#1

Network topologies under simulation:

802.11 wireless mobile

802.15.4 wireless mobile

#2

Parameters under variation

The number of nodes, flows, packets and node speed were the parameters that were varied during the simulation. The different values of these metrics for which the code was tested are:

Node count: 20, 40 (default), 60, 80, 100

Flow count: 10, 20 (default), 30, 40, 50

Packet count: 100, 200 (default), 300, 400, 500

Speed: 5, 10 (default), 15, 20, 25

#3

Modifications made in the simulator

ns-2.35/tcp/tcp.cc

```
/* my code starts */
// bool use_rto_modified = false;
int data_taken = 0;

int A = 12;
int B[5] = {23, 20, 17, 15, 13};
int decimal_B[5] = {1, 0.875, 0.77, 0.67, 0.59};
//vector<int> medianParams;
vector<double> previousRTO;
//deque<int> previousRTO;

int my_rtt_estimate(int prev_srtt_)
{
    vector<int> medianParams;
    deque<int>::iterator it;
    //queue<int> tempQueue = previousRTO;
    for(int i=0; i<A; i++)
        medianParams.pb(prev_srtt_);

    for(int i=0; i<previousRTO.size(); i++)
    {
        //int data = *it;
        for(int j=0; j<B[i]; j++)
            medianParams.pb(previousRTO[i]);
    }

    sort(medianParams.begin(), medianParams.end());
    return (medianParams[49] + medianParams[50])/2; // a: rtt_estimate
}

int my_rtt_var()
{
    int expected_rto = 0;
    for(int i=0; i<previousRTO.size(); i++)
        expected_rto += (previousRTO[i] * decimal_B[i]);
    int temp = 0;
    for(int i=0; i<previousRTO.size(); i++)
        temp += ((previousRTO[i] - expected_rto) * decimal_B[i]);
    return temp/expected_rto;
}

int my_insert(double value) { previousRTO.insert(previousRTO.begin(), value); }

int my_remove() { previousRTO.pop_back(); }
```

```
646
647  void TcpAgent::my_rtt_update_modified()
648  {
649      // printf("here\n");
650      t_srtt_.val_ = my_rtt_estimate(t_srtt_.val_);
651      t_rttvar_.val_ = my_rtt_var();
652
653      double miu = 4.5;
654      double lambda = 1 + miu * t_rttvar_.val_;
655      t_rtxcur_ = lambda * t_srtt_.val_;
656
657      my_insert(t_rtxcur_);
658      my_remove();
659  }
660  /* my code ends */
661
662  /* This has been modified to use the tahoe code. */
663  void TcpAgent::rtt_update(double tao)
664  {
665      // printf("here 2\n");
666      double now = Scheduler::instance().clock();
667      if (ts_option_)
668          t_rtt_ = int(tao /tcp_tick_ + 0.5);
669      else {
670          double sendtime = now - tao;
671          sendtime += boot_time_;
672          double tickoff = fmod(sendtime, tcp_tick_);
673          t_rtt_ = int((tao + tickoff) / tcp_tick_);
674      }
675      if (t_rtt_ < 1)
676          t_rtt_ = 1;
677
678      // my code starts
679      if(use_rto_modified && data_taken > 4)
680      {
681          my_rtt_update_modified();
682          return;
683      }
684      // my code ends
```

ns-2.35/tcp/tcp.h

```
174  class TcpAgent : public Agent {
175      friend class XcpEndsys;
176  public:
177      int use_rto_modified;
```

```
288      virtual void rtt_update(double tao);    /* update RTT estimate */
289      virtual void my_rtt_update_modified(); /* update modified RTO by me */
```

```
        qs_window_(0), qs_cwnd_(0), frto_(0), use_rto_modified(1)
{
    bind("use_rto_modified", &use_rto_modified);
```

ns-2.35/tcl/lib/ns-default.tcl

```
Agent/TCP set use_rto_modified 0
```

ns-allinone-2.35/tclcl-1.20/tracedvar.h

```
    int val_;       // value stored by this trace variable; made public by me
protected:
    virtual void assign(const int newval);
    // this line was shifted to line 129
};
```

#4

Summary findings

Two unfortunate events occurred during the simulation. The assigned topologies (802.11 & 802.15.4 static) could not be executed without errors. Hence, the forced choice was to move to mobile nodes, for which the light of hope was there. But yet, another unfortunate event during the simulation was that the modified algo threw floating point exception for some inputs. So, results for the other inputs are analyzed here.

Varying Nodes

In case of 802.15.4, the modified algo failed to outsmart the legacy algo for all metrics. But for 802.11, it succeeded over the existing approach.

Varying Flows

The modified algo performed slightly better in many aspects in case of 802.11. But for 802.15.4, none could demonstrate a visible domination, moreover, they did unpredictable behavior too.

Varying Packets

In case of 802.15.4, there was a race between the two algos for performance on the different metrics. However, a surprising fact to mention, there were two flat lines in all the graphs which indicates varying the number of facts did not have any impact on any of the metrics.

Varying Speed

Well, the surprising fact that was mentioned in the last point occurred here too. The comments are more or less the same as the previous point.

The modified algo performed slightly better in many aspects in case of 802.11. But for 802.15.4, none could demonstrate a visible domination, moreover, they did unpredictable behavior too.