International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org, editorijaiem@gmail.com
Volume 2, Issue 7, July 2013 ISSN 2319 - 4847

QOS MANAGEMENT IN COMPUTER NETWORKING

[1]Sushruta Mishra, [2]Saswati Sahoo

[1,2]Gandhi Engineering College, BBSR

ABSTRACT

This paper presents the underlying concepts of managing Quality of Service in a computer network. It shows that techniques used, namely 1) over-provisioning; 2) buffering; 3) traffic shaping; 4) leaky bucket algorithm; 5) token bucket algorithm; 6) resource reservation; 7)admission control; are fundamental building blocks required for managing quality of services in a network. This paper provides basic understanding of all the techniques given above.

Keywords: Traffic shaping, best-effort network, over-provisioning, datagrams, bursty data, fair queueing algorithm, weighted fair queuing, buffering, token bucket, virtual circuit.

1. INTRODUCTION

QoS is a term widely used in the last recent years in the area of wire-based networks. QoS stands for Quality of Services and the truth is that there is much debate on what exactly QoS is supposes to mean. Most vendors implement QoS protocols having in mind speci_c scenarios and taking into consideration different parameters, network topologies and variables. The United Nations Consultative Committee for International Telephony and Telegraphy (CCITT) Recommendation E.800, has defined QoS as: "The collective set of service performance which determines the degree of satisfaction of a user of the service". This is a widely accepted definition since it doesn't make any reference to any minimum characteristics, such as Bandwidth or Delay. QoS is the capability of a network to provide better service to selected network traffic over various underlying technologies like Frame Relay, ATM, IP and routed networks etc. In other words, it is that feature of the network by which it can differentiate between different classes of traffic and treat them differently. QoS in a entire network involves capabilities in the the networks that carry the data being sent back and forth from one end host to another. The various formal metrics to measure QoS are

1 *Service Availability* : The reliability of users' connection to the internet device.
2. *Delay* : The time taken by a packet to travel through the network from one end to another.
3. *Delay Jitter* : The variation in the delay encountered by similar packets following the same route through the network
4. *Throughput* : The rate at which packets go through the network.

Any network design should try to maximize 1 and 4, reduce 2, and try to eliminate 3.

Today's networks need to support multiple kinds of traffic over single network links. Different kinds of traffic demand different treatments from the Network
Therefore, the need to design networks which
1. *can deliver multiple classes of service - that is they should be QoS conscious.*
2. *is Scalable - so that network traffic can increase without affecting network performance*
3. can support emerging network intensive, mission critical applications which will be the key determinant of a company’s success in the global world. Qos finds application in different fields of networking:

- *Streaming media specifically*
- *IP telephony also known as Voice over IP (VoIP)*
- *Videoconferencing*
- *Telepresence*
- *Storage applications such as iSCSI and FCoE*
- *Circuit Emulation Service*
- *Safety-critical applications such as remote surgery where availability issues can be hazardous*
- *Network operations support systems either for the network itself, or for customers' business critical needs*
- *Online games where real-time lag can be a factor*
- *Industrial control systems protocols such as Ethernet/IP which are used for real-time control of machineries*

2. QOS MANAGEMENT METHODS

The common methods for offering QoS discussed in this paper are over-provisioning, buffering, traffic shaping, resource reservation, admission control and packet scheduling. With the growth of multimedia networking, attempts at guaranteeing quality of service through network and protocol design are needed. No single technique provides efficient, dependable QoS in an optimum way. Instead, a variety of techniques have been developed, with practical solutions often combining multiple techniques[2]. One way of supporting QoS in the Internet is through traffic shaping, where routing of QoS packets does not follow the traditional IP routing protocols(i.e. OSPF and BGP), but instead takes into account available resources and expected traffic on the various network links. Following this approach, some paths can be over-provisionedand used for the most demanding packet flows (marked with the appropriate label), whereas others could be left for the best-effort traffic [1].

An independent but important aspect in QoS provision isthe existence and operation of policy and admission control inside the routers. Policy control determines whether the requesting user is entitled to make the requested reservation, while admission control determines whether the node has sufficient resources to facilitate the reservation. If both checks succeed, the requested reservation can be established by configuring the respective router parameters. The techniques for managing quality of service are described below[2].

1. OVER-PROVISIONING

An easy solution is to provide so much router capacity, buffer space, and bandwidth that the packets just fly through easily. The trouble with this solution is that it is expensive. As time goes on and designers have a better idea of how much is enough, this technique may even become practical. To some extent, the telephone system is over-provisioned. It is rare to pick up a telephone and not get a dial tone instantly. There is simply so much capacity available there that demand can always be met[4].

2. BUFFERING

Flows can be buffered on the receiving side before being delivered. Buffering them does not affect the reliability or bandwidth, and increases the delay, but it smoothes out the jitter. For audio and video on demand, jitter is the main problem, so this technique helps a lot[4]. In Fig. 1we see a stream of packets being delivered with substantial jitter. Packet 1 is sent from the server at t = 0 sec and arrives at the client at t = 1 sec. Packet 2 undergoes more delay and takes 2 sec to arrive. As the packets arrive, they are buffered on the client machine. Att = 10 sec, playback begins. At this time, packets 1 through 6 have been buffered so that they can be removed from the buffer at uniform intervals for smooth play. However, packet 8 has been delayed so much that it is not available when its play slot comes up, so playback must stop until it arrives, creating an annoying gap in the music or movie. This problem can be alleviated by delaying the starting time even more, although doing so also requires a larger buffer[4].

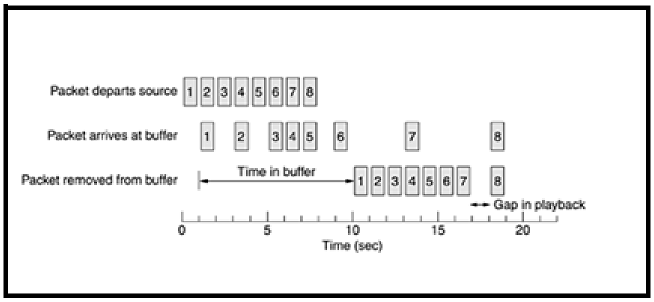

Fig. 1 Packets delivery with jitters

3. TRAFFIC SHAPING

Traffic shaping aims at limiting the variations of transmission rates within a negotiated per user flow rate andburstiness[3].Traffic shapingis a traffic managementtechnique which delays some or alldatagramsto bring them into compliance with a desiredtraffic profile.In the above example, the source outputs the packets with a uniform spacing between them, but in other cases, they may be emitted irregularly, which may cause congestion to occur in the network. Non-uniform output is common if the server is handling many streams at once, and it also allows other actions, such as fast forward and rewind, user authentication, and so on. Also, the approach we used here (buffering) is not always possible, for example, with videoconferencing. However, if something could be done to make the server (and hosts in general) transmit at a uniform rate, quality of service would be better. Traffic shapingsmooths out the traffic on the server side, rather than on the client side[4].

Traffic shaping is about regulating the average rate (and burstiness) of data transmission. When a connection is set up, the user and the subnet (i.e., the customer and the carrier) agree on a certain traffic pattern (i.e., shape) for that circuit. Sometimes this is called a service level agreement. As long as the customer fulfills her part of the bargain and only sends packets according to the agreed-on contract, the carrier promises to deliver them all in a timely fashion. Traffic shaping reduces congestion and thus helps the carrier live up to its promise. Such agreements are not so important for file transfers but are of great importance for real-time data, such as audio and video connections, which have stringent quality-of-service requirements[4].

4. THE LEAKY BUCKET ALGORITHM

Imagine a bucket with a small hole in the bottom, as illustrated in Fig. 2(a). No matter the rate at which water enters the bucket, the outflow is at a constant rate, r, when there is any water in the bucket and zero when the bucket is empty. Also, once the bucket is full, any additional water entering it spills over the sides and is lost (i.e., does not appear in the output stream under the hole)[4]. Figure 2.(a) A leaky bucket with water. (b) A leaky bucket with packets. The same idea can be applied to packets, as shown in Fig. 2(b). Conceptually, each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more processes within the host try to send a packet when the maximum number is already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulated by the host operating system. It was first proposed by Turner (1986) and is called the leaky bucket algorithm. In fact, it is nothing other than a single-server queueing system with constant service time[4]. The host is allowed to put one packet per clock tick onto the network. Again, this can be enforced by the interface card or by the operating system. This mechanism turns an uneven flow of packets from the user processes inside the hostinto an even flow of packets onto the network, smoothing out bursts and greatly reducing the chances of congestion. When all the packets are of the same size (e.g., ATM cells), this algorithm can be used as described above. However, when variable-sized packets are being used, it is often better to allow a fixed number of bytes per tick, rather than just one packet. Thus, if the rule is 1024 bytes per tick, a single 1024-byte packet can be admitted on a tick, two 512-byte packets, four 256-byte packets, andso on. If the residual byte count is too low, the next packet must wait until the next tick[4]

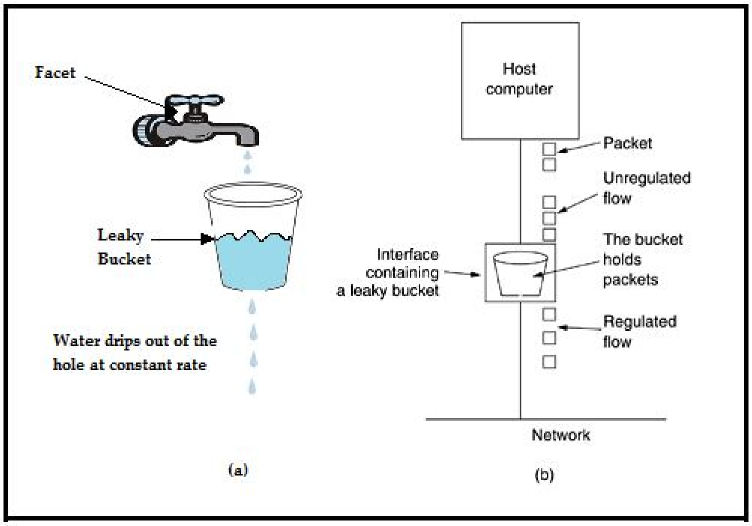

Fig. 2 A leaky bucket with water and a leaky bucket with packets

5. THE TOKEN BUCKET ALGORITHM

The leaky bucket algorithm enforces a rigid output pattern at the average rate, no matter how bursty the traffic is. It is important allow the output to speed up somewhat when large bursts arrive, so a more flexible algorithm is needed, preferably one that never loses data. One such algorithm is the token bucket algorithm. In this algorithm, the leaky bucket holds tokens, generated by a clock at the rate of one token every □T sec. In Fig. 3(a)we see a bucket holding three tokens, with five packets waiting to be transmitted. For a packet to be transmitted, it must capture and destroy one token. In Fig. 3(b)we see that three of the five packets have gotten through, but the other two are stuck waiting for two more tokens to be generated[4]. The token bucket algorithm provides a different kind of traffic shaping than that of the leaky bucket algorithm. The leaky bucket algorithm does not allow idle hosts to save up permission to send large bursts later. The token bucket algorithm does allow saving, up to the maximum size of the bucket, n. This property means that bursts of up to n packets can be sent at once, allowing some burstiness in the output stream and giving faster response to sudden bursts of input.

Another difference between the two algorithms is that the token bucket algorithm throws away tokens (i.e., transmission capacity) when the bucket fills up but never discards packets. In contrast, the leaky bucket algorithm discards packets when the bucket fills up. Here, too, a minor variant is possible, in which each token represents the right to send not one packet, but k bytes. A packet can only be transmitted if enough tokens are available to cover its length in bytes. Fractional tokens are kept for future use. The leaky bucket and token bucket algorithms can also be used to smooth traffic between routers, as well as to regulate host output as in our examples. However, one clear difference is that a token bucket regulating a host can make the host stop sending when the rules say it must. Telling a router to stop sending while its input keeps pouring in may result in lost data. The implementation of the basic token bucket algorithm is just a variable that counts tokens. The counter is incremented by one every T and decremented by one whenever a packet is sent. When the counter hits zero, no packets may be sent. In the byte-count variant, the counter is incremented by k bytes every T and decremented by the length of each packet sent. Policing all these schemes can be a bit tricky. Essentially, the network has to simulate the algorithm and make sure that no more packets or bytes are being sent than are permitted. Nevertheless, these tools provide ways to shape the network traffic into more manageable forms to assist meeting quality-of-service requirements[4].

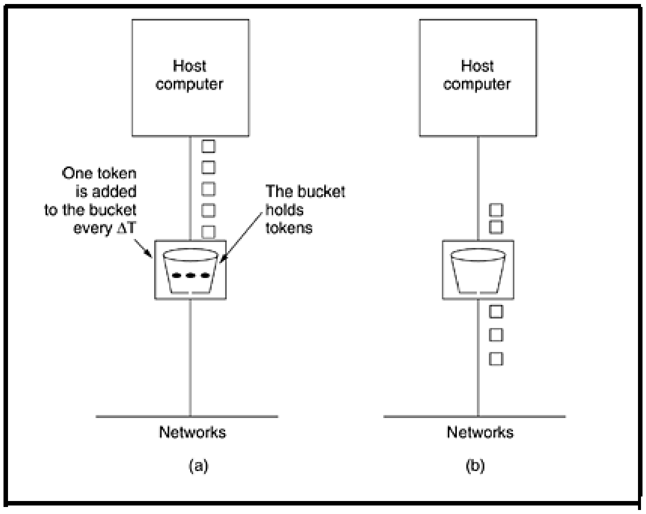

Fig. 3 Token bucket algorithm (a) before (b) after

6. RESOURCE RESERVATION

Being able to regulate the shape of the offered traffic is a good start to guaranteeing the quality of service. However, effectively using this information implicitly means requiring all the packets of a flow to follow the same route. Spraying them over routers at random makes it hard to guarantee anything. As a consequence, something similar to a virtual circuit has to be set up from the source to the destination, and all the packets that belong to the flow must follow this route. Once we have a specific route for a flow, it becomes possible to reserve resources along that route to make sure the needed capacity is available. Three different kinds of resources can potentially be reserved:

1. Bandwidth.
2. Buffer space.
3. CPU cycles.

The first one, bandwidth, is the most obvious. If a flow requires 1 Mbps and the outgoing line has a capacity of 2 Mbps, trying to direct three flows through that line is not going to work. Thus, reserving bandwidth means not oversubscribing any output line[4]. A second resource that is often in short supply is buffer space. When a packet arrives, it is usually deposited on the network interface card by the hardware itself. The router software then has to copy it to a buffer in RAM and queue that buffer for transmission on the chosen outgoing line. If no buffer is available, the packet has to be discarded since there is no place to put it. For a good quality of service, some buffers can be reserved for a specific flow so that flow does not have to compete for buffers with other flows. There will always be a buffer available when the flow needs one, up to some maximum. Finally, CPU cycles are also a scarce resource. It takes router CPU time to process a packet, so a router can process only a certain number of packets per second. Making sure that the CPU is not overloaded is needed to ensure timely processing of each packet. At first glance, it might appear that if it takes, say, 1 µsec to process a packet, a router can process 1 million packets/sec. This observation is not true because there will always be idle periods due to statistical fluctuations in the load. If the CPU needs every single cycle to get its work done, losing even a few cycles due to occasional idlenesscreates a backlog it can never get rid of[4].

7. ADMISSION CONTROL

Now we are at the point where the incoming traffic from some flow is well shaped and can potentially follow a single route in which capacity can be reserved in advance on the routers alongthe path. When such a flow is offered to a router, it has to decide, based on its capacity and how many commitments it has already made for other flows, whether to admit or reject the flow. The decision to accept or reject a flow is not a simple matter of comparing the (bandwidth, buffers, cycles) requested by the flow with the router's excess capacity in those three dimensions. It is a little more complicated than that. To start with, although some applications may know about their bandwidth requirements, few know about buffers or CPU cycles, so at the minimum, a different way is needed to describe flows. Next, some applications are far more tolerant of an occasional missed deadline than others. Finally, some applications may be willing to haggle about the flow parameters and others may not. For example, a movie viewer that normally runs at 30 frames/sec may be willing to drop back to 25 frames/sec if there is not enough free bandwidth to support 30 frames/sec. Similarly, the number of pixels per frame, audio bandwidth, and other properties may be adjustable. As there are many parties may be involved in the flow negotiation (the sender, the receiver, and all the routers along the path between them), flows must be described accurately in terms of specific parameters that can be negotiated. A set of such parameters is called a flow specification. Typically, the sender (e.g., the video server) produces a flow specification proposing the parameters it would like to use. As the specification propagates along the route, each router examines it and modifies the parameters as need be. The modifications can only reduce the flow, not increase it (e.g., a lower data rate, not a higher one). When it gets to the other end, the parameters can be established. As an example of what can be in a flow specification, consider the example of Table1. It has five parameters, the first of which, the Token bucket rate, is the number of bytes per second that are put into the bucket. This is the maximum sustained rate the sender may transmit, averaged over a long time interval. The second parameter is the size of the bucket in bytes. If, for example, the Token bucket rateis 1 Mbps and the Token bucket size is 500 KB, the bucket can fill continuously for 4 sec before it fills up (in the absence of any transmissions). Any tokens sent after that are lost.The third parameter, the Peak data rate, is the maximum tolerated transmission rate, even for brief time intervals. The sender must never exceed this rate.The last two parameters specify the minimum and maximum packet sizes, including the transport and network layer headers (e.g., TCP and IP). The minimum size is important because processing each packet takes some fixed time, no matter how short. A router may be prepared to handle 10,000 packets/sec of 1 KB each, but not be prepared to handle 100,000 packets/sec of 50 bytes each, even though this represents a lower data rate. The maximum packet size is important due to internal network limitations that may not be exceeded. For example, if part of the path goes over an Ethernet, the maximum packet size will be restricted to no more than 1500 bytes no matter what the rest of the network can handle[4].

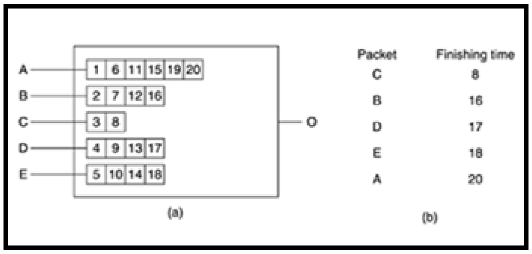

Fig. 4 . (a) A router with five packets queued for line O.
(b) Finishing times for the five packets

8. PACKET SCHEDULING

If a router is handling multiple flows, there is a danger that one flow will hog too much of its capacity and starve all the other flows. Processing packets in the order of their arrival means that an aggressive sender can capture most of the capacity of the routers its packets traverse, reducing the quality of service for others. To thwart such attempts, various packet scheduling algorithms have been devised. One of the first ones was the fair queueing algorithm. The essence of the algorithm is that routers have separate queues for each output line, one for each flow. When a line becomes idle, the router scans the queues using round robin, taking the first packet on the next queue. In this way, with n hosts competing for a given output line, each host gets to send one out of every n packets. Sending more packets will not improve this fraction. Although a start, the algorithm has a problem: it gives more bandwidth to hosts that use large packets than to hosts that use small packets. An improvement is done to this algorithm in such a way so as to simulate a byte-by-byte round robin, instead of a packet-by-packet round robin. In effect, it scans the queues repeatedly, byte-for-byte, until it finds the tick on which each packet will be finished. The packets are then sorted in order of their finishing and sent in that order. The algorithm is illustrated in Fig.4. In Fig.4(a)we see packets of length 2 to 6 bytes. At (virtual) clock tick 1, the first byte of the packet on line A is sent. Then goes the first byte of the packet on line B, and so on. The first packet to finish is C, after eight ticks. The sorted order is given in Fig.4(b). In the absence of new arrivals, the packets will be sent in the order listed, from C to A. One problem with this algorithm is that it gives all hosts the same priority. In many situations, it is desirable to give video servers more bandwidth than regular file servers so that they can be given two or more bytes per tick. This modified algorithm is called weighted fair queueing and is widely used. Sometimes the weight is equal to the number of flows coming out of a machine, so each process gets equal bandwidth[4].

3. CONCLUSION

The key contributions of this paper were to provide details for managing the quality of services in a network. Network quality of service is a critical factor of a successful converged networking design. Although over-provisioning the network bandwidth may provide adequate QoS temporarily, additional mechanisms including traffic shaping, buffering and admission control should be designed into the network infrastructure. Significant advances are constantly taking place to increase the capabilities and its use in wireless devices also.

REFERNCES

[1.] Atsushi Iwata and Norihito Fujita, “A Hierarchical Multilayer QoS Routing System with Dynamic SLA Management”, IEEE Journalon Selected Areas in Communication, Vol. 18, No. 12, pp. 2603-2616, December 2000.
[2.] Dimitra Vali, Sarantis Paskalis, Lazaros Merakos andAlexandros Kaloxylos, “A Survey of Internet QoS Signalling”, IEEE Communication Surveys and Tutorials, Vol. 6, No. 4, pp. 32-43, 2004.
[3.] Theofanis G. Orphanoudakis, Christos N. Charopoulos and Helen Catherine Leligou, “Leaky-Bucket Shaper Design Based on Time Interval Grouping”, IEEE Communications Letters, Vol. 9, No. 6, pp. 573-575, June 2005.
[4.] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, March 2003.