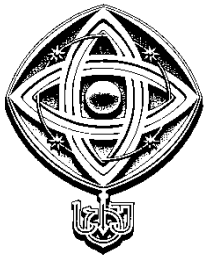
Georgian Technical University

M.Kartvelishvili,O.Kartvelishvili

QoS AND PACKET QUEUEING STRATEGIES IN MULTISERVICE HETEROGENOUS TCP/IP NETWORKS



Tbilisi
2009

Introduction

Data networks became one of the most important branches of IT industry. Social life cannot be considered without global data exchange. The history of computer networks began in early 70s when several nodes were joint together with 56kbps links and formed ARPANET, which was initially military experimental project. This was precursor of what is called Internet today.

Today Internet covers almost every country in the world allowing millions of people to obtain grains from the enormous global knowledge base.

In the beginning networks were intended for pure data exchange among nodes. Almost all the applications were tolerant to the lack and variation of data path parameters. If there was an application requiring certain level of service, it should take path other than for standard traffic. This meant activating of additional physical circuits, which leaded to inefficient use of network capacity. But there were not a big number of such applications. Data and voice networks were absolutely separate using different hardware and circuits.

These days situation has changed. Data and voice are merging rapidly, sharing the same infrastructure. Growing network capacity brought possibility to run many new services, which introduce different requirements to the network parameters. The clear example of such a service is video conferencing (duplex) and video streaming (simplex). There is also a demand for integration of legacy technologies, like SNA, into contemporary network infrastructure allowing combination of processing power of mainframes with capacities of today's data links.

Finally, with the grows of popularity of enterprise networks – so called intranets, there appeared a demand for secure connectivity of several geographically distributed branch offices with the headquarters allowing to exchange important data over public facilities (or private facilities of another company) between them as if there are physical connections. This type of network is also called Virtual Private Network – VPN.

All these gave birth to multiservice networks, which extended functionality of legacy networks. In such networks several applications with different requirements are launched sharing the same network infrastructure. To fulfill these requirements there must be some way to differentiate between different services at the edges and in the backbone.

This differentiation is known as "Quality of Service" - QoS. This is capability of network to dynamically respond to application requirements allocating necessary bandwidth and maintaining other network parameters.

As the global networks use different technologies as their data-link layer, which may or may not support QoS, some mechanism is needed to provide QoS at the network layer, because users need to have end-to-end guarantees of network parameters. This type of networks, that use different data link layer protocols, are called heterogeneous networks. In most cases end users are not interested what

technologies are used in the backbone or along the data path. That's why it is necessary to separate QoS functionality from Data Link layer up to the network layer.

The main advantage of moving QoS functions to upper layers is the opportunity to directly use network (and upper) layer packet header fields for traffic differentiation. In case of data link layer QoS the mapping of some subset of traffic into certain virtual circuit must be configured statically at each switched network node and then QoS parameters applied to these VCs. From this point of view in network layer QoS more flexible policies can be specified. This feature is known as "Content Aware Networking". There is no need for data link virtual circuits any more. Of course, this also reduces network complexity.

One of the most widespread network layer protocols in the world is IP. But originally IP didn't support any QoS features. Additional mechanisms are needed to make the service differentiation work over IP networks. Along the data path IP may use ATM or frame relay supporting some QoS or Ethernet and token ring, PPP or HDLC have not a slight idea what QoS is.

In this environment deploying QoS mechanisms in such heterogeneous IP networks is very significant topic. This will allow using the same media, as a transport for different types of traffic providing corresponding required characteristics for each of them leaving unused bandwidth for best effort traffic.

Another topic is providing of QoS signaling, such as RSVP and MPLS, throughout complex networks, which give the possibility for network to dynamically change it's traffic patterns according to the changing application demands and congestion situations in the core.

Improvement of QoS features will cause the birth of new services that weren't possible using old methods of congestion management. This will open new future perspectives for WANs as multiservice integrated environments for data exchange.

This document mainly provides overview and analysis of different methods of congestion management giving possibility of maintaining QoS parameters. There are also some examples illustrating the methods of design of multiservice networks and showing the actual traffic distribution in different cases.

Most of the features shown in this document are not completely standardized and are still vendor specific. Partially the work is based on corresponding IETF (Internet Engineer Task Force) drafts, which are still under development. The sources also include documents by Cisco Systems – network equipment manufacturer, leader in this field.

This document doesn't cover every aspect of the topic, as it is very wide, but it is sufficient for designing of networks with IP QoS deployed. In Georgia the need for QoS just appeared on the IT market and I think will develop very rapidly.

1. Traffic Management

Data networks are becoming more and more complex. That's why it is necessary to predict load distribution and traffic patterns during network design and to have the possibility to manage its flow in time.

First steps in this direction were made when in 1986 the new flow control model for TCP protocol was proposed by Van Jacobson for congestion management in Internet. Much later appeared demand for differentiated services. In this chapter are discussed the problems related to this topic.

1.1. Congestion Control

Congestion is the state of network node when incoming traffic destined to certain interface is more than its physical capacity. Presence of congestion in the network means that available resources are not sufficient to process offered load. In most cases congestion leads to the packet drops. For most types of traffic (except real time streams like voice and video) sources have to retransmit lost packets making congestion situation even worse. As a result network will meet significant performance degradation, especially if two or more consequent packets per session are lost.

Here appears a need of some congestion control mechanism, which means that in case of congestion flows experiencing packet drops must slow down their transmission rate. This behavior is called flow control. Many protocols such as TCP and HDLC include this functionality, but this is not sufficient. The main danger is that not all protocols are behaving correctly.

In this document the main focus is on TCP/IP networks and that's why in this section will be considered TCP flow control mechanism.

In multiservice networks, where several protocols are usually used at the same time there is a possibility that some of them will employ flow control features and some of them not. In this scenario "misbehaving" streams may monopolize bandwidth making "correct" applications starve for bandwidth. As it will be shown in one of the following sections this problem can be solved using intelligent queuing technique and per flow scheduling.

All this leads to performance degradation – so called congestion collapse. It occurs when an increase in load offered to network leads to dramatically decrease of useful throughput performed by this network. There are two main reasons of congestion collapse to occur: The first reason is the presence of unnecessary packet retransmissions and the other is undelivered traffic.

First reason takes place when flow control algorithm is not optimized and can be easily solved by adjusting retransmission timers. The second problem is more complex. It happens if lots of packets are dropped just prior to destination after crossing the whole network wasting valuable bandwidth. This is extremely dangerous for protocols without flow control, which can not adapt to congestion along the data path.

Let's consider the following case shown in Fig. 1.

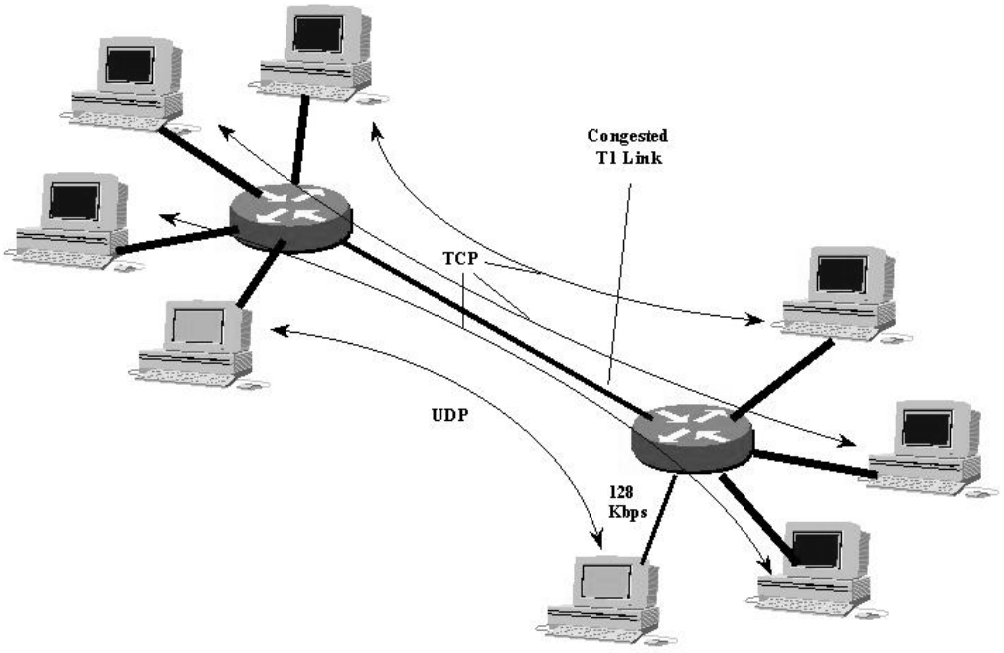


Fig. 1

Network backbone is represented by one T1 (1.544Mbps) link, which is in state of congestion. There are 4 senders and 4 receivers at each side. 3 of these sender-receiver pairs have TCP sessions between them; the last pair has UDP session. All the end nodes, except UDP traffic receiver, are connected to backbone by 10Mbps access links. UDP traffic receiver has 128Kbps link to network, which is only 8.4% of congested link bandwidth. TCP load is assumed to be constant and the entire network is using FIFO queuing. Here is the table of network goodput for TCP, UDP and total flows for different UDP loads offered. All the statistics are given as percentage of the capacity of congested link.

Table 1. Network load statistics for different incoming UDP rates

UDP Arrival Rate %	UDP Goodput %	TCP Goodput %	Total Goodput %
0.7	0.7	98.5	99.2
1.8	1.7	97.5	99.1
2.6	2.6	96.0	98.6
5.3	5.2	92.7	97.9
8.8	8.4	87.1	95.5
10.5	8.4	84.8	93.2
13.1	8.4	81.4	89.8
17.5	8.4	77.3	85.7
26.3	8.4	64.5	72.8
52.6	8.4	38.1	46.4
58.4	8.4	32.8	41.2
65.7	8.4	28.5	36.8
75.1	8.4	19.7	28.1
87.6	8.4	11.3	19.7
105.2	8.4	3.4	11.8
131.5	8.4	2.4	10.7

In Fig. 2 is shown the graphical representation of the results of the experiment for clearer understanding of the case.

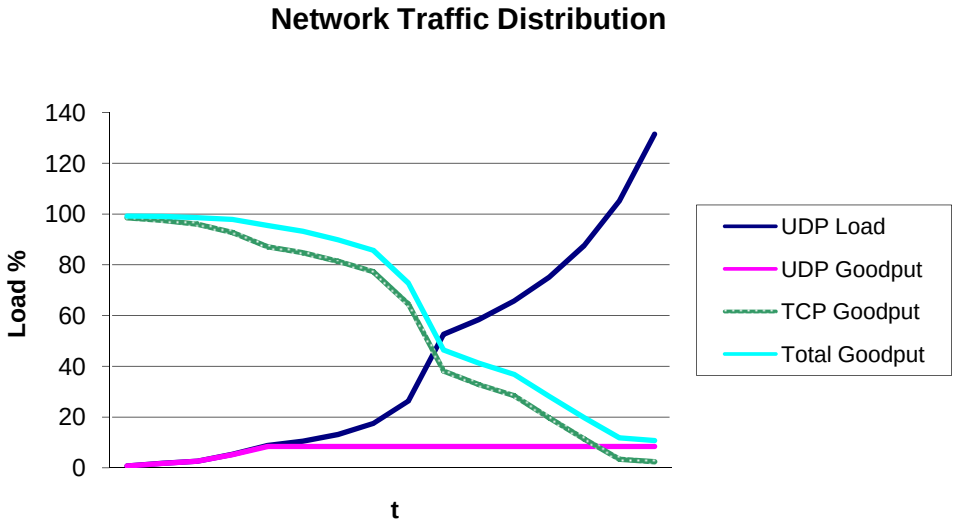


Fig. 2

From the statistics shown above we can see that increase of incoming UDP traffic above the rate of receiver's access link doesn't lead to increase of UDP goodput any more. It only results the absolute degradation of TCP goodput and of total goodput as well. It has to be noted that TCP fast performance degradation almost stops at the end of the curve as soon as UDP load reaches 100% of backbone capacity. No more traffic is allowed into congested link and the rest of UDP traffic is dropped before getting the bandwidth. These results also show the unfairness of bandwidth distribution among separate flows.

This problem can be reduced by using more complex queuing strategies like weighted fair queuing or by adding flow control features to upper layer protocols that are using UDP as transport.

This shows that congestion control is very important subject concerning network performance, but it becomes insufficient when there is a need for traffic differentiation. New QoS measures are to be deployed in the network to make this differentiation possible.

1.2 QoS Concepts

These days in many cases fair traffic flow distribution is not sufficient any more. With the appearance of real-time traffic, which required different treatment at the network nodes, it became necessary to create possibility to differentiate among different traffic flows. That is different types of traffic need different levels of QoS.

Here are some of the benefits offered by QoS feature in networks:

- ❖ Control over network resources that significantly boosts the efficiency of network usage and also increases flexibility of network management.
- ❖ Coexistence of mission critical applications even in situations of heavy congestion by creating different policies for different traffic types.
- ❖ More predictable traffic behavior reducing unpredicted traffic bursts leading to congestion in the network.

As shown in the Fig 3. QoS architecture includes three components:

1. QoS within a single network node. This task can be completed using several queuing and/or congestion avoidance mechanisms. This methods have to be deployed at every hop router inside QoS network, that's why this component of QoS architecture is also called "Per Hop Behavior" (PHB).
2. QoS signaling. Some information must be exchanged among network nodes enforcing QoS policy and end-nodes. The information received from end-nodes must include requests for certain types of QoS. As a response to such requests network nodes have to make some reconfiguration to serve the traffic from this end-nodes. The information exchanged between network nodes must include current states of links, especially load, unused bandwidth and optionally packet propagation delay, delay variation, link reliability, etc. This information allows choosing the optimal path, which is capable of delivering the requested level of QoS.
3. QoS policy, management and accounting. These features allow controlling and administering end-to-end traffic across a network, monitoring its performance.

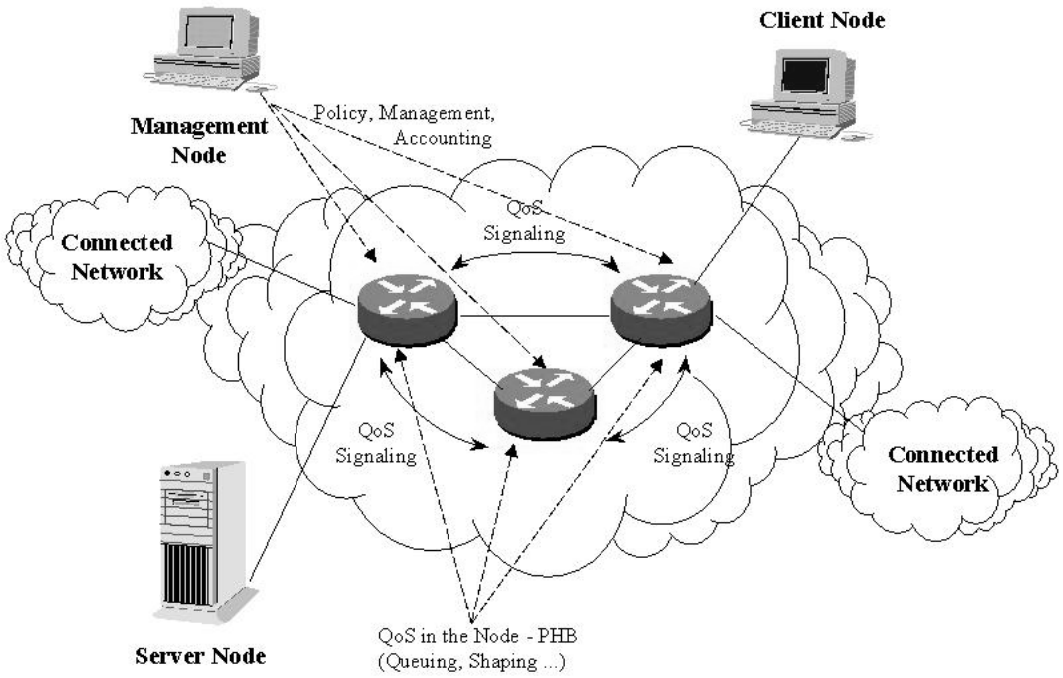


Fig. 3

Network traffic in QoS network may be split into three categories:

- Best-effort traffic, which is served when the network is idle and there is no traffic of other types.
- Differentiated traffic, which is given a priority over best-effort traffic. It has no special bandwidth guarantee in the network, but it is dequeued first.
- Guaranteed traffic is usually given the highest priority and it will always get its guaranteed bandwidth even if the other types of traffic are congested, but not more than that. Guaranteed traffic exceeding its bandwidth limit is likely to be dropped.

The following network traffic diagram presented in Fig. 4 illustrates the general concepts concerning different traffic types.

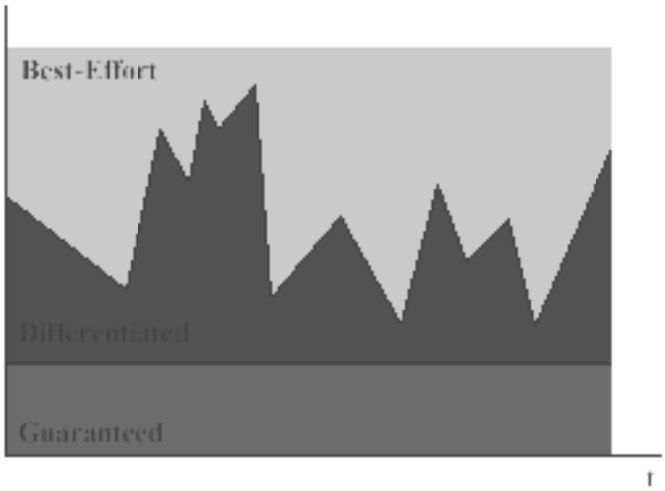


Fig. 4

As we can see in some cases best-effort (and even differentiated) traffic may starve for bandwidth, but only because of poor network design. It is recommended to make some undersubscription of guaranteed and differentiated services to allow best-effort traffic to receive minimal level of service even in case of congested links. Of course the universal way of removing congestion situations is link upgrade to higher rates.

1.3 Architecture of Differentiated Services

The main building block of differentiated services (or DiffServ in short) is ToS field in IP packet header. This 8 bit field in this environment is called DS field. Currently only low 6 bits also known as differentiated service code point (DSCP) are used for service differentiation. The rest of bits (high order 2 bits) are currently unused. The location and structure of DS field if shown in Fig. 5.

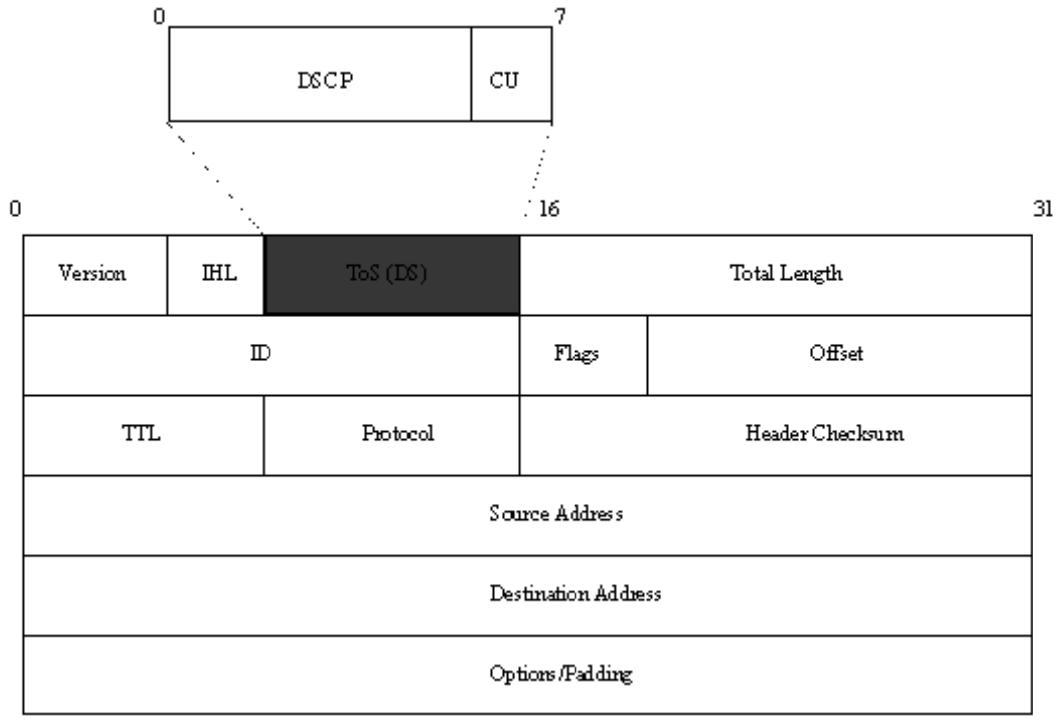


Fig. 5

Before the creation of DS specifications the recommended behavior for ToS field was to use 3 low order bits for packet classification, giving 8 possible classes of traffic. This 3-bit field was called IP Precedence. All the classes were mapped to predefined per hop behaviors (PHBs) according to a rule: higher IP precedence values were mapped to higher priority PHBs, lower values mapped to lower priority PHBs. The default IP precedence value was 0, which corresponded to best-effort traffic.

Currently some compatibility with IP precedence is still retained. 3 low order bits of DS field are called Class Selector and are mapped to PHBs according to the same rule as described above and default DS value is 0. This allows DS compliant nodes to interact with IP precedence compliant nodes.

DSCP field value space is divided into 3 groups. Their meanings are displayed in Table 2.

Table 2. DSCP space division.

xxxxx0	Standard Action
xxxx11	Experimental/Local Use
xxxx01	EXP/LU (possible Standard Action)

Another important element of DiffServ architecture is DS domain. DS domain is a contagious set of DS-capable nodes using the same traffic policy. The nodes in the DS domain can be divided into two groups:

1. Boundary nodes that are on the either sides of the link connecting two DS domains or connecting DS domain to non-DS capable network. Almost all the work is done at this point including traffic classifying, conditioning and marking.
2. Interior nodes that are inside DS domain and mainly do traffic classification according DS field in the IP header. After this classification certain PHB must be applied to the packet. PHB maybe implemented by using some queuing mechanism or some dropping strategy, or combination of them.

The main concepts of the diffserv architecture are the following:

1. Any packet arriving to the DS domain ingress boundary node it is classified according to prespecified set of rules, then conditioned to conform to internal domain QoS policy and then optionally marked with some DS value. Traffic conditioning may include: meter to find temporal characteristics of traffic stream; policer and shaper that will limit traffic flows to predefined temporal values either by delaying or dropping excess traffic; dropper to perform drops of excess traffic according to some dropping policy; and marker that marks packets identified for some level of service with certain DS value before transmitting them into domain. The complete scheme of functions of boundary node is shown in Fig.6.

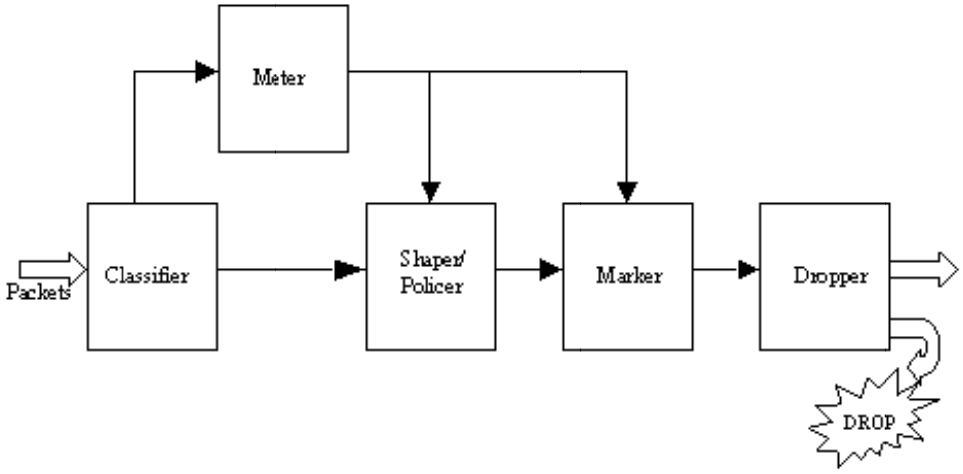


Fig. 6

2. Packets allowed to enter DS domain are routed over interior nodes and treated according to configured PHB mappings, which may be implemented using several queuing and dropping strategies. Packets are routed toward egress boundary node of DS domain.
3. At the egress boundary node the same steps may be performed including packet re-marking (recolouring), which depends on the Service Level Agreement (SLA) with another DS domain to which packet is moving. All these steps may be skipped if these two neighboring DS domains are using the same QoS policy.

All the features implemented in boundary nodes may be also included into interior nodes, but in this case the network may become less scalable.

As it was mentioned above Service Level Agreement (SLA)/Traffic Conditioning Agreement (TCA) must be signed between customer and service provider or between two neighboring DS domains, which will determine traffic policy at boundary nodes. These agreements will include traffic classes (i.e. Gold, Silver, Premium, etc) and service level they will receive in destination DS domain.

2. Requirements of Different Types of Traffic

As it was mentioned above different traffic types need different network resources and parameter guarantees and as a result different network design. Generally traffic can be divided into following categories to be considered separately:

- File transfers and general data transfer applications;
- Digital audio;
- Digital Video;
- Circuit emulation and VPNs.

What network parameters affect service quality over the network? Here are some of them:

Consumed bandwidth is minimal throughput of the data flow to achieve service delivery with sufficient quality. It is usually measured in 'kilobits per second' (kbps).

Delay is the amount of time needed for packet to reach its destination. This characteristic is usually applied to unidirectional streams. For bi-directional streams roundtrip time (RTT) is mainly used, which is consisted of packet delays in both directions. Delay and RTT are measured in seconds (or milliseconds). Packet delay can be divided into several components: signal propagation over wire, which is equal to the speed of light and adds a very little delay, so can be ignored; packet send delay is introduced at each link because of finite link speed; processing delay is added in routing and switching devices during determination of output link and is limited by processing power of each node; queuing delay which appears during congestion when packets are buffered into queues and waiting there for transmission.

Jitter is defined as end-to-end delay variation, which means that packet delay may change in time, which may be caused by temporal bursts of traffic from other sources in poorly designed networks.

Loss ratio defines how many packets are dropped during data transfer. This may be caused by queue overflows during congestion, transfer errors or administratively set drop policies. It is usually referred as percentage of lost packets out of totally sent. Sometimes loss ratio represents probability of the fact that packet will be dropped in the network.

2.1. File transfers and general data transfer applications

These applications include classical Internet applications like email, web, and ftp file transfers. These applications are very tolerant to all network parameters and don't usually require certain level of QoS. So classical best effort behavior introduces by IP protocol absolutely satisfies their demands. On the other hand traffic types that need higher degrees of guarantees must be separated from the best effort traffic. The main reason for this is the fact that best effort traffic can consume all the resources, causing other traffic flows to starve for bandwidth. Even fair flow distribution is not sufficient, as quantity of best effort flows is almost unpredictable at any moment of

time and as the result real-time flows may experience significant performance degradation.

All this arguments show the necessity to differentiate between ordinary data flows and real-time interactive flows that require some level of QoS.

2.2 Digital Audio

Audio streams are usually represented by Voice over IP (VoIP) technology, which becomes very popular these days. Voice by its nature is analog signal and of course is needs to pass through digitization process before transmitting over data networks. This digitizing process is usually performed by special software or hardware coders/decoders (CODECs), which digital audio output as series of samples of analog source. The functional scheme of codec is illustrated in Fig. 7.

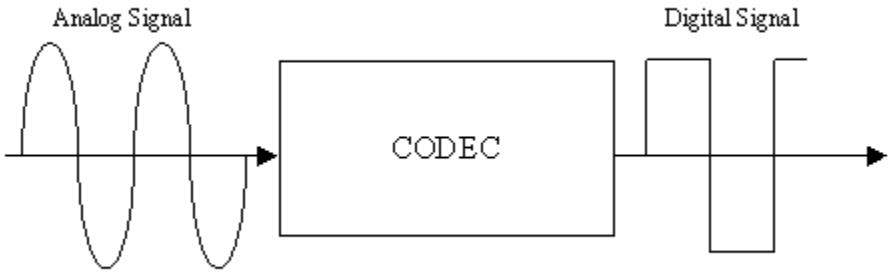


Fig. 7

Standard uncompressed voice stream needs 64kbps (56kbps) to be transmitting accurately. According to Nyquist's theorem analog signal must be sampled at least at twice its initial rate. As voice human voice bandwidth is 4000Hz (300Hz-4300Hz) it must be sampled 8000 times per second. If 8 bits are taken for each sample, the total stream will be 64kbps.

Today many compression standards and algorithms are introduced allowing reducing the bandwidth needed for audio stream transmission. Compression is usually lossy, that is compressed and decompressed streams may differ introducing some quality degradation, which depends on compression algorithm and ratio.

Voice traffic is very sensitive to delay. End-to-end delay must be kept under 200 milliseconds, so RTT must not exceed 400ms. After digitization of audio signal needs to be put into packets or frames, which may introduce additional delay. This framing delay in classical 64kbps stream is 0.125ms as each byte is transmitted separately, but in other standard is usually larger. Another type of delay is added during process of compression, which mainly depends on complexity of the algorithm. Bandwidth and framing and compression delay requirements of different audio coding standards are

listed in Table 3. The detailed description of audio compression standards is beyond the scope of this document.

Table 3. Comparison of different voice compression standards

Compression Method	Bit Rate (kbps)	Framing delay (ms)	Compression delay (ms)
G.711 PCM	64	0.125	0.75
G.729 CS-ACELP	8	10	10
G.723.1 MP-MLQ	6.3	30	30
G.723.1 ACELP	5.3	30	30

Audio traffic is tolerant to packet loss and only after loss of 2-3 consequent packets it experiences significant quality degradation. On the other hand audio is very sensitive to jitter which may voice clicks and distortion. This problem maybe solved using buffering at the receiver. Packets experiencing jitter are places into fixed-length buffer from which they are extracted at constant rate. Of course, the mean packet arrival speed must be greater or equal to speed of packet extraction from buffer. Otherwise the traffic will constantly get into buffer underruns which will also affect voice quality.

2.3 Digital Video

Digital is also real-time type of traffic as it was in case of voice and mostly has similar problems as audio streams. Video stream is usually transmitted as series of image frames or still pictures and if transmitted uncompressed consumes a large amount of bandwidth. For digital video the following components are critical:

- Resolution – The horizontal and vertical dimensions of the video sessions measured in pixels. The full screen video streams usually have 640x480 resolution.
- Color depth – The number of bits that are used for color presentation. High quality video streams use 24 bit color depth which is capable displaying 16.7 million colors and low quality video uses 8 bit depth with 256 colors respectively.
- Frame rate – The number of frames (pictures) that are displayed per second. This component usually varies from 25 fps to 30 fps.

Bandwidth required for uncompressed video stream can be easily calculated. High quality (24 bit) full screen video stream will consume 640 x 480 x 3 x 30 = 27.648 MBps = 221.184 mbps which is not a small value. To avoid such enormous load on network the following techniques are used:

- Video Capture Manipulation
- Video Compression

Video capture manipulation is applied at video source or codec. Reduction of bandwidth consumption is achieved by reducing video stream quality. In case of video session at 320x200 resolution, 8-bit depth and 15fps rate the bandwidth

requirement drops to 9.216Mbps. This level of bandwidth is achievable in contemporary LAN technologies.

Video compression is a set of algorithms that allow source video stream to be transmitted in more compact way. The effectiveness of compression algorithm may be evaluated by how the algorithm affects video stream quality and how effectively it can reduce video data rates without significant quality loss. Compression may be implemented in software or in hardware. Video compression mainly uses lossy compression algorithms, which means that after transmission the video data stream is not identical with initial one. This gives the opportunity to obtain greater compression ratios – from 2:1 to 300:1 compression. Compression methods may be divided into two groups:

- Interframe compression – Compression applied between frames, also known as temporal compression, because it is applied along the time dimension. This method eliminates redundant information between frames. The video stream is transmitted as sequence of key frames (compressed or uncompressed) and interleaved with delta frames which contain only changes in key frames or other delta frames. There are several standards of interframe compression algorithms: These are:
  – MPEG1 – This compression is optimized f or 1.5 mbps bandwidth, which coincide with audio CD data rate.
  – MPEG2 – This is a standard for high bandwidth between 4 and 9 mbps. It is used in high quality TV broadcasts.
  – MPEG4 – Low bit rate compression algorithm intended for DS0 (64kbps) connections. Because of the low bandwidth consumption this algorithm may be used widely in video conferencing, but it is still under development.
- Intraframe compression – Compression within individual frames, also known as spatial compression. This method is performed independently from interframe compression. It operates with information solely contained in a single frame. Here are the best known of these algorithms:
  – M-JPEG – Video stream is transmitted as series as JPEG compressed still images.
  – Apple Video – Video compression algorithm implemented in Apple Quicktime video conferencing application.

All types of algorithms mentioned above may be implemented in hardware, but MPEG family encoders are usually implemented as hardware modules because of algorithm complexity, while decoders may hardware or software.

2.4 Circuit Emulation and VPNs

Frequently some enterprise needs to interconnect its branch offices with each other or with central office. The classical way of achieving this aim for an enterprise is creation of its own private network infrastructure. But this is very expensive and complex method. It is much easier to use existing public network infrastructure to

transport enterprise data. But public and private networks use different network design techniques and this may create problems concerning deployment of enterprise level applications over public network.

The most natural solution in this case is to map private network topology over public network infrastructure. The main idea is to make network look like as if it is private network for the upper layer protocols while using public network at lower layers. This type of private links is usually called virtual private networks. This concept is illustrated on Fig. 8.

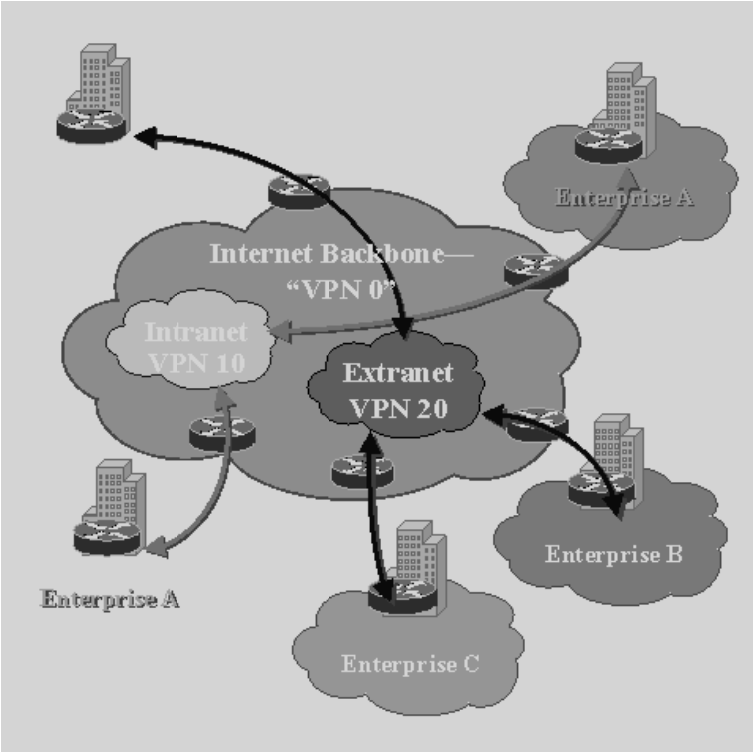


Fig. 8

This technique may be divided into three different subtypes: circuit emulation and data-link layer tunneling and network layer tunneling. Circuit emulation is usually emulating physical circuits over public network for network layer protocols, while data link and network layer virtual private networks generally use emulation of correspondingly data link or network layer protocol over network layer infrastructure. Let's discuss each of them in TCP/IP environment.

Circuit emulation is much more complex and rare in practice that network layer VPNs. It is not effective, because of large overhead. Physical layer bitstream is usually packed into IP packets. This technique is mostly popular for emulation of point-to-point DS1 and DS3 links. This method is used only when it is necessary

to keep the physical layer signalisation. The main complexity of this method is the need for clock restoration at the egress end of the emulated circuit. Several algorithms are created regenerating synchronization for emulated circuits but this adds additional overhead to transmitted data stream. Circuit emulation may be deployed in the following cases: transmission of multiplexed data stream over public network facilities; connecting remote PBX to the local access server with PRI link over public infrastructure and so on. Virtual circuits are most sensitive to network parameters, because they have to be much like real physical circuits. They are not affected by delay, but must have constant bandwidth, no jitter and no loss. Otherwise clocking will be lost which will lead to mass data loss until new handshake and clocking will be established.

Data link layer VPNs emulate particular data link layer technology over network layer protocols. In this case data link layer frames are encapsulated into network layer packets and transmitted over public facilities. Here's also significant overhead because we have two network layer and two data-link layer headers. This method is used when connecting two or more homogenous networks without protocol translation. As an example can be considered two frame relay networks which need to be interconnected over public cloud. The most frequently used data link layer tunneling protocols are PPTP, L2F and L2TP.

Network layer VPNs are the most effective types of tunneling, because only network layer header are duplicated. In TCP/IP networks network layer VPNs are usually created by encapsulation of original IP packet into another IP packet in which destination address is the IP address of egress node of the tunnel, where it'll be deencapsulated. These types of tunnels are the most popular in contemporary enterprise networks. The typical protocols of this family are IP-to-IP and GRE.

Another issue to be considered in tunneling is data security. As public facilities and links are not sufficiently secured, some measures are necessary to prevent the confidential data to be stolen or spoofed. In this environment IPSec protocol was invented which makes IP protocol more secure by encapsulating original IP datagram in encrypted form into another datagram.

The last two types by their nature don't require any quality of service, but this some level of QoS may be needed for content transmitted over them. This requirements are negotiated in the SLA (Service Level Agreement) and entirely depends on the needs of VPN customer.

3. Queueing Mechanisms

The general function of each intermediate node along the packet path is to receive packet, check its integrity, decide output interface and transmit it through that interface. Of course the transmit rate of packets is dependent on the speed of output interface. Very often situations happen when temporal bursts of incoming traffic may exceed the possibilities of the output line. Without any additional measures the packets in the burst exceeding output requirements will be dropped. To avoid this behavior additional memory buffers are added for each interface to temporally store packets that cannot be transmitted immediately. This allows sustaining packet bursts without increase of the loss probability. These buffers are usually called packet queues.

Packet queues play very significant role in dynamics of packet flows. Their parameters such as queue length and serve police greatly affect traffic patterns passing through the corresponding interface. That's the reason the queueing strategies are usually considered as the primary instrument in provisioning and tuning well designed multiservice networks.

The definition of term 'flow' is not quite clear. Generally this is a set of packets that follow the same route and require the same level of service. The idea of flow may change between different implementations. In today's networks the meaning of flow has changed. Flow is the set of packets, which have common source and destination and belong to the same application. Most network implementations distinguish flows according to their source and destination address and destination port value. Some systems increase the granularity of flows by also including into classification of source port. This allows considering a flow as one transport layer session. This latter meaning of flow is used in this document.

Queueing does not affect congestion directly. It only determines the way outgoing packets interact with each other and the order in which they are multiplexed into outgoing channel. The queueing discipline generally deals with three fundamental quantities, which determine characteristics of egress traffic flow. These are:

- Bandwidth allocation, which determines which packet is to be sent at each moment *t*;
- Delay injection, which decides when each packet have to be sent;
- Jitter appearance, which is mainly due to differentiation in packet sizes;
- Buffer space allocation, which determines, which packets have to be accepted into queue and which of them have to be dropped.

Queueing strategies are used in providing PHB features in QoS networks, as each packet is processed separately at each node along its path and at each intermediate node queueing decisions are made independently from other node queueing behaviors.

There are many different queueing algorithms (disciplines), which significantly differ in their application fields and the possibilities they offer:

- FIFO
- Priority Queueing
- Round Robin Queueing
- Fair Queueing

The algorithms are presented from the most simple (FIFO) to the most complex (FQ).

In this section will be discussed the effect of queueing strategies on the resource distribution among flows with different characteristics and will be considered the most effective applications of each discipline.

3.1 FIFO

FIFO queueing strategy is one of the simplest ones. FIFO stands for "First In First Out" or it is also called FCFS – "First Come First Served". This discipline assumes that there is a single queue per each interface. All excess traffic is queued into this buffer without any differentiation. The working scheme is extremely simple: After incoming packet is received it is immediately queued to the top of the queue. On the other side the buffer is dequeued consequently until it is empty. The packets are sent from queue in the same order they were stored. As a result the moment *t* of packet arrival completely determines bandwidth, delay and buffer space allocation. This behavior is shown on Fig.9.

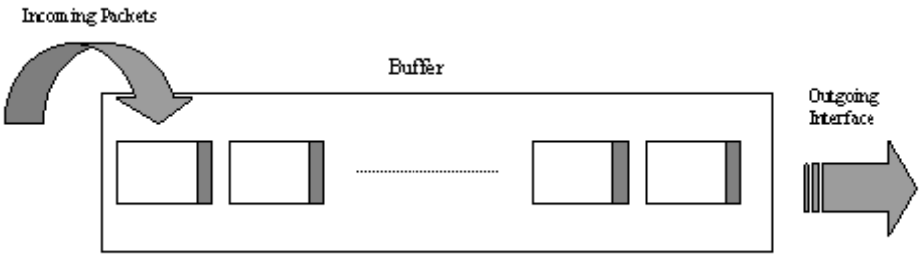


Fig. 9

This type of queueing requires the least processing power, that's why it is recommended to be used for high-speed interfaces like fast ethernet.

As all the traffic is mixed into a single buffer there is no way to treat different types of traffic in different ways, tuning consumed bandwidth, latency and other network parameters on per service basis.

FIFO queueing is also vulnerable to misbehaved flows, which may consume all the bandwidth available making other flows to starve for bandwidth and increasing average delays to large values. This means that is completely depends on proper implementation of end-to-end congestion control mechanism. But even in case of standardization of some congestion control algorithm, such as Van Jacobson algorithm for TCP, There will be always possibility of some malicious user or host breaking this rules due to malfunctioning or intentional algorithm modification for improving individual performance and receiving more network resources at the expense of others.

Let's discuss the drawbacks of FIFO queueing in detail. For the clear discussion let's assume that we have a FIFO queueing system with a single flow of interactive traffic (i.e. telnet) with Poisson distribution having intensity λ and average service delay μ_1. Of course total service delay of the queue in case of single flow will be equal to $\mu = \mu_1$. If we assume that the probability of two packets arrive during a very small time interval δ is almost zero then according to theory Markovian processes the probabilities of the fact that number of packets will increase by 1 or decrease by one will be correspondingly equal to $\lambda\delta$ and $\mu\delta$.

If we consider the equation of detailed balance between two sets of states *{0, 1, …, n}* and *{n+1, n+2, …}*, then we will obtain

$$p_n \lambda\delta = p_{n+1} \mu\delta$$

which states that the probability of system moving from state n into state n+1 is equal to probability of system moving from state *n+1* into state *n*, where p_n is stationary probability of system being in state *n*.

$$p_{n+1} = \rho p_n = \rho^{n+1} p_o = \rho^n (1-\rho), \qquad n = 0, 1, \ldots$$

where $\rho = \lambda/\mu$ is system load coefficient.

The average number of packets in the queueing system (including the one being serviced) is

$$N = \sum_{n=0}^{\infty} n p_n = \sum_{n=0}^{\infty} n\rho^n (1-\rho) = \rho(1-\rho) \sum_{n=0}^{\infty} n\rho^{n-1} = \rho(1-\rho) \frac{\partial}{\partial\rho} \left(\sum_{n=0}^{\infty} \rho^n \right) = \rho(1-\rho) \frac{1}{(1-\rho)^2} = \frac{\rho}{(1-\rho)} = \frac{\lambda}{(\mu-\lambda)} \qquad (1)$$

If we use Little's theorem we will receive average total packet delay in FIFO queueing system

$$T = \frac{N}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu - \lambda}. \quad (2)$$

It is clear that using the recent equation we can calculate average queueing delay for FIFO queue.

$$W = T - \frac{1}{\mu} = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\lambda}{\mu(\mu - \lambda)} = \frac{\rho}{\mu - \lambda}.$$

In the same way we calculated number of packets in the queueing system N, we can calculate average queue size N_Q.

$$N_Q = \lambda W = \frac{\rho^2}{1-\rho}.$$

Fig. 10 shows dependence of queue length on usage level of the queueing system.

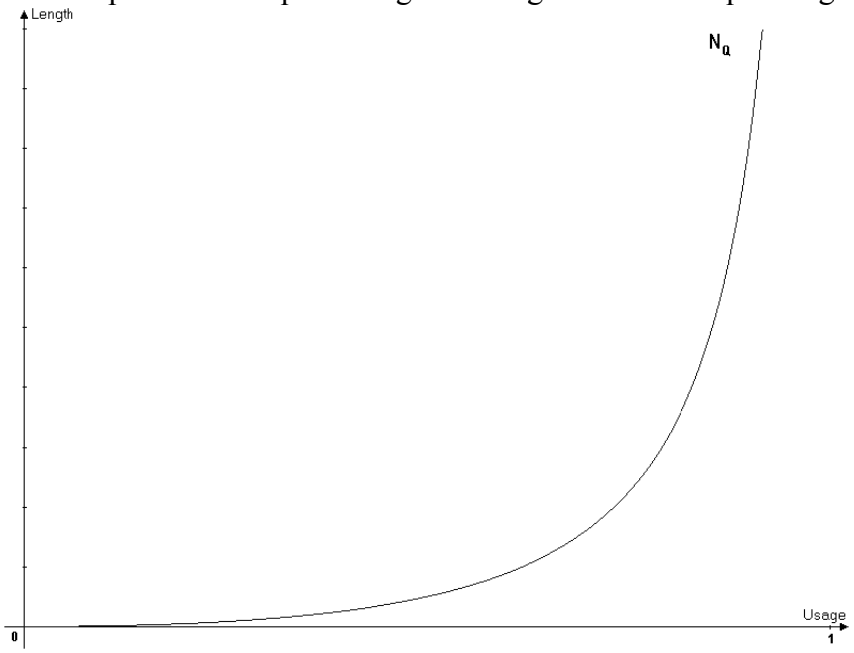


Fig.10

In case of several Poisson flows with corresponding intensities λ_1, λ_2,… they merge into summary Poisson flow with total intensity of $\lambda = \sum_i \lambda_{i.}$. And each flow will avesage coefficient ρ_i.

In this case we will have similar situation as we discussed for a single data flow.

It was mentioned above, any user can consume arbitrarily high fraction of bandwidth of outgoing interface. This can be easily achieved by simply increasing the intensity of incoming packets for data flow j and reaching the state, where $\forall i\ \lambda_j \gg \lambda_i$, $i = 1, 2, \ldots\ i \neq j$. As we can see from (2) average delay will also significantly increase even in case all the flows will use the same packet size C/μ, where C is bandwidth of outgoing interface. This shows how unfair FIFO queueing is. The problem mentioned above appears when some aggressive flow does not have end-to-

end congestion control feature and does not slow down its transmission rate as a response to congestion.

Generally in case of two flows: normal flow A with rate λ_A and aggressive flow B with rate λ_B such, that $\lambda_A<C$, $\lambda_B<C$ and $\lambda_A+\lambda_B>C$, where C is total link bandwidth. This means that the link is congested. For this illustration of them have equal packet transmission times $1/\mu$. Flow B will always try to consume all the unused bandwidth available. This means that at any moment t, the bandwidth consumed by flow B will be equal to $R_B=C-R_A$, where R_A is bandwidth received by flow A. Let's assume that all the flow A is transmitted with its incoming rate $R_A = \frac{\lambda_A}{\mu}C$. But as flow B consumes all the rest of bandwidth $R_B = \left(1-\frac{\lambda_A}{\mu}\right)C$, that is unused, link becomes congested and as a result flow A reduces its rate to $\lambda_A'<\lambda_A$. Flow B immediately will capture the bandwidth $\lambda_A-\lambda_A'$ just freed up, making flow A to reduce its transmission rate once more. This process will continue until flow B will reach its maximum rate $R_B = \frac{\lambda_B}{\mu}C$, while flow A will receive $R_A = \left(1-\frac{\lambda_B}{\mu}\right)C$. In case of, $\lambda_B>C$, flow B will monopolize all the bandwidth making flow A starve. As the aggressiveness completely depends on source implementation, the situation discussed above may obviously happen.

This poor behavior shown above can be improved by using buffer management feature for existing FIFO queue. To provide buffer management buffer is partitioned into segments and each segment is related to some flow. Let's consider the previous example, where we have flow A with arrival rate λ_A and aggressive flow B, which consumes all the unused bandwidth. Buffer of size B is split into two parts of sizes B_A and B_B for each of the flows respectively. The following notations will be used:

- $Q_A(t)$ and $Q_B(t)$ – buffer occupancy levels at time t respectively for flows A and B.
- $A_A(t)$ and $A_B(t)$ – number of packets admitted to buffer by time t respectively for flows A and B.
- C – Total link bandwidth.

Also suppose that $B_A = \frac{B\lambda_A}{C}$ and $B_B = B - B_A$. Flow A experiences the first packet loss at some moment $u>0$. This means that buffer already contains threshold number of packets for this flow: $Q_A(u)=B_A$. The oldest packet of flow A at time u had to arrive to the queue at some moment v, such that $A_A(v) = A_A(u) - B_A$. At moment v buffer contains $Q_A(v)$ packets of flow A and $Q_B(v)$ packets of flow B. It is clear that at moment v buffer usage threshold for flow A could not be reached as the first packet loss happened at moment $u>v$. This mean that

$$Q_A(v) < B_A = \frac{B\lambda_A}{C}.$$

According to FIFO definition packet that arrived at moment v could not spend more than time $(Q_A(v)+ Q_B(v))/C$ in the queue. This leads us to inequation

$$u - v \le \left(Q_A(v) + Q_B(v)\right)/C.$$

Number of packets that belong to flow A arrived between moments *v* and *u* cannot exceed the value $\lambda_A \cdot (u - v)$ and number of packets in the queue at moment *u* is $Q_A(u) \le Q_A(v) + \lambda_A(u - v)$. As $Q_A(v) - 1$ packets will be already dequeued, bacause the newest packet at the moment *v* is the oldest one at the moment *u*,

$$Q_A(u) \le \lambda_A (u - v)$$

$$Q_A(u) \le \lambda_A \frac{Q_A(v) + Q_B(v)}{C}$$

$$Q_A(u) < \lambda_A \frac{B_A + (B - B_A)}{C}$$

$$Q_A(u) < \frac{B\lambda_A}{C}.$$

This inequation shows, that to guarantee lossless transfer of flow with rate λ, it is sufficient to set buffer usage threshold for this flow to $\frac{B\lambda}{C}$.

At this point another subject of discussion appears: in the scheme mentioned above if some flow does not use the bandwidth that was guaranteed for it, this unused bandwidth cannot be used by another flow. In order to avoid this ineffectiveness some mechanism is needed to fairly share this bandwidth among existing active flows.

The intuitive solution of this problem is to partition unused buffer space among active flows according to the proportion of their reserved bandwidth. In order to reduce the impact of this scheme on rate guarantees some part of this spare buffer space, known as headroom, must be reserved for flows that didn't reach their threshold. The rest of the buffer left after headroom reservation is called holes and is free to be shared by flows. The algorithm for incoming packet *p* allowing fair distribution of free buffer space may be work as follows:

Enqueueing Module:

```
i = ExtractFlow(p);
if (Threshold(i) <= CurOccupation(i))
 {
  if (CurOccupation(i) – Threshold(i) < FreeHoles(i))
   {
    HolesSize(i) -= Size(p);
    Enqueue(i,p);
   }
   else Drop(p);
 }
else
 {
   if (Size(p) <= FreeHoles(i))
```

```
    {
     HolesSize(i) -= Size(p);
     Enqueue(i,p);
    }
  else if (Size(p) <= FreeHead(i))
    {
     HeadSize(i) -= Size(p);
     Enqueue(i,p);
    }
  else Drop(p);
 }
```

In the algorithm shown above fairness is achieved by allowing non-conformant flow to use spare buffer space only if its current usage of holes space is less than unused space in holes, while headroom can be used only by flows that are currently under their threshold.

On the other hand when packet leaves system, resources used by it must be freed up. The recovery algorithm for each outgoing packet can be like this:

Dequeueing Module:

```
while(true) do
for(i=0; i<n; i++)
 {
  p = QueueHead(i);
  HeadSize(i) += Size(p);
  HolesSize(i) = max(HeadSize(i) - MAXHEADSIZE,0);
  HeadSize(i) = min(HeadSize(i), MAXHEADSIZE);
  Dequeue(i);
  Send(p);
 }
```

This algorithm ensures that the space freed up after packet successive transmission will first of all increment free headroom size and only after it reaches its maximum value, it increases hole space.

The recent discussion shows, that in spite of FIFO queueing drawbacks bandwidth guarantees are achievable even in this case, if in addition buffer management is used for the queueing system.

3.2 Priority Queueing

The natural way of avoiding drawbacks introduced by FIFO queueing strategy is to use separate queues for different classes of traffic. In this case another question appears: what is the queue service discipline, or in other words what is the order in which these queues are queried for data ready and how much of is transferred during each round.

One of the types of such mutiqueue strategies is priority queueing system. It can't be used for delivery of service fairness among users, but it allows service differentiation among different data flows.

The main idea of priority queueing is as follows: All the traffic is giver certain priority level and is placed into one of the queues of corresponding priority. Traffic classification may be performed according to different packet fields: source address, destination address, protocol, source port, destination port, packet size, incoming interface, etc. There may be arbitrary number of priorities and queues associated with them. For example routers by Cisco Systems are using priority queueing system with 4 queues. We will consider number of queues equal to *n*.

In priority queueing higher priority queues have absolute priority over lower priority ones. This means, that some priority queue is not serviced until all its higher priority queues are empty. This working scheme implementation for Cisco routers is presented in Fig. 11.

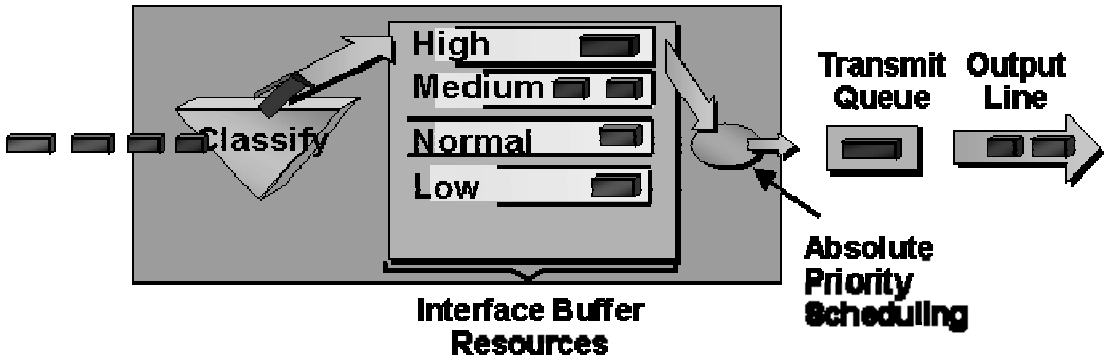


Fig. 11

The check for active queue with highest priority may be implemented using special variable *TopActive Queue*, which updates its value as soon as a new packet arrives to the system. After this description this algorithm can be presented in more strict form. Enqueueing module for incoming packet *p* is quite simple.

Enqueueing Module:

```
i = ExtractFlow(p);
if (Size(p) > Free(i))
  Drop(p);
if Empty(i)
 {
  if (Priority(i)>Priority(TopActiveQueue))
    TopActiveQueue = i;
  Enqueue(i,p);
 }
```

On the other hand there is some complexity in dequeueing module.

Dequeueing Module:

```
while(true) do
 {
  if (Empty(TopActiveQueue)) continue;   /*The system is
empty*/
  p = QueueHead(TopActiveQueue);
  Dequeue(TopActiveQueue);
  Send(p);
  if (!Empty(TopActiveQueue)) continue;
  while(QueuesLeft()&!Empty(Queue(Priority(TopActiveQueue)--
)));
 }
```

TopActiveQueue always points to the queue with packets waiting except the case when system is absolutely empty. In this case this variable is set to the lowest queue value and which does not hold any packet. The algorithm will execute nothing but spare loops.

The priority queueing scheme ensures that critical traffic will be handled in the fastest way. This queueing strategy must be used with care, because ill-behaved high priority flow may oppress the lower priority flows making them starve consuming the most part of the bandwidth available. But low priority traffic cannot affect the performance of higher priorities any more.

Priority queues are not intended to provide fair resource distribution among data flows. They were initially created for implementation of differentiated services.

The main drawback of priority queueing strategy is the fact that it allows resource reservation for certain service only in the indirect way. That is one can't get guaranteed quantity of bandwidth. All reservations are only relative to each other. This means that if suddenly the volume of high priority traffic increases this will affect the lower priority flows and as a result they may fail to receive their own guaranteed share of bandwidth. In the worst case the lower priority traffic may get out of service. This open the way for various "Denial of Service" attacks resulting performance degradation of the network.

As a result of the fact mentioned above there appears another drawback of priority queueing: it is very static and it cannot adapt to changing network requirements.

As it was mentioned earlier, priority queueing is mostly used in networks where one or more network applications are extremely sensitive to delay, because of the smallest response time among queueing methods. Now let's investigate characteristics of priority queueing discipline.

In this document we will discuss only nonpreemptive priority queues. These are the queues that don't interrupt transmission of lower priority packet even if higher priority packet arrives to the system.

Let's introduce the following designations for further use:

N_Q^k - Average number of packets in the queue of priority k.

W_k – Average packet waiting time in the queue of priority *k*.

$\lambda_k, \overline{X_k} = 1/\mu_k, \overline{X_k^2}$ - Incoming flow intensity, first and second moments of average packet service delay for the queue of priority *k*.

$\rho_k = \lambda_k / \mu_k$ - Usage coefficient for the queue of priority *k*.

R – Mean residual packet service time.

Using Polachek-Hinchin formula for the highest priority queue we will obtain:

$$W_1 = R + \frac{1}{\mu} N_Q^1$$

By applying Little's theorem for excluding of N_Q^1 from the equation we will receive:

$$N_Q^1 = \lambda_1 W_1,$$

$$W_1 = R + \rho_1 W_1,$$

Finally,

$$W_1 = \frac{R}{1-\rho_1}. \qquad (3)$$

Now let's consider the second priority queue, which is served immediately after highest priority queue 1 is empty. For this queue the queueing delay will be equal to

$$W_2 = R + \frac{1}{\mu_1} N_Q^1 + \frac{1}{\mu_2} N_Q^2 + \frac{1}{\mu_1} \lambda_1 W_2.$$

It is clear that the first two members of this sum represent the service time of the highest priority queue, the third member is the service time of the packets which are already in the queue, the last member takes into account highest priority packets that arrived during the queueing delay of the second queue.

If we apply Little's theorem again, then we'll receive:

$$W_e = R + \rho_1 W_1 + \rho_2 W_2 + \rho_1 W_2,$$

$$W_2 = \frac{R + \rho_1 W_1}{1-\rho_1-\rho_2}.$$

In this final equation we can substitute W_1 with the expression calculated above in (3).

$$W_e = \frac{R}{(1-\rho_1)(1-\rho_1-\rho_2)}.$$

We can easily generalize this formula for the system with *k* priority queues:

$$W_k = \frac{R}{(1-\rho_1-\ldots-\rho_{k-1})(1-\rho_1-\ldots-\rho_k)}. \qquad (4)$$

At this point we have to find residual packet service time *R*. Polachek-Hinchin formula directly leads to

$$R = \frac{1}{2}\left(\sum_{i=1}^{n} \lambda_i\right)\overline{X^2}.$$

The second moment of average packet service delay is averages according to corresponding flow intensities λ_i. As a result

$$R = \frac{1}{2}\sum_{i=1}^{n} \lambda_i \overline{X_i^2}.$$

If we insert this value into (4), then the equation for queueing delay will get the following form:

$$W_k = \frac{\sum_{i=1}^{n} \lambda_i \overline{X_i^2}}{2(1-\rho_1-\ldots-\rho_{k-1})(1-\rho_1-\ldots-\rho_k)} \qquad (5)$$

In case of exponential distribution of service periods second moment will be equal to $\overline{X_i^2} = \frac{2}{\mu_i^2}$. If we apply this expression to (5), then it will have the following form:

$$W_k = \frac{\sum_{i=1}^{n} \rho_i / \mu_i}{(1-\rho_1-\ldots-\rho_{k-1})(1-\rho_1-\ldots-\rho_k)}.$$

This is formula of queueing delay for priority *k* queue in *M/M/1* system.

The total transmission delay at the intermediate node will be equal to

$$T_k = \frac{1}{\mu_k} + W_k. \qquad (6)$$

One of the most important issues for priority queueing is choosing packets for high priority queues. It is crucial to give high priorities to services with low service times because this will reduce the overall average service time. To make this point clear let's consider priority queueing system with two queues: A and B with corresponding incoming flow intensities and service time equal to λ_A, λ_B, $1/\mu_A$ and $1/\mu_B$. At this point the average service time for overall system will be

$$T = \frac{\lambda_A T_A + \lambda_B T_B}{\lambda_A + \lambda_B}.$$

From the expression shown above, it is clear, that in case of $\mu_A > \mu_B$, it is obvious, that *T* will have smaller value, when A has higher priority then B. This fact must be considered during priority queue design at network nodes. For example voice traffic in this queueing model must have higher priority over database transactions as it has smaller average packet lengths and is more sensitive to delay.

3.3 Round Robin Queueing

Round robin queueing is also called custom queueing[1]. It was created in attempt to reach fair resource distribution among flows while consuming the least processing power. This queueing system flows also map to a set of queues as it was for priority queueing. The main difference in dequeueing strategy.

[1]Term "Custom Queueing" is mainly used in literature by Cisco Systems and beginning from IOS version 12.1 corresponds to Deficit Round Robin algorithm described later in this section. Before version 12.1 Cisco used classical round robin algorithm with all the drawbacks described in this section.

All the queues in round robin discipline are serviced in sequential circular manner. Certain number of bytes or packets called quantum is dequeued from each queue and for fair distribution this quantum is equal for all queues in the system. In case of unequal quantums (weights) for queues the algorithm is called weighted round robin queueing (WRR). After dequeueing quantum from the current queue algorithm proceeds to another queue in sequence until it reaches the end of queues. After this the process repeats.

The working scheme of weighted round robin queueing discipline is illustrated in Fig. 12.

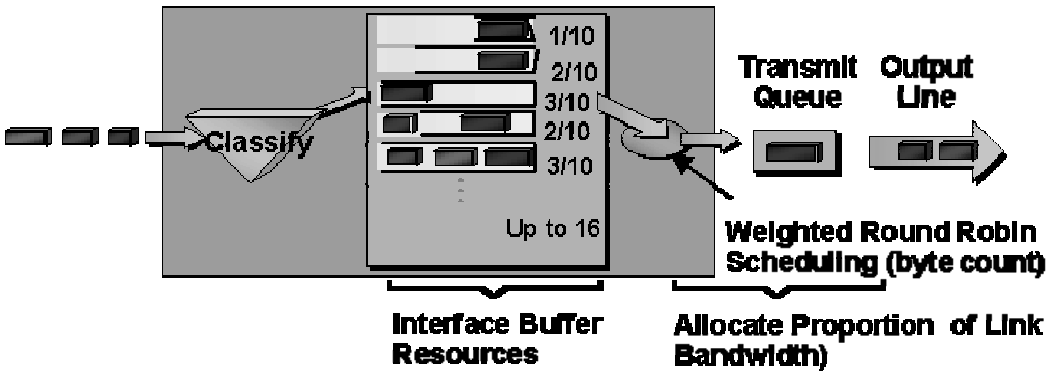


Fig. 12

The main drawback of this algorithm is that it significantly depends on average packet lengths in each queue. If quantum values are set as number of packets to be transmitted this dependence is quite clear.

In case of byte-count quantum values, in order to transmit exactly quantum bytes, there are two ways: all incoming packets must be fragmented to fit into quantum integer times, which extremely inefficient method, because different routers along the path will defragment some packet several times according to their needs and as a result receiver will get a set of packets instead of one packet; another way is to transmit packet number that fits into quantum, but in this case some part of quantum will be lost, that will lead to unfairness. Of course, there is the possibility to adjust quantum values according to average packet sizes for each flow, but this method is too complex to be implemented and used.

The solution to this problem is Deficit Round Robin algorithm (DRR). The idea is for each queue to transmit as many packets as fit into quantum value, but the remainder will be carried to the next round.

Before we begin definition and investigation of this algorithm let's define initial conditions and terms that are used later in algorithm definition.

First of all it is assumed, that flows are mapped to queues using hashes. This idea was introduced by McKenney in stochastic fair queueing algorithm. Another initial conjecture is that all the flows in the queueing system are always backlogged. This means that they always have packets to transmit when algorithm applies to each of them.

As soon as we argue about fairness of algorithm, some quotient must be defined to measure the fairness level. Let's denote total number of bytes sent by flow *i* until time *t* by $sent_{i,t}$. Let $sent_t$ be the total number of bytes sent by all *n* flows by time *t*. It is quite natural to define fairness quotient as worst-case ratio of the bytes sent by flow *i* and total number of bytes sent by all flows. This will be simply the worst-case share of bandwidth consumed by flow *i*. As we need to the quotient not to be dependent on time, we assume that time *t* tends to infinity. Fairness quotient for flow *i* will look like:

$$FQ_i = \max(\lim_{t\to\infty} \frac{sent_{i,t}}{sent_t})$$

Maximum value must be obtained for any possible packet length distribution across all *n* flows in the queueing system.

For measurement purposes of how perfect the fairness is, we'll define some value of desired "ideal" fairness for flow *i* f_i which defines a desired share of flow *i* in the total bandwidth. Naturally this "ideal" distribution will have its own "ideal" fairness quotient, which as it was defined above, will be equal to:

$$FQ_i^{'} = \frac{f_i}{\sum_{j=1}^{n} f_j}.$$

We don't need any maximum values or limits as this value is supposed to be constant in time and must not depend on packet length distributions. In the simplest case when this "ideal" quotients are equal for all flows, $FQ_i^{'} = 1/n$. Finally, we will define Fairness Index value for flow *i* as

$$FI_i = \frac{FQ_i}{FQ_i^{'}} = \frac{FQ_i \cdot \sum_{j=1}^{n} f_j}{f_i}.$$

The Fairness Index value will be primarily used in algorithm fairness measurements. Now let's begin with more strict definition of DRR algorithm.

In the Deficit Round Robin algorithm quantum value Q_i for each queue completely determines the share of bandwidth f_i it receives. Incoming flows are distributed among different queues using hash. If we denote with $bytes_{i,k}$ the number of bytes dequeued from queue *i* during round *k*, then it is easy to see that for the first round $bytes_{1,k} \le Q_i$. The total number of deficit $Q_i - bytes_{i,k}$, that is carried between rounds is stored in Deficit Counters DC_i for each queue *i*. If some queue after being served is empty, corresponding deficit counter is set to 0 and nothing is carried to the next round.

Special list of active flows is maintained to avoid wasting processing power on examination of empty queues. Whenever packet arrives to empty queue its index is appended to the end of the list. At each round at most Q_i+DC_i bytes are dequeued from the queue at the top of the active list. If at the end of the round the queue contains at least one packet, then it is moved to the end of active list, otherwise it is removed from the list and its deficit counter is dropped to 0.

Enqueueing and Dequeueing modules Deficit Round Robin algorithm can be described in the following way:

Enqueueing Module:

```
i = ExtractFlow(p);
if (!ExistsInActiveList(i))
 {
  InsertActiveList(i);
  DeficitCounter_i = 0;
 }
if (!FreeBuffersAvailable())
FreeBuffer();
Enqueue(i,p);
```

Dequeueing Module:

```
while(true) do
 {
  if(!Empty(ActiveList))
   {
    i = Head(ActiveList);
    DeficitCounter(i) += Quantum_i;
    while((DeficitCounter(i)>0)and(!Empty(Queue(i))))
do
     {
      PacketSize=Size(Head(Queue(i)));
      if(PacketSize ≤ DeficitCounter(i))
       {
        p = Dequeue(i);
        Send(p);
        DeficitCounter(i) -= PacketSize;
       }
      else break;
     }
    if(Empty(Queue(i)))
      DeficitCounter(i) = 0;
    else InsertActiveList(i);
   }
 }
```

In case of all buffer space is already full, buffer stealing method is used, which drops the last packet from the longest queue. This is what exactly does function `FreeBuffer()`.

Now let's see the changing bounds of deficit counter at any point of algorithm execution. Initially $DC_i=0 \Rightarrow DC_i<Q_i$. As deficit counter value changes only after corresponding queue has been changed, we can distinguish between two behaviors: 1) Queue is empty after being serviced. In this case according to algorithm scheme show above deficit counter is set to 0. 2) Queue contains one or more packets after being serviced. This means that next packet in the queue was strictly larger that deficit counter value and was not transmitted during current round. As lengths of all packets are limited by some value *Max*, it is clear, that $0 \le DC_i < Max$.

We will introduce the following notations:

- $DC_{i,K}$ – value of deficit counter of flow *i* at the end of round *K*.
- $bytes_{i,K}$ – Number of bytes sent by queue *i* during round *K*.
- $sent_{i,K}$ – Number of bytes sent by flow *i* during rounds 1 through *K*.

It is easy to see that $sent_{i,K} = \sum_{k=1}^{K} bytes_{i,k}$. (7)

The state of algorithm before it begins to execute is as follows: $DC_{i,0} = = bytes_{i,0} = 0$. If we assume, that all the flows are always backlogged, then during the execution of the algorithm the following equation is correct: $bytes_{i,k} + DC_{i,k} = Q_i + DC_{i,k-1}$, which is clear from algorithm description. It shows the amount of bandwidth allocation for flow *i* during round *K*. This equation can be rewritten in the following way:

$$bytes_{i,k} = Q_i + DC_{i,k-1} - DC_{i,k}.$$

This equation can be summed up for K rounds and after taking into account (7) and algorithm initial conditions, it will transform into

$$send_{i,K} = K \cdot Q_i - DC_{i,K}. \quad (8)$$

As it was shown above deficit counter value never exceeds maximum packet length *Max*. On the other hand, $send_{i,K} = K \cdot Q_i$ is the portion of bandwidth granted to flow *i* during ideal distribution of bandwidth. It is clear, that (8) equation proves that difference between real and ideal distributions will not exceed maximum packet length *Max*.

Now let's examine the system, which always has *n* active flows at any moment of time. Packet lengths vary between *Min* and *Max*. As it was just proved,

$$K \cdot Q_i - Max \le sent_{i,K} \le K \cdot Q_i$$

If we sum this inequation for *n* active flows in the system:

$$K \cdot \sum_{i=1}^{n} Q_i - n \cdot Max \le sent_K \le K \cdot \sum_{i=1}^{n} Q_i$$

Finally,

$$\frac{K \cdot Q_i - Max}{K \cdot \sum_{i=1}^{n} Q_i} \le \frac{sent_{i,K}}{sent_K} \le \frac{K \cdot Q_i}{K \cdot \sum_{i=1}^{n} Q_i - n \cdot Max}. \quad (9)$$

It is obvious, that if $t \to \infty$, also $K \to \infty$. If time tends to infinity (9) transforms into the following equation:

$$FQ_i = \frac{Q_i}{\sum_{i=1}^{n} Q_i}.$$

Now it is possible to calculate fairness index for DRR algorithm for the given initial conditions and assumptions. As $f_i = Q_i$, we get:

$$FI_i = \frac{FQ_i \cdot \sum_{i=1}^{n} f_i}{f_i} = 1. \quad (10)$$

These results were received with several limiting initial conditions: 1) All the flows are backlogged; 2) Measurements are performed at the end of a round. The results concerning fairness may significantly differ from the ones shown above. The fairness of the algorithm greatly depends on burstiness of flows. This dependence will get clear, if we imagine the following situation: system serves bursty and non-bursty flows, which have the same quantum values. Bursty traffic will be backlogged for some rounds while for some rounds not. It will obviously lose the opportunity to transmit packets during nonbacklogged rounds, and these lost opportunities will not be carried to the next round, during which a burst will occur. In this case the algorithm may be unfair.

In order to investigate protocol fairness not on round boundary, let's create another fairness measure for arbitrary $[t_1;t_2]$ period. It is difference of data volumes $W_i(t_1,t_2)$ sent from any two queues, altered by corresponding weights $r_i = Q_i$. It is obvious that for ideal case this difference will equal to 0.

$$\left| \frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j} \right| = 0.$$

In practice this difference can be only close to 0 value. This means, that fairness of every algorithm can be measured by the upper bound of this difference $H(i,j)$.

$$\left| \frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j} \right| \le H(i,j). \quad (11)$$

It was proved, that for any packet scheduling algorithm

$$H(i,j) \ge \frac{1}{2}\left(\frac{Max_i}{r_t} + \frac{Max_j}{r_j} \right). \quad (12)$$

We can apply this fairness measure to DRR algorithm on $[t_1;t_2]$ time interval of length T and see how close it is to (12). As $[t_1;t_2]$ is chosen arbitrarily, it is possible that it won't match the round boundary. This means, that flow j may complete K rounds, while flow i will complete only $(K-1)$ rounds. It was shown above, that on round boundaries the difference of bytes sent by each flow from the ideal distribution will never exceed maximum packet length Max. As a result the following system of inequation can be written:

$$\begin{cases} sent_{i,T} \ge (K-1) \cdot r_i - Max_i \\ sent_{j,T} \le K \cdot r_j + Max_j \end{cases}$$

Using this system of inequations we can calculate (11) expression as follows:

$$\left| \frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j} \right| \le \left(1 + \frac{Max_i}{r_i} + \frac{Max_j}{r_j} \right) \quad (13)$$

This result is not quite good, as we will see for other FQ algorithms. If $Max_i = Max_j = 1$ and $r_i = r_j = 100$, fairness of DRR $H(i,j) = 1.02$,which is 100 times worse than the best possible case (12) and is 50 times worse than most FQ algorithms.

Now let's investigate delay introduced by DRR algorithm. Let's assume that packet *p* arrives to DRR queueing system. It is clear that in the worst case it will have to wait for all active flows to transmit their quantums. This means, that maximum bound for packet queueing delay is $\frac{\sum_{i=1}^{n} Q_i}{C}$, where *n* is the number of active flows in the system and *C* is the capacity of the outgoing link. As quantum values for flows can be arbitrarily high, leads to arbitrarily high delay values. This is very bad result as it makes DRR unusable for latency-critical applications like voice transmission.

The solution to this problem may be combination of DRR with other queueing disciplines. For example the system can be modified in the following way: one DRR system can be split into two, one for latency-critical and one for best-effort data and this two DRR systems will be served in PQ manner with higher priority given to latency-critical queues. The main change in algorithm will be, that if packet *p* arrives to some delay-critical queue *f*, then *f* is placed at the top of the active flow list, not at the end as it is done for other best-effort queues. Of course, each of the delay sensitive flows must not send more than *b* bytes during some time period *T* and *T* is large enough to send at least one packet of size *b*, because otherwise it will affect performance of best-effort flows that are served by the other DRR system. The values *b* and *T* must be regulated by traffic contract or SLA. All the excess traffic violating this contract must be dropped. In this case the maximum packet delay for latency-critical packets will be the following:

$$W_{\max} = \frac{\left((n' \cdot b) + Max \right)}{C}.$$

where *n'* is the number of latency-critical flows in the system. This equation means, that packet which arrived for latency-critical won't be delay by more, than time necessary to transmit small b-size packets from all latency-critical flows plus transmission time of maximum size packet from best effort queues. The value of delay can be easily reduced by reducing the number of delay-critical queues *n'* and latency-critical application packet sizes[2].

One more thing to be mentioned about DRR algorithm is, that as we will see later, it is much simpler, than other fair queueing algorithms. This fact makes it more suitable for implementation on high speed network interfaces. At the same time it does not require additional processing power to be included into routers.

---

2 The value of delay can be reduced even further if we use preemptive algorithm. In this case the delay will be smaller by *Max* value.

Finally, DRR algorithm is mostly static as PQ algorithm was. It cannot adapt to changing network requirements and work in dynamic network environments.

3.4 Fair Queueing

Fair queueing algorithm family (FQ) was created to avoid static behavior of other algorithms such as WRR, DRR and PQ. It is intended to be used in dynamically changing network conditions.

In FQ algorithms the entire traffic stream is broken up into flows or conversations. Incoming packets are mapped into per-flow queues using special hash function based on the following packet header fields:

- Source IP address
- Destination IP address
- Source TCP or UDP port
- Destination TCP or UDP port
- Transport protocol
- ToS field

Generally number of per-flow queues in the system must be larger that number of flows. In some implementation of FQ algorithm several flows with similar characteristics can be mapped into the same queue. In this case all the traffic is split into classes. Such algorithms are called Class Based Fair Queueing (CBFQ).

FQ algorithm has to operate properly even under heavy loads distributing available bandwidth fairly. It must preserve low volume interactive traffic flows from being suppressed by high volume flows, such as file transfers.

Another aim for FQ is to provide sensible delay for latency sensitive flows. It has to break up high volume traffic trains (sets of consequent packets that belong to the same flow) and interleaving them with interactive packets, allowing keeping delay bounds under certain values for delay sensitive data.

As it was mentioned above, all the conversations are mapped to corresponding queues using special hash functions. All this queues then served according to FQ scheduler, which will be described a bit later. After the scheduler orders packets for transmission, they are multiplexed into output interface. Fig. 13 presents FQ working scheme[3].

---

3 Cisco Systems in its products has implemented Weighted Fair Queueing – one of the simplest algorithms from FQ family. This algorithm is optimized for interface speeds under 2048 Kbps.

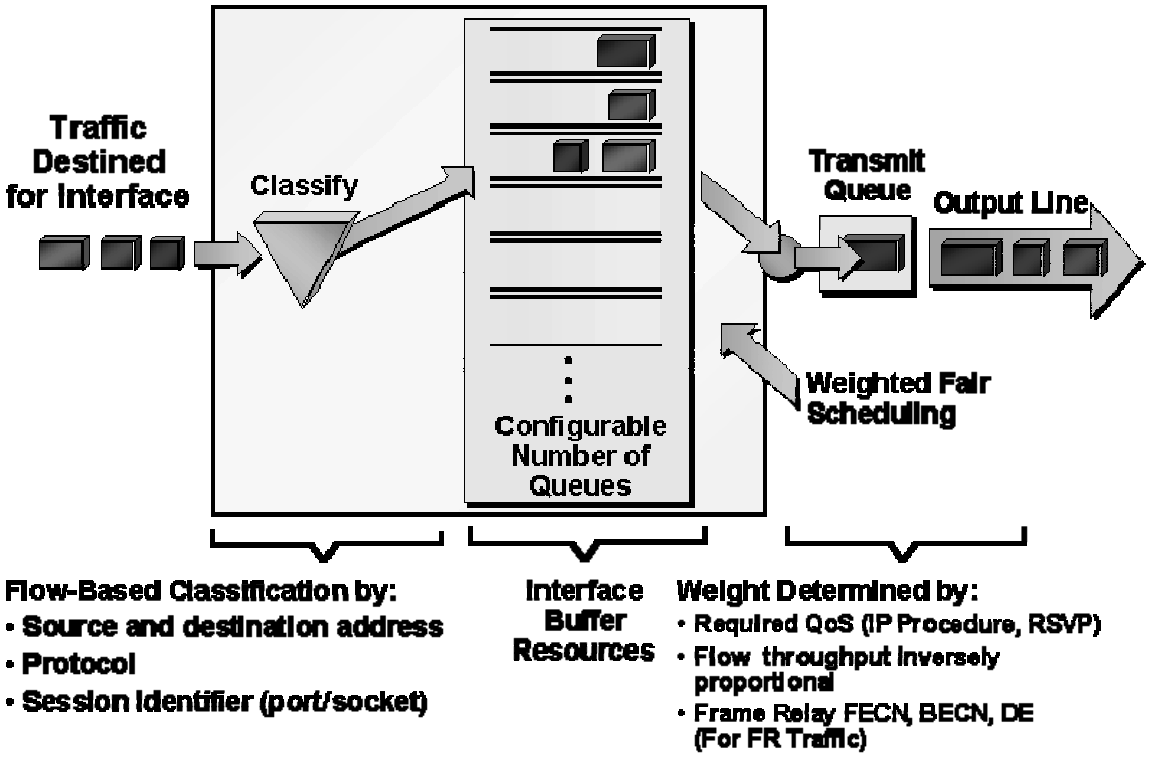


Fig. 13

The main aim of FQ algorithm is to combine predictability of network parameters in TDM networks while preserving efficiency of packet switched networks.

By splitting the overall traffic into conversation and mapping them into separate queues, FQ algorithm builds firewall that protects well-behaving flows from ill-behaving ones. If some flow sends data too quickly, it only increases the length of its own queue not affecting other flows. The first FQ algorithm was created by John Nagle in 1985, but it was very simple and as a result lots of improvements like Weighted Fair Queueing (WFQ) were created. Nagle's algorithm simply stored packets of each flow into separate queues and then served them in round robin manner. Anyway this was significant improvement over legacy FIFO queueing.

It is clear, that Nagle's algorithm performing fair packet allocation significantly depends on packet lengths of each flow during bandwidth allocation, which finally makes this allocation unfair. On the other hand there is absolutely no improvement in delay over standard round robin algorithm discussed in the previous section. At the same time, there is no method of delay management in this algorithm, which makes it not suitable for interactive applications.

In order to perform fair bandwidth allocation it is necessary to take into account lengths of packets to be transmitted. This can be easily achieved, if so called Virtual Clock mechanism is implemented.

TDM completely eliminates interference among competing flows by splitting them according to real-time clock. In order to emulate TDM behavior, we must imagine that packets are arriving to the queue at a constant rate. This may be achieved by associating special virtual clock VC_i to each of the queues. If we assume for

simplicity that flows have constant packet sizes, one tick of each virtual clock may be chosen as average packet interarrival gap. Each packet to be transmitted must be stamped with the virtual clock value, which the flow had at the moment of its arrival. Virtual clock value is increased by one tick each time packet is received. Queues are ordered for packet transmission according to their current virtual clock values. The working principle of virtual clock is illustrated in Fig. 14.

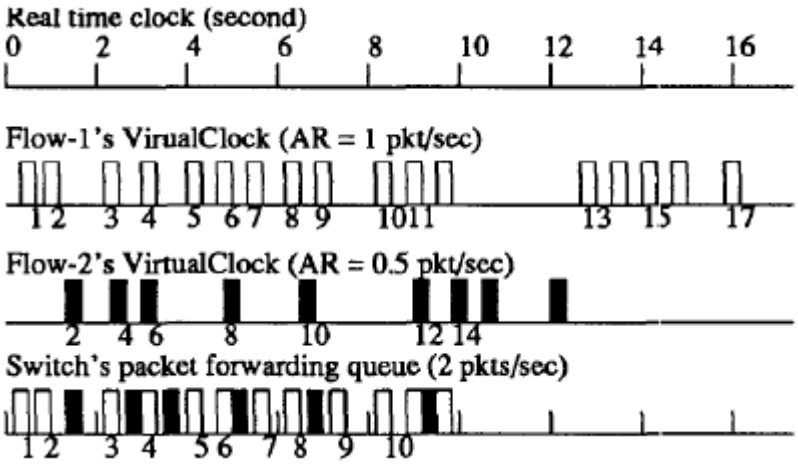


Fig. 14

Virtual clock implementation needs to obtain average packet arrival rate AR_i to calculate mean interpacket gap. This parameter may be negotiated during session establishment or statically configured by network administrator. This initial value for virtual clock tick must be initialized separately for each flow.

The Virtual Clock Fair Queueing algorithm described in the formal way looks like the following:

```
Initializing Module:

foreach(flow(i))
 {
  VTick(i) = 1/AR(i);
  VClock(i) = RealTime();
 }

Enqueueing Module:

i = ExtractFlow(p);
VClock(i) += VTick(i);
Stamp(p) = VClock(i);
Enqueue(i,p);
```

Dequeueing Module:

```
while(true) do
 {
  if(Size(p)>FreeSpace(i)) DropLast(i);
  MinFlow = Stamp(1);
  foreach(flow(i))
    if (MinStamp<Stamp(Top(i))) MinFlow = i;
  p = Dequeue(i);
  Send(p);
 }
```

In variable packet size environment it is necessary to include packet size value in per-flow virtual clock tick calculation. For example tick value may be changed to be proportional to incoming packet length l_p, $VTick_i = \frac{l_p}{AR_i}$.

Virtual Clock algorithm can be also used as flow control mechanism. The deviation of per-flow virtual clock value from real-time clock shows if the flow confirms with its requested average rate. Some threshold of deviation value can be chosen, such that the flow exceeding this threshold must be penalized and/or forced to reduce its rate.

Another issue related to the virtual clock flow control mechanism is deviation sampling rate. This value is application specific. It must not be chosen too large, as it will react to network events too slowly. After each measurement interval if virtual clock is less than real-time, it can be either set to real-clock value of retain its current value unchanged. In the first case, there is less freedom for bursty traffic, as unused bandwidth is not accumulated and carried over intervals.

Priority queueing can also be implemented by the means of virtual clock algorithm. It is necessary to change only initialization routine of standard algorithm. Virtual clock of priority queue must be initialized in the following way:

```
VClock(i) ← RealTime(i) – P;
```

where P represents the level of priority. This type of priority queueing still preserves relative fairness of utilization. If priority flow runs faster then its reserved rate, it will soon run ahead of real-time clock and loose its priority. The value for P must be chosen large enough to keep virtual clock of priority queue far behind other queues even in case of limited burstiness.

The main strength of Virtual Clock algorithm is that it provides TDM-like behavior for flows while not affecting statistical multiplexing benefits. It simply reorders packets to be transmitted according to their virtual time-slot values.

The significant drawback is that Virtual Clock algorithm must be implemented in conjunction with corresponding flow control mechanism, which was described earlier. On the other hand, this flow control mechanism prevents flows from

consuming spare bandwidth and reduces total efficiency of the system. Each flow has to overbook resources to have possibility to consume spare bandwidth, if it is available.

It is necessary to provide some strict basis for virtual clock calculation during algorithm execution. Let's consider ideal round robin queueing discipline, where scheduler dequeues one bit from each queue during each round. This queueing strategy is called Bit-by-bit Round Robin (BR). Of course this discipline is completely virtual, as in practice we can't dequeue less than a single packet. This queueing discipline is ideally fair, as each flow receives its fair share of bandwidth at any instant of time. We can imagine that the system is running using BR discipline and associate some value *R(t)* with each real-time moment *t*, which is the number of BR rounds that passed by this moment in the system.

Thus, we denoted by *R(t)* number of rounds that passed in virtual BR system by time *t* and $N_{ac}(t)$ is the number of active flows (conversations) in the system at time *t*. Then $\frac{\partial R}{\partial t} = \frac{C}{\sum_{j \in N_{ac}(t)} r_j}$, where *C* is total bandwidth of outgoing interface. It is clear that if packet with length *P* gets service in BR system at time t_0, its service will be completed *P* rounds later at time *t*. $R(t) = R(t_0) + P$.

Let's denote the moment of arrival of *i*th packet belonging to flow *f* by $A\left(p_f^i\right)$ and denote values *R(t)* at the moment, when service of packet *i* was started and finished correspondingly by $S\left(p_f^i\right)$ and $F\left(p_f^i\right)$. l_f^i is the size of packet *i* from flow *f*.

For each packet in the system service start and finish times for BR discipline can be calculated using the following formulas:

$$F\left(p_f^i\right) = S\left(p_f^i\right) + l_f^i; \qquad S\left(p_f^i\right) = \max\left(F\left(p_f^{i-1}\right), R\left(A\left(p_f^i\right)\right)\right). \qquad (14)$$

Of course BR system cannot be implemented in practice. Some method is needed to emulate BR behavior for packet-by-packet transmission. One of the ways to achieve this is to order packet for transmission according to their finish tags $F\left(p_f^i\right)$.

Of course BR system cannot be implemented in practice. Some method is needed to emulate BR behavior for packet-by-packet transmission. Each time the system finishes transmission of a packet it chooses next packet having the smallest value of $F\left(p_f^i\right)$. It can be proved that for any instant of time the deviation of number of data sent by each flow from each other is bounded by l_{max} value, where l_{max} is maximum packet length.

Now we can consider delay allocation in FQ algorithm. It can be natural to assign less delay to flows that don't consume their fair share of bandwidth. This kind of traffic separation can be performed using some nonnegative parameter δ. Packets from each flow have to be ordered according some quantity called bid and denoted by $B\left(p_f^i\right)$, which is calculated according to the following formula:

$$B\left(p_f^i\right) = \max\left(F\left(p_f^{i-1}\right), R\left(A\left(p_f^i\right)\right) - \delta\right) + l_f^i. \quad (15)$$

Other quantities will stay unchanged. The sending order of packets now is determined by *B*'s, not *F*'s. Parameter δ in (15) gives some preference to the packets, that arrive into inactive conversation and variation of its value allows us to control the level of preference.

The role of parameter δ becomes clear, if we consider two extreme cases of its values: $\delta=0$ and $\delta=\infty$. It is clear, that if $R\left(A\left(p_f^i\right)\right) \le F\left(p_f^{i-1}\right)$, then flow *f* is active or in other words backlogged. In this case behavior of the algorithm will be the same as described in (14), because the bid value will be equal to the finish tag value of the previous packet $F\left(p_f^{i-1}\right)$ and does not depend on δ. On the contrary, when $R\left(A\left(p_f^i\right)\right) > F\left(p_f^{i-1}\right)$, the flow *f* is in the inactive state. When $\delta=0$, from (15) we get $B\left(p_f^i\right) = R\left(A\left(p_f^i\right)\right) + l_f^i$, which means, that bid value depends only on the actual time of packet arrival ignoring the history of previous packets. On the other hand, when $\delta=\infty$, according to (15) $B\left(p_f^i\right) = F\left(p_f^{i-1}\right) + l_f^i$ and the bid value depends only on the finish tag of the previous packet irrespective of how many rounds ago that packet was served. Usually δ is chosen somewhere between these extreme values and expresses the time period, during which the packet history is examined when choosing each packet bid and, thus, what is the level of promptness preference, that inactive flows receive over active ones.

In the discussion it was assumed, that we need to distribute bandwidth fairly among flows, but FQ algorithm can be easily converted to provide unequal resource allocation according to weights assigned to each flow. In weighted fair queueing (WFQ) weights may be presented as fractions of outgoing bandwidth or as bandwidth amount r_f assigned to each flow[4].

If direct bandwidth values are used as weights, (14) formula can be changed to adapt to weighted allocation scheme:

$$F\left(p_f^i\right) = S\left(p_f^i\right) + \frac{l_f^i}{r_f}.$$

According to this formula it is clear, that virtual clock for large weight flows will run r_f times slower, than for ones with smaller weights.

Now let's return to the problem of ill-behaved flows. All queues in the system are finite. Let's assume that for purpose of flow firewalling if packet arrives and its queue is full, then the last packet from the longest queue is dropped. Moreover it is an idea not to change current start and stop tags $S\left(p_f^i\right)$ and $F\left(p_f^i\right)$ for that flow. This provides some penalty for high volume ill-behaved flows. All the time its virtual clock is increased, but total length of transmitted data is much smaller then it had to be.

It is to be said that WFQ algorithm uses complex calculations to maintain virtual clock values. The amount of work done by this algorithm can be presented as

---

[4] It is assumed, that for weights defined as direct bandwidth values $\sum_{f \in Q} r_f \le C$ holds, where *Q* is the set of all flows in the system and *C* is outgoing link capacity. The event where this inequality is violated is called resource overbooking and it is beyond the scope of this document.

O(logQ). This limits the application of WFQ algorithm to relatively low speed interfaces (usually 2Mbps). Additional work is to be done to adapt FQ algorithm to high-speed environments.

There are several variations of this algorithm, which improve some of its characteristics. One of the modifications suggest to stamp incoming packets not with finish tags of packets, but with their start tags. Start and finish tags of packets are calculated in the same way as in WFQ algorithm.

$$S\left(p_f^i\right) = \max\left(F\left(p_f^{i-1}\right), R\left(A(p_f^i)\right)\right)$$

$$F\left(p_f^i\right) = S\left(p_f^i\right) + \frac{l_f^i}{r_f},\ i \geq 0.$$

Here r_f is the weight of flow *f*. This algorithm is called Start-Time Fair Queueing (SFQ).

Virtual time *R(t)=0*, when *t=0*, but it is not calculated in the contiguous manner according to equation $\frac{\partial R}{\partial t} = \frac{C}{\sum_{j \in N_{ac}(t)} r_j}$ (16), as it consumed a lot of processing resources. Value of virtual time *R(t)* at time *t*, when it is backlogged, is defined as $S\left(p_f^i\right)$ of some packet p_f^i, which is in service in time *t*. If flow is empty, then *R(t)* is set to finish tag of the last serviced packet by time *t*. This makes *R(t)* more easy to calculate during algorithm execution. It is clear that *R(t)* value changes only when transmission of a packet is completed on the outgoing interface.

All the packets are ordered according to their start tag values. Ties[5] are broken arbitrarily. It is easy to see, that as in (16) calculations of *R(t)* are dependent on *C*, it must be assumed constant. This means, that this method will not work in variable bandwidth environments. Of course we can change constant *C* with function of time *C(t)*, but this will significantly increase computational complexity of the algorithm. On the other hand, SFQ algorithm associates *R(t)* with start tag value, which does not depend on *C*. This means that, this algorithm won't loose its efficiency during bandwidth variations.

It can be easily proved, that the fairness measure for the SFQ algorithm is no more than factor of two from its minimal possible value. In other words, for any two flows *f* and *m*, which are backlogged in $[t_1,t_2]$ time interval, the following inequation holds:

$$\left|\frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m}\right| \leq \frac{l_f^{\max}}{r_f} + \frac{l_m^{\max}}{r_m} \qquad (17)$$

(17) shows significant improvement in fairness of the algorithm in comparison with DRR and is not worse than WFQ.

---

5 Tie is the situation when several packets have the same tag. Some additional mechanism is needed to order these same-value packets. Some methods may be more effective than others may, but in this document we assume that the method of tie breaking is arbitrary.

The variable rate channel can be described either by Fluctuation Constrained (FC) or by Exponentially Bounded Fluctuation (EBF) models.

FC channel can be described by two parameters: average rate C and burstiness $\delta(C)$. If we denote number of bytes transmitted by the server during $[t_1,t_2]$ time period, then FC server satisfies the following condition:

$$W(t_1,t_2) \geq C(t_2 - t_1) - \delta(C)$$

EBF uses more complex model to describe variable rate server and it is beyond the scope of this document.

SFQ algorithm establishes strict guarantees on the packet throughput and delay for each flow in either FC or EBF variable rate servers. Let's consider FC server with parameters $(C,\delta(C))$. It can be proved, that if server capacity is not exceeded, or in other words $\sum_{n\in Q} r_n \leq C$, for all $[t_1,t_2]$ time intervals, where flow f is backlogged, $W_f(t_1,t_2)$ always satisfies inequation:

$$W_f(t_1,t_2) \geq r_f(t_2 - t_1) - r_f \frac{\sum_{n\in Q} l_n^{\max}}{C} - r_f \frac{\delta(C)}{C} - l_f^{\max} \qquad (18)$$

(18) makes this algorithm predictable and also allows calculating the worst-case traffic pattern on the network design stage. If more stochastic calculations are needed for network, EBF model is usually more preferred.

We can also calculate packet delay guarantees for FC traffic model with parameters $(C,\delta(C))$. We assume that virtual time calculation method is kept the same and server capacity is not exceeded. In order, to calculate delay guarantee of the algorithm, for each packet p_f^j with the assigned rate r_f^j, we need to introduce the notion of Expected Arrival Time (EAT), which determines deadline for each packet and is calculated using the following formula:

$$EAT(p_f^j, r_f^j) = \max\left(A(p_f^j), EAT(p_f^{j-1}, r_f^{j-1}) + \frac{l_f^{j-1}}{r_f^{j-1}} \right), \qquad j \geq 0$$

Now we can evaluate the delay guarantee size for each packet p_f^j from flow f. It can be shown, that for $(C,\delta(C))$ variable rate FC server, capacity of which is not being exceeded, departure time L of packet p_f^j is bounded by inequation:

$$L(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n\in Q \wedge n\neq f} \frac{l_n^{\max}}{C} + \frac{l_f^j}{C} + \frac{\delta(C)}{C}$$

This maximum delay guarantee calculation for SFQ algorithm is suitable for integrated service networks, where some services are served using priority queueing and low priority flows are served using SFQ discipline. If high priority flows are policed or shaped using token bucket algorithm[6] with rate ρ and burst size σ, then the residual of the total bandwidth can be treated as fluctuation constrained variable rate channel with parameters *(C-ρ,σ)*. If high priority packet arrivals form Poisson process, then FC model is not suitable for the bandwidth left for low priority flows. In this case EBF model must be applied to the low priority packet rate.

The maximum packet delay bounds for SFQ can be compared with the corresponding values for WFQ algorithm. For WFQ maximum delay bound can be presented as $EAT\left(p_f^j, r_f^j\right) + \frac{l_f^j}{r_f^j} + \frac{l_{\max}}{C}$. If we denote the difference in maximum delay values for WFQ and SFQ algorithms by $\Delta\left(p_f^j\right)$, then

$$\Delta\left(p_f^j\right) = \frac{l_f^j}{r_f^j} + \frac{l_{\max}}{C} - \sum_{n \in Q \wedge n \neq f} \frac{l_n^{\max}}{C} - \frac{l_f^j}{C} \qquad (19)$$

For simplicity let's assume, that $l_f^j = l_{\max} = l_n^{\max} = l$ and $r_f^j = r_f$. Then formula (19) will change as follows:

$$\Delta\left(p_f^j\right) = \frac{l}{r_f} - \frac{\left(|Q| - 1\right) \cdot l}{C}$$

where *Q* denotes the number of flows in the system. It is easy to see, that $\Delta\left(p_f^j\right) \geq 0$ only if

$$\frac{1}{|Q| - 1} \geq \frac{r_f}{C}$$

This inequation holds for low throughput flows, as the share of bandwidth they consume is usually much less than $\frac{1}{|Q| - 1}$.

Besides maximum delay value SFQ also efficiently reduces average delay bound. This can be seen intuitively, as SFQ orders packets according to their start tags always stamps them with smaller value and queueing them at the earlier instance, then WFQ which uses finish tags for packet stamping. The results of maximum and average delay analysis are shown in Fig. 15 and Fig. 16[7].

---

6 Token bucket algorithm is usually used for admission control at the traffic flow source or network boundary, which it crosses. Admission control issues and token bucket algorithm will be discussed in detail in the next chapter.

7 These simulation results were received by P. Goyal and were presented in [17].

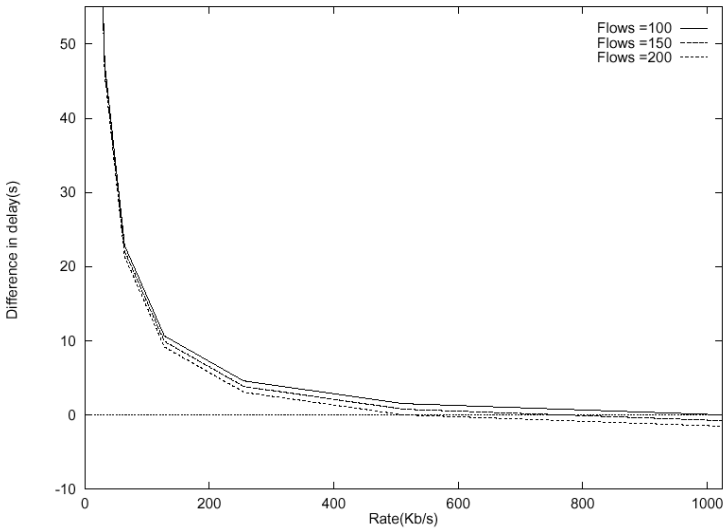


Fig. 15

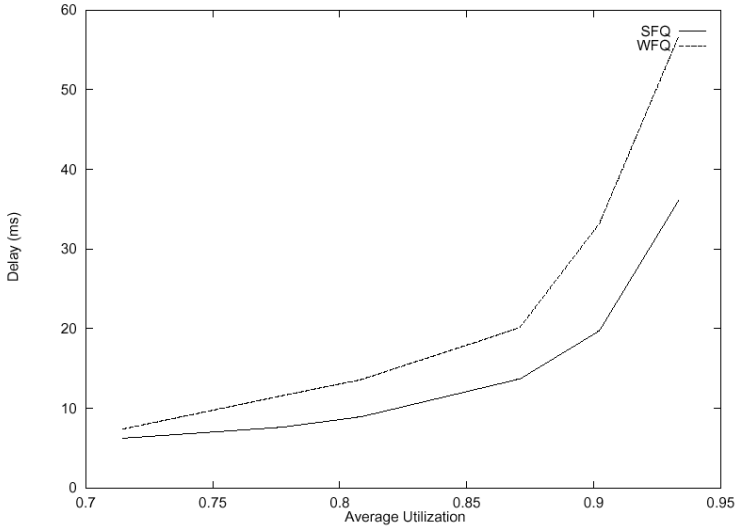


Fig. 16

In Fig. 15 it is shown the change of maximum delay difference $\Delta\left(p_f^j\right)$ for different flow rates r_f and numbers of flows Q. As it was mentioned this difference is quite large for low throughput flows and becomes negative for high throughput ones, but not much. This means, that WFQ guarantees a bit less maximum delay than SFQ for flows with high rate (>600-800Kbps).

Fig. 16 illustrates the dependence of average delay on the server (link) utilization. It is clear, that SFQ provides strictly better average delay values for all utilization levels. These delay calculations can also be used for estimation of end-to-end delays for multihop network paths.

In this chapter different queueing disciplines were presented. Each of them was investigated in detail, including significant parameter evaluations. Most of these queueing algorithms are widely deployed in contemporary packet switched networks, but some of them are currently on the development stage and their implementations will be integrated into data networks in the nearest future.

4. Congestion Avoidance Mechanisms

Up to this moment we discussed congestion control mechanisms, which begin to work only when congestion has already occurred. All the queueing algorithms are mainly designed to reduce the overall impact of congestion on the total network performance.

Another aspect of network design is to find way to avoid congested situation in the network. The class of mechanisms, that prevent congestion in advance just before it will happen, are called congestion avoidance algorithms.

There are two different ways to achieve this goal. One is the method of limiting incoming traffic to conform to some predetermined characteristics at the source or network boundary, making traffic more predictable at the core nodes. This limiting policy may be applied to a flow on conditional basis: for example if it does not confirm to some traffic policy. This assumes that traffic is continuously measured in time to detect the event of policy violation. This makes it easier to design the network with minimized probability of congestion and to simulate network performance in real-time in advance.

Another approach to congestion avoidance problem is so called Random Early Detection (RED). This method can be deployed either at the network border or in its backbone. The main idea is that when some node uses this mechanism it monitors status of the network and when it approaches congestion by some threshold starts special measures toward the whole (or subset of) traffic it is crossed by. One of the possible behaviours of RED node in such situation is to drop packets according to some predefined drop policy.

As it was shown in chapter 1, in multiservice network traffic must be differentiated at the network border and optionally at each node it passes through. This differentiation can be easily used by admission control algorithm to apply different admission rules to different types of traffic along the network. The use of packet differentiation in admission control inserts additional packet classification stage into packet processing sequence.

The admission control does not assume that there is some signaling method among network nodes, but it must be able to take advantage of it. The scope of admission control algorithm is usually limited to a single node.

The total scheme of admission control mechanism performed for each active policy in the system is clearly depicted below in Fig. 17:

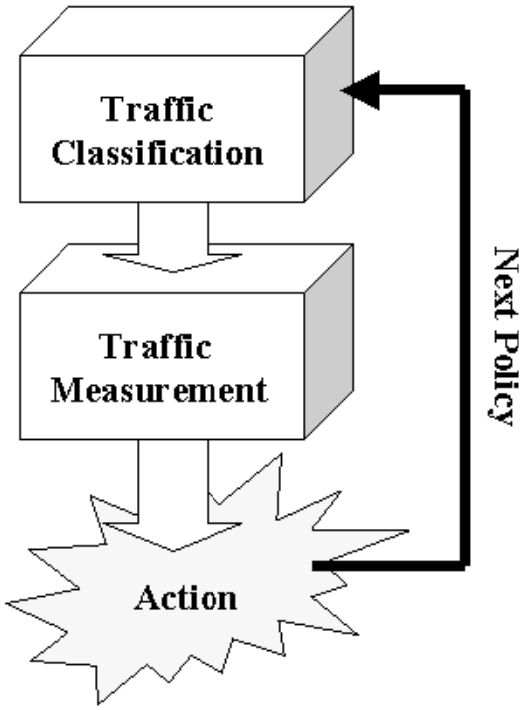


Fig. 17

Now we will have a detailed look into different algorithms of congestion avoidance. Let's start with traffic admission control.

4.1 Traffic Admission Control

Admission control module in the network node determines in what to do with the incoming packet. The packet is finally transmitted or dropped according to some traffic policy predefined by the network administrator.

As it was mentioned above, first of all any incoming packet must be classified to belong to some class of service. This differentiation must be done based on some protocol specific fields. For example for TCP/IP these can be source and/or destination addresses/ports, protocol type, type of service or any routing or QoS signaling derived information.

Traffic admission control can be divided into two different behavior types: traffic policing and traffic shaping. The main difference between them is the following: if an incoming packet violates some predefined traffic policy, traffic policer[8] drops this packet, while traffic shaper simply delays its transmission by queueing it into special buffer. In other words the presence of queue admission control module determines if

[8] Traffic policer is the software module in the network node that performs traffic policing action. The similar definition is correct for traffic shaper.

it is policer or shaper. Traffic policing process is also called Committed Access Rate (CAR).

Admission control can use cascade of policies, which are examined in sequence. This process can be interrupted at any round and special action can be performed. This allows several policies to be applied to the same packet performing more granular flow control.

3

Admission control mechanism is usually implemented at the network border, as close to the source of traffic as possible. The most frequently it is applied at the either side of the link connecting CPE and the network edge nodes. The place of admission control in network topology is shown in Fig. 18.

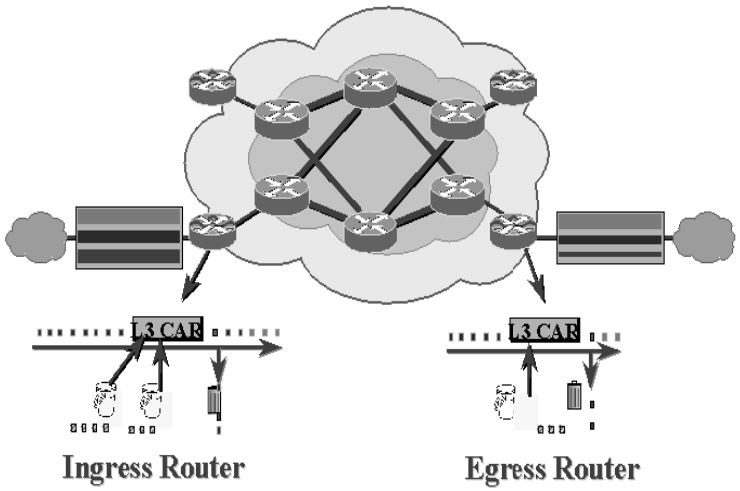


Fig. 18

Traffic policing and shaping allows make traffic pattern at the core to conform to some predefined characteristics making it more predictable. Traffic policing can be also used for traffic coloring and recoloring (marking packets with QoS values).

After checking the packet to conform to some policy (which will be defined in detail a bit later), one of the following actions can be taken:

- Transmit packet
- Drop packet
- Continue with the next policy
- Set precedence bit and transmit
- Set precedence and continue with the next policy

Finally packet is either transmitted or dropped and the process is repeated for the next packet.

Traffic shaping instead of cutting off the excess traffic shapes is using special algorithm. Packets non-conforming to some rules defined by this algorithm are queued for later transmission. Of course packets are lost in case this queue overflows. Traffic shaping feature can be used to generate congestion signals in case it receives some threshold in the queue, if the underlying protocol supports such signaling. For example, in frame relay network in case

of shaping becomes active, it can sent BECN and FECN signals to source and destination of the flows to which shaping is applied. This can be used to dynamically adapt to changes in traffic characteristics. For IP networks there is no native explicit congestion notification mechanism. Admission control can be implemented to respond to resource reservation protocol (RSVP) requests and include them into its decision process.

The working scheme of traffic shaping is presented in Fig. 19.

As it can be seen from the figure, any queueing discipline can be implemented with traffic shaping.

Now let's get down to traffic limiting on which packet admission decision is based either during traffic policing or shaping. It is called Token Bucket or Leaky Bucket algorithm. This algorithm is described by a model of a bucket of depth σ with a hole. Special token is put into the bucket at a constant rate ρ. These two parameters completely describe any token bucket filter. Parameter σ is also called burst size.

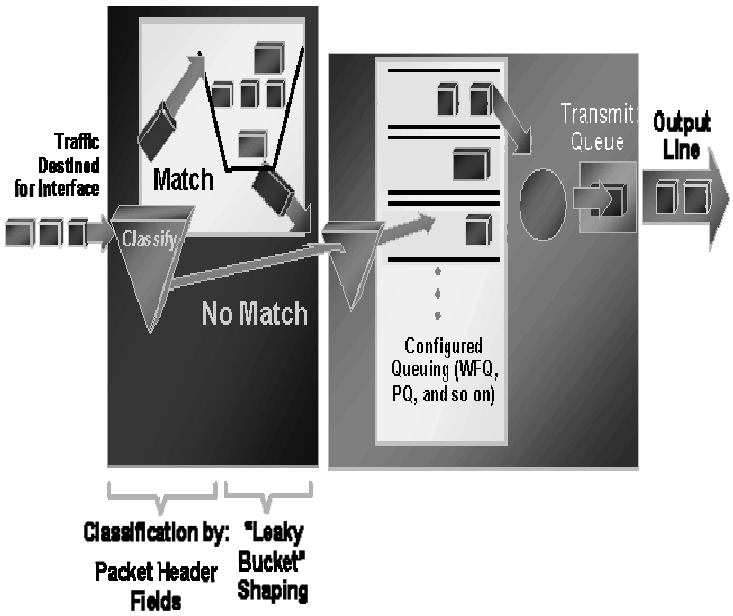


Fig. 19

When a packet arrives to empty queue[9] and it is to be transmitted through the egress interface, a certain number of tokens equivalent to the size of packet are extracted from the bucket, until the bucket gets empty. If the incoming packet finds bucket empty or insufficient tokens are available, packet is either dropped[10] or delayed.

[9] Here we assume that we have shaper filter with input queue

[10] Packets are immediately dropped in traffic policers, as they don't have any queue. In shapers packets can be also dropped when their queue overflows.

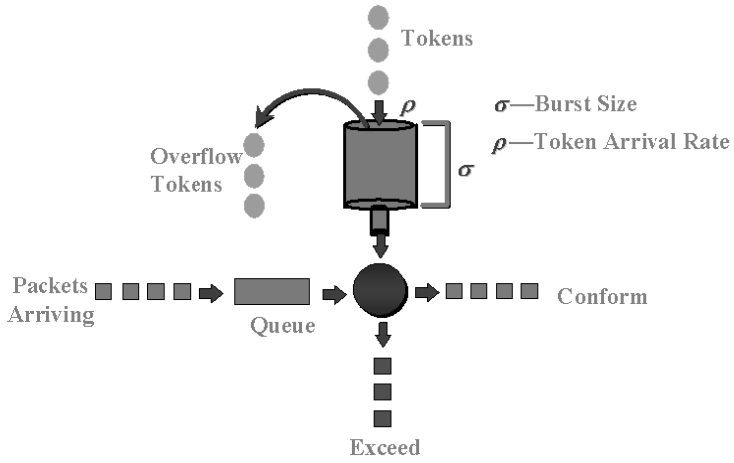


Fig. 20

If token bucket is full, excess tokens that arrive to the bucket are lost and can't be used for future packets.

One of the significant parameters of token bucket that appear at run-time is load factor γ, which represents the relative load created by input packet stream. It can be calculates as $\gamma = \frac{\lambda}{\rho} \cdot 100\%$, where λ is the average packet arrival rate and ρ is the token generation rate.

Token bucket filter makes characteristics of bursty traffic more predictable by limiting it burst size with σ value. This parameter determines how many packets can be sent out of the interface back-to-back. It is clear, that when $\sigma = 1$, the output stream will be of constant rate ρ and the burstiness of incoming traffic will be completely eliminated. On the other hand, for $\sigma = \infty$, the behavior of output traffic depends on the value of γ. For $\gamma < 100\%$ no filtering will be applied as number of tokens in the bucket will continuously grow and will be able to pass any burst[11]. If $\gamma < 100\%$, then all the bursts will be cut, as most of the time the bucket is empty. For boundary condition $\gamma = 100\%$, the behavior of the token bucket completely depends on the initial state of the filter, which is described the number of tokens in the bucket at $t=0$.

Now let's give more formal definition to the token bucket algorithm[12].

```
Foreach(p)
 {
  if (Size(Queue)==0) then
   {
```

[11] The filter is assumed to be in stationary state. During initialization process there is possibility of some shaping or packet drop in case of the small initial number of tokens in the bucket.

[12] The algorithm is presented for traffic shaping case, as for more complex one. It can be easily simplified to policer by removing queue and performing packet drop, when bucket is empty.

```
    if(TokenInBucket>=Size(p))

{
      Send[13](p);
      TokensInBucket -= Size(p);
     }
    else Enqueue(p);
   }
  else
   {
```

`TokensInBucket` variable is usually initialized with 0, but this value can be arbitrary from the interval $[0,\sigma]$. Queue service discipline can be chosen arbitrarily. Token arrival distribution is deterministic and has constant rate ρ. It was shown through simulations, that this method of token generations is the most effective by packet loss ratio and packet delay quantities.

Leaky bucket algorithm can be improved to use virtual clock method, which will use two additional values[14]: real time, which will flow normally and virtual clock, which will measure size of burst, that is being served by the filter. Here's the formal definition of this algorithm:

```
Foreach(p)
 {
  VirtualTime = max(RealTime, VirtualTime);
  if(VirtualTime+Size(p)/Rate > RealTime + BucketDepth)
   Drop(p);
  else
   Send(p);
  VirtualTime += Size(p)/Rate;
 }
```

It is to be mentioned that for this modification of the algorithm bucket depth is measured in seconds and represents time it will take to transmit all the data from the full bucket. `Rate` variable holds the value of capacity of outgoing link.

Token bucket algorithms can also be cascaded at the network boundary to make traffic differentiation based on its rate. As an example let's consider dual leaky bucket algorithm, which is widely used in ATM networks, but can be easily adapted for packet switched IP networks. Let's describe its implementation.

Let's imagine the case, when we have three classes of traffic arriving to the network boundary node with egress link capacity C and we need to give them different PHBs. The first class receives guaranteed service of constant rate ρ_α. Service α is marked with DSCP value x. The second service β receives differentiated with maximum allowed rate ρ_β. It is marked with

[13] Here it is assumed, that Send() procedure will place the packet directly to output queue to avoid congestion in case interface is busy.

[14] Virtual clock leaky bucket is recommended algorithm by ATM Forum to be used for traffic policing in ATM switches, but here it is adapted to police variable length packet switched networks.

DSCP *y*. Finally, packet that don't belong to classes α or β are placed into best-effort class ω, which receives at any moment *t* non-reserved spare bandwidth $\rho_\omega = C - \rho_\alpha - \rho_\beta(t)$.

Here we will use triple token bucket algorithm, which consists of three buckets *A*, *B* and *Z*. One stream of generated tokens is placed into bucket *A* at rate ρ_α, which is performing according to the single leaky bucket algorithm as described earlier[15]. In the same way tokens at rate ρ_β are placed into bucket *B*. For best effort traffic we need additional bucket *Z*. It will be forced to have incoming token rate $\rho_\omega = C - \rho_\alpha - \rho_\beta$. Packet that belongs to service ω will also receive bandwidth not consumed by differentiated service β. Let's denote bucket depths by σ_A, σ_B and σ_Z respectively. Then triple bucket scheme suitable for networks model described above can be built as it is shown in Fig. 21.

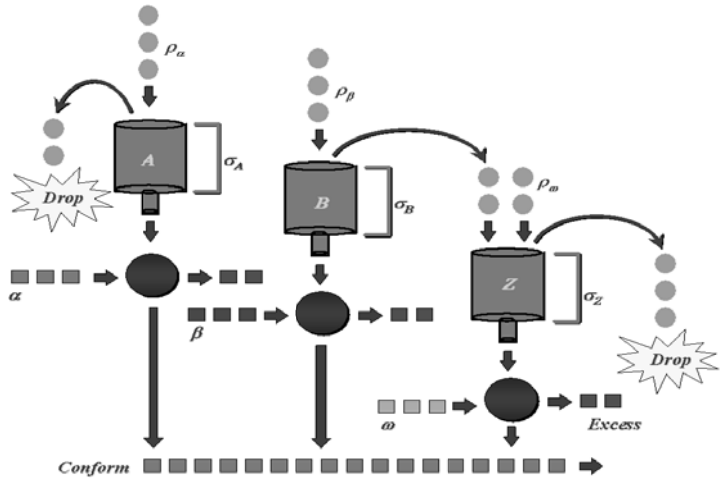


Fig. 21

In the Fig. 21 the difference between guaranteed and differentiated services is clearly seen. Guaranteed service does not share its reserved bandwidth with other classes and all its overflowed tokens are simply dropped. Behavior of differentiated service is absolutely different. In case it underloads its maximum allowed bandwidth, its spare bandwidth is given to other classes. In our example it is passed to best-effort traffic. This behavior is performed by putting tokens, that overflowed the bucket *B* into bucket *Z*, that corresponds to best effort service ω, in addition to its own packet arrival process. Tokens, that overflow bucket *Z* are dropped as they violate the condition $\rho_\alpha + \rho_\beta + \rho_\omega \leq C$.

Excess packets, as it was discussed earlier, are either queued or dropped. But the choice is not limited by these two actions. To achieve more granular behavior, Random Early Detection (RED)[16] mechanism can be applied to excess packets as well as any other packet drop policy. Thus token bucket algorithm is very flexible tool for network design and flow management.

We just discussed admission control on per-packet basis. Another issue is the per-flow admission control. This type of admission control is especially important for real-time connection oriented services like VoIP. During each call setup decision must be made to accept or reject new flow based on its predicted parameters and current state of system. It is natural to accept a new flow, if this doesn't lead to QoS policy violations for other flows.

[15] Both versions of the algorithm presented in this paper are suitable.

[16] Random Early Detection will be discussed in the next section.

These flow parameters must be negotiated at session setup stage. To simplify decision process it is quite useful to approximate expected flow parameters with token bucket model. Thus characteristics of each flow can be presented by its average rate ρ and burstiness σ. This method makes it easier calculation of the effect of flow acceptance into the system.

There are two methods to evaluate current state of the system. The first method is to calculate system state using values negotiated during session setup. This gives the worst-case model, which will lead to low resource utilization. The other method is to base the system state estimation on measured values. This method allows achieving higher level of utilization. The main drawback of real-time measurements is, that they need more processing power and there is some positive probability p_f, that after flow is accepted, QoS guarantees for other flows will be violated. But as some real-time applications seem to be relatively tolerant toward short-time policy violations, this method looks very attractive

This measurement method can't be applied to the new flow parameters estimation process, as there is no information about its past history available in advance. But as it is accepted and starts to send data, this initial value won't be used any more, dynamically measured parameters of the flow will be used instead.

The key to solve this task is to calculate the possible effect of the new flow on the system performance. Let's analyze an example: the traffic passing through packet switch is differentiated into guaranteed flows with instant measured bandwidth $\hat{\rho}_G$ and sum of reserved rates R_G and n classes predictive flows, that are served using priority discipline. Each class is characterized by measured bandwidth $\hat{\rho}_j$, measured queueing delay $\hat{d}_j$ and pre-defined delay D_j. C denotes total link capacity.

Now let's consider some new flow α, which requests real-time service from the network. We assume that this flow can be approximated by average rate ρ_α and bucket depth σ_α. Here two cases are possible: 1) If flow α requests guaranteed service, then it will impact both guaranteed and predictive flows; 2) if flow α requests predictive service of class k, it will only affect predictive flows of class k and lower. Now let's investigate these two cases in detail.

It was proved, that if i flows network traffic consists of confirm to token bucket model, then their worst-case delay of class j for strict priority service discipline of FIFO queues can be evaluated using the following equation:

$$D_j = \frac{\sum_{i=1}^{j} \sigma_i}{C - \sum_{i=1}^{j-1} \rho_i} \quad (20)$$

This means, that the worst-case delay happens when all the classes 1 through j dump their buckets simultaneously, while sources of classes 1 through *j-1* still keep sending at their reserved rates.

Now change in delay for different requirement cases of arrived flow α can be calculated. If class α requires a predictive service k, then it will impact either class k and all the lower priority classes j, while guaranteed and higher priority classes are kept unaffected. These impacts can be evaluated using formula (20) and will respectively look like:

$$D_k^* = \hat{D}_k + \frac{\sigma_k^\alpha}{C - \hat{\rho}_G - \sum_{i=1}^{k-1} \hat{\rho}_i} \qquad (21)$$

$$D_j^* = \hat{D}_j \frac{C - \hat{\rho}_G - \sum_{i=1}^{j-1} \hat{\rho}_i}{C - \hat{\rho}_G - \sum_{i=1}^{j-1} \hat{\rho}_i - \rho_k^\alpha} + \frac{\sigma_k^\alpha}{C - \hat{\rho}_G - \sum_{i=1}^{j-1} \hat{\rho}_i - \rho_k^\alpha} \qquad (22)$$

For flow a requesting guaranteed service will affect either other guaranteed flows and all the predictive flows and for this case (20) can be easily rewritten for the predictive service *j* in the following way:

$$D_j^* = \hat{D}_j \frac{C - \hat{\rho}_G - \sum_{i=1}^{j-1} \hat{\rho}_i}{C - \hat{\rho}_G - \sum_{i=1}^{j-1} \hat{\rho}_i - \rho_G^\alpha} \qquad (23)$$

From equations (21) – (23) it is clear, that the worst impact on the delay of predictive class flows has the admission of high priority predictive flow. The equations (21) – (23) can be easily used in admission control criteria evaluation. This criterion can be extremely easy:

For predictive class flows request must be denied if:

1. The requested reserved rate of the new flow α makes total traffic rate exceed available bandwidth *C*: $C < \rho_k^\alpha + \hat{\rho}_G + \sum_{i=1}^{N} \hat{\rho}_i$.
2. If new delay estimation calculated using (21) and (22), that takes into consideration the new flow violates reservation policy of some predictive flow already admitted into packet switch.

The condition of request denial can be presented in the similar way for the case of the new flow requesting guaranteed service. The request is denied if:

1. The total bandwidth reserved for guaranteed flows including the requesting flow exceeds total interface bandwidth available: $C < \rho_G^\alpha + \hat{\rho}_G$.
2. If new delay calculations made according to (23) violate reserved values of any predictive or guaranteed flow.

It is easy to see, that if measures values $\hat{D}$ and $\hat{\rho}$ are substituted with absolute pre-defined values *D* and ρ, then this algorithm will transform into classical worst-case admission control scheme.

It is to be mentioned that this algorithm does not depend on measurement algorithm selected and it can be chosen according to current needs. The simplest method is to calculate arithmetical average over constant time periods. Another more complex choice may be exponential averaging over variable periods of time. But one must remember that complex measurements are processor intensive tasks and have to be implemented with extreme care.

4.2 Random Early Detection

If network node is in congested state, then it's queue is permanently full. If a packet arrives to this nodes and finds, that there is no place in the queue for it, it is dropped. Optionally congestion notification is sent to the source (and sometimes destination). In response to packet drop, which is usually detected using timeouts or congestion notification well-behaved source must decrease its packet sending rate toward congested node. This type of network node behavior, when packet arriving to the full queue is dropped it called Tail Drop scheme.

Tail drop is extremely inefficient method, as it is applied only when congestion has already happened. In TCP networks it creates effect of so called global synchronization, which occurs when packets are dropped consequently from several TCP sessions. This makes TCP simultaneously reduce its windows for these sessions, which significantly reduces utilization level. At the same time, as experiments showed, packet drop percentage distribution for flows is unfair relative to their consumed bandwidth shares. This effect is the strongest in case of FIFO queueing.

There are special methods that allow avoiding congested situation before it really happens. One of such methods is Random Early Detection (RED). It was designed to avoid drawbacks if tail drop method named previously.

1. Remove global synchronization effect from TCP sessions;
2. Maintain fair bandwidth usage even in case of FIFO queueing;
3. More control on average queue size;
4. Relatively small computation per-packet overhead.

RED is based on the idea to start packet drop before congestion has occurred, when there is more freedom to choose which packet to discard. Some dropping policy, which will dynamically adapt to changing traffic pattern, must be applied to traffic flows. It is clear, that RED can be easily changed to work in case of traffic differentiation. In this environment different drop characteristics can be applied to different classes of service. This will allow more granular traffic parameter tuning: increasing reliability of some classes of flows on the expense of others. This type of algorithm is called Weighted Random Early Detection (WRED) and its essence is shown in Fig. 22.

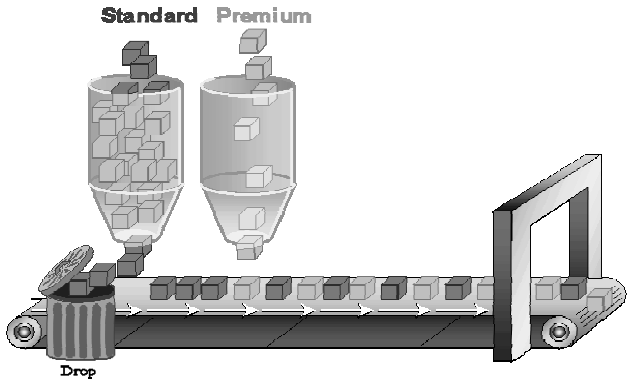


Fig. 22

Fig. 22 illustrates working scheme of WRED algorithm in case of two traffic classes: standard and premium. Premium traffic has higher importance and must experience less drop rate. This can be achieved by increasing preliminary drop rate for standard flow if

some congestive threshold is reached, making its transport protocol flow control algorithm reduce standard traffic flow rate leaving more unused bandwidth to be consumed by premium traffic.

It is natural to make the drop rate proportional to the level of congestion with linear dependence function. At the same time packet drops have to be uniformly distributed along time axis. Of course the dependencies may be also nonlinear, but this will insert into algorithm additional unnecessary complex calculations.

Now let's see what happens to the traffic from the packet drop probability point of view in both, tail drop and RED queue management.

In Fig. 23 are shown packet drop probability distribution functions for both packet drop schemes. In case of tail drop (Fig. 23a) as queue reaches its limit max_b, the packet drop probability eventually reaches 1, which means that every packet is dropped. Before max_b limit no drops are preformed.

For RED algorithm (Fig. 23b) there are no drops while average queue size is lower, than some min_{th} value. After average queue size exceeds min_{th} but is still under max_{th}, it starts to drop packets with probability, which increases linearly, until it reaches boundary max_p value. When average queue gets higher than max_{th}, all packets are dropped similarily as in tail drop case.

It must be admitted, that we use not the actual queue size but its average value. This is done to reduce its bias toward bursty traffic patterns. Average queue size will be calculated using weighted moving average algorithm with weight w_q. The weight value determines how quickly averaging algorithm will adapt its value to change in actual queue size and its value changes in *[0,1]* range. The formula for calculation of weighted moving average has recursive nature:

$$avg_n = (1 - w_q) \cdot avg_{n-1} + w_q \cdot q \quad (24)$$

To make clear the role of w_q parameter, let's consider its boundary values: *0* and *1*.

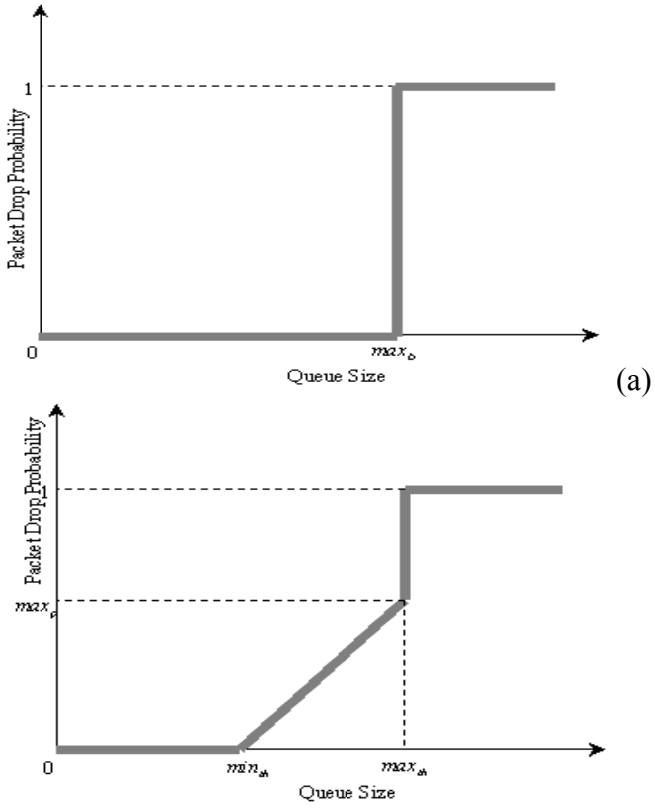


Fig. 23

- $w_q = 1$. Value of average will coincide with the actual queue sizes, as $avg_n = q$. In this case drops will be triggered for all packets that violate threshold limits, even in short bursts.

- $w_q = 0$. Value of average will not adapt to queue size changes at all, as $avg_n = avg_{n-1}$. It will only depend on its initial value and stay without change.

The conclusion is, that actual average values $0 < w_q < 1$ allow more precise definition of the maximum duration and size of burst that will be accepted without triggering drop mechanisms. It is recommended not to set w_q parameter less than 0.001. The value widely used in different implementations is 0.002.

As it was stated above, other significant parameters in RED tuning are min_{th} and max_{th}. min_{th} determines the lower bound queue size for packet drops and must be higher for bursty traffic patterns. min_{th} also represents preferred average queue size, while max_{th} corresponds to maximum average queue size and works as upper limiting factor. It is recommended not to set these threshold values too far one from another to avoid global synchronization effect in the network, when all the sessions reduce their window sizes at the same time significantly reducing link utilization level. The classical dependence is $max_{th} = 4 \cdot min_{th}$. Widely used values for these parameters are $max_{th} = 5$ and $min_{th} = 20$.

There are two parameters related to a single packet drop probability in RED algorithm: initial and final packet drop probability. Initial drop probability is calculated as linear function

of average queue size. It also depends on threshold values chosen. Initial packet drop probability is computed as follows:

$$p_b = \max_p \cdot \frac{avg - \min_{th}}{\max_{th} - \min_{th}} \quad (25)$$

Here max_p is the maximum desired packet drop probability in case, when average queue size *avg* reaches max_{th}.

Final packet drop probability determines the distribution of inter-drop gap, which means the number of packets that are transmitted between adjacent packet drops. This quantity is uniformly distributed variable in the $[1,1/p_b]$ range. If we denote random value of inter-drop gap by *X*, then *X* probability density distribution function is the following:

$$\begin{cases} P[X = n] = \dfrac{p_b}{1-(n-1)p_b} \prod_{i=0}^{n-2} \left(1 - \dfrac{p_b}{1-ip_b}\right) = p_b, 1 \le n \le 1/p_b; \\ P[X = n] = 0, n > 1/p_b. \end{cases} \quad (26)$$

Where *n* is the current number of packets since the last drop. This method gives significantly more fair distribution of packet drops over time axis, than simple geometric random variable method. The average value for *X* gap is $E[X] = 1/(2p_b) + 1/2$.

The formula for calculation of final packet drop probability can be performed in the following way based on the previous discussion:

$$p_a = \frac{p_b}{1 - n \cdot p_b} \quad (27)$$

It is clear, that final drop probability will tends to p_b for $n \to \infty$.

In RED algorithm the system idle state is treated in special way, because it is executed only at packet arrivals it will be unaware of the idle state period and as a result the average queue length will have unrealistic value as it will not dissolve correctly during idle time. Special measures are taken to avoid this incorrect behavior of the algorithm. When recovering from idle state, algorithm assumes that during this idle time it received and served *m* small packets, that arrived back-to-back and were served immediately, thus keeping queue length at *0*. If we use (24) formula in this case, then the value at the end of the idle period will be

$$avg_n = (1 - w_q)^m \cdot avg_{n-1} \quad (28)$$

This formula will allow average queue value to reduce exponentially to the idle time elapsed.

Now we can present RED algorithm, which will be executed for each arrived packet *p*, as a whole in the formal notation:

```
Initialization Module:
avg = 0;
count = -1;

RED Enqueueing Module:

foreach(p)
```

```
{
 q = Len(Queue);
 if(!Empty(Queue))
  avg = (1-wq)*avg + wq*q;
 else
  {
   m = Linear(CurrentTime - IdleStartTime);
   avg = (1-wq)^m*avg;
   IdleStartTime = -1;
  }
 if(avg>minth&&avg<maxth)
  {
   count++;
   pb = maxp*(avg-minth)/(maxth-minth);
   pa = pb/(1-count*pb);
   if (Drop(p, pa))
     count = 0;
   else
     Enqueue(p);
  }
 elseif (avg>=maxth)
  {
   Drop(p);
   count = 0;
  }
 else
  {
   count = -1;
   Enqueue(p);
  }
 if(Empty(Queue)&&IdleStartTime==-1)
  IdleStartTime = CurrentTime;
}
```

In this algorithm description function `Linear()` represents linear dependence of its value on its argument. The function `Drop(p, pa)` performs packet drops with the given probability p_a. It returns 'true' if succeeds and 'false' if fails and packet is enqueued. `count` is equivalent for inter-drop gap counter n presented earlier. This variable is set to –1, when average queue size if below min_{th} and packet is not subject to drop.

`IdleStartTime` variable has nonnegative values only during idle periods, otherwise it is set to –1. It is also to be mentioned that idle time processing algorithm is not quite correct. Let's imagine the situation, when packet arrives to the queue when both its instant and average queue sizes are equal to 500. The next packet arrives only after 250 packet times. It is clear, that instant queue size this new packet discovers will be 250. But as the averaging algorithm works only at packet arrivals, average queue size value will be a bit less than 500. This may lead to unnecessary packet drops and as a result low link utilization level. The solution to this

problem is to include average calculation at packet departures too. The addition for each *p* packet departure to the algorithm presented above is quite simple:

```
RED Dequeueing Module:

if(!Len(Queue))
  avg = (1-wq)^m*avg;
Dequeue(p);
```

This module provides accurate queue average estimation even during absence of incoming packets.

The RED algorithm version presented here is packet oriented, which means, that it uses packets as a queue measure, but it can be easily adapted to byte oriented environment. In order to transform packet oriented RED in byte oriented one, p_b parameter before being used in p_a calculation (27) needs to be normalized according to packet sizes:

$$p_b = p_b \frac{PacketSize}{MaxPacketSize} \qquad (29)$$

After this normalization large packets will have higher drop probability than small ones.

Another issue about RED algorithm is its performance. As its code needs to be executed at each packet arrival and departure, it must be extremely effectively implemented not to affect the total interface throughput. On the other hand, calculations presented above look quite complex.

After a couple of changes and careful parameter choice RED algorithm can be very efficient. Average queue formula (24) can be rewritten in the following way:

$$avg = avg + w_q(q - avg) \qquad (30)$$

If w_q will be chosen as a negative power of 2, then this operation can be implemented using only one shift and two additions. For the case with empty queue, table lookup can be made for different idle time intervals with certain granularity to get a value for $(1 - w_q)^{idletime/s}$. Here *s* is transmission time of a small packet.

Now let's turn to drop probability processing. Equation for p_b calculation (25) can be presented as

$$p_b = C_1 \cdot avg - C2 \quad ,$$

where

$$C_1 = \frac{\max_p}{\max_{th} - \min_{th}};$$

$$C_2 = \frac{\max_p \cdot \min_{th}}{\max_{th} - \min_{th}}.$$

Here C_1 and C_2 depend only on static predefined parameters and can be calculated in advance. Parameters max_p, max_{th} and $\min_{th}$ can be selected so, that C_1 will become power of two. In this case (25) can be implemented using only one shift and one addition.

When $min_{th} \leq avg \leq max_{th}$ special random number $R \in [0,1]$ is generated to determine if current packet will be dropped. Another efficiency gain can be obtained by generating R only after packet drops and keep it unchanged if packet is placed into output queue.

RED algorithm has some drawbacks to be mentioned here. Namely, the idea, that during pure random drop flow consuming larger bandwidth fraction will receive more packet drops seems nice for general traffic scheme, but its efficiency is not clear in TCP traffic environment. The main problem is that TCP implementing slow-start algorithm even in case of a single packet shuts down its window size too quickly[17]. Thus difference between of losing only one and several consequent packets for flows is not very big. This behavior reduces fairness in bandwidth distribution among flows and lowers link utilization. This drawback can be avoided by using per flow state mechanism. Of course, this will make the implementation more complex and less efficient, but will make RED drop packet only from flows that exceed their bandwidth fair share.

In this chapter different congestion avoidance algorithms were discussed beginning from network periphery toward its core according to algorithm implementation location. Congestion avoidance makes network control process easier than in the congestion control case, as it tries not to reach nearly link saturation without subsequent overload leading to congestion situation and as a result increase in queueing delay and packet loss. Thus efficient implementation of congestion avoidance mechanisms is of extreme importance for performance of contemporary data networks.

[17] Most TCP implementations reduce current window to 1 after each packet loss.

5. QoS Signaling and Information Distribution

Recently we have discussed mechanisms that can be deployed on the per node basis to achieve QoS and make traffic more predictable. In this chapter will be presented some measures to join these per-node QoS guarantees into a whole system.

This type of integration is performed by means of QoS signaling protocols that provide possibilities of coordination between standalone packet switching nodes. Each of these protocols plays its in unique role in this integration process.

First tries in this direction were made in early 90's, when preliminary draft for new resource reservation protocol (RSVP) was proposed. This protocol will be discussed first in the following section.

As RSVP performs well only inside limited autonomous system (AS), additional measures were introduced to extend QoS coordination beyond AS boundary. A new standard for QoS propagation over BGP (QPPB) was created.

Later some methods were developed to deploy ATM like behavior in packet switched networks[18]. This class of protocols can be presented by Multiprotocol Label Switching (MPLS). The main idea lying under MPLS is policy routing paradigm, which significantly extends classical routing process. QoS routing is another large topic so it is beyond the scope of this document and will be subject for my future work.

The greatest improvement built by means of QoS signaling methods over ATM networks is so called Content Aware Networking. In ATM network when data gets encapsulated into ATM cells by SAR sublayer information about its payload type is usually lost.

VCs are provisioned statically over ATM network fabric and QoS are applied to traffic on a per-VC basis. Network does not know anything about content it carries in each VC, as cell switches have access to only to 5 byte ATM header of the cell.

On the contrary QoS signaling like MPLS and RSVP are integrated with (classical or policy) routing process. They have full access to all packet headers including transport layer TCP/UDP headers giving a complete information about application data being carried by each packet. This feature inserts great power into decision making process on each intermediate switch allowing them to effectively distinguish data belonging to different types of applications on dynamic basis flexibly adapting to changing network environment.

Thus QoS signaling can be named among mandatory optional of QoS networking model, which can become mandatory with the network size and complexity growth. At the same time this methods are usually kept independent from routing and packet scheduling methods used by each network node. This makes them quite universal in use.

[18] Here variable packet switched networks are assumed, contrary to ATM fixed length cells.

5.1 Resource Reservation Protocol

RSVP is a network QoS distribution and control protocol, which enables applications to reserve desired volume of resources along the whole path through the network. It is adapted to work not only with point-to-point applications, but also with multicast ones.

RSVP does not perform any functions besides QoS signaling and control. It is decoupled from routing (either unicast or multicast) and per-node packet scheduling mechanisms, which makes it absolutely independent.

RSVP is receiver-oriented protocol. This means, that it receiver, who makes actual reservation along the network path. This is quite natural, because usually sender is capable of sending arbitrary volume of traffic toward receiver and only receiver has to decide how much it wants to accept. RSVP makes reservations only for unidirectional flows. Naturally for duplex flows duplex flows distinct reservations are needed in both directions.

Typical RSVP network is shown on Fig. 24. It in this sample there is one sender to several receivers and internetwork consists of either RSVP or non-RSVP routers.

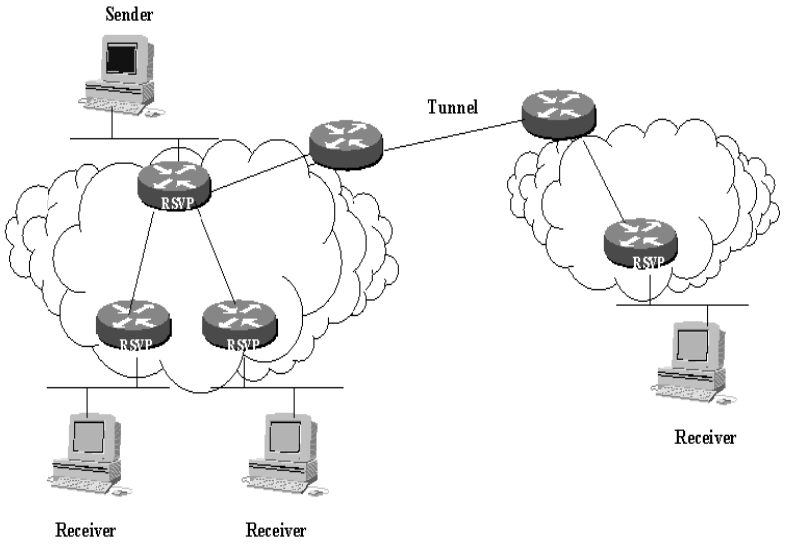


Fig. 24

With multicast reservation additional difficulties appear, as senders and receivers are free to join and leave multicast groups using IGMP protocol and as a result different types of reservations are possible to implement. First of all reservations can be distinguished as shared and distinct. In case of shared reservation reservation the total amount of resources being reserved on a single link to some receiveris equal to the largest reservation request toward the same receiver or the group of receivers. If reservation larger than requested is already made somewhere on the path, no additional reservation is made beyond this point. This type of reservation is extremely useful for audio conferencing, as at any moment of time only a single person is speaking and it is sufficient to reserve resources only for one audio stream and maybe a bit more to allow opponent exclamations, interruptions, etc.

Distinct reservation is performed always independent of other reservations toward the same group of receivers. If several reservation requests are made along a single link to the same receiver, the total reservation made will be the sum of all reservation requests. This type of reservation can be used for video traffic, when several streams (channels) are transmitted simultaneously.

Another type of classification that can be applied to types of reservation breaks it into the following categories:

- no-filter or wildcard reservations
- fixed-filter reservations
- dynamic-filter reservations[19]

This classification concerns the ability of the node, where reservation is applied to distinguish among packets that can use reserved resources. These two distinct types of reservation classification clearly illustrates, that packet filtering and control is separated from resource reservation process itself in RSVP operational model.

Wildcard reservation does not control the packets that can use reserved resources at all. This means that in this case no sender information is stored at intermediate nodes along the packet path.

For filtered reservations on reservation establishment stage a list of permitted senders is propagated by receiver, which determines the packets that are allowed to consume reserved resources. For fixed-filter reservations this list of senders cannot be altered on the fly, while dynamic-filter reservations allows receiver to change the list of senders. For fixed-filter and wildcard reservations aggregation is possible, thus minimizing necessary reservation level.

As an example we can present situation when fixed-filter reservations are made over a single link. If some of these reservations are made for the same sender, these reservations can be aggregated and use shared reservation, as the set of senders won't change in time. Wildcard reservation does not distinguish among senders at all. For dynamic-filter such an aggregation is not possible, because the list of senders may change and reservation must be sufficient for the worst case situation when no allowed senders coincide and reservations must be made distinct.

The operational model of RSVP is built using soft-state machine at transit nodes. First of all the senders issue a path request toward all their receivers. These path requests use unicast or multicast routing information to reach receivers[20]. Each node that receives path request sets path state for this sender-receiver pair, which includes the previous hop and optionally sender information. After this receiver is free to send reservation request to any sender at any moment. Reservation request includes information about amount of resources to be reserved and also type of reservation. Reservation request follows reverse route that was passed by path request. This avoids difficulties with asymmetric routing scheme. After admission control, when transit node checks for sufficient resource availability, the node switches to reservation state. If any of the transit nodes fails to make a reservation due to insufficient resources, it sends special reservation error message to the reservation initiator (receiver), which clears all the reservation states along all its path back. In the same way path error messages are generated toward sender in case of some routing problems during path message propagation.

[19] Cisco systems implements only first two types of reservations in its routers.

[20] Unicast and multicast routing algorithm discussion is beyond the scope of this document.

As RSVP needs to adapt to changing topology, path and reservation requests (keepalives) are sent periodically with period, which is negotiated during the first request. First request also usually carries timeout value. If this timeout expires and no new keepalive is received by node, which is reservation state, this state is cleared and special teardown message is sent toward receiver, as initiator of reservation. This is extremely useful if original teardown message was lost on transit or end-node was unable to send it at all. In this case reservation will be cleared anyway after timeout expiration. It is natural to set timeout to at least twice keepalive period, as a single keepalive loss won't cause unnecessary reservation teardown.

It is clear, that RSVP as a new protocol won't be deployed all over the world at the same time. This is the main reason not to make mandatory RSVP peers to be directly reachable. There may be several non-RSVP capable routers on the path between two RSVP peers. This scheme is called RSVP tunnel. This allows connecting two RSVP networks by any non-RSVP internetwork. Of course, it is assumed that this internetwork has enough capacity to carry all the traffic between the two RSVP networks. In case of RSVP tunneling when path request is entering the tunnel area it carries the last RSVP capable router address as a pervious hop information and underlying routing protocol must be used to route reservation requests through the tunnel along the reverse path. The place RSVP tunnel has in the overall model is shown in Fig. 24.

RSVP includes several entities, which interact with one another according the scheme shown in Fig. 25. It presents interaction of end-node with transit router node.

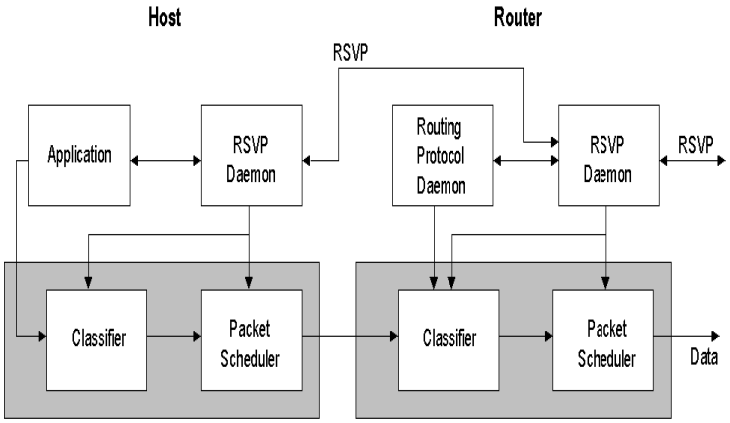


Fig. 25

Packet classifier uses filters to associate each packet with corresponding QoS class. Packets scheduler allocates resources from corresponding class pool. If data-link layer supports QoS concepts, packet scheduler is responsible for interaction with data-link

layer protocol, otherwise it allocates packet transmission capacity, CPU resources and buffers on QoS-passive medium.

RSVP daemon plays the main role in each network node. This is the entity application interacts with and implements soft-state scheme as discussed earlier. It also interacts with other RSVP daemons on routers or end-nodes sending them requests. It gets information from local routing table maintained by routing algorithm to determine the next hop for path requests. RSVP daemon sets packet filters for classifiers as it gets them from reservation requests and directly manages packet scheduler setting and updating QoS policies based on the reservation messages received from applications or other RSVP daemons.

Sometimes RSVP is used in conjunction with other protocols like MPLS or QPPB, which perform policy routing features. This usually leads to increased network efficiency. These protocols will be discussed in the next sections.

5.2 Multiprotocol Label Switching

In the connectionless network layer environment at each packet switching node packet routing decision is made independently from other nodes based on the packet's network layer header information. The routing process can be broken up into two stages: first router partitions entire traffic stream into subsets of packets called Forward Equivalence Classes (FEC) and then looks up the next hop information in its routing table, maintained by some routing protocol. All the packets belonging to the same FEC will be forwarded to the same next hop node[21]. The routing table is indexed by variable length destination network addresses using longest match rules, which makes it rather complex. This process is repeated at each router along packet path.

It will be natural to simplify this process by adding some fixed length label to each packet determined by it FEC at the network ingress edge and make forwarding decision based on this label value further in the network. This is what Multiprotocol Label Switching (MPLS) exactly does. Now routing table and next-hop lookup process is extremely simplified as routing table is now indexed by fixed length relatively short value. This approach allows significantly simplify implementation code and even move it to hardware.

[21] In case of multipath load balancing routing schemes packet from the same FEC will be routed to the same set of alternate next hop nodes.

Inside MPLS network packet network layer header analysis must be performed only at the ingress border router, where it will be marked with the label corresponding to its FEC. As packet is placed into FEC only at the border, this process may become more complicated without need to change routers inside the network. Of course, process of assignment of a packet to FEC can take into consideration as criteria not only packet destination address, but also many other parameters like port and precedence information.

Letter 'M' in MPLS stands for Multiprotocol, this means that it is not limited to IP. MPLS can work with any routable network layer protocol.
As packet enters MPLS network it is marked with special label. At each intermediate node inside MPLS network label value is analyzed after which routing decision is make. After this label swapping is performed: original incoming label in packet is substituted by outgoing label which is meaningful for next hop packet switch. All the bindings between pairs of incoming and outgoing interfaces/labels are maintained in Label Information Base (LIB). The MPLS forwarding scheme is illustrated on Fig. 26. Packets are transmitted through MPLS network consisting of 4 LSRs. While transmission along two hops, packets are assigned corresponding labels according to each LSR's LIB.

Labels are distributed across the network using Label Distribution Protocol (LDP). Several instances of LDP are being standardized. LDP may be special purpose protocol designed specially for label distribution or may be enhanced version of existing dynamic routing protocol. Currently LDP extension of OSPF protocol is under development.

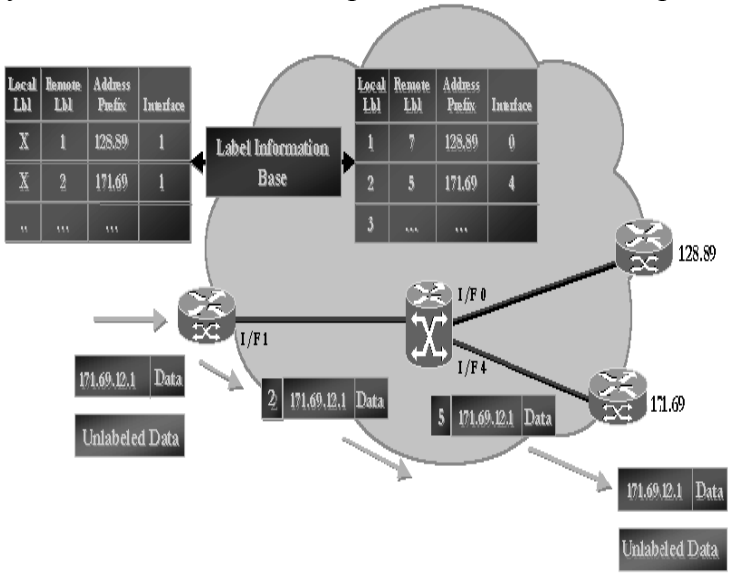


Fig. 26

Label distribution process may be unsolicited or on demand. In unsolicited method LSR distributes its label bindings to all its LDP peers, while on demand distribution propagates LDP bindings only if being requested by upstream peers for that FEC. On demand LDP is used in ATM LSPs and with RSVP traffic engineering.

Label propagation process begins from the MPLS capable node closest to the ultimate destination – egress LSP node. It binds each FEC it is crossed by to the label from its space of available labels.Each label-to-FEC binding is of local significance and label values at each LSR along the LSP path are independent. These bindings are propagated to all the label distribution peers of the current router. This process continues upstream until it reaches LSP ingress router.

The set of packet switches packet traverses while moving to its destination over MPLS network is called Label Switched Path (LSP). In the same way as in the classical routing model, LSPs may be set up manually by network administrator or using some dynamic protocol. There are two types of LSPs available: hop-by-hop routed LSPs and explicit routed LSPs. Hop-by-hop routed LSPs perform in like classical routing manner. Decision is made at each transit node independently based on information available at that node and represented

by label switching table. On the contrary, in explicit routed LSP routing decision is made by a single node, usually LSP ingress or egress switches and is based on information on network topology collected analyzed by some link-state routing algorithm, like SPF.

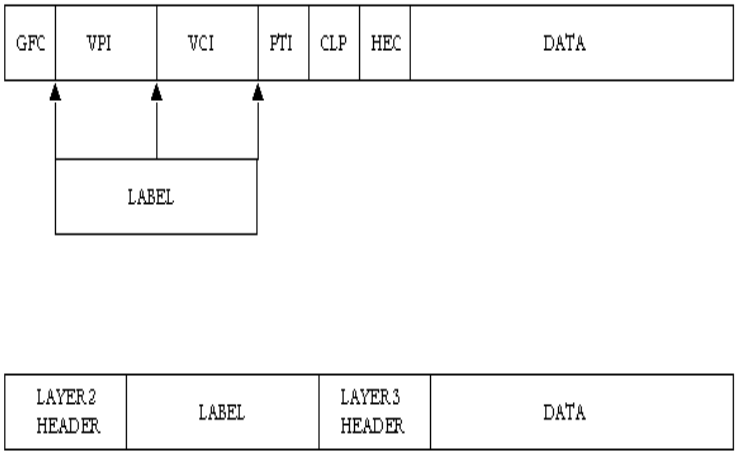


Fig.27

There are two options for inserting label value into PDUs as can be seen from Fig. 27. The first option is so called shim

header, when label value is inserted between layer 2 and layer 3 headers. Another option is to put label value into some field inside layer 2 header. For example when deploying MPLS in ATM network it is possible to put label into VPI/VCI fields pair[22].

In corresponding RFC standard label format is not strictly standardized and may vary in different vendor implementations. Cisco as a leader in MPLS development proposed the MPLS shim header format, which is presented in Fig. 28.

The meaning of label and ttl fields is clear. The purpose of exp field is not defined yet. S bit set to 1 only for last label in the sequence of labels in the stack. For all other labels it is equal to 0.

Fig. 28

When used with ATM MPLS header does not need any TTL or S fields as ATM switches don't implement TTL decrease functions and number of labels in stack available inside ATM header is usually limited and limited (max 2) by encapsulation type used. The rest of the label (if such exist) must be placed into cell data potion.

[22] VPI and VCI fields can also be used as label containers separately. Encapsulation type also exists where VPI/VCI field pair can contain two-label stack.

In the network it is often necessary forward packet to next hop router, which is not directly connected. In this case it is possible to encapsulate the original packet into another network layer packet with the destination address equal to the next hop address for the original packet. This method is usually referred as tunneling. Tunnels can also be hop-by-hop or explicitly routed. The same method can be implemented in MPLS architecture using LSPs. If tunnel is LSP consisted of the label switching routers (LSRs) $\{R_1, R_2, \ldots, R_n\}$, then R_1 is transmit endpoint and R_n is receive endpoint of the tunnel.

LSP tunneling is done with so called label stacks, when each packet is marked with a set of labels. Only top level label is analyzed at transit LSRs. As packet arrives at the tunnel transmit endpoint first standard label swapping is performed with label received from the tunnel egress node via LDP. Then another label is pushed into stack, which is received from the next hop LSR inside the tunnel, for instance R_2. After this packet is forwarded along the tunnel.

The tunnel label is usually popped at the penultimate LSR of the tunnel LSP. This allows tunnel egress endpoint node to receive the packet with the label, which has sense to it. This avoids excess per-packet processing at the LSP egress points and only one forwarding decision per-packet. LSP tunneling is mainly used in complex BGP/IGP interaction and VPN solutions.

The main advantage of such approach to VPN deployment is that there no need for mesh of point-to-point virtual connections among all the sites of a single VPN. MPLS allows creation of VPN in point to cloud connectivity manner, which leads to significantly simplified logical topology.

MPLS allows seamless ATM and IP integration, significantly more effective, than classical IP over ATM or MPOA overlay technologies. Network complexity is reduced by maintaining only one routing algorithm instance. ATM specific routing protocols like PNNI are not needed any more. All the job is done by IP routing protocol, which is aware of ATM network physical structure as well as its logical structure (VC topology).

It is easy to achieve diffserv integration with MPLS. This integration assumes some method of mapping diffserv values from IP header into MPLS fields. Currently two such methods exist: One method is to use EXP field for carrying diffserv information inside a single LSP. In this case number of CoS per LSP is limited by 8 (3 bits in EXP field). Another method is to use the whole label to encode diffserv value. This method has no limitation, but it requires separate LSP to be established for each CoS along the data path.

5.3 QoS Policy Propagation over BGP

This section is mainly a logical extension of the previous section, as partially it is extension to MPLS technology.
Quite often one network needs to force another network to use some QoS parameters for certain type of traffic. This must be regulated in SLA signed between these two networks. The network from where traffic is originated can play either passive or active role in QoS parameter assignment based on the information received from another network.
If the two networks in the latter case have their own autonomous systems and run BGP routing between each other, this task can be performed using QoS Policy Propagation over BGP (QPPB) method. QPPB is part the subject called policy routing or constraint-based routing, which is too large to be included into scope of this document. So QPPB will be discussed in short here.
QoS information can be transmitted in different elements of BGP architecture. One idea is to use
AS_PATH attribute when setting QoS policy at the network edge router. This method is relevant only for situation, when decision must be made mainly by traffic originator, as AS_PATH attribute can be easily customized. This method is not very flexible and has only objective criterias.
Another more suitable method is to store QoS information in the COMMUNITY attribute of BGP update. Separate community value can be created for each traffic class the network wishes to receive and forward over previously engineered paths. Each route advertised to the BGP neighbor is marked with certain COMMUNITY value, which represents which type of traffic AS wants to be forwarded along that path. The COMMUNITY to QoS mapping must be defined at the upstream AS and it also performs marking. QoS can be represented by means of diffserv or using some QoS ID, that has only local meaning. For this method it is not necessary for QPPB peer networks to be direct BGP neighbors. COMMUNITY attribute can be successfully propagated through any QoS incapable networks. The QPPB working scheme with COMMUNITY attribute is shown in Fig. 29.
In the picture AS 20 marks route to network 10.0.0.0 with COMMUNITY attribute 60:1 and propagates it to AS 10 where all the packets that follow this route are marked with diffserv value of 2. If community to QoS mapping scheme is defined and approved by both networks in advance, then AS 20 can dynamically manage packet marking process in AS 10 by simply varying COMMUNITY attribute in BGP route updates. This allows implementation of rather flexible, scalable and effective traffic engineering, which is not limited by AS boundaries.

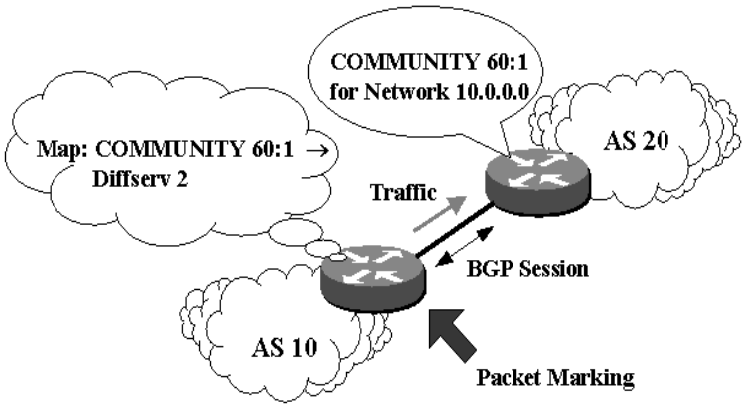


Fig. 29

For MPLS policy propagation between autonomous systems special QPPB method is defined. Obviously the methods mentioned above can also be successfully used for this purpose.

BGP multiprotocol extensions can be used to carry QoS information as defined in RFC 2283. Specifically it can be encoded into Network Layer Reachability Information (NLRI) field of multiprotocol BGP update together with special subsequent Address Family Identifier (SAFI) value 4, which reflects that MPLS information is carried inside the update. NLRI field will consist of one or more triples of the format shown in Fig. 30.

Length contains size (in octets) of label + prefix. Label and prefix parts represent two sides of binding. This means, that the LSR mentioned in the NEXT_HOP attribute will use this incoming label for packets with the prefix from NLRI. In this case BGP is used as LDP for MPLS. It is clear, that this capability must be negotiated on BGP session establishment stage between two BGP peers.

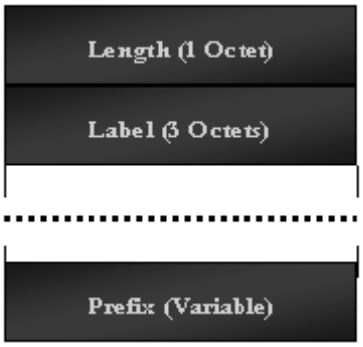


Fig. 30

It must also be mentioned, that BGP speakers must not be adjacent to each other to exchange label bindings. If BGP peers are separated by several IGP routers, then can first push label corresponding to the BGP peer into update and then push label of the next upstream IGP router along the path.

This chapter showed how several QoS capable nodes can be joint into one whole using different QoS signaling and information exchange strategies. Three sections presented three main aspects of QoS information distribution and usage: Resource reservation, traffic engineering and policy routing. As the scope of the document is limited, all these were given in short and are subject for further investigation.

6. Multiservice Network Design Concepts Including QoS Features

QoS networks must be carefully designed before being constructed. Each of the QoS architectural components must be deployed in its designated place inside the network. These architectural components include:

- Queueing and RED as PHBs
- Admission Control
- Resource reservation and traffic engineering

Service provider (SP) network usually consists of the core, which is built with core routers/switches (P nodes). Each customer is connected to the provider' edge routers (PE nodes) by customer edge equipment (CE nodes). This architecture is shown in Fig. 31.

One or more ASs may cover provider's network core. All the P and PE nodes must run some IGP protocol like OSPF or EIGRP, while interacting with users by BGP[23].

Network core can run arbitrary data-link technology, but as the subject of discussion is TCP/IP networks, we will assume, that core is a contiguous network of IP capable devices. Now let's move through the provider's network design from its edges (customers) towards its core

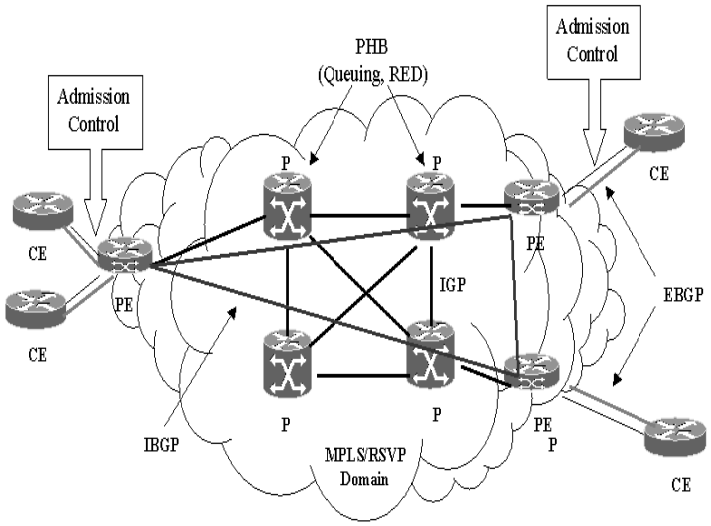


Fig. 31

[23] BGP is not mandatory component of the architecture, though in large systems from the scalability point of view it is preferred to separate customer routing scheme from SP core routing even by running several instances of IGP. But BGP is good solution for customer connectivity, as it introduces very small per-route processing overhead.

First of all SP must control the traffic volume and pattern, that enters its network. At this point it is useful to deploy admission control algorithms and apply them to the links connecting CE and PE equipment. The choice between traffic policing and shaping depends on the type of traffic being transmitted. If the traffic is critical to delays, traffic policing is more preferred method as it immediately drops packets violating policy without any buffering. On the other hand, if reliability is an issue and delay variation is not so important, then traffic shaping will suit into admission control scheme at SP boundary, as it drops packets only if buffer gets overflowed and is more resistant to temporal traffic bursts without packet loss.

If customer network consists of several sites, which are connected to provider backbone through a single link, it is necessary to somehow distinguish their traffic. The traffic separation can be performed at either side of the boundary link and can be implemented using either MPLS or diffserv mappings using filters. Of course, the first method is preferred if MPLS is supported at the CE equipment, as it allows using diffserv field for further traffic segregation.

Besides admission control functions, PE routers must perform packet marking action. Marking action includes label swapping/pushing and/or diffserv field setting actions according to some networkwide policy. If SP uses MPLS all the PE and P devices must support label switching feature.

Each of the P and PE devices can perform PHB to different classes of traffic, which were marked at the network edge. PHB usually include:

- Different packet queueing parameters and disciplines for different traffic types;
- Different RED characteristics.

These PHBs allow treating traffic according to its priority and resource requirements. These PHBs can be implemented into core statically or adapt dynamically using some constrain-based routing protocol, like QoS extended OSPF, which can perform LDP functions for MPLS as well.

Different queueing strategies must be chosen according to delay and jitter characteristics they apply to the traffic being passed. For example to voice traffic the queueing scheme with the least delay must be applied. These characteristics were thoroughly discussed in chapter 3. RED must be treated with more care. As it assumes that some of the packets will be dropped, it must be usually applied to the best effort traffic, which is not very sensitive to packet loss. It is especially useful when trying to bind different priority to several classes of best effort traffic.

It is very important to keep external routes learned from customers separate from internal routes that maintain reachability of PE routers. PE routers must contain both internal and external routes but injection of exterior routes into core P routers must be avoided to keep the network scalability. This is extremely significant when deploying VPNs service over SP network.

Now let's design some sample multiservice network to make it clearer how this all can be implemented and then simulate how it will perform.

Let's assume that the sample network core consists of 4 P and 4 PE routers. The latter routers are connected to two customers, one "Customer A" of which has 1 site and the other "Customer B" – 3 sites. P and PE routers are interconnected by E1 lines (2048 Kbps) with partial mesh topology. Customer A site is connected to the backbone via 512 Kbps link, while all the customer B sites are having 1 Mbps connections to the core each. In addition Internet transit traffic may also transit the backbone, increasing its load.

SP backbone belongs to AS 10, site of customer A is in AS 20 and sites of customer B are in AS 30.

One of the sites of customer B needs to have 4 simultaneous voice channels (4x64 = 256 Kbps) to the site of customer A and at the same time customer B needs to have full mesh VPN connection (in this context it doesn't matter if it is secure or insecure) between all its sites with guaranteed 512 Kbps. The rest of traffic must be used for internet connectivity and treated as best-effort. The customer service scheme, which is to be implemented by SP, is presented in Fig. 32.

First of all, to implement the scheme described above, it is necessary to establish sane routing architecture between the network nodes.

Inside the network core we will run extended OSPF, which supports label switching. All PE routers will establish EBGP sessions with their directly connected customers and full mesh IBGP between each other. BGP routes are never redistributed into OSPF to minimize P router load.

As it was mentioned above, at the customer access links SP must perform admission control procedures. In this case they must be applied to voice traffic limiting it to previously specified value – 256 Kbps, as if it increases without control, it may degrade performance of all the other classes of traffic according to queueing method we will choose. For voice traffic admission control must be used traffic policing feature, as it is very sensitive to delays. VPN traffic may pass without admission control or may be applied traffic shaping method, as its increase will degrade performance only of best-effort traffic.

Queueing is designed in the following manner: Voice traffic must be all placed into priority queue which will have the highest priority over others. VPN traffic has to be placed into deficit round robin or weighted fair queue. In case of weighted fair queueing VPN and best-effort queue weights must be set in proportion 2:1.

SP network backbone will perform MPLS forwarding between PE routers. LSPs must be created along all the IBGP adjacencies also creating full mesh topology. OSPF will perform constraint-based routing and label distribution procedures and will reroute LSP traffic round network failures in the core.

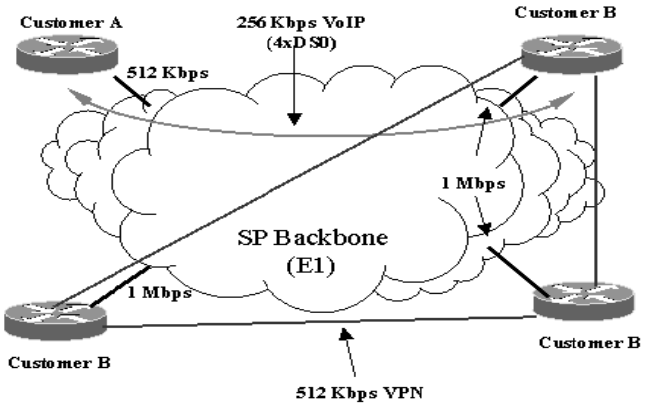


Fig. 32

RSVP can be used either for voice and VPN connections. It will be most effective for VoIP, because of its switched behavior. VPN traffic reservation has permanent nature and can easily be implemented using explicit traffic engineering.

Now let's see how packet will be forwarded through the SP network backbone between its customers. The following is the sequence of actions that are performed on packet entering SP network, for instance, from one of the customer B sites and destined to another customer B site:

1. At the PE router packet get classified and forwarded to the input of corresponding admission control algorithm, where it is accepted or rejected.
2. If the packet is accepted, routing table (label information base in this case) is checked for the destination prefix. When the suitable entry is found, label is pushed into label stack and packet is placed into its corresponding output queue of the interface from routing table entry. If customer also uses MPLS in its network, it can propagate its label over BGP to the SP network. The label used by CE router is propagated from it to its EBGP neighbor egress PE router and then the latter propagates it to ingress PE router over IBGP. In this case, first label learnt over BGP corresponding to the BGP next-hop router is pushed into stack and then label received from OSPF to the IGP next-hop router is pushed. Optionally RSVP channel is established (if not already) between ingress and egress PE routers.
3. The packet is served according to queueing discipline defined in the PE router and sent into the core toward next-hop P router, which is learnt over OSPF (IGP).
4. Packet is forwarded over the core hop-by-hop according to local LIB derived from OSPF database at each intermediate P router until it reached egress PE router.
5. At the egress PE router top level label is removed from the packet. And the packet is sent to CE router either without any label or with label advertised by CE router over EBGP, so it can be correctly routed in the customer network, where they will reach their destination.

This are the stages of the packet transmission across the multiservice network described in short.

This chapter showed how principles outlined in the previous chapters can be used in real network. It summarizes all the methods of multiservice network management and design and presents each of them in their own place inside the whole system.

Conclusion

QoS is one of the fastest developing service attributes in today's networks. It allows complex multiservice networks to be built, that are aware of the content they transfer and treating it according to its needs. This is large step away from circuit switched and multiplexed networks in network evolution. QoS development is extremely complex discipline in contemporary network design requiring deep knowledge of theory of probability and other mathematical theories.

During implementation QoS can be deployed either at the data-link or network layers. The examples of data-link layer QoS mechanisms are frame relay and ATM. Placing QoS into data-link layer makes it easier to implement, but it usually lacks some features that are present at packet level QoS. They example is content aware networking feature, which allows to apply different PHBs to individual packets using they payload type as one of the criterias in the decision process. This significantly increases network scalability, as makes it easy to dynamically adapt to changing traffic patterns.

This document describes some of the aspects of QoS implementation. First several per-node mechanisms are described, like different queueing algorithms and admission control methods together with different types of traffic. All the algorithms are analyzed and investigated thoroughly. Different variations are presented to improve some of their characteristics. Several mathematical models are used to evaluate attributes of each algorithm.

In the subsequent chapters several QoS signaling and information distribution protocol are presented. They allow creating network systems that perform as a whole with all per-node mechanisms previously described being synchronized. This is the most young and rapidly developing part of the QoS architecture today. Most of protocols shown here are not standardized yet by IETF and are available only as drafts.

Finally, the general QoS network design principles are discussed based on the material presented earlier. Some examples of multiservice networks are also available.

This document does not show all components of QoS architecture in detail. Routing related side of QoS was not considered, as this is another large subject. Only QPPB feature in present, as it is close to label switching and QoS features.

The terminology used in this document is taken from IETF and Cisco Systems documents. As these are the leading authorities in this field.

QoS needs more in depth investigation in the future as this new discipline is currently in the development process. Anyway this document is a step toward understanding of the significance and complexity of QoS features for further network service improvements.

References

1. Awduche D., Malcolm J., Agogbua J., O'Dell M., McManus J., *Requirements for Traffic Engineering over MPLS*, RFC 2702, 1999.
2. Bajaj S., Breslau L., Shenker S., *Uniform versus Priority Dropping for Layered Video*, SigComm Proceedings, 1998.
3. Benmohamed L.M., Dravida S., Harshavardhana P., Lau W.C., Mittal A.K., *Designing IP Networks with Performance Guarantees*, Bell Labs Technical Journal, 1998.
4. Bennett J.C.R., Zhang H., *Hierarchical Packet Queueing Algorithms*, SigComm Proceedings, 1996.
5. Bertsekas D., Gallager R., *Data Networks*, Prentice-Hall International, 1989.
6. Blake S., Black D., Carlson M., Davies E., Wang Z., Weiss W., *An Architecture for Differentiated Services*, RFC2475, 1998.
7. Bolot J.C., *End-to-End Packet Delay and Loss Behavior in the Internet*, SigComm Proceedings, 1993.
8. Caceres R., Danzig P.B., Jamin S., Mitzel D.J., *Characteristics of Wide-Area TCP/IP Conversations*, SigComm Proceedings, 1991.
9. Cao Z., Wang Z., Zegura E., *Rainbow Fair Queueing: Fair Bandwidth Sharing Without Per-Flow State*, InfoCom Proceedings, 2000.
10. Christiansen M., Jeffay K., Ott D., Donelson Smith F., *Tuning RED for Web Traffic*, SigComm Proceedings, 2000.
11. Demers A., Keshav S., Shenker S., *Analysis and Simulation of a Fair Queueing Algorithm*, SigComm Proceedings, 1989.
12. Dovrolis C., Stiliadis D., Ramanathan P., *Proportional Differentiated Services: Delay Differentiation and Packet Scheduling*, SigComm Proceedings, 1999.
13. Floyd S., *Congestion Control Principles*, IETF Draft, 2000.
14. Floyd S., Fall K., Tieu K., *Estimating Arrival Rates from the RED Packet Drop History*, 1998.
15. Floyd S., Jacobson V., *Random Early Detection for Congestion Avoidance*, IEEE/ACM Transactions on Networking, 1993.
16. Floyd S., *RED: Discussions of Setting Parameters*, Email Message, 1997.
17. Goyal P., Vin H.M., Cheng H., *Start-time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks*, SigComm Proceedings, 1996.
18. Guerin R., Kamat S., Peris V., Rajan R., *Scalable QoS Provision Through Buffer Management*, SigComm Proceedings, 1998.
19. Jacobson V., Karels M.J., *Congestion Avoidance and Control*, IEEE/ACM Transactions on Networking, 1988.
20. Jacobson V., Nichols K., Poduri K., *An Expedited Forwarding PHB*, RFC2598, 1999.

21. Jacobson V., *Notes on Using RED for Queue Management and Congestion Avoidance*, NANOG 13, 1998.
22. Jain R, Routhier S.A., *Packet Trains – Measurements and a New Model for Computer Network Traffic*, IEEE Journal on Selected Areas in Communications, Vol. Sac-4 №6, 1986.
23. Jamin S., Danzig P.B., Shenker S.J., Zhang L., *A Measurement-Based Admission Control Algorithm for Integrated Service Packet Networks*, IEEE/ACM Transactions on Networking, Vol. 5, № 1, 1997.
24. Kleinrock L., *On the Modeling and Analysis of Computer Networks*, Proceedings of IEEE, Vol. 81, №8, 1993.
25. Kleinrock L., *Queueing systems, Vol. 2*, New-York, John Wiley & Sons, 1979.
26. Lai K., Baker M., *Measuring Link Bandwidths Using a Deterministic Model of Packet Delay*, SigComm Proceedings, 2000.
27. Lin D., Morris R., *Dynamics of Random Early Detection*, SigComm Proceedings, 1997.
28. Mankin A., *Random Drop Congestion Control*, SigComm Proceedings, 1990.
29. Nagle J., *On Packet Switch With Infinite Storage*, RFC 970, 1985.
30. Nichols K., Blake S., Baker F., Black D., *Definition of the Differentiated Services Field in the IPv4 and IPv6 Headers*, RFC2474, 1998.
31. Ozveren C., Simcoe R., Varghese G., *Reliable and Efficient Hop-by-Hop Flow Control*, SigComm Proceedings, 1994.
32. Paxson V., *End-to-End Internet Packet Dynamics*, SigComm Proceedings, 1997.
33. Prithviraj N., Williamson C.L., *A Simulation Study of Token Generation Policies for a Leaky Bucket Traffic Shaper*, 1994.
34. Prue W., Postel J., *A Queueing Algorithm to Provide Type-of-Service for IP Links*, RFC1046, 1988.
35. Rekhter Y., Davie B., Katz D., Rosen E.C., Swallow G., *Cisco Systems' Tag Switching Architecture Overview*, RFC 2105, 1997.
36. Rekhter Y., Rosen E.C., *Carrying Label Information in BGP-4*, IETF Draft, 2001.
37. Rosen E., Rekhter Y., *BGP/MPLS VPNs*, RFC 2547, 1999.
38. Rosen E.C., Viswanathan A., Callon R., Multiprotocol Label Switching Architecture, IETF Draft, 2000.
39. Shreedhar M., Varghese G., *Efficient Fair Queueing Using Deficit Round Robin*, SigComm Proceedings, 1995.
40. Spohn D.L., *Data Network Design*, McGraw-Hill, Second Edition, 1997.
41. Stoica I., Shenker S., Zhang H., *Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks*, SigComm Proceedings, 1998.
42. Stoica I., Zhang H., *Providing Guaranteed Services Without Per Flow Management*, SigComm Proceedings, 1999.
43. Wang Z., Crowcroft J., *Analysis of Burstiness and Jitter in Real-Time Communications*, SigComm Proceedings, 1993.
44. Weiss W., *QoS Differentiated Services*, Bell Labs Technical Journal, 1998.
45. Willinger W., Paxson V., *Where Mathematics Meets the Internet*, Notices of the American Mathematical Society, 1998.
46. Zhang L., Deering S., Estrin D., Shenker S., Zappala D., *RSVP: A New Resource Reservation Protocol*, Preliminary Draft, 1993.

Table of Contents