CSE 322 PROJECT

FINAL REPORT

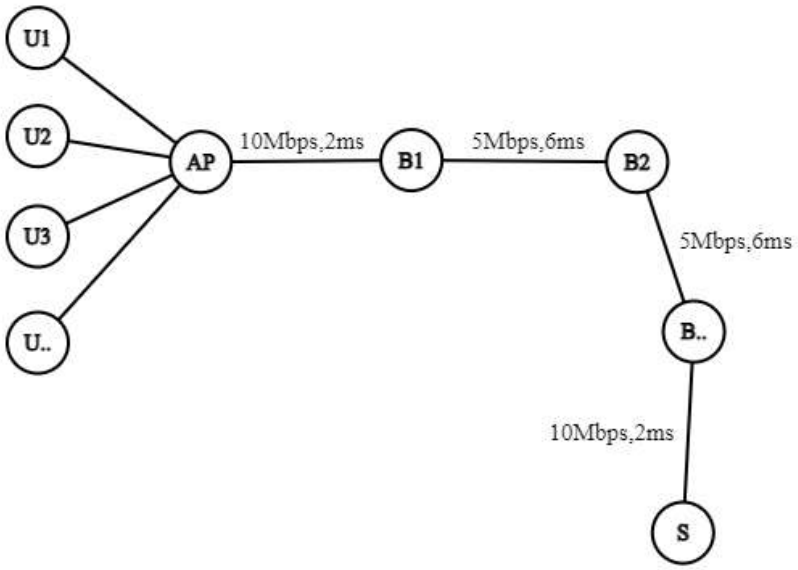
Submitted by: Iftekhar Hakim Kaowsar

ID: 1705045

Task A:

Simulating Wireless high-rate 802.11 (mobile) network.

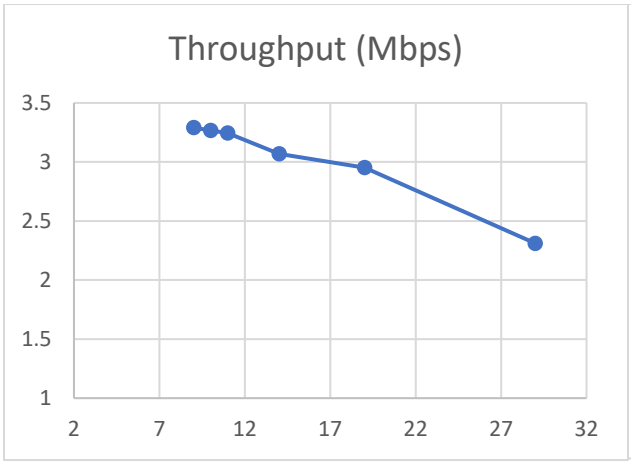
Topology:

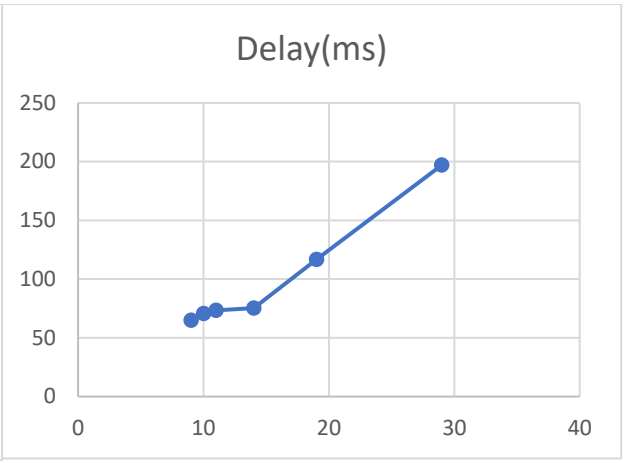


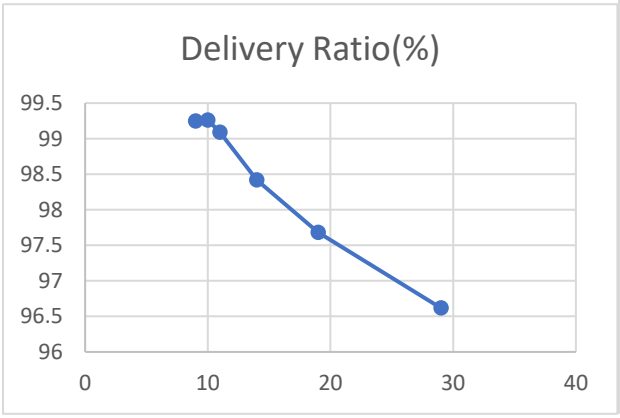
Number of bottleneck nodes, flows, speed of user nodes and packet per second have been varied.

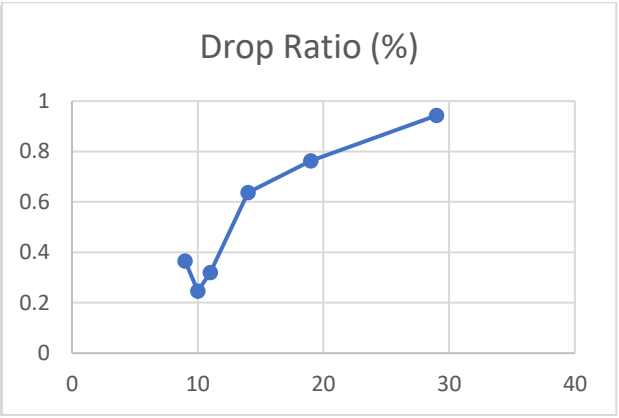
Results:

Varying number of bottleneck nodes or total nodes:

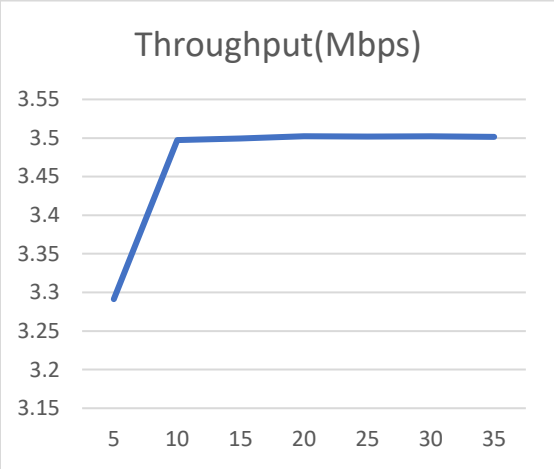


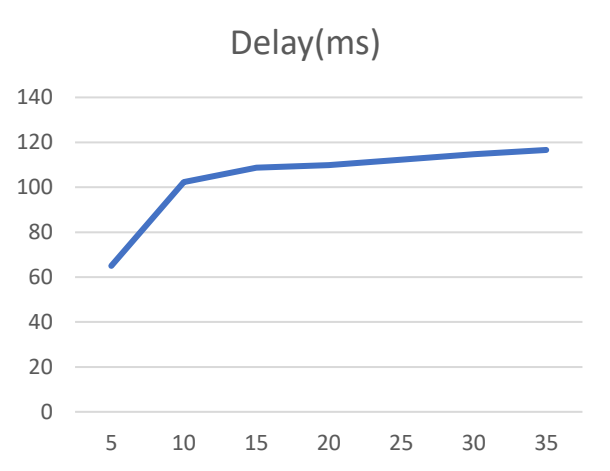


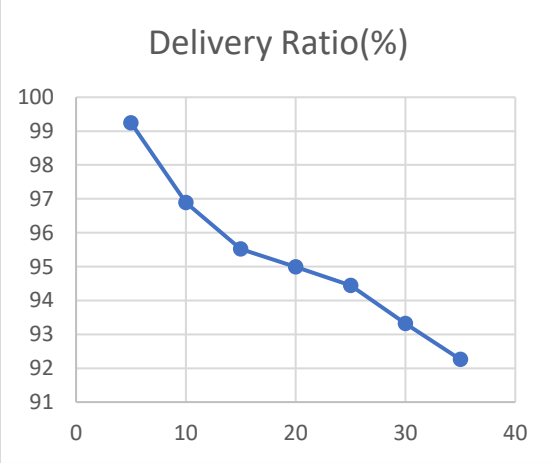


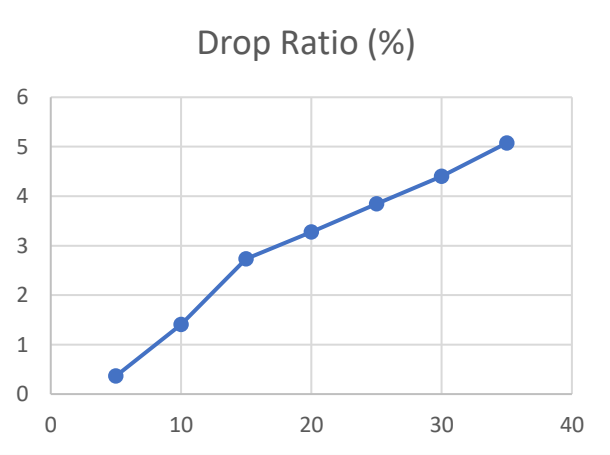


Varying number of flows:

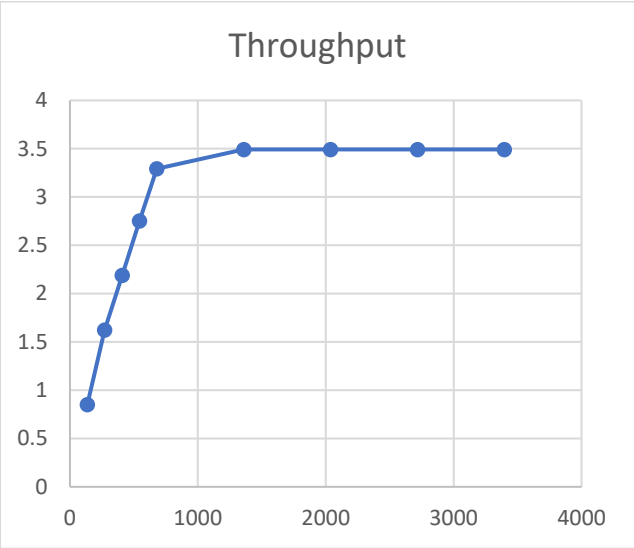


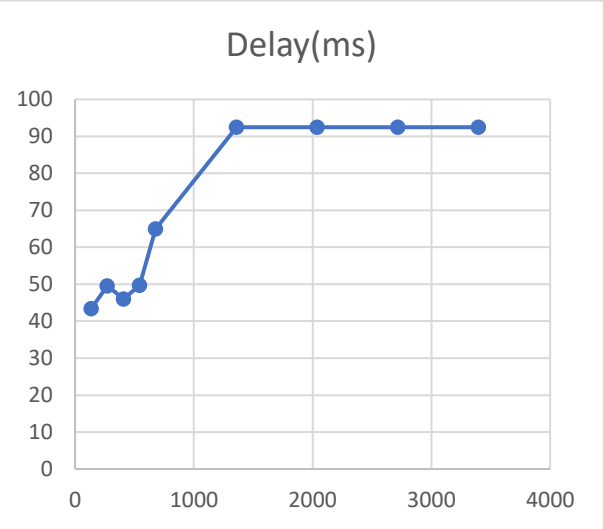


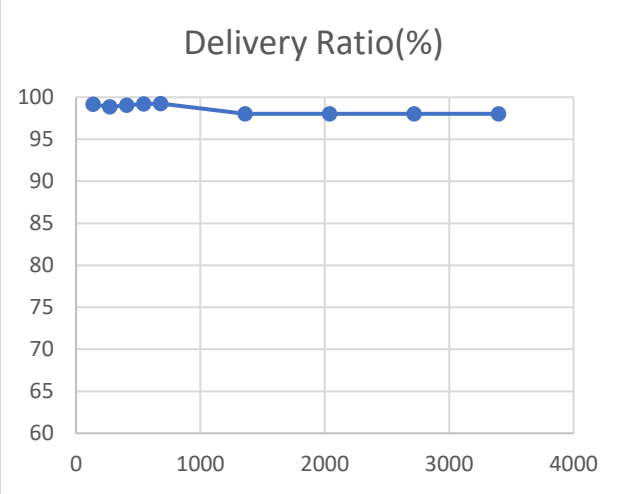


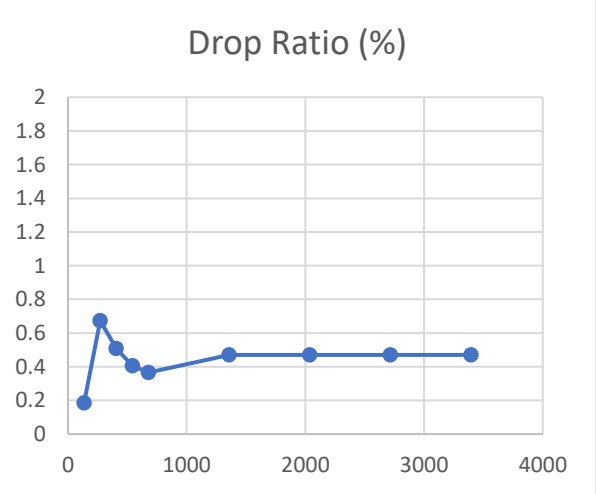


Varying average packets per second:

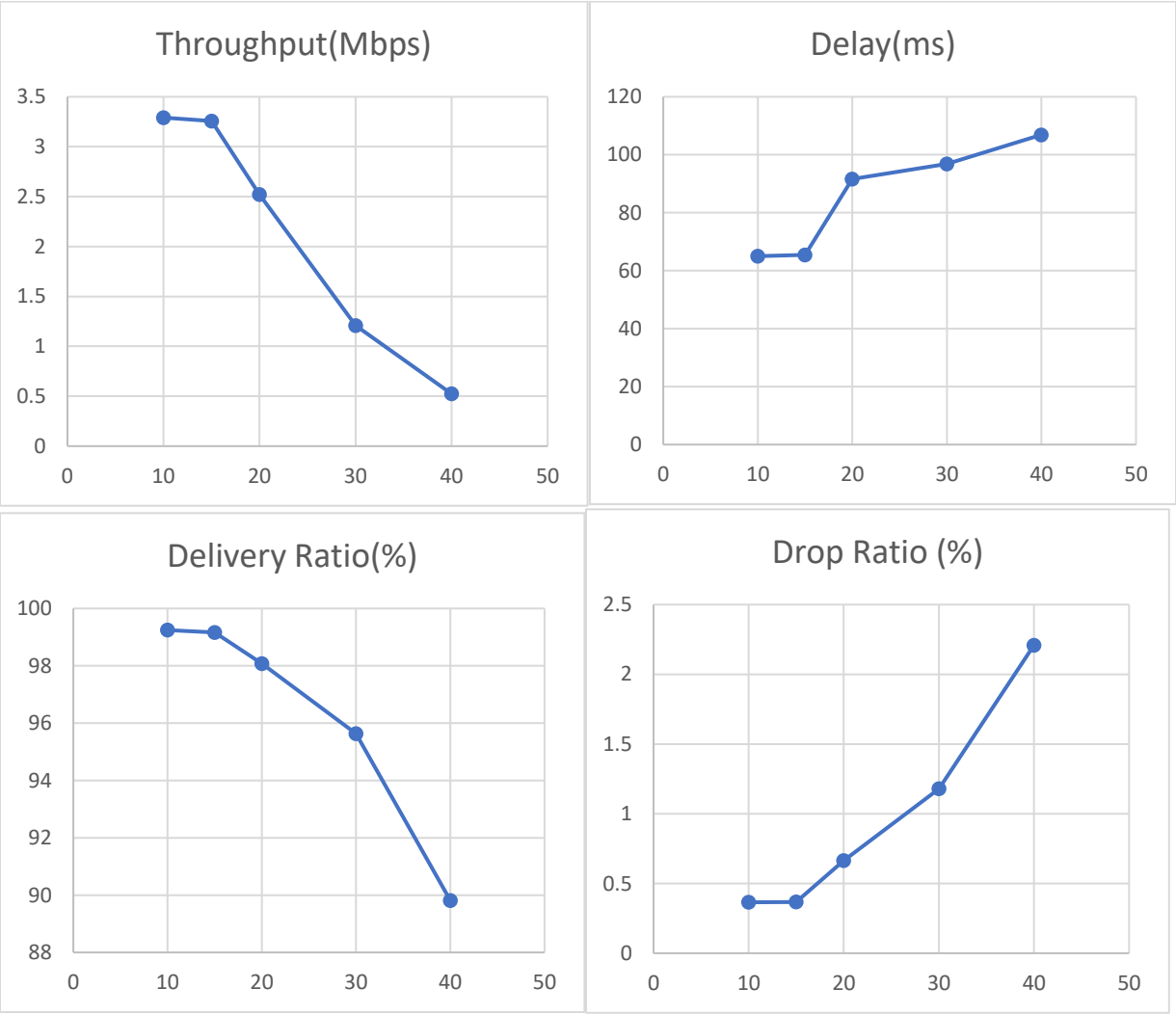








Varying speed (m/s) of user nodes:



Summary:

Graphs here are self-explanatory. Among all the variations, a proportional or inverse proportional (not necessarily linear) relation with metrics have been seen. About throughput, it seemed to saturate in 3.5Mbps (bottleneck has 5Mbps datarate) whenever it could reach upto that. Average delay ratio seemed to almost linearly proportional to number of bottleneck nodes which is intuitive too. Delivery ratio almost always decreased with the increment of all parameter, and the opposite case happened to drop ratio.

Task B:

Implementation of RRED: Robust RED Algorithm to Counter Low-Rate Denial-of-Service Attacks. Link

Overview:

LDoS exploits TCP's retransmission timeout mechanism. RRED will deny attacking packets to enter RED queue. RRED is supposed to keep **throughput** better in presence of LDoS. It is implement over the DoEnqueue() method of the usual RED queue discipline.

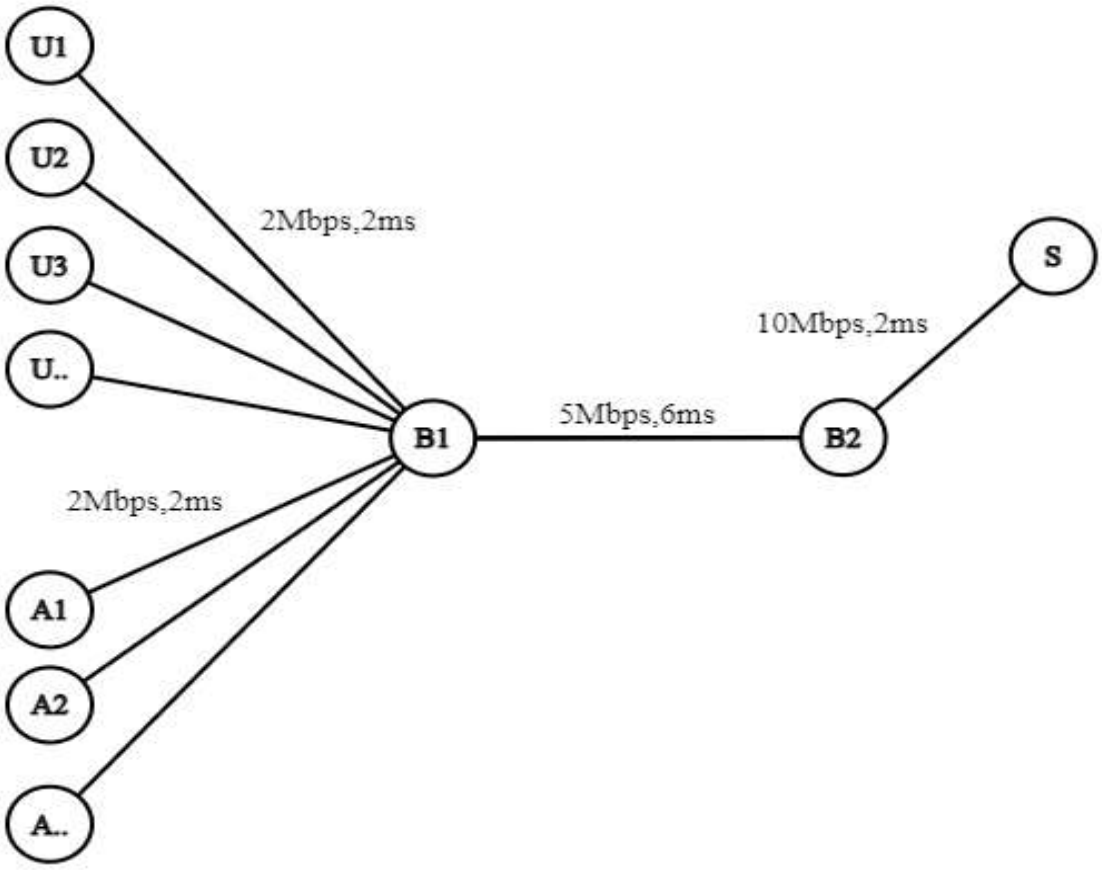
Whenever a packet *p* tries to be enqueued in RED queue, RRED will check it first. If packet *p* belongs to flow *f* and *f.T1* is the arrival time of last dropped (by RRED) packet from flow *f* and *T2* is the arrival time of last dropped (by RED) packet of all flows, RRED will discard *p* given that its arrival time is less than *max(f.T1,T2)+10 ms.*

Modifications in simulator:

1. In **red-queue-disc.cc:**
   - Checking of RRED on top of DoEnqueue() function
   - Hash function of each incoming packet (based on flow, same hashvalue of same flow).
   - Updated constructor for added member variables.
2. In **red_queue-disc.h:**
   - Including necessary member variables (T2, flow array), attributes (RRED enable, default value=false).
3. In **sB.cc:** Implemented a wired topology to check RRED and compare.

Topology:

Wired topology.



Direct observation: Throughput (Mbps) [10 users, 10 attackers, 1 Mbps datarate]

	With RRED	Without RRED
With Attack	**4.63268**	**0.197837**
Without Attack	4.68096	**4.64563**

Without RRED and with LDoS attack, throughput gets devastated. With RRED it comes close to the throughput with out any LDoS attack.

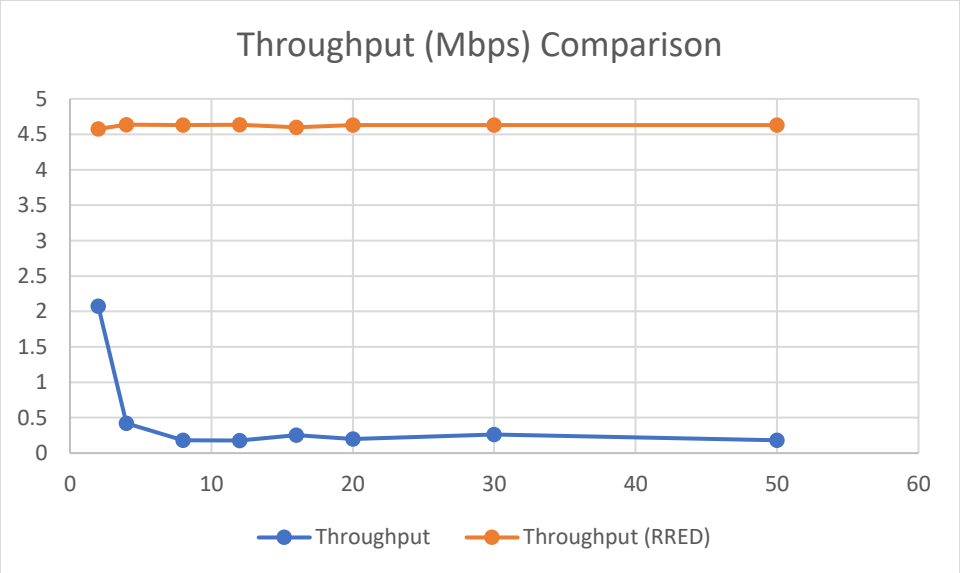
Graphs:

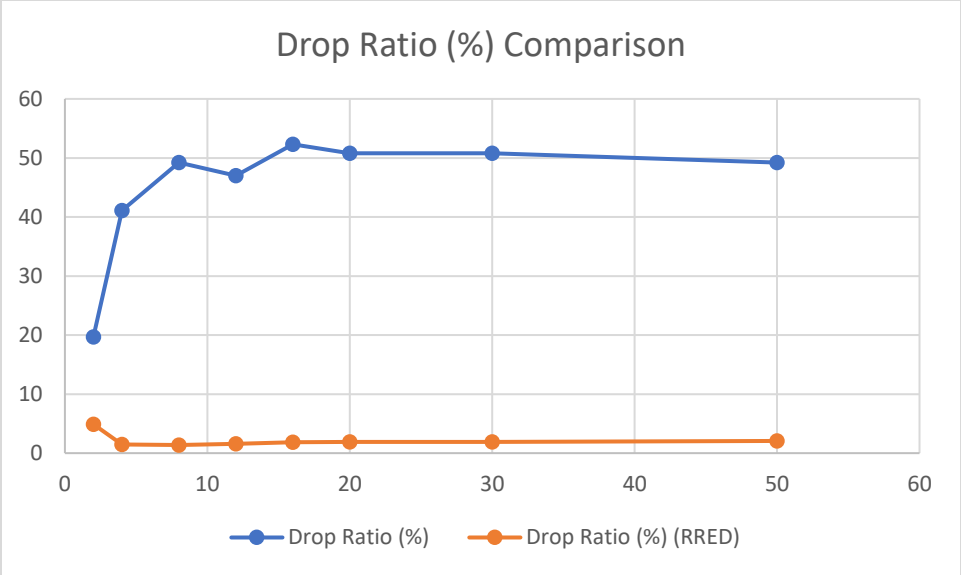
Number of Users=10

Datarate=1Mbps

Attacking nodes count has been varied to calculate **TCP** throughput and drop ratio.

#Attacker	Throughput	Drop Ratio (%)	Throughput	Drop Ratio (%)
2	2.07493	19.6787	4.57615	4.89642
4	0.420403	41.0959	4.63621	1.45379
8	0.178995	49.2308	4.6315	1.37615
12	0.177818	46.9697	4.63268	1.59405
16	0.252006	52.3077	4.59735	1.83908
20	0.199014	50.7692	4.62915	1.89944
30	0.262605	50.7692	4.62915	1.89944
50	0.18135	49.2308	4.62915	2.06422





Summary:

RRED almost restores TCP throughput in presence of LDoS attack. It completely outperforms usual RED. However, in absence of LDoS attack too, it seems to improve throughput.