n";
      // why 5?
      // for now 5 = weighted median dhore nisi; dhoro je RTO ber korar 5 ta method ase; eta 5th method
      if(CheckIfRttWeightedMedian==5){
        // std::cout<<"CheckIfRttWeightedMedian==5 found\n";
        std::vector<double> m_actual_rtt_vect1 = m_rtt->GetVectorRttActual();
        // std::vector<double> m_estimated_rtt_vect1 = m_rtt->GetVectorRttEstimate();

        double zeta1_paper = calculateZeta(m_actual_rtt_vect1);
        double lambda_paper = 1.0 + 4.5* zeta1_paper;

        if(lambda_paper<0){
          TcpSocketFile1<<"lambda_paper<0 found\n";
        }
        // double oldRttVar = m_estimatedVariation.ToDouble(Time::S);
        double last_estimated_rtt1 = m_rtt->GetEstimate ().ToDouble(Time::S);

```

Figure 4.2: TcpSocketBase RTO update

7 Results Explanation Task B

7.1 Congestion Window

CWND - a TCP sender's congestion window, which is the computed number of bytes that the sender is allowed to send in the next RTT

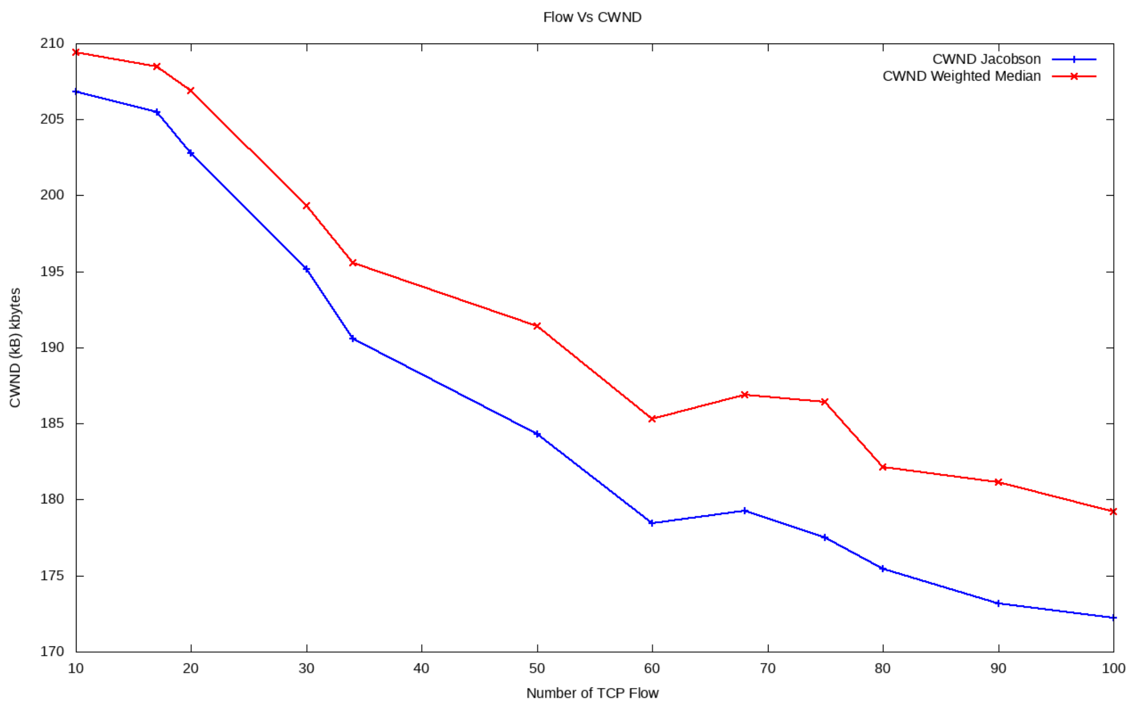


Figure 7.1: Flow Vs Congestion Window

This simulation is done in a Mobile Ad hoc Network (MANET) of 20 nodes.

Observation

- Weighted Median Algorthm the RTT predictions and hence avoiding unnecessary retransmissions.

- As a result, TCP's congestion window (cwnd) is higher on average

- The gap between Jacobson and Weighted Median CWND increases with increasing congestion conditions (when Number of flow is higher).

- In case of Jacobson Algo when number of flow increases CWND size decreases rapidly. But in Weighted Median Algo we took into consideration previous M RTT observed values and estimated RTT can be scaled, where the scale factor reflects the RTT variability.

- So it is able to shield TCP's congestion control from wide RTT fluctuations.

7.2 *Mean RTT Error*

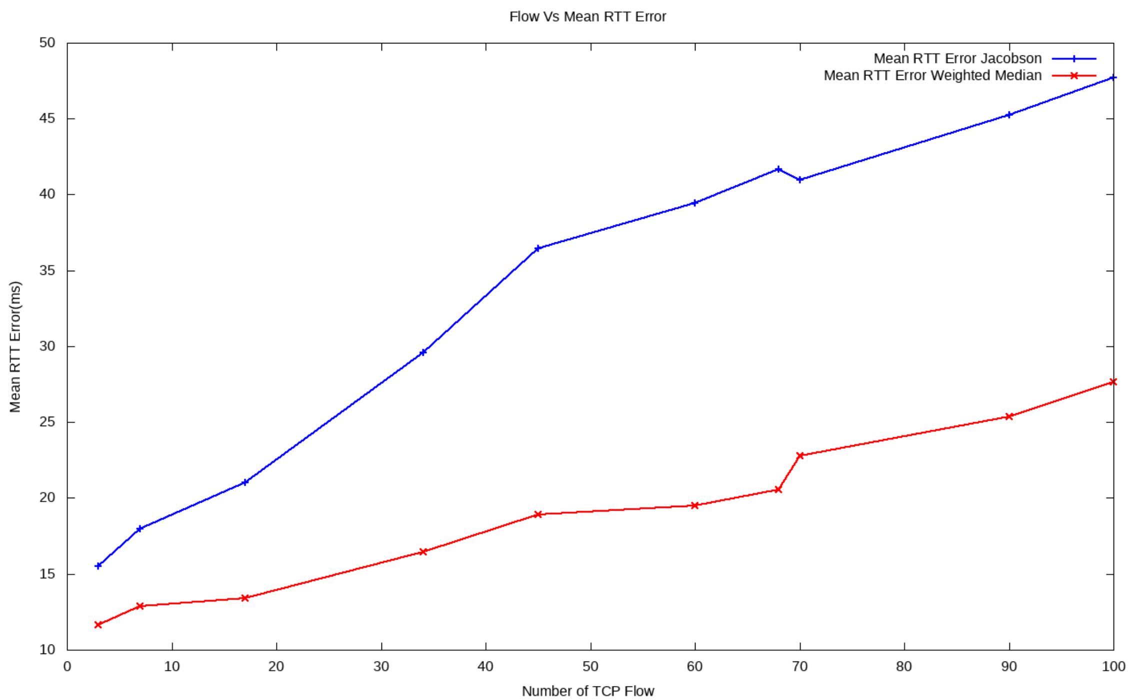


Figure 7.2: Flow Vs Mean RTT Error

Observation

- Mean RTT Error is lower than Jacobson Algorithm.
- In Weighted Median Algo we took into consideration previous M RTT observed values and estimated RTT can be scaled, where the scale factor reflects the RTT variability.
- So RTT Weighted Median Algorithm RTT estimation accuracy considerably as the load in the network increase (when Number of flow is higher).

7.3 Retransmitted Packet Compare

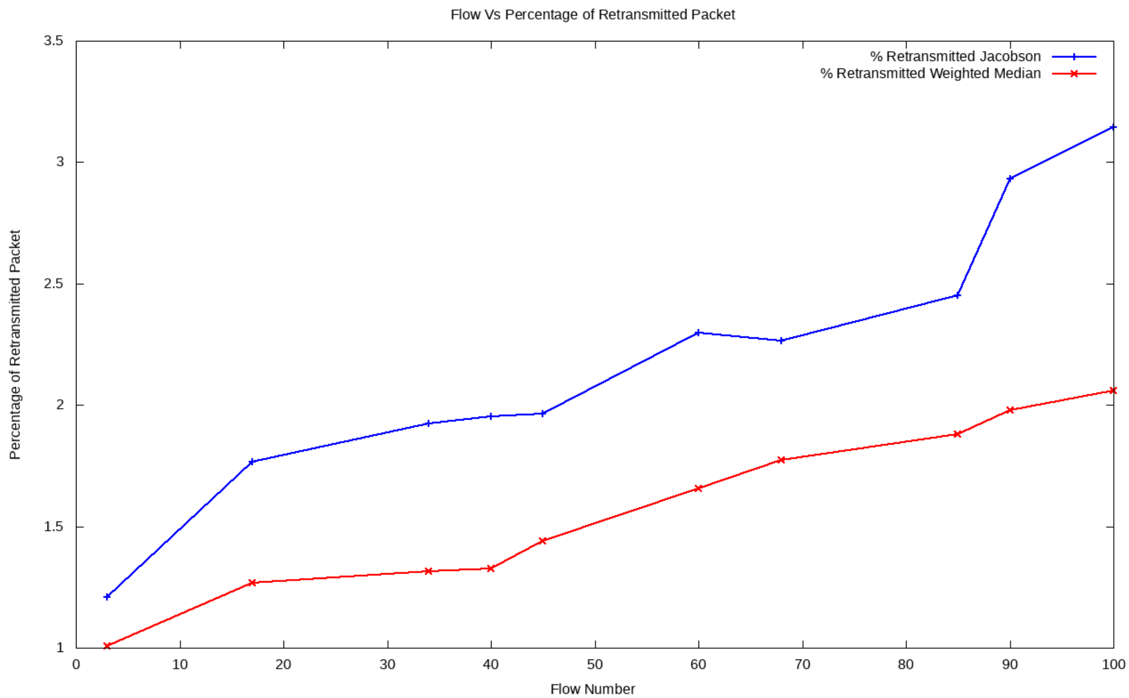


Figure 7.3: Flow Vs Percentage Retransmitted Packet

Observation

- RTT Weighted Median Algorithm RTT estimation accuracy considerably as the load in the network increases(when Number of flow is higher).

- This leads to more accurate setting of the RTO timer, which, in turn, reduces the relative number of packets retransmitted significantly, especially under heavy traffic conditions.

- Retransmissions are reduced substantially without decreasing the number of packets sent.

7.4 RTT RTO Compare Wired Topology

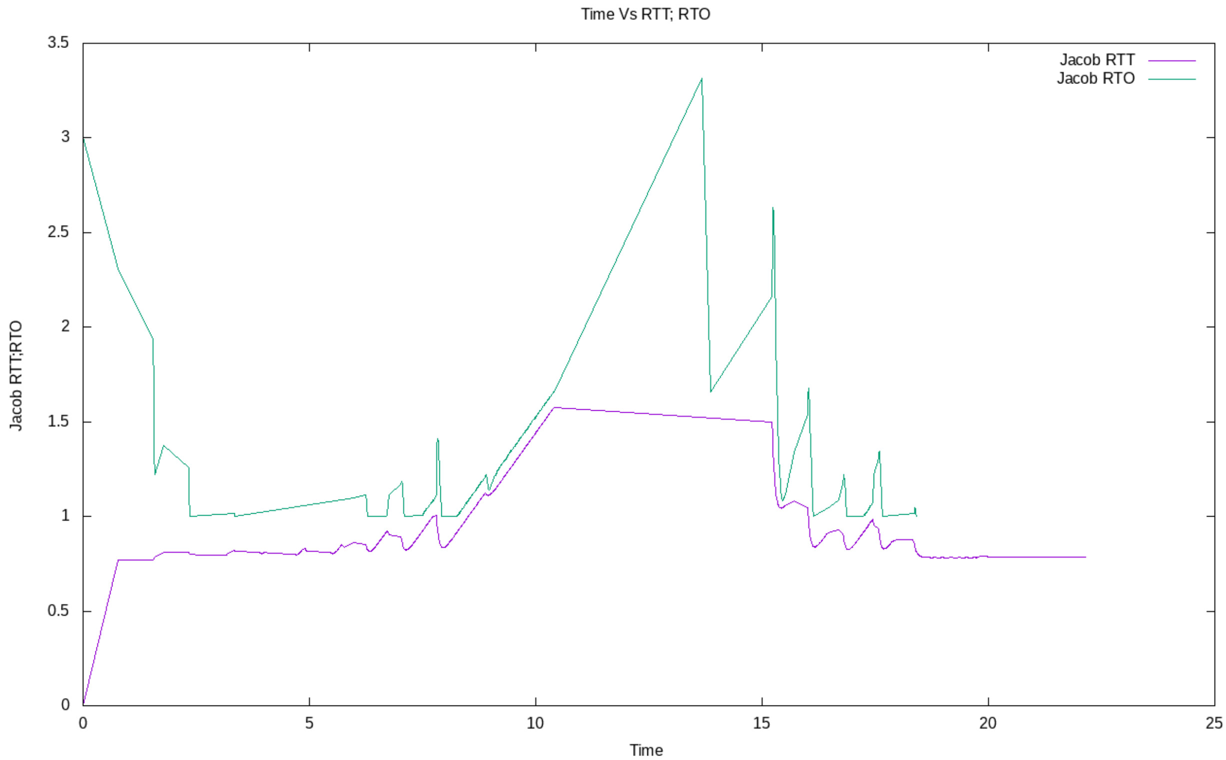


Figure 7.4: Time vs RTT, RTO for Jacobson Algorithm

Observation

- For RTT Jacobson Algorithm sudden increase in RTT results in huge discripancy for RTO calculation. As a result TCP Retransmission Timeout becomes large. So we have to wait for a long time to retransmit a packet. (which is much larger than the actual RTT).

- RTT Weighted Median Algorithm RTT even if sudden increase in RTT the RTO is close. [2]

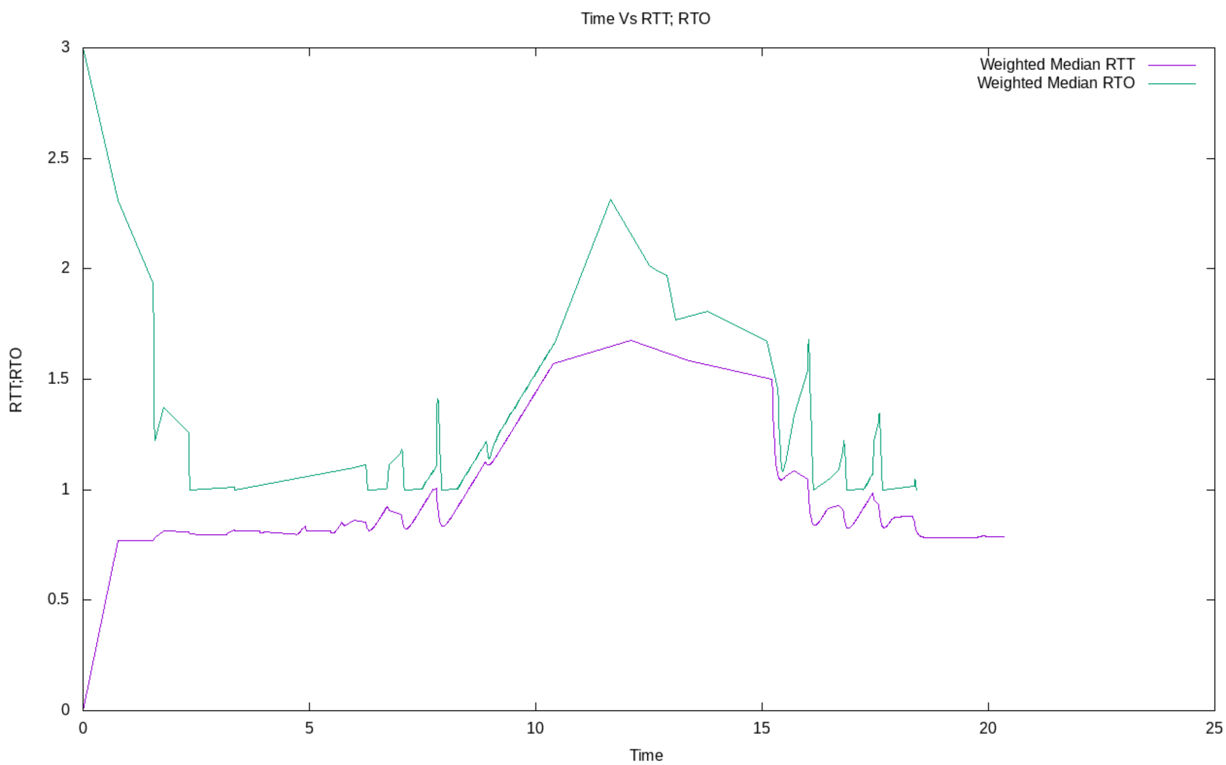


Figure 7.5: Time vs RTT, RTO for Weighted Median Algorithm

References

[1] B. A. Arouche Nunes, K. Veenstra, W. Ballenthin, S. Lukin, and K. Obraczka, "A machine learning framework for tcp round-trip time estimation," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, pp. 1–22, 2014.

[2] L. Ma, G. R. Arce, and K. E. Barner, "Tcp retransmission timeout algorithm using weighted medians," *IEEE Signal Processing Letters*, vol. 11, no. 6, pp. 569–572, 2004.