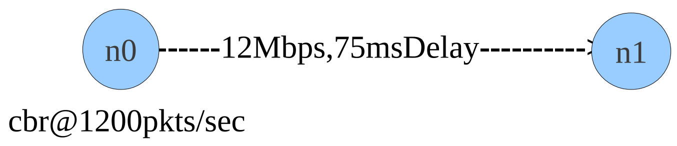
Simulation and Comparison of Queue Scheduling Algorithms

Abstract

This report presents a study of queue scheduling algorithms. We study the behaviour of queues (queue size and throughput) by simulating different queue scheduling algorithms like Deficit Round Robin (DRR) and Fair Queueing (FQ). We also make observations on the variation of TCP congestion window enabling RED and DropTail. The simulations were done using ns-2 simulator.

1 Simulation of D/D/1 queue (M/D/1 - with cbr)

1.1 Topology



1.2 Execution Format

```
1. $ns md1.tcl <Packet Size in Bytes> <Simulation Duration in secs>
2. $nam out.nam –to view the simulation
Arguments:
 <Packet Size in Bytes> : 100, 400, 700, 1000, 1200 etc.
 <Simulation Duration in secs> :  default is 3secs
Example
$ns md1.tcl 1300 7
$nam out.nam
```

1.3 Output details

The following files are obtained as output, when the simulation is complete.

1.3.1 out.nam : File that contains details for nam to view the animation.

1.3.2 qm.out : File containing details of the queue that is being monitored (n0-n1).

1.3.3 avgql.out : File containing time and corresponding average queue length.

1.4 Values Obtained

Packet Length (Bytes)	Simulation Time (secs)	Arrival Rate (Mbps) [λ]	Service Rate (Mbps) []	Average Queue Length (Theoretical)	Average Queue Length (Result)
100	4.4	0.96	12	0.0	0.0
400	4.4	3.84	12	0.0	0.0
700	4.4	6.72	12	0.0	0.0
1000	4.4	9.6	12	0.0	0.0
1200	4.4	11.52	12	0.0	0.4
1300	4.4	12.48	12	203.077	194.5

1.5 Formulae used

Since this is a constant bit rate source,

1.5.1 If λ <μ, then the queue length will be zero (i.e queue is unutilized). This is because the packets are coming at a constant rate (there is no burstiness) and the rate at which the packets are being serviced is faster than the rate at which they arrive.

1.5.2 When λ >μ, packets continue to arrive at a constant rate but cannot be served by the link at that pace. So, more and more packets keep getting pushed into the queue. Hence, the size of the queue is directly proportional to the difference in packets generated and serviced per unit time. And it also increases with the simulation time.

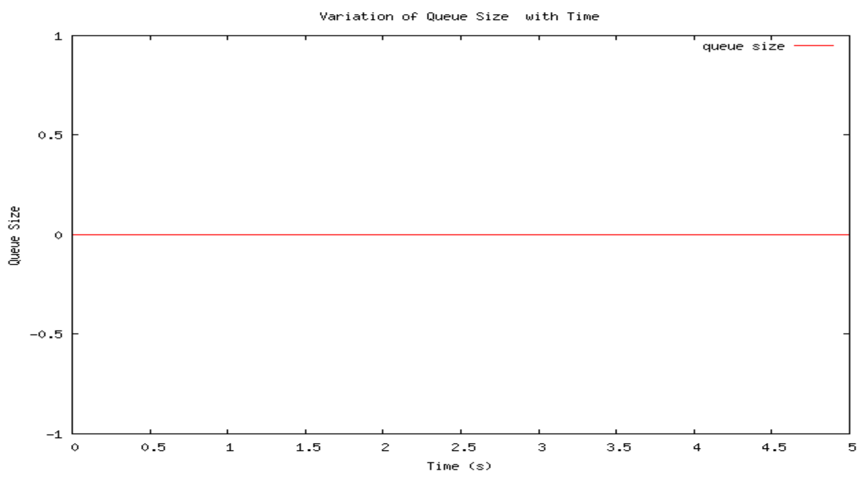
Formula used:

$$Queue\,Length=((\mu-\lambda)/(packetSize))*simulationTime$$

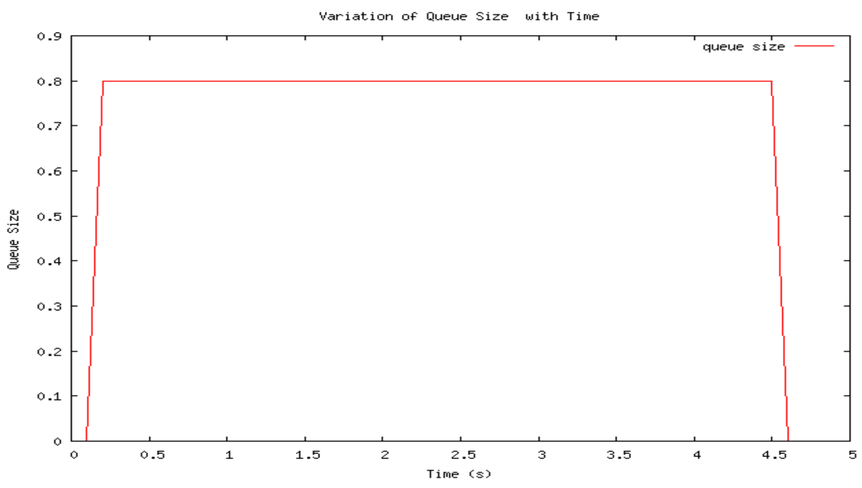
1.6 Results

1.6.1 Queue Length vs Time

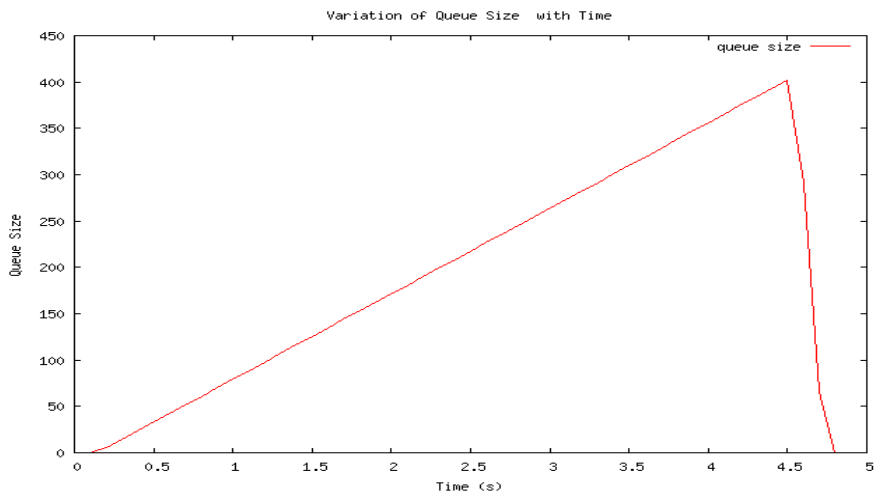
1.6.1.1 packet size: 100B, 400B, 700B, 1000B



1.6.1.2 packet size: 1200B

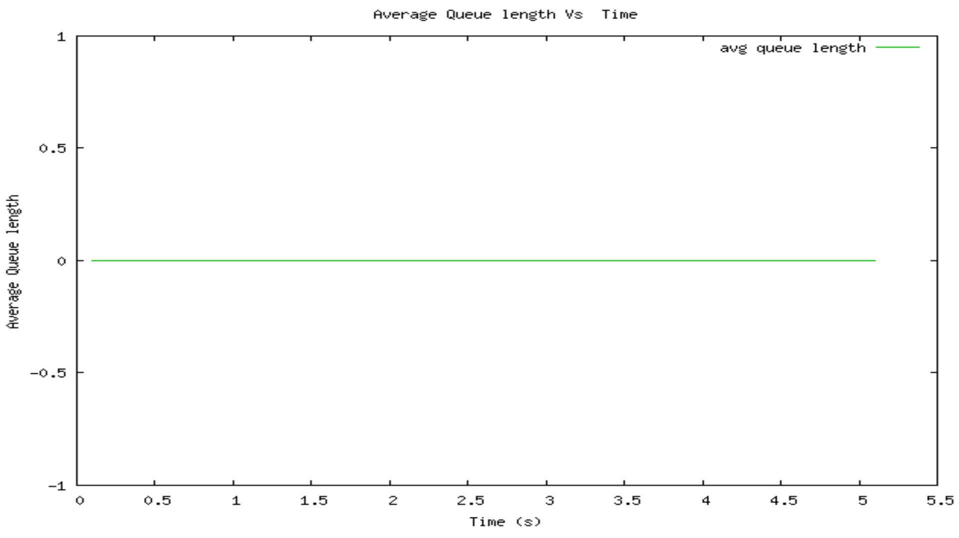


1.6.1.3 packet size: 1300B

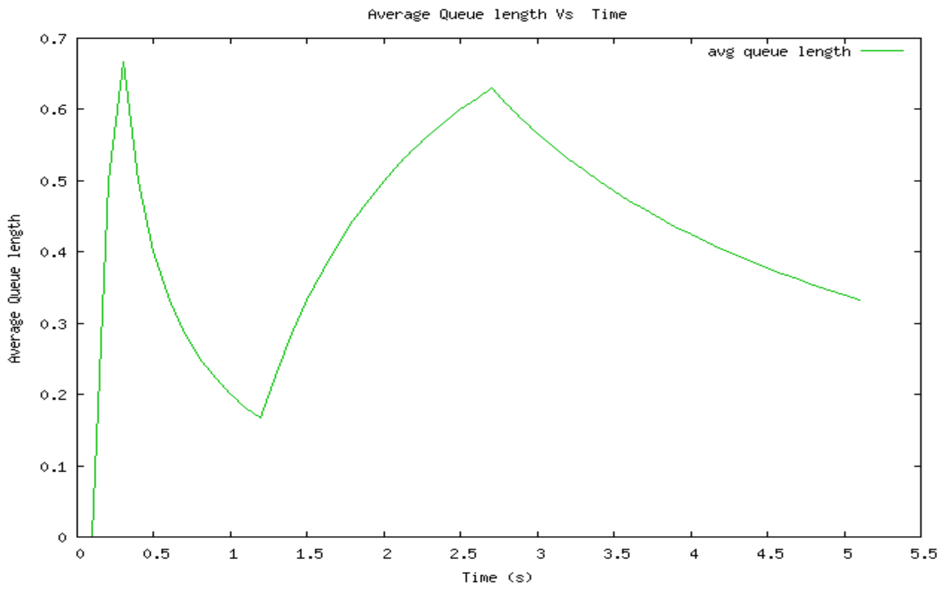


1.6.2 Average Queue Length vs Time

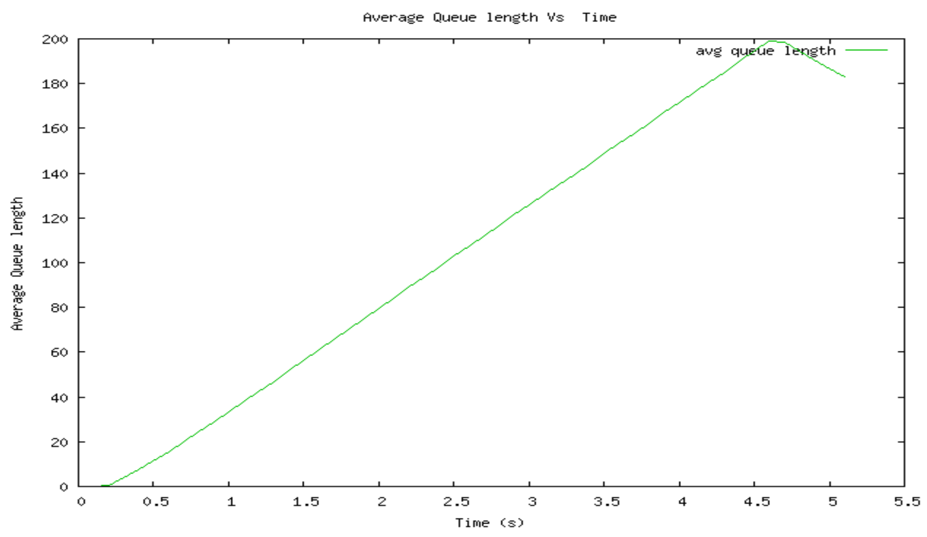
1.6.2.1 Packet Size: 100B, 400B, 700B, 1000B



1.6.2.2 Packet Size: 1200B



1.6.2.3 Packet Size: 1300B



1.7 Observation/Conclusions

The results are in concensus with the theoretical values.

1.7.1 When the rate of generation is less than the service rate, the queue is untilized.

1.7.2 With packet size 1200B, the load on the server is very close to one, in this case server is just able to meet the demand and hence the queue size fluctuates between 0 to 1 i.e it constantly grows and shrinks by a fraction.

1.8 Implementation Details

1.8.1 CBR over UDP

```
#setup the udp connection
set udp [new Agent/UDP]
$ns attach-agent $S $udp
set null [new Agent/Null]
$ns attach-agent $D $null
$ns connect $udp $null
$udp set fid_ 1

#setup a cbr over the udp connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ $pktsiz
#lambda needs to be 1200pkts/s in megabits, so compute accordingly
set a 1200.0
set lambda_ [expr [expr [$cbr set packet_size_] * 8] * $a]
set lambdaMbps_ [expr $lambda_ / 1000000]
puts "packet size: [$cbr set packet_size_] lambda: $lambdaMbps_
Mbps"
#continue with cbr parameters
$cbr set rate_ [expr $lambdaMbps_]mb
$cbr set random_ false
```

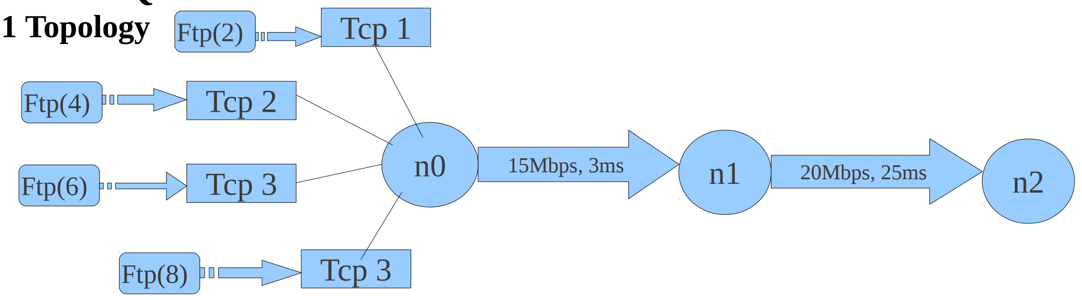
1.8.2 Queue Monitoring

```
set qmonitor [$ns monitor-queue $S $D [open qm.out w] ];
[$ns link $S $D] queue-sample-timeout;

#procedure to compute average queue length
proc queueLength {sum number file} {
    global ns monitor qmonitor
    set time 0.1
    set len [$qmonitor set pkts_]
    #set len [$monitor set size_]
    set now [$ns now]
    set sum [expr $sum+$len]
    set number [expr $number+1]
    #write the average queue length in to a file
    puts $file "[expr $now+$time] [expr 1.0*$sum/$number]"
    $ns at [expr $now+$time] "queueLength $sum $number $file"
}
$ns at 0 "queueLength 0 0 $avgf"
```

2 DRR vs FQ

2.1 Topology



2.2 Execution Format

```
1. $ns fqdrr.tcl <DRR|FQ> <simulationTime (s)> [uniform_cwnd]
2. $nam out.nam –to view the simulation
```

Arguments:

```
 <DRR|FQ> : To choose between DRR and FQ.
 <Simulation Duration in secs> :  default is 3secs
[uniform_cwnd] : optional parameter, to set all tcp sources to generate
packet at the same/uniform rate by keeping a uniform congestion window.
By default, the sources will have 2Mbps,4Mbps,6Mbps and 8Mbps rates.
```

Examples

```
$ns DRR 10
$ns FQ 15
$ns FQ 10 40 –here all ftp flows will generate at uniform rate.
$nam out.nam
```

2.3 Output Details

'out.nam' contains the nam trace details for the animation.

2.4 Results

2.4.1 TCP with different flow rates

commands: $ns fqdrr DRR 4.4 and $ns fqdrr FQ 4.4

Flow Id.	Generation Rate	DRR Throughput	FQ Throughput
1	2 Mbps	1.132 Mbps	1.479 Mbps
2	4 Mbps	2.236 Mbps	2.266 Mbps
3	6 Mbps	3.188 Mbps	4.829Mbps
4	8 Mbps	4.475 Mbps	4.849 Mbps

2.4.2 TCP with uniform flow rate

commands: $ns fqdrr DRR 4.4 40 and $ns fqdrr FQ 4.4 40

Flow Id.	Generation Rate	DRR Throughput	FQ Throughput
1	13.74 Mbps	3.389 Mbps	3.299 Mbps
2	13.74 Mbps	2.226 Mbps	3.294 Mbps
3	13.74 Mbps	3.989 Mbps	3.291 Mbps
4	13.74 Mbps	3.551Mbps	3.289 Mbps

2.4.3 TCP with varying flowrates and larger time duration

commands: $ns fqdrr DRR 50.4 and $ns fqdrr FQ 50.4

Flow Id.	Generation Rate	DRR Throughput	FQ Throughput
1	2 Mbps	1.109 Mbps	1.765 Mbps
2	4 Mbps	2.215 Mbps	2.349 Mbps
3	6 Mbps	5.332 Mbps	5.084 Mbps
4	8 Mbps	4.418 Mbps	5.086 Mbps

2.4.4 TCP with uniform flowrates and large simulation duration
commands: $ns fqdrr DRR 50.4 50 and $ns fqdrr FQ 50.4 50

Flow Id.	Generation Rate	DRR Throughput	FQ Throughput
1	13.74 Mbps	2.920 Mbps	3.428 Mbps
2	13.74 Mbps	2.970 Mbps	3.4276 Mbps
3	13.74 Mbps	2.862 Mbps	3.4273 Mbps
4	13.74 Mbps	2.752 Mbps	3.4271 Mbps

2.5 Observations

2.5.1 DRR throughput rates, in case of varying flow rates, are in proportion to the generation rate of the TCP source.
2.5.2 FQ throughput rates are more closer to the max-min fair allocation rates.
2.5.3 When flows generate at a uniform rate, FQ allocates bandwidth slightly more uniformly than DRR for small time durations.
2.5.4 The total throughput (of the entire node – n0) is marginally higher when FQ is used for scheduling than when DRR scheduling is used.
2.5.5 When simulation is run for a longer duration DRR and FQ more or less seem to be allocating a similar ratio of the bandwidth for non-uniform generating sources.
2.5.6 For longer simulation times and uniform generation, DRR and FQ both equal share to all their flows. But here again, over throughput is higher when using FQ than when using DRR.

2.6 Conclusions

FQ provides a fairer allocation of bandwidth.

3 TCP with RED

3.1 Topology [same as 2]

3.2 Execution Format

```
1. $ns red.tcl <RED|DropTail> <simulationTime (s)> [uniform_cwnd]
2. $nam out.nam –to view the simulation
Arguments:
 <RED|DropTail> : To choose with RED and without.
 <Simulation Duration in secs> :  default is 14.4secs
[uniform_cwnd] : optional parameter, to set all tcp sources to generate
packet at the same/uniform rate by keeping a uniform congestion window.
By default, the sources will have 2Mbps,4Mbps,6Mbps and 8Mbps rates.
Examples
$ns RED 10
$ns DropTail 15
$ns RED 23.5 8000 –here all ftp flows will generate at uniform rate.
$nam out.nam
```

3.3 Output Details

3.3.1 out.nam : Contains the trace output for nam animation.

3.3.2 redq.tr : Contains the trace of the queue being monitored at node n0.(n0-n1)

3.3.3 redq2.tr : Contains the trace of the queue being monitored at node n1.(n1-n2)

3.3.4 timewin* : This contains the details of the tcp congestion window for all flows.

3.4 Results

3.4.1 Varying TCP flow generation rates

3.4.1.1 Throughput values

Flow Id.	Generation Rate	Without RED (DropTail)	With RED
1	2 Mbps	0.936 Mbps	1.147 Mbps
2	4 Mbps	1.869 Mbps	2.115 Mbps
3	6 Mbps	4.652 Mbps	4.597Mbps
4	8 Mbps	6.182 Mbps	3.984 Mbps

3.4.1.2 Comparison of QueueSize vs Time

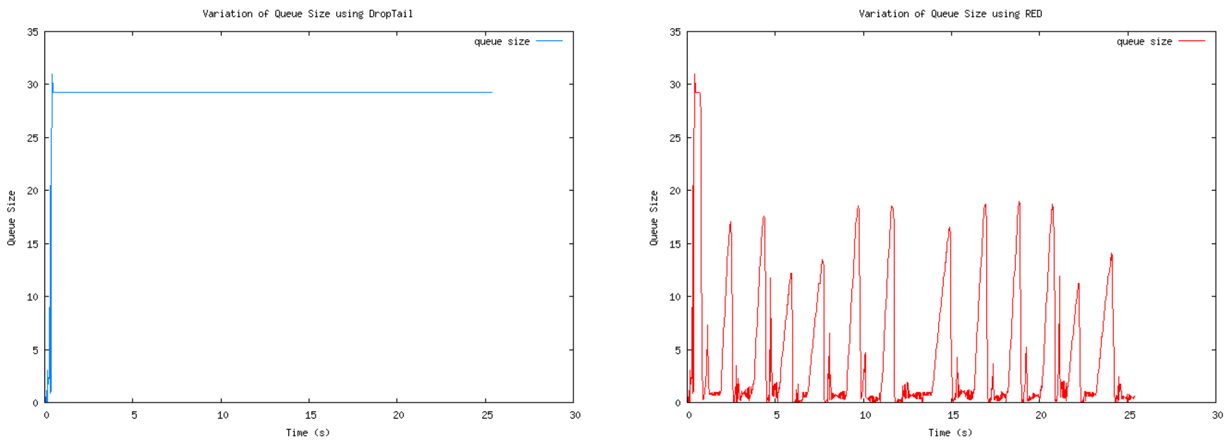
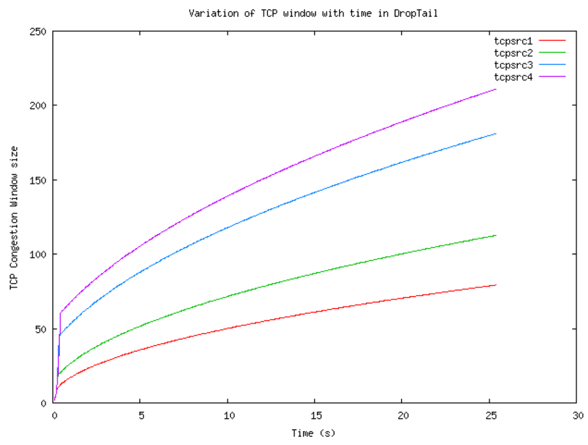


Fig: Comparison of Queue Sizes without RED and with RED

3.4.1.3 Comparison of Congestion Window vs Time



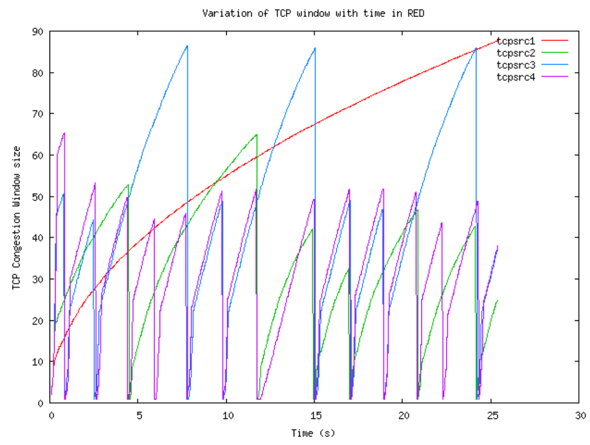


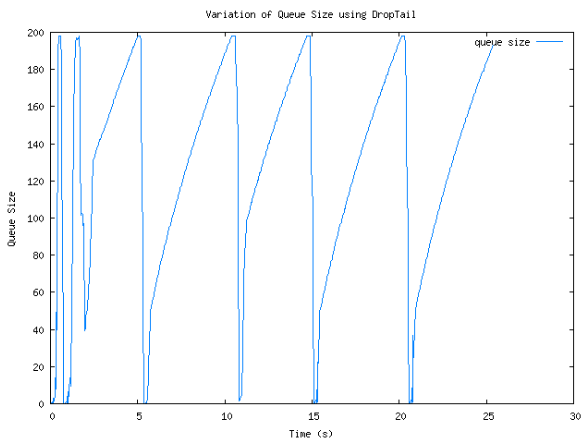
Fig: TCP congestion window variation without and with RED

3.4.2 Uniform TCP flow rates

3.4.2.1 Throughput Values

Flow Id.	**Generation Rate**	**Without RED (DropTail)**	**With RED**
1	13.74 Mbps	2.974 Mbps	3.147 Mbps
2	13.74 Mbps	4.265 Mbps	2.741 Mbps
3	13.74 Mbps	3.315 Mbps	2.395 Mbps
4	13.74 Mbps	2.616 Mbps	2.906 Mbps

3.4.2.2 Variation of Queue Size vs Time



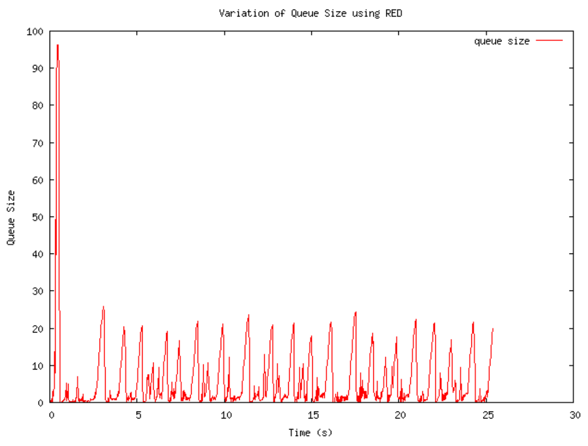
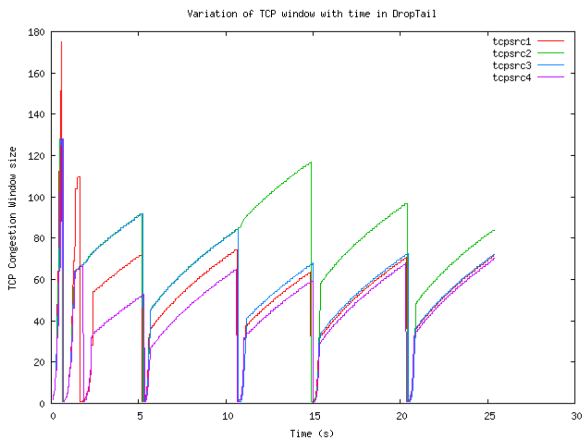


Fig: QueueSize vs Time without RED and with RED

3.4.2.3 Variation of Congestion window vs Time



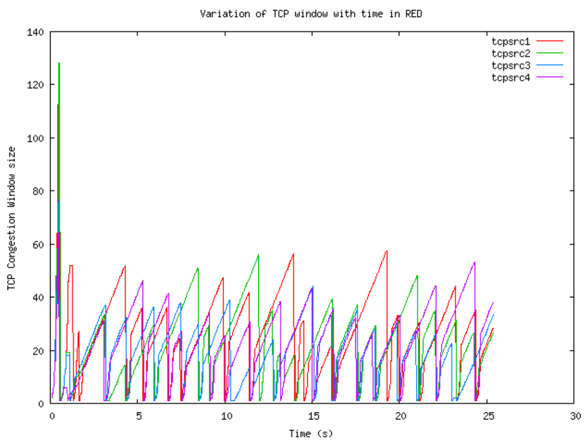


Fig: Variation of TCP congestion window without and with RED.

3.5 Observations

3.5.1 The throughput of the link with RED is lower than without it. But, on increasing the threshold values (thresh_ and maxthresh_) it was observed that the throughput matched to that of DropTail, particularly for larger congestion window sizes.

3.5.2 When generation rates of the flow are varying (but not all are backlogged) the queue size without RED is nearly constant.

3.5.3 With RED, for varying flow rates, the queue length is always between the maximum and minimum threshold values (the default maxthres_ is 15) and the correspondence of this to variation in queue length can be observed in the plot.

3.5.4 In case of simulation without RED, congestion window size of varying TCP flows correspond to the rate at which they are generating packets. And the window size keeps increasing gradually till it reaches a maximum.

3.5.5 With RED the congestion window sizes of the flows fluctuate, in a synchronized manner. We can clearly observe the “Sawtooth” in of tcp congestion window. Also, because RED is being used, different flows' windows peak and fall at different times.

3.5.6 In case of uniform flows, the queue size with just DropTail leads to the queue getting full and then falling and repeating over again. Whereas with RED, the queue size is more or less stable – in the range from 0 to 20 at all times.

3.5.7 Without RED the congestion window in uniform flows show an interesting pattern. i.e All of them increase and fall all at the same time. Which is also reflected in the queuesize variation.

3.5.8 Congestion window variation of flows with RED makes each source reach it's peak at different times. Hence it keeps the queue size more stable.

3.6 Conclusions

3.6.1 RED does not allow the average queue size to increase much. The variation in queue size is within a limited range, determined by the threshold values set.

3.6.2 DropTail don't perform well for large bursts of traffic, it causes the congestion window to decrease and start over from the beginning.

3.6.3 RED prevents synchronization in TCP congestion window. In case of DropTail all the tcp flows kept increasing and resetting their congestion window sizes at the same time. Whereas with RED, this was prevented and different flows reset their windows at different times. This ensures better bandwidth utilization.

3.6.4 Queuing delay in RED is lower. This follows from 1. Since, the average queue size is controlled well by RED, it ensures that the delay for the packets is low.

4 References

4.1 http://nile.wpi.edu/NS/ (NS by example)
4.2 http://www.isi.edu/nsnam/ns/
4.3 http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/ns.htm