CSE 322 PROJECT

FINAL REPORT

Submitted by: Iftekhar Hakim Kaowsar

ID: 1705045

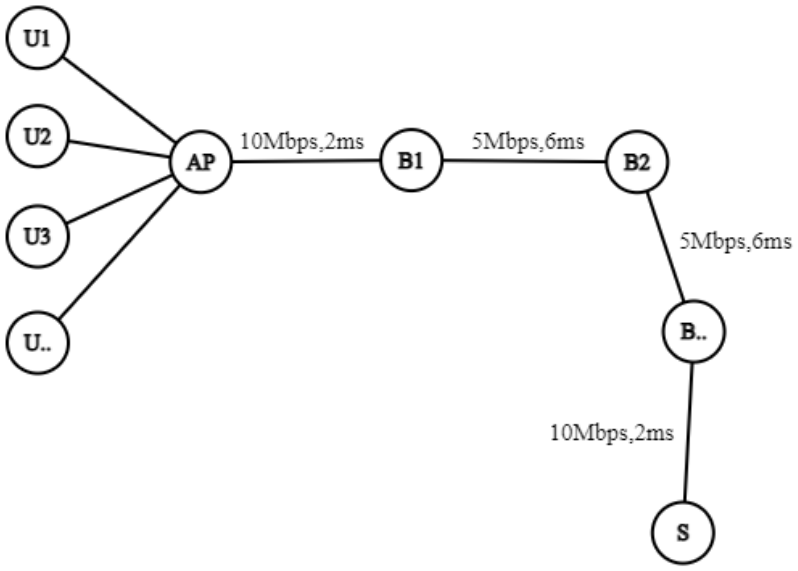
Task A.1:

Simulating Wireless high-rate 802.11 (mobile) network.

Added File:

- Simulation file **sA1.cc**

Topology:



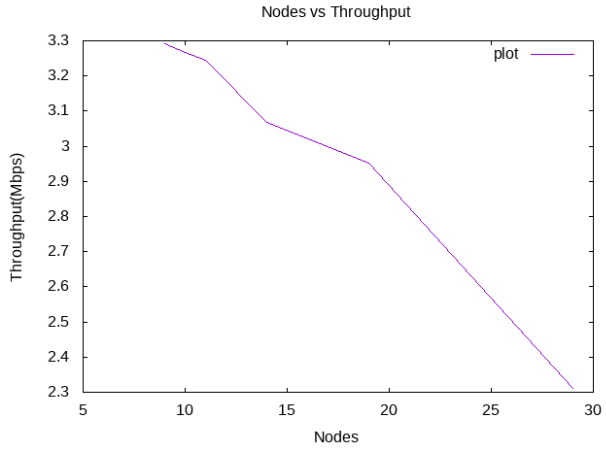
User nodes (including AP) are in wireless high-rate network. Rests are in wired network. We can vary bottleneck nodes with same attributes in each link. All user nodes are mobile and echo the server node with TCP sockets.

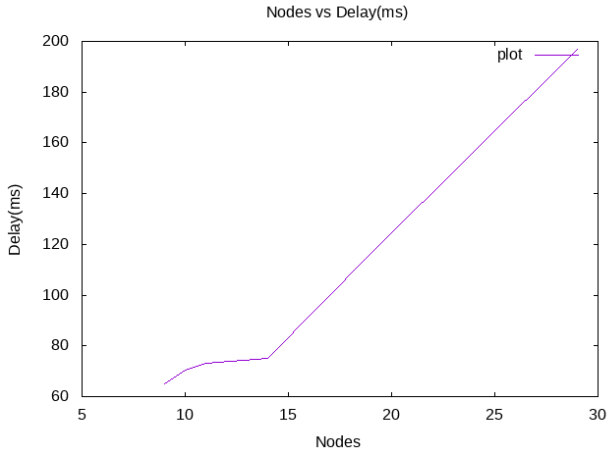
Number of bottleneck nodes, flows, speed of user nodes and packet per second have been varied.

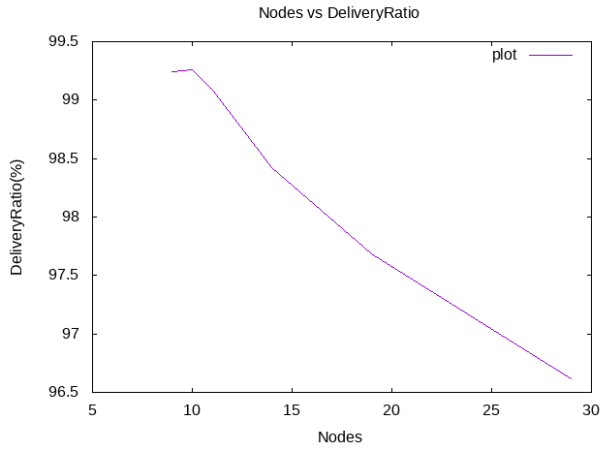
Results:

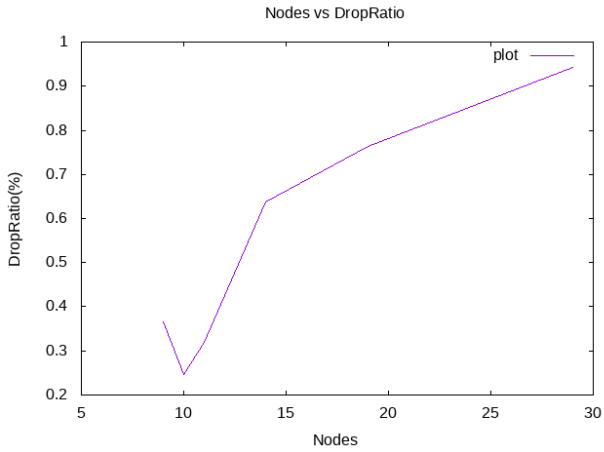
Varying number of bottleneck nodes or total nodes:

Fixed parameters: 5 user nodes, 5 flows, 1 m/s speed and 1Mbps data rate from each source.







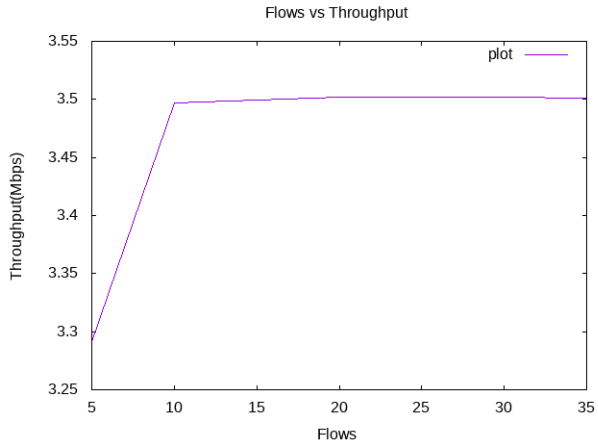


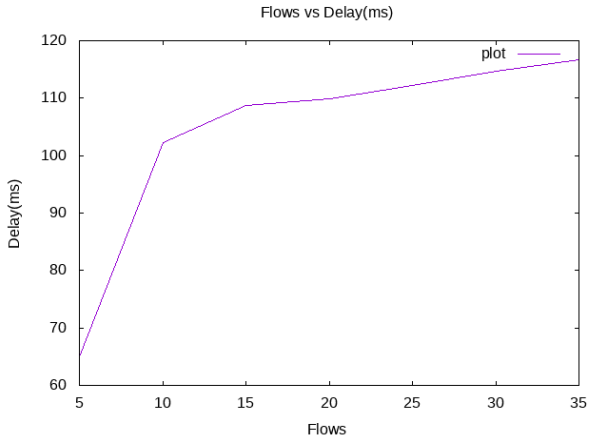
As we increase the nodes, hopcount from source to sink increases here. As a result, throughput decreases as node count increases. Delay increases with the

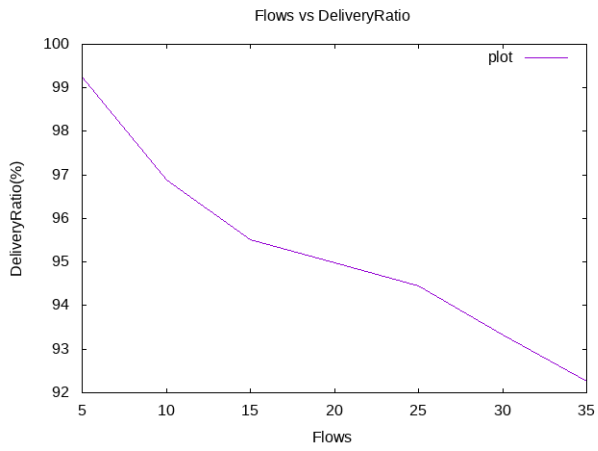
increment of nodes, because it needs to flow through more channels. Delivery ratio decreased and drop ratio decreased mostly for the same reason.

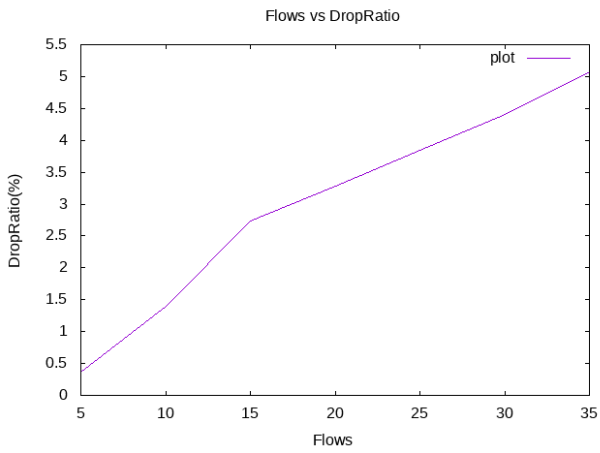
Varying number of flows:

Fixed parameters: Total 9 nodes, 1 m/s speed and 1Mbps datarate from each source.





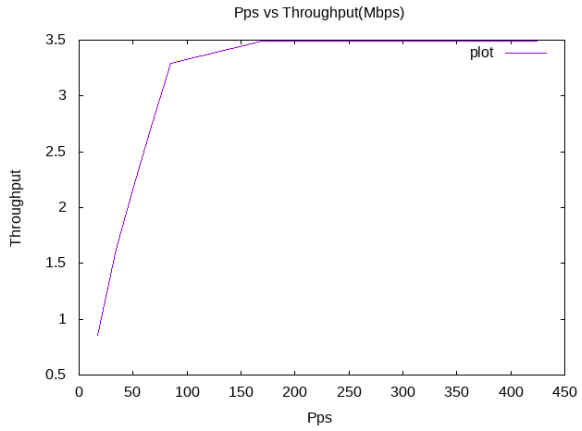


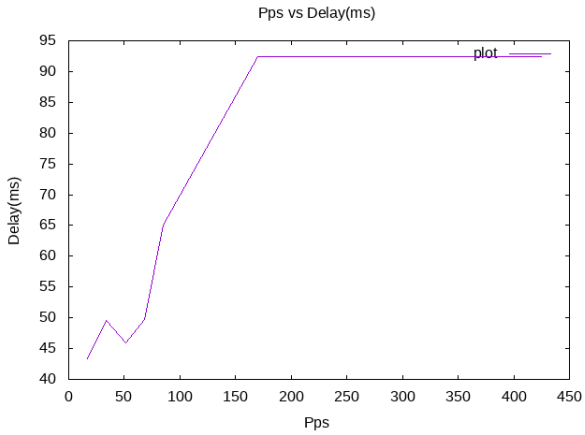


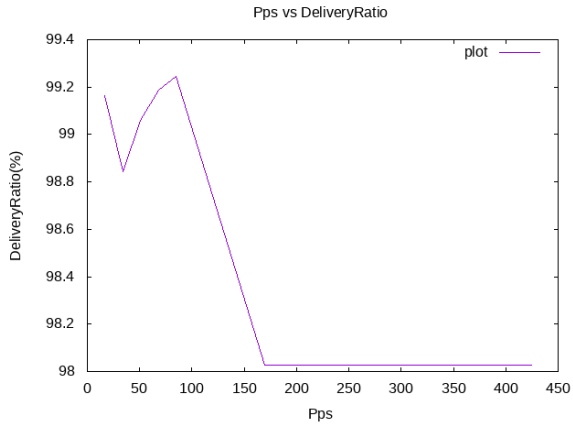
As number of flows increase, throughput first increases and then saturates. Every network has a peak throughput (we introduced bottleneck link here too). Delay also increases with the presence of more traffic. For the similar reason, delivery ratio falls and drop ratio grows.

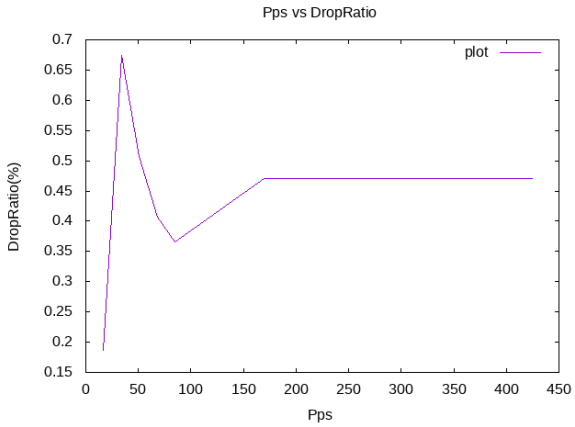
Varying average packets per second:

Fixed parameters: Total 9 nodes, 9 flows, 1 m/s speed.





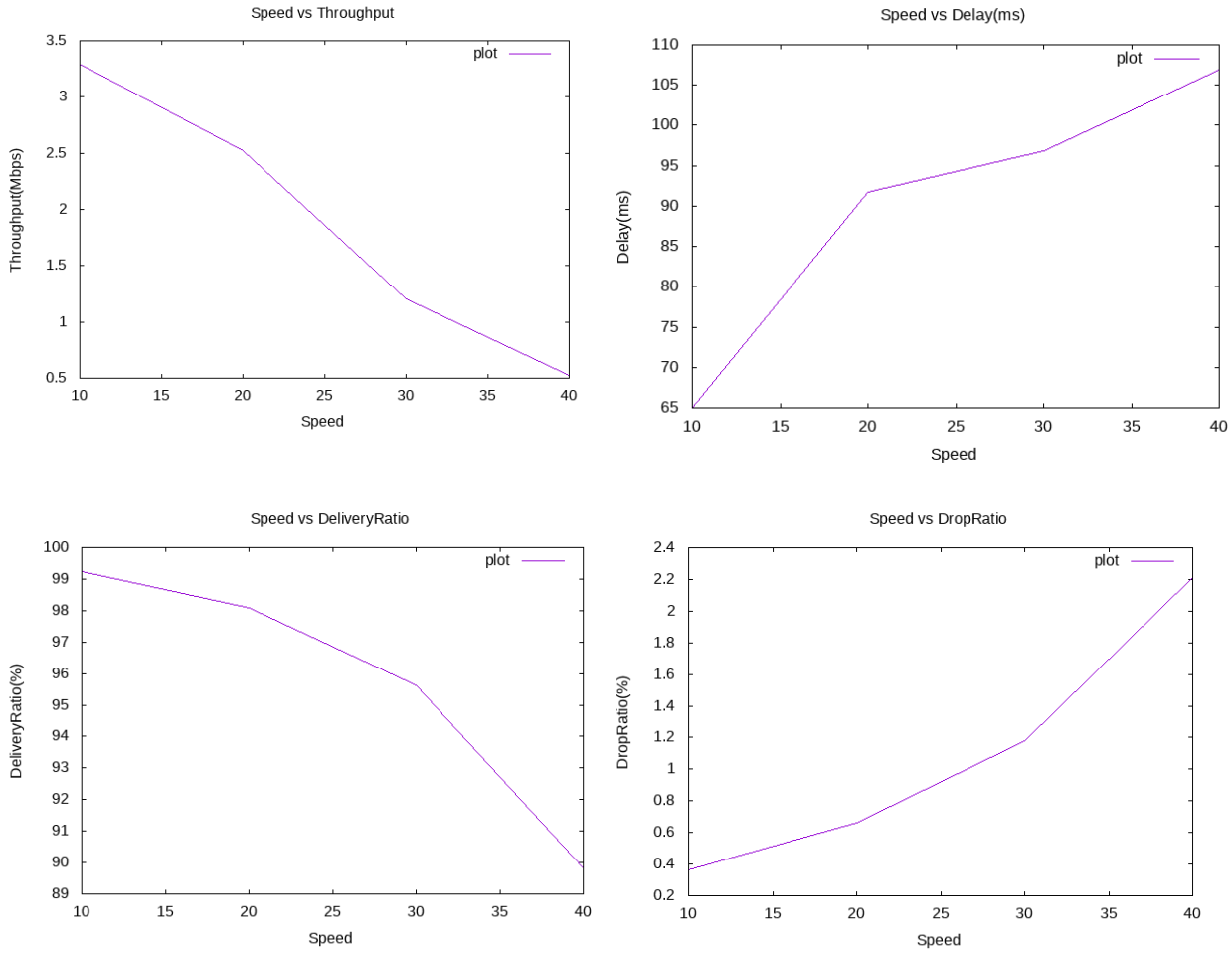




With the increment of packet per second or data rate of sources, throughput first increases and saturates to 3.5Mbps, which is close to bottleneck link's data rate. Delay seemed to increase mostly as queues get fulfilled faster with more datarate. But delivery ratio mostly decreased and drop ratio mostly increased. But they have little random behavior too when network does not reach saturation throughput.

Varying speed (m/s) of user nodes:

Fixed parameters: Total 9 nodes, 9 flows and 1 Mbps datarate from each source.



As we vary speed of the wireless nodes, metrics followed a proportional or inverse proportional behavior. All of them here are as expected. As speed increases, throughput and delay increases which is intuitive. Similarly, delivery ratio falls and drop ratio grow, with more speed of wireless 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 data rate) 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.

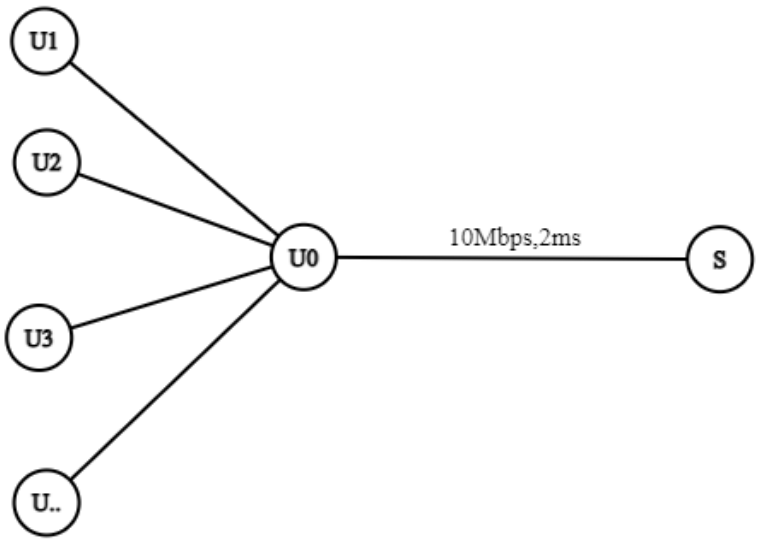
Task A.2:

Simulating Wireless low-rate 802.15 (mobile) network.

Added File:

- Simulation file **sA2.cc**.

Topology:



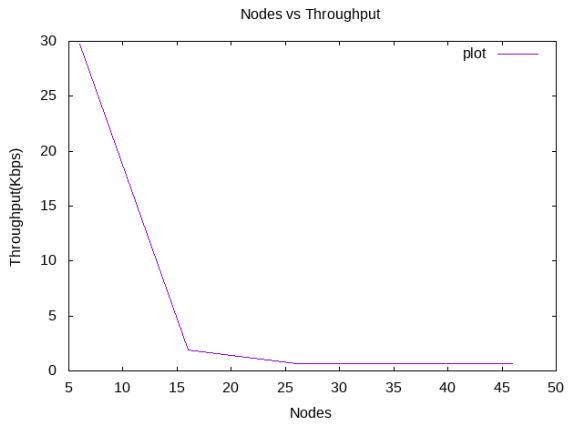
User nodes here are in wireless low-rate network. User node (except U0) are mobile. Channel between U0-Server is wired.

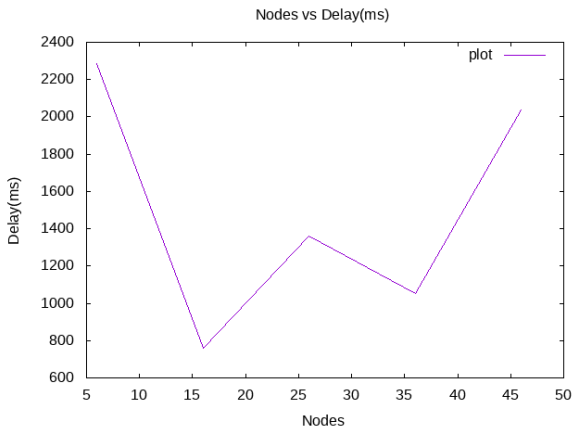
Number of user nodes, flows, speed of user nodes and packet per second have been varied.

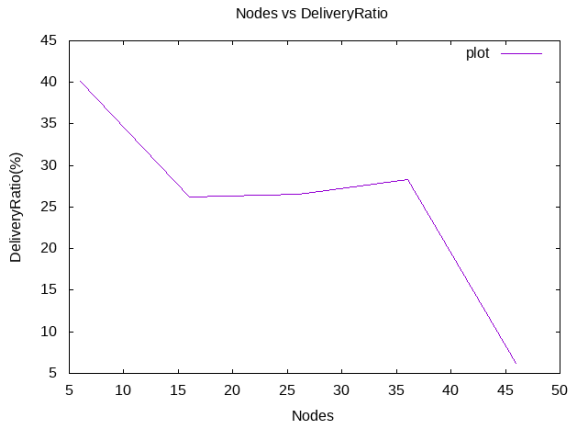
Results:

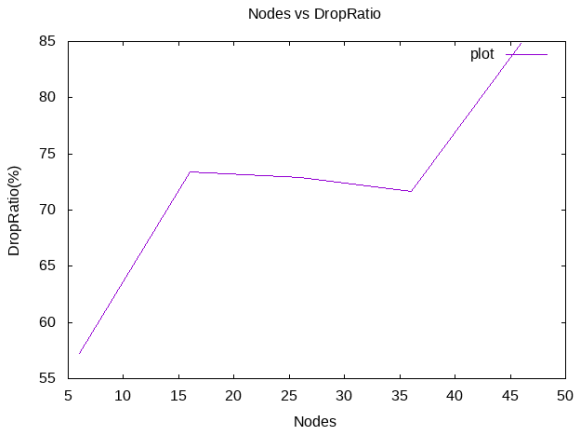
Varying user nodes or total nodes:

Fixed parameters: 4 flows, 0.01 m/s and 10 Kbps data in each source.





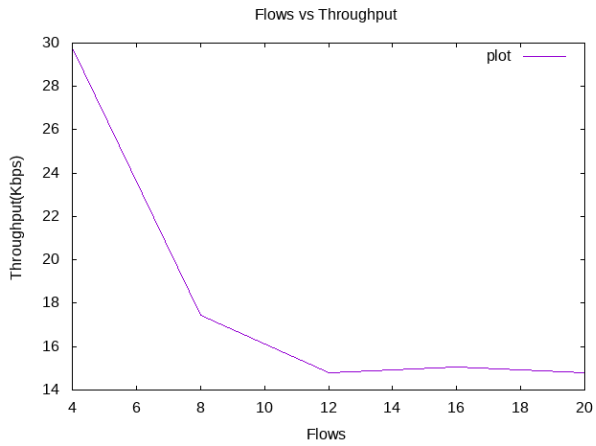


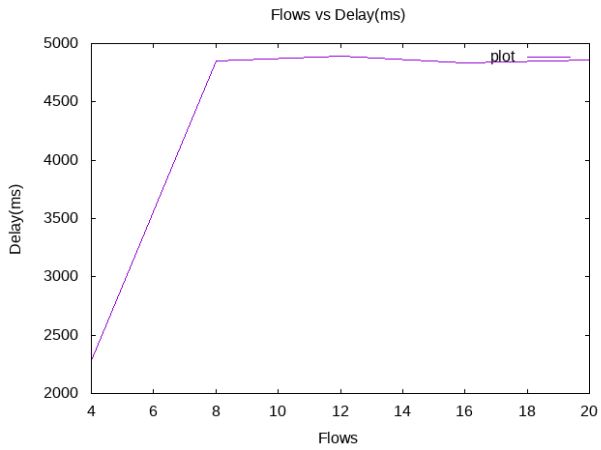


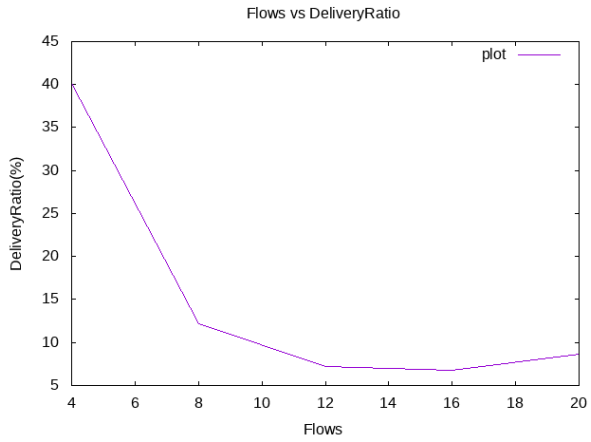
As number of user nodes varied, throughput decreases. Here number of flow is fixed. So this can be because of beacon corresponding with nodes that do not include any TCP flow. Delivery ratio decreases and drop ratio increases for similar reason. Delay ratio seems to be quite random. Here, as we increase node count, we do not really increase intermediate nodes count. Presence of high drop ratio dominates all the metrics.

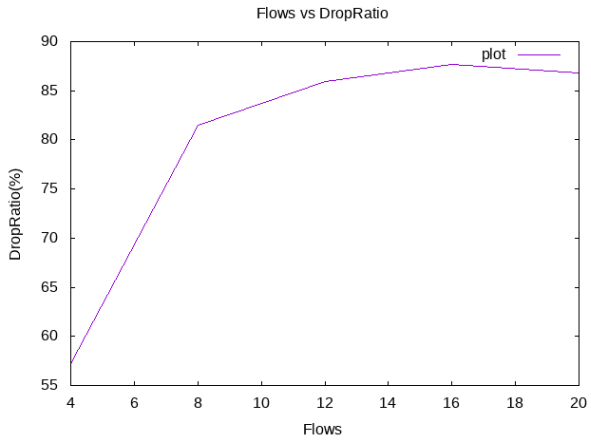
Varying number of flows:

Fixed parameters: 6 total nodes, 0.01 m/s speed for each wireless mobile node and 10 Kbps datarate.





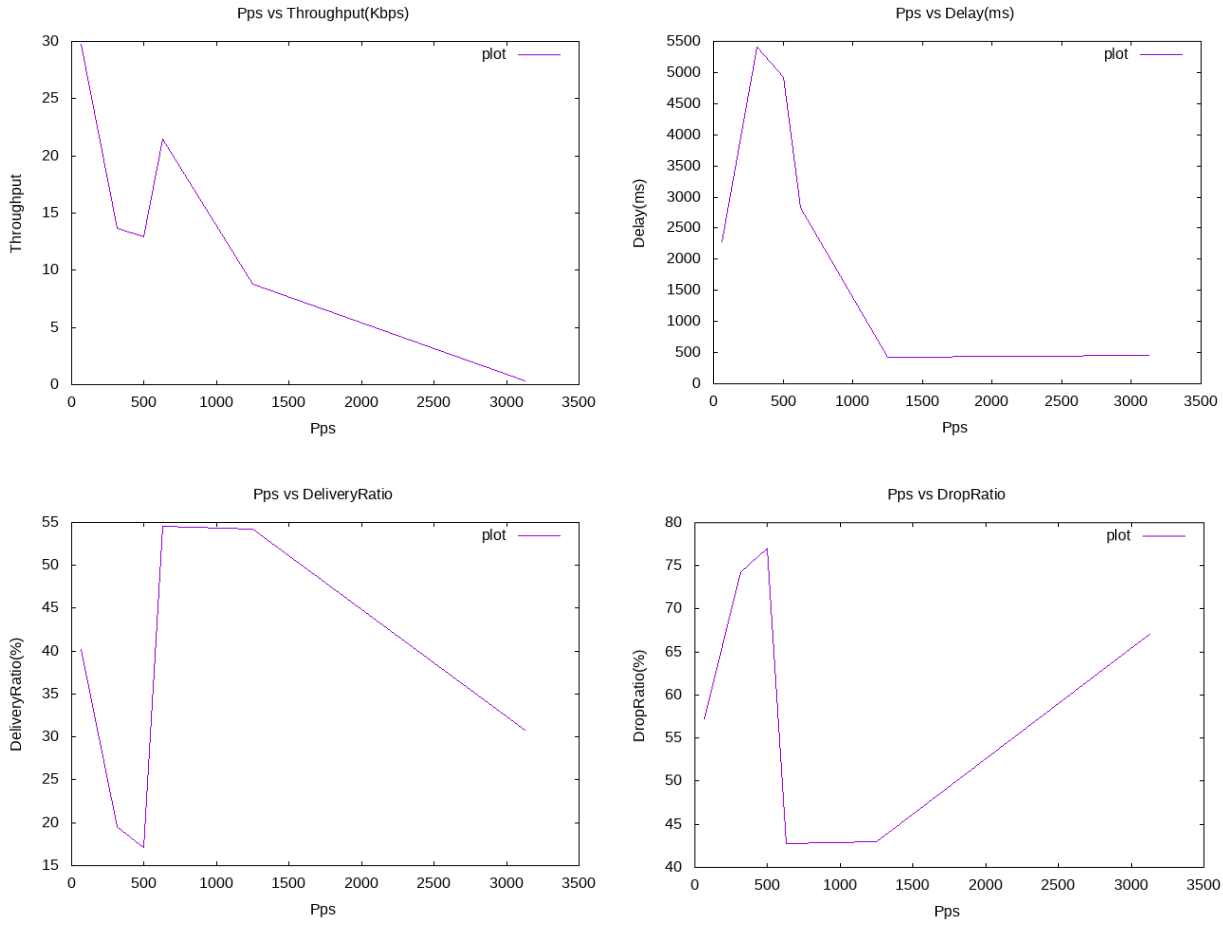




As flows increases, there are more traffic in overall network and queues get fulfilled faster. As a result, throughput increases, delay ratio increases almost upto a saturation point. More or less, delivery ratio decreases and drop ratio increases.

Varying average packets per second:

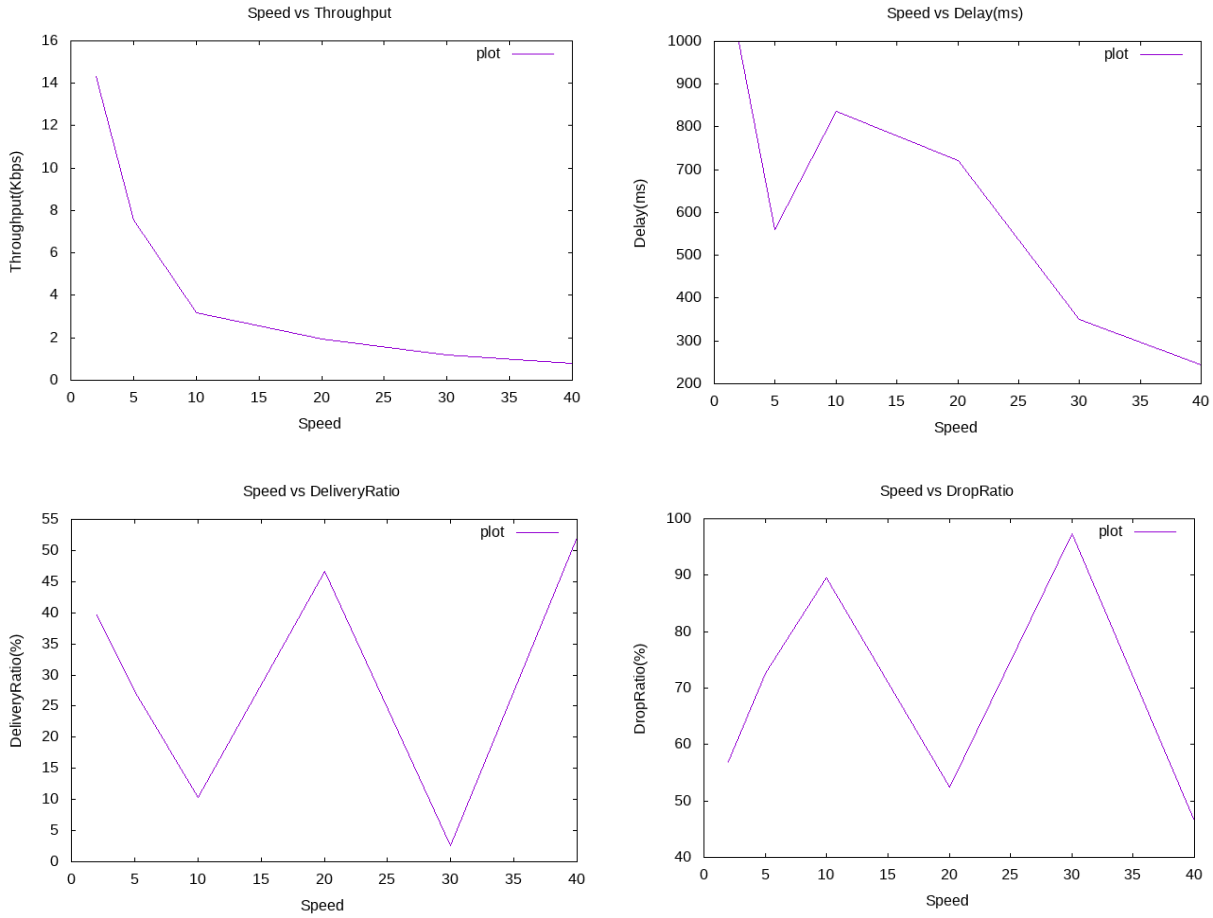
Fixed parameters: 6 total nodes, 1 m/s speed and 4 flows.



Here most of the values do not directly follows an generic schema. This can be because of high drop ratio being more dominating than packet per second.

Varying speed (m/s) of user nodes:

Fixed parameters: 6 total nodes, 4 flows and 10 Kbps datarate.



As we increase speed, throughput decreases as expected. But end-to-end delay seemed to decrease mostly. Delivery ratio and drop do not follow fixed sequence, which can be the result of packets being dropped in wireless low-rate network in high rate.

Summary:

In overall metrics, we saw that it has a very high drop ratio in average (greater than 50%). This actually dominates overall performances and other calculated metrics.

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.

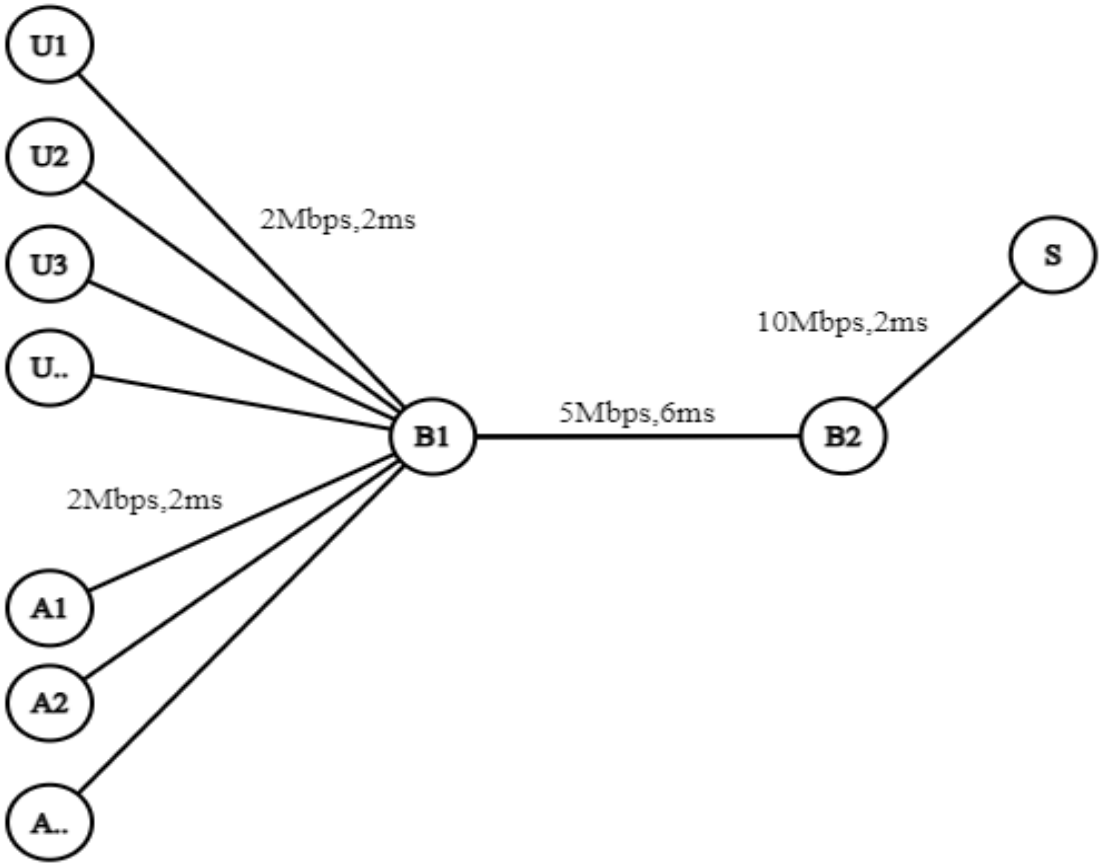
Whenever a packet *p* tries to be enqueued in RED queue, RRED will check it first. If packet *p* belongs to flow *f* and *f.T1* is the arrival time of last dropped (by RRED) packet from flow *f* and *T2* is the arrival time of last dropped (by RED) packet of all flows, RRED will discard *p* given that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

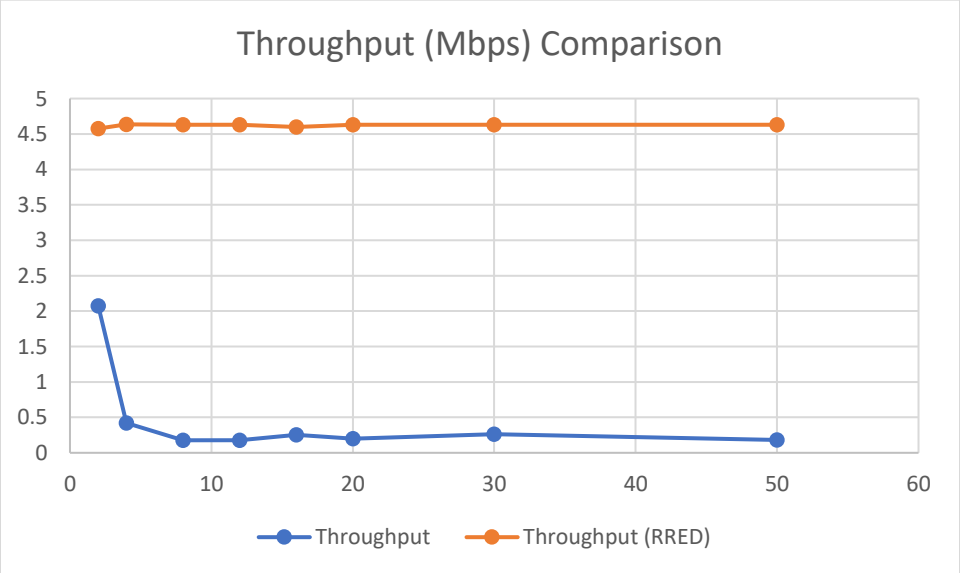
Graphs:

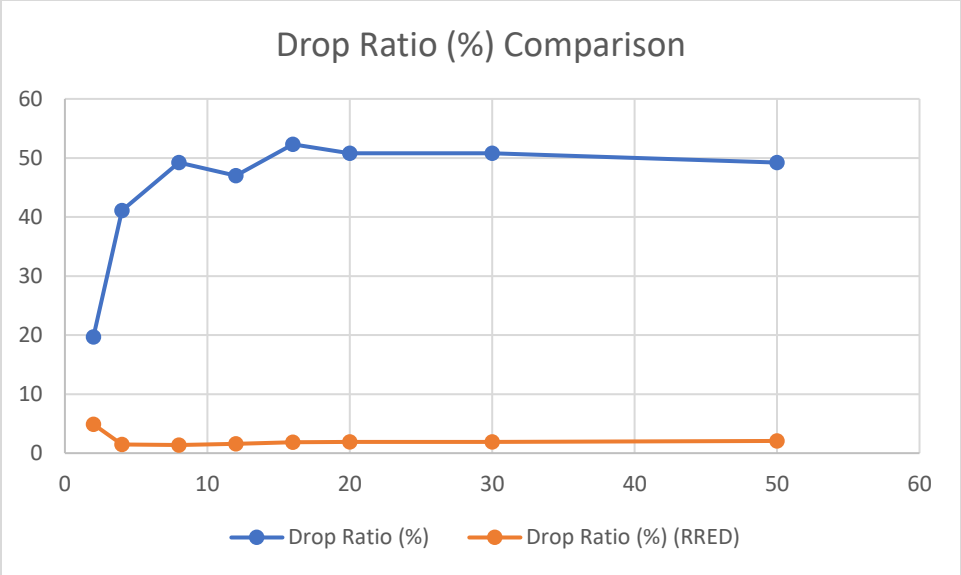
Number of Users=10

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.