Network and Complex Systems
ISSN 2224-610X (Paper) ISSN 2225-0603 (Online)
Vol.5, No.2, 2015

Analysis and Evaluation of Quality of Service (QoS) Router using Round Robin (RR) and Weighted Round Robin (WRR)

John Niyi Iyanda
ICT Department, Joseph Ayo Babalola University, Ikeji-Arakeji. Osun State.Nigeria
P.M.B 5006, Ilesa. Osun State. Nigeria
E-mail: johniyanda@gmail.com

Abstract
The paper discuses a scheduling system for providing Quality of service (Qos) guaranteed in a network using Round Robin (RR) and Weighted Round Robin. It illustrates the simulation and analysis of data by evaluating the performance of Round Robin (RR) and Weighted Round Robin (WRR) schedulers. The evaluation and analysis of this schedulers' is based on different parameters such as the throughput, loss rate, fairness, jitter and delay Also, in analysis and evaluation of the two scheduling using different charts to demonstrate the effects of each parameter in order to decide an efficient algorithm between Round Robin (RR) and Weighted Round Robin (WRR.).The simulated output of the experiment enabled us to determine different result of parameter used and proof the schedulers that are best to used and that will help in improving the Qos in differentiated services.
Keywords: Quality of Service(QoS), Round Robin (RR), Weighted Round Robin(WRR), Throughput, Scheduling, loss rate, fairness, jitter and delay

INTRODUCTION

Quality of Service refers to a set of rules or techniques that help the network administrators use the available network resources optimally to manage the effects of congestion and to treat the applications according to their needs. It also refers to the capability of a network to provide better service to selected network traffic over various technologies, including Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and IP-routed networks that may use any or all these underlying technologies. As computer networks grew, a desire for new multimedia services such as video conferencing and streaming audio arose. These applications were thought of as being workable only with support from within the network. In an attempt to build a new network that supports them via differentiated and accordingly priced service classes, A TM was designed. A best-effort network or service does not support quality of service. An alternative to complex Qos control mechanism is to provide high quality communication over a best-effort network by over-provisioning the capacity so that it is sufficient for the expected peak traffic load. Quality of service comprises requirement on all the aspects of a connection, such a service response time, loss, signal-to-noise ratio, cross-talk, echo, interrupts, frequency response, loudness levels, and so on. The queue management tool used for congestion avoidance raises priority by dropping lower priority flows before higher-priority flows. Policing and shaping provide priority to a flow by limiting the throughput of other flows. Link efficiency tools limits large flows to show a preference for small flows.

1. Bandwidth - the maximum data transfer rate possible between the source and the destination,
2. Delay or Latency - the time a packet takes to traverse from the source to the destination, and
3. Reliability or Packet loss- the average error rate of the source to the destinatoin

The QoS solution is largely aimed at controlling one or more of these metrics for a particular data communication session between a sender and a receiver. **In** a DiffServ network, these performance metrics are controlled by packet schedulers and droppers. DiffServ can be provided in an absolute or relative manner. **In** the former approach, DiffServ strives to achieve IntServ like performance guarantees; whereas in the latter, a traffic class is treated relative to another traffic class.

The relative service differentiation has several advantages over the absolute service differentiation. The absolute service differentiation makes use of several mechanisms including, admission controls, bandwidth brokerage, and resource reservation. Hence like lntServ, the absolute service differentiation suffers from scalability issues.

QoS tools can help alleviate most congestion problems. However, many times there is just too much traffic for the bandwidth supplied. Many things can happen to packets as they travel from origin to destination, resulting in the following problems as seen from the points of view of the sender and receiver;

- Dropped packets
- Delay
- Jitter
- Out-of-order delivery
- Error

1.1 Quality of Service Building Blocks

Services are constructed from somewhat independent logical building blocks. Depending on their specific instantiation and combination, numerous types of QoS architectures can be formed.

1.1.1 Packet classification: If any kind of service is to be provided, packets must first be classified according to header properties. For instance, in order to reserve bandwidth for a particular end-to-end data flow, it is necessary to distinguish the IP addresses of the sender and receiver as well as ports and the protocol number (this is also called a *jive-tuple).* Such packet detection is difficult because of mechanisms like packet fragmentation (while this is a highly unlikely event, port numbers could theoretically not be part of the first fragment), header compression and encryption.

1.1.2 Meter: A meter monitors traffic characteristics (e.g. 'does flow behave the way it should?') and provides information to other blocks.

1.1.3 *Policing:* Under certain circumstances, packets are policed (dropped) - usually, the reason for doing so is to enforce conforming behavior. For example, a limit on the burstiness of a flow can be imposed by dropping packets when a token bucket is empty. If the quality of services of the new flow can be satisfied without violating Qos of existing flows, the flow is accepted; otherwise the flow is rejected. The Qos may be expressed in terms of maximum delay, loss probability, delay variance, or other performance measures.

1.1.4 Admission control: Other than the policing block, admission control deals with failed requirements by explicitly saying 'no'; for example, this block decides whether a resource reservation request can be granted.

1.1.5 Marking: Marking of packets facilitates their detection; this is usually done by changing
Something in the header. This means that, instead of carrying out the expensive ***multi-jield classification*** process described above, packets can later be classified by simply looking at one header entry. This operation can be carried out by the router that marked the packet, but it could just as well be another router in the same domain. There can be several reasons for marking packets - the decision could depend on the conformance of the corresponding flow and a packet could be marked if it empties a token bucket.

1.1.5 Switching fabric: The switch fabric is the logical block where routing table lookups are performed and it is decided where a packet will be sent.

1.1.6 Queuing: This block represents queuing methods of all kinds - standard FIFO queuing or active queue management alike.

1.1.7 Scheduling*:* Scheduling decides when a packet should be removed from which queue. The simplest form of such a mechanism is a round robin strategy, but there are more complex variants; one example is *Fair Queuing (FQ),* which emulates bitwise interleaving of packets from each queue. Also, there is its weighted variant *WFQ* and *Class-Based Queuing (CBQ),* which makes it possible to hierarchically divide the bandwidth of a link. Its burstiness. A *leaky bucket* is a simple example of a traffic shaper: in the model, packets are placed into a bucket, dropped when the bucket overflows and sent on at a constant rate (as if there was holes near the bottom of the bucket). Just like a token bucket, this QoS building block is normally implemented as a counter that is increased upon arrival of a packet (the 'bucket size' is an upper limit on the counter value), and decreased periodically - whenever this is done, a packet can be sent on.

2.1 Weighted Round Robin (WRR)

Weighted Round Robin (WRR) is the foundation for a class of queue scheduling that is designed to address the limitations of the FQ and PQ models. WRR addresses the limitations of FQ model by supporting flows with significantly different bandwidth requirements. With WRR queuing, each queue can be assigned a different percentage of the output port's bandwidth. It also addresses the limitation of the strict PQ model by ensuring that lower-priority queues are not denied access to buffer space and output port bandwidth. With RR queuing at least one packet is removed from each queue during each service round.WRR overcome the limitation of FQ by scheduling service classes that has different bandwidth requirements. WRR overcomes the limitation of strict PQ by ensuring that lower-priority queue are not bandwidth starved.

WRR queuing ensures that all service classes have to at least some configured amount of network bandwidth to avoid bandwidth starvation. It also provides an efficient mechanism to support the delivery of differentiated service classes to a reasonable number of highly aggregated traffic flows.

2.2 Round Robin (RR):

Round Robin (RR) is one of the simplest scheduling algorithms for processes in an operating system, which assigns time slices to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is both simple and easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks.

The name of the algorithm comes from the round-robin principle known from other fields, where each person takes an equal share of something in turn. In the round robin scheduling, processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time-slice or a quantum. If a process does not complete before its CPU-time expires, the CPU is pre-empted and given to the next process waiting in a queue. The pre-empted process is then placed at the back of the ready list.

Round Robin Scheduling is pre-emptive (at the end of time-slice) therefore it is effective in time-sharing environments in which the system needs to guarantee reasonable response times for interactive users

2.3 Deficit Weighted Round Robin (DWRR)

Deficit Weighted Round Robin (DWRR) queuing was proposed by M. Shreedhar and G. Varghese in 1995.DWRR is the basis for a class of queue scheduling disciplines that are designed to address the limitation of WRR and WFQ models. DWRR addresses the limitation of the WRR model by accurately supporting the weighted fair distribution of bandwidth when servicing the queues that contain the variable-length packets.

DWRR addresses the limitation of WFQ model by defining a scheduling discipline that has lower computational complexity and that can be implemented in hardware. This allows DWRR to support the arbitration of output port bandwidth on high-speed interfaces in both the core and at the edges of the network.

2.4 Analysis of simulation result

In order to analyze the performance of our scheme and the differences between two different end-to-end Qos schedulers that will increase throughput, minimize delay, reduce loss, improve fairness, no jitter and help in improving the Qos routers for differentiated services internet model. Also to decide an efficient resource scheduling algorithm that would integrate the good features of Diffserv by evaluating its performances based on the following: throughput, minimize delay reduce loss; improve fairness, jitter and latency.

For the simulation of this experiment, we use ns-2 with OTcl as front-end and C++ as the backend, the compiler translates each line in the OTcl script to an event and the functionality of the events runs at the back-end (C++). The code written in C++ serves as the backbone, which was extended to bring out the functionality expected of the scheduler, also to provide classes that will provide a wide array of network features such as packet generator, queue management, routing and reporting. In order to actually achieve our said goal, the DiftServ architecture which provides Qos by dividing traffic into different categories, marking each packet with a code point, and scheduling packets accordingly was concentrated on.

The topology used in this experiment is shown below which show the network nodes, links, routers, bandwidth and delay

Notes: A DS_RED script that uses CBR traffic agents and the Token Bucket policer

```
#
#     ----
#    |s1| ------
#   ---10 Mb\
#       5ms \
#   \----                ----                    -----
# |el| ------------------|core| ---------------|e2|-----------|dest|
# /---------10Mb -------5Mb ----------10Mb ---------
# ----------------------/
# |s2| --------------------
# -----------10Mb
# 5ms
#
# ------------------------------------------------------------------
```

```
Set ns [new Simulator]

set cir O 1000000
set cbsO 10000
set rateO 4000000
```



```
set cirl 1000000
set cbsl 10000
set ratel 4000000

set testTime 85.0
set packetSize 1000

# Set up the network topology shown at the top of this file:
set s 1 [15ns node]                .
set s2 [$ns node]
set el [$ns node]
set core [$ns node]
set e2 [5ns node]
set dest [$ns node]

$ns duplex-link $sl $el 10Mb 5ms DropTail
$ns duplex-link Ss2 Sel 10Mb 5ms DropTail

Sns simplex-link $e1 Score 10Mb 5ms dsRED / edge
$ns simplex -link $core $e 110Mb 5ms dsRED / core
$ns simplex-link $core $e2 5Mb 5ms dsRED / core
$ns simplex-link $e2 Score 5Mb 5ms dsRED / edge

$ns duplex-link-op $s 1 $e1 orient down-right
$ns duplex-link-op $s2 $e1 orient up-right
$ns duplex-link-op $e1 Score orient right
$ns duplex-link-op $core $e2 orient right
$ns duplex-link-op $e2 $dest orient right

set qE1C [[$ns link $e1 $core] queue]
set qE2C [[$ns link $e2 $core] queue]
set qCE 1 [[$ns link $core $e 1 ] queue]
set qCE2 [[$ns link :lkore $e2] queue]


set ff3 [open dean-rr.tr w]

$ns trace-queue $core $e2 $ff3
 #$ns trace-all $ff3

# Set DS RED parameters from Edge1 to Core:
$qE 1C meanPktSize $packetSize
$qE 1C set numQueues_ 1
$qE1C setNumPrec 2
SqE1C addPolicyEntry [$sl id] [$dest id] TokenBucket 20 $cirO $cbsO
$qE1C addPolicyEntry [$s2 id] [$dest id]TokenBucket 10 $cir1 $cbs1
$qE1C addPoliccrEntry TokenBucket 10 11
$qE1C addPolicerEntry TokenBucket 2021
$qE1C addPHBEntry 1000
$qE 1 C addPHBEntry 11 0 1
$qE 1 C addPHBEntry 20 0 0
$qE1C addPHBEntry 21 01
$qE1C configQ 0 0 20 40 0.02
$qE1C configQ 01 10200.1

# Set DS RED parameters from Edge2 to Core:
SqE2C meanPktSize $packetSize
$qE2C set numQueues_ 1
$qE2C setNumPrec 2
```



```
$qE2C addPolicyEntry [$dest id] [$sl id] TokenBucket 20 $cirO $cbsO
$qE2C addPolicyEntry [$dest id] [$s2 id] TokenBucket 10 $cir1 $cbs1
$qE2C addPolicerEntry TokenBucket 10 11
$qE2C addPHBEntry 10 0 0
$qE2C addPHBEntry 11 0 1
$qE2C addPHBEntry 20 0 0
$qE2C addPHBEntry 21 0 1
$qE2C configQ 0 0 20 40 0.02
$qE2C configQ 0 1 10200.10

# Set DS RED parameters from Core to Edge 1:
$qCE1 meanPktSize $packetSize
$qCE 1 set numQueues_ 1
$qCE1 setNumPrec 2
$qCE1 addPHBEntry 10 0 0
$qCE1 addPHBEntry 11 0 1
$qCE 1 addPHBEntry 20 0 0
$qCE1 addPHBEntry 21 0 1
$qCE1 configQ 0 0 20 40 0.02
$qCE1 configQ 0 1 10200.10

# Set DS RED parameters from Core to Edge2:
$qCE2 setSchedularMode RR
$qCE2 meanPktSize $packetSize
$qCE2 set numQueues_ 2
$qCE2 setNumPrec 2
$qCE2 addPHBEntry 10 0 0
$qCE2 addPHBEntry 11 0 1
$qCE2 addPHBEntry 20 1 0
$qCE2 addPHBEntry 21 1 1
$qCE2 configQ 0 0 20 40 0.02
$qCE2 configQ 0 1 10 20 0.10
$qCE2 configQ 1 0 20 40 0.02
$qCE2 configQ 1 1 10 20 0.10

# Set up one CBR connection between each source and the destination: s
et udpO [new Agent/UDP]
$ns attach-agent $sl $udpO
set cbrO [new Application/Traffic/CBR]
$cbrO attach-agent SudpO
$cbrO set packecsize_$packetSize
$udpO set packetSize_ $packetSize
$cbrO set rate_ $rateO
set nullO [new Agent/Null]
$ns attach-agent $dest $nullO
$ns connect $udpO $nullO
set udp1 [new Agent/UDP]
$ns attach-agent $s2 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbrl attach-agent $udp1
$cbr1 set packecsize_ SpacketSize
$udp 1 set packetSize_ $packetSize
$cbrl set rate_ $rate1
set null1 [new Agent/ ull]
$ns attach-agent $dest $null1
$ns connect $udp1 Snull1

proc finish {} {
        global ns ff3
```



```
        close $ff3
        exit 0
}

$qE1C printPolicyTable
$qE 1 C printPolicerTable
$ns at 0.0 "$cbrO start"
$ns at 0.0 "$cbrl start"
$ns at 20.0 "$qCE2 printStats"
$ns at 40.0 "$qCE2 printStats"
$ns at 60.0 "$qCE2 printStats"
$ns at 80.0 "SqCE2 printStats"
$ns at $testfime "$cbrO stop"
$ns at $testTime "$cbrl stop"
$ns at [expr $testTime + 1.0] "finish"

$ns run
```

We looked at two nodes sl and s2 which are the nodes that inject different traffics into the network. Also there are two edge routers, e 1, e2 and a core router, core, receiving the traffic mark with a separate DSCP and performing necessary operations on it. From the three sources and the edges router, e 1, the bandwidth and delay are 10MB and **5ms** respectively. Between the edge router, el and the core router, core is bandwidth of 10MB and 5ms delay From core to edge router, e2 is bandwidth of 5MB and delay of 5ms in order to cre8 congestion in between them. And lastly is a bandwidth of 10MB and 5ms of delay between the edge router, e2 and the destination node, dest.

2.5 The Diffserv Module and Red Configuration

We decided to use token bucket as the policer which uses a CIR and a CBs and two d precedence. A policy is established between the sources and destination node. An arriving packet is marked with a lower precedence if and only if it is larger than the token bucket. The traffic parameters for the simulation are listed in the table below:

Table 1: DMRC

Token Bucket Parameters	Compliment Code Point(CP)	Non-compliment Code Point	Committed Information Rate(CIR)	Committed Burst Size (CBS)
Source I	10	11	1000000	10000
Source 2	20	21	1000000	10000

The table above show the token bucket has identified the source 1 with DSCP 10 downgraded to 11, source2 has DSCP 20 downgraded to 21. The committed Information Rate (CIR) for the two sources remains 1000000 bps with committed Burst rate (CBS) of 10000bytes.

The RED with in and out profile is used as a buffer management scheme for edge and core routers. We have queue 1 with CP (10 and 11), queue 2 with CP (20 and 21).

2.6 Evaluation of Result

The experiment as carried out for a total of 85secs with packets statistics. The actual output for the Round Robin (RR) and Weighted Round Robin (WRR) is shown respectively.

2.7 The Calculation Based On Throughput

The through result was obtained after running the simulation for 85secs. The through status of WRR for non-compliment is higher than that of RR because of the weight applied the queues. Also, for WRR, the throughput ratio of compliant and non-compliant packet 3:7, while that of the RR is also 3:7. However, all packets in the entire queue using RR scheduling were attended to and serviced almost equally. The effect of the weight applied to the queue of WRR made the different in the throughput compared to RR. With the application of this weight, we see from the chart the effect into the real life scenario, thus, guaranteeing absolute service delivery for all real time packets. The overall throughput for RR is 50,027,000 bytes and overall throughput for WRR is 39,957,000 bytes. We calculate the percentage work done for RR:

Table 2. The Total Packet Dequeue

1ST	2480	3783	2493	3771	12527
2ND	2500	3753	2500	3749	12502
3RD	2500	3750	2500	3754	12504
4TH	2500	3747	2500	3747	12494
TOTAL	9980	15033	9993	15021	50,027

Table 3 The Total Packet Enqueue

1ST ROUND	9994	9995	
2ND ROUND	19994	19995	
3RDROUND	29995	29995	TOTAL
4TH ROUND	39994	39995	79,989

% work done = Total Dequeue * 100%
Total Enqueue

% work done = (50027/79989)*100%
=62.54235

For WRR, percentage work done is given below:

Table 4. The Total Packet Dequeue

1ST	2389	1384	2508	6254	12535
2ND	2500	1246	2500	6259	12505
3RD	2500	1254	2500	6246	12500
4TH	2500	1247	2500	6239	12486
TOTAL	9889	5131	10008	24998	50,026

Table 5. The Total Packet Enqueue

1ST ROUND	9994	9995	
2ND ROUND	19994	19995	
3RD ROUND	29994	29995	
4TH ROUND	39994	39995	**79,989**

% work done = Total Dequeue * 100%
Total Enqueue

% work done = (50026/79989)*100%
=62.54109

Conclusion

The analysis and evaluation of the Round Robin (RR) and (WRR) with different kind of parameters that could help in achieving quality of service such as throughput, packet transfer delay, packet loss rate and jitter. The simulations of protocols were done, varying the parameters or configurations and quick explore the changing scenarios. On the other hand, I used a system programming language, like C++ that efficiently handles bytes and implement algorithm efficiently. The NS-2 which is made up of Otcl and C++ which helps to achieve the analysis and evaluation Round Rob (RR) and Weighted Round Robin (WRR) which can be viewed as doing two different things The performance of this network in managing delay, jitter, bandwidth provision and packet can better be achieved by a network administrator or architects. The result for the evaluation of different parameter used is based on the result generated of the higher throughput, a minimal packet loss rate, a scheduler with better fairness, low end-to-end delay and jitter. Finally, from the result Weighted Round Robin (WRR) is a better choice of Qos scheduler for the flow with higher weight, but using Round Robin (RR) may cause delay when thousands of queues are to be serviced on each port of a router.

References

[1] Chang. W.Z., Jianpin Y., Zhiping C. and Weifeng C. (2010).RRED: Robust RED Algorithm to Counter Low-rate Denial-of-Service Attacks, IEEE communications Letters, vol. 14, pages 489-491.
[2] Chiu.D. and Jain .R. (June 1989). "Analysis of the Increase/Decrease Algorin Congestion Avoidance in Computer Networks", Journal'of Computer ISDN, vol. 17, no. **1,** pages. 1-14.

[3] Chiu, D and Jain, K. R. Ramakrishnan. (Aug. 1987). Congestion avoidance in Computer Networks with a connectionless network layer". Tech. Rep. DEC- TR-506, Digital Equipments
[4] Corporation (August 1991), Gateway Congestion Control Survey.
[5] International Journal of Computer Science and Network Security, (January 200: No.1. Jain .R. (May 1990). "Congestion Control in Computer Networks issues and Tn Network Magazine, pages 24-25,
[6] John N. (1987), Fair Queuing
[7] Monarch Project Extension to NS-2, Retrieved (March 31, 2012).
[8] The Network Simulator-NS-2 Retrieved (March 31, 2012)

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
http://www.iiste.org

CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

Prospective authors of journals can find the submission instruction on the following page: http://www.iiste.org/journals/ All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

MORE RESOURCES

Book publication information: http://www.iiste.org/book/

Academic conference: http://www.iiste.org/conference/upcoming-conferences-call-for-paper/

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar