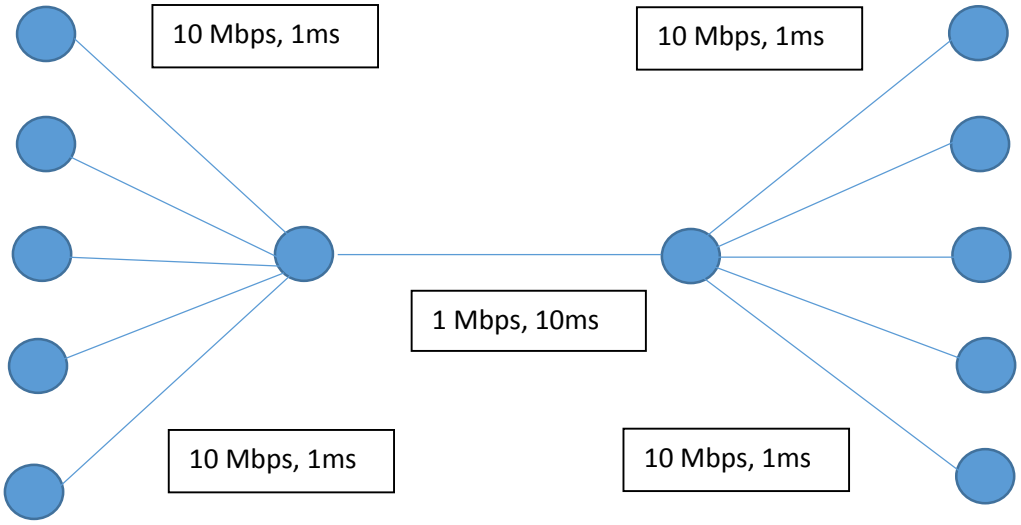

Setup:

Five sources and five sinks are connected with two routers in between. All the router to source links and router to sink links are 10 Mbps. The link between routers is 1 Mbps and is the bottleneck, where we analyze the queue. Delay on links are varied to study their effects on goodput.

Procedure:

In some set of experiments, all the sources are TCP, using *BulkSendHelper()* at source and *PacketSinkHelper()* at the corresponding sink. We also tried *OnOffHelper()* for TCP, with sources On and Off times uniformly distributed between 0 and 1 seconds. In another set of experiments, UDP source and corresponding sink are installed on the last source-sink pair. Effectively, there are 4 TCP pairs and one UDP pair in those experiments. UDP sources use *OnOffHelper()* and sinks use *PacketSinkHelper()*. Different sets of experiments are performed by changing queue type (RED or Tail Drop), queue size, TCP segment size, maximum receiver advertised window, using either TCP Tahoe or Reno, UDP sending data rate and delay on links. Burstiness is created in the network by using links with large delay-bandwidth product and setting small receiver advertised windows.

Red mechanism drops packets earlier than before congestion occurs in the network and packets are actually dropped at the router queues. Red has a few parameters to tune, Minimum Threshold (MinTh), Maximum Threshold (MaxTh), MaxP and queue size, which can be tuned in NS3. Paper by Sally Floyd suggests setting values of these parameters for optimal results, TCP receiver advertised window should be set equal to delay-bandwidth product of the link where queue is implemented. Maximum buffer size of queue should be set to approximately twice the delay-bandwidth product. MinTh should be $1/4^{th}$ of maximum buffer size. MaxTh should be three times MinTh. RED keeps track of running weighted average of number of bytes in queue. If average number of bytes remains below MinTh, the packet is accepted in queue. If the average number of packets in queue is between MinTh and MaxTh, the packet is dropped with

probability as a function of 1/linterm, maximum and minimum threshold and current value of average queue size. If the average size is above MaxTh, the packet is definitely dropped. TCP sources would go into fast retransmit and fast recovery mode when their packets are dropped and would slow down in sending data in response to congestion. UDP sources do not adjust their sending rate in response to congestion and packet drop. They keep on flooding the network. In our simulations, UDP sources send the data all the time with always on duty cycle. When their combined sending rate approaches the bottleneck link capacity, they completely shut out the TCP sources using the entire network bandwidth. TCP Reno is used in the simulations. TCP Reno has Fast Recovery mechanism in addition to all the features of Tahoe version.

MinTh = 0.25 * Queue Size
MaxTh = 0.75 * Queue Size
LInTerm = 1/Pmax = 50
Simulation Time = 100 seconds
TCP Segment Size = 1024B

We calculate the average of all goodput values at all the sinks. This is not a good measure as it does not take fairness into account. But it gives a general trend with change of parameters. For looking at fairness, we look at goodput values of individual flows.

List of Parameters to tune in simulations

windowSize, queueSize, tcpType (0 for Reno, 1 for Tahoe), segSize, rtt (at the bottleneck), queueType (0 for tail drop, 1 for red), udpRate (in kbps), noUdp (0 for presence of UDP, 1 for absence), bulkSend (for TCP traffic, 0 for on off model, 1 for bulksend)

Effect of UDP on goodput

No matter, whatever combinations of parameters we use, UDP flow achieves goodput almost equal to the sending rate as long as it is below the bottleneck link capacity. If sending rate is more than the bottleneck link capacity, it still achieves bottleneck link capacity. The reason is that UDP source we are using is always on and it does not decrease its sending rate in response to dropped packets. By changing the on duration in duty cycle, goodput is accordingly scaled. For example, 50% on duration results is halving of goodput as compared to full on duty cycle. If UDP sending rate is limited to $1/5^{th}$ of bottleneck link capacity, it gets its share leaving the remaining share to be divided among TCP flows. This results in fair allocation. There is slight improvement in Red case compared to Drop Tail as UDP packets would definitely be dropped when average queue size exceeds maximum threshold (which would be earlier than tail drop policy). But still UDP flow occupies single shared queue most of the time. As an example, when UDP is sending at 500 Kbps:

Flow	Goodput
1	94.705kbps
2	87.081kbps
3	120.996kbps
4	110.961kbps
5	487.424kbps

In all the below examples, we just consider TCP flows because the results would almost be the same with UDP getting its share of bandwidth leaving the rest for other flows.

Comparison of Red and Tail drop in terms of fairness

Bulk send application

In one set of experiments, we set the same parameters for Red and Tail drop. We observed that even though the average goodput of these scenarios was the same, Red was much fairer in allocation of resources as compared to tail drop. Since the sources are sending at continuous rate, some sources lock out the queue, leaving very less bandwidth for the other flows.

Red	Tail drop
191.857	304.169
194.478	304.087
174.817	304.005
210.125	18.5958
178.094	18.5958

On Off model

In another set of experiments, with on off model of TCP sources (with source on time and off time uniformly distributed between 0 and 1), fairness of drop tail was improved because the sources would be in off mode for a while giving chance to other sources to send data.

Red	Tail drop
184.074	197.55
176.742	203.776
205.414	152.781
195.174	142.787
181.576	224.461

Results for Red with TCP flows:

In all the experiments below default value of segment size is taken to be 1024 Bytes, because it gives optimal results for goodput which was found after rigorous experiments changing its values. Most of the results are for *BulkSendApplication*. Results for *OnOff* model are quite similar in case of using only TCP flows. The default values of delay for leaf node links and router to router are 1ms and 10ms respectively.

Effect of changing queue size with fixed window size

If we set window size equal to delay bandwidth product and change queue size, there is no effect of changing queue size on goodput. Queue Size can vary from delay bandwidth product to any higher value. Goodput is limited by maximum receiver advertised window, which is less than or equal to queue size at bottleneck link. No matter how much large queue we put up, it will never be filled up to result in decrease in goodput.

Delay (one way)	Bandwidth	Delay × Bandwidth	Window Size	Queue Size	Average Goodput
10ms	1Mbps	2500B	2500B	2500B	189.874kbps
10ms	1Mbps	2500B	2500B	5000B	189.847kbps
10ms	1Mbps	2500B	2500B	7500B	189.847kbps

Effect of changing window size with fixed queue size

Setting Queue Size to twice the delay bandwidth product, we get optimal results when queue size is half of window size i.e. equal to delay bandwidth product. Above and below that value, goodput is decreased.

Delay (one way)	Bandwidth	Delay × Bandwidth	Window Size	Queue Size	Average Goodput
10ms	1Mbps	2500B	1250B	5000B	34.9798kbps
10ms	1Mbps	2500B	2500B	5000B	189.847kbps
10ms	1Mbps	2500B	5000B	5000B	182.321kbps

Since we have used segment size of 1024 Bytes, smaller window of 1250B means we will get short segments at the receiver resulting in very low goodput value.

Effect of changing RTT at the Bottleneck

Keeping queue size equal to twice delay-bandwidth product of bottleneck link (1Mbps*20ms = 2500 B) i.e. 5000 Bytes and window size equal to 2500 B, we see the effect of changing delay of bottleneck link. Changing delay and adjusting window size and queue size accordingly does not result in change in goodput.

Delay (one way)	Bandwidth	Delay × Bandwidth	Window Size	Queue Size	Average Goodput
10ms	1Mbps	2500B	2500B	5000B	189.847kbps
20ms	1Mbps	5000B	5000B	10000B	188.973kbps
30ms	1Mbps	7500B	7500B	15000B	189.448kbps

Effect of changing RTT at leaf nodes

In one set of experiments, we set delay of bottleneck link to 10ms and bandwidth to 1Mbps. We change the delay on leaf node links to observe effect of average goodput at leaf nodes. As long as delay is less than or equal to the delay of bottleneck link, there is no effect on goodput. Increasing delay beyond bottleneck delay results in decrease in goodput. Increasing delay at leaf node links results in burstiness of data. These results are more pronounced if window size is kept small.

Delay (one way) at leaf	Average Goodput
1ms	189.847kbps
10ms	189.809kbps
20ms	136.872kbps
30ms	102.482kbps

Effect of changing values of MaxTh and MinTh

There was no effect on goodput values by setting MinTh to $1/4^{th}$ of queue size and changing values of MaxTh from 0.35 of queue size to 0.75 of its value with any set of parameters. May be our topology and choice set of parameters does not allow these changes to take effect. We can try in some other setting to see their effects in future.

Conclusion:

In our scenarios, there is not much difference in Red and Tail Drop apart from much better fairness in case of Red. Fairness in Tail Drop can be improved by using OnOff model. UDP flows hog resurces. To avoid that service differentiation can be used which means that we should give separate queue to UDP and TCP flows. We can use Red algorithm on each queue separately depending on relative weight of flow in that queue. This mechanism is known as weighted red. This is not currently supported in NS3.