TXC (Tecnologies de Xarxes de Computadors): Quality of Service (QoS) in networks and Internet architectures

Jose M. Barcelo Ordinas, Jorge García Vidal, Pau Ferrer Cid

November 13, 2023

Contents

1 Quality of Service in Internet protocols and networks **3**

1.1 QoS definition 3

1.2 QoS in network architectures 4

1.3 Static and dynamic bandwidth allocation 5

1.4 QoS parameters 6

1.5 Types of traffic 8

1.6 End-to-end QoS levels (service models) 9

2 Queueing management **11**

2.1 Explaining congestion in the queue 11

2.2 Queueing scheduling disciplines 14

2.3 First-in-first-out (FIFO) + drop-tail 14

2.3.1 Random early detection (RED) 15

2.3.2 Explicit congestion notification (ECN) 19

2.4 Resource allocation in computer networks 20

2.4.1 Uniform fair allocation 21

2.4.2 Maximum throughput allocation 21

2.4.3 Max-min fair allocation 22

2.4.4 Proportionally fair allocation 23

2.5 Queueing service stratgies 23
2.5.1 Round robin (RR) 23
2.5.2 Weighted round robin (WRR) 24
2.5.3 Weighted deficit round robin (WDRR) 24
2.5.4 Priority queueing with round robin (PQ-RR) 24
2.5.5 Priority queueing with weighted round robin (PQ-WRR) 25
2.6 Weighted fair queueing (WFQ) 26
2.6.1 General processor sharing (GPS) 27
2.6.2 Weighted fair queueing (WFQ): a virtual time implementation of PGPS (Optional material) 29
2.6.3 Class-based weighted fair queuing (CBWFQ) 31
2.6.4 Low latency queueing (LLQ) 32

3 Traffic shaping and policing 32

3.1 Token/Leaky bucket algorithms 34
3.2 Traffic parameters 35
3.3 Leaky bucket algorithm with GPS scheduler 38

4 QoS models 41

4.1 Best effort 41
4.2 Integrated services (IntServ) 41
4.3 Differentiated services (DiffServ) 43

The subject TXC (Tecnologies de Xarxes de Computadors) is part of the Bachelor's Degree in Computer Engineering at the Faculty of Computer Science of Barcelona (FIB). The course provides the knowledge and skills necessary to understand the design of connectivity solutions between computers and computing devices beyond the local environment. The basis of the course is the study of the architecture, communication protocols and the fundamentals of data transmission technologies that support wide area computer networks, and in particular the Internet.

These lecture notes deal with "Quality of Service" in networks and Internet architectures and are intended to cover about three and a half weeks of the course (one quarter of the course). The theoretical part will consist of about 10-11 hours of lectures.

1 Quality of Service in Internet protocols and networks

We will use the terms flow, traffic and session interchangeably throughout the document. We define a flow as an end-to-end connection without defining the type of network (layer 2 or layer 3). For example, in terms of Internet traffic, a flow is a 5-tuple defined by the source and destination IP address, the source and destination port (port L4) and the protocol type (TCP or UDP). However, when we talk about classes of service we are assuming that a set of flows are grouped into a class and it is the class that receives the QoS treatment. Classes are identified by precedence bits (some bits of the packet header).

1.1 QoS definition

There are various ways of defining Quality of Service (QoS) according to different authors, manufacturers or regulatory bodies:

- QoS refers to the ability of a network to provide better service to selected network traffic through various technologies, such as Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and 802.1 networks, SONET and IP routed networks that may use any or all of these underlying technologies;
- QoS is the manipulation of traffic so that the network device forwards it in a manner consistent with the behaviours required by the applications generating that traffic;
- Quality of service measures the ability of a network to deliver high quality services to an end user. More specifically, it designates the mechanisms and technologies to manage data traffic and control network resources;

- QoS prioritises certain types of network traffic over others based on their level of importance or sensitivity to delay, loss or other network conditions;
- The QoS solution allows policies to request network priority and bandwidth for TCP/IP applications throughout the network.

QoS makes it possible to provide a better service to certain flows or connections by increasing the priority of one flow or limiting the priority of another. This can be done in several ways:

- **congestion management tools** attempt to increase the priority of a flow or a group of flows by queuing and servicing (scheduling) queues in different ways. The queue management tool used to avoid congestion increases the priority by discarding lower priority flows before higher priority flows. Examples are priority queuing (PQ), weighted fair queuing (WFQ), class-based weighted fair queuing (CBWFQ), low latency queuing (LLQ), etc, although other mechanisms attempt to prevent congestion before it occurs such as random early detection (RED) or explicit congestion notification (ECN);
- **traffic shaping and policing functions** give priority to one flow or a group of flows by limiting the speed of other flows. The idea is to define which traffic **conforms** to a specification, and to enforce that specification by classifying the traffic into conforming and non-conforming. Some action is taken on the non-compliant (or non-conforming) traffic (e.g. discarding, labelling or shaping the traffic). Examples are token/leaky bucket (TB/LB) algorithms;
- **link efficiency tools** maximize bandwidth use and reduce delay for packets accessing the network by limiting large flows and showing preference for small flows. Examples are the use of TCP header compression or link compression mechanisms.

We can observe that to provide QoS, networks allocate resources (bandwidth, queues, buffer priorities, etc.) and to measure the goodness of service (what quality the user receives), the network measures quality in terms of metrics (delay, jitter, loss, etc.).

1.2 QoS in network architectures

QoS has been applied in different network architectures over time [4, 7]:

- **circuit-switching:** used in the telephone system allocate fixed resources when a connection is initiated and these resources remain in use until the connection is terminated. The advantage is the low and predictable delay

of these networks. However, if more users try to call and the resources are not available, the user is denied the call. Also, only two users can be on a call simultaneously (if Bob talks to Alice and wants to talk to Mary as well, he has to hang up Alice and dial Mary's phone number);

- **packet-switching:** networks such as ATM, Frame Relay or Internet allow everyone to be on the network at the same time and dynamically allocates resources among the active users of the network. A packet-switched network allows several devices to communicate simultaneously and needs to allocate sufficient resources to them to communicate. If there are many users, they receive fewer resources, but are not prevented from communicating.

What are the challenges, in terms of quality of service, of packet-switched networks? (i) The lack of predictability for real-time applications such as VoIP, online gaming, video conferencing and IPTV, and (ii) the high jitter that occurs whenever a large number of packets pass from a faster network link to a slower one or when several network links are merged into a single link. When this happens, network devices such as routers and switches get stuck and force packets to wait inside their memory buffers, which increases the time it takes for packets to traverse a network.

The goal is therefore to provide packet-switched networks, such as the Internet, with the necessary resources and performance characteristics required by real-time applications, just like circuit-switched networks.

1.3 Static and dynamic bandwidth allocation

Links have a C capacity that can be shared between users (or flows). However, there are several ways to share that capacity. The idea behind **static bandwidth allocation** [4, 7] is that flows are allocated a fixed amount of bandwidth from the link capacity, and therefore that fixed amount of link bandwidth cannot be used by other flows. If the flow does not use its allocated bandwidth, then the bandwidth is not used. Examples of techniques that allow static allocation are TDMA.

On the other hand, **dynamic bandwidth allocation** is a technique where flows are allocated an amount of bandwidth, but if the flow does not use that bandwidth, it can be used by other flows. Dynamic bandwidth allocation is based on the idea of statistical multiplexing.

Example 1.1 (Dynamic bandwidth allocation) *Assume a* 100 Mbit/s *link, and 10 users are receiving* 10 Mbit/s*. If a user uses an average of* 1 Mbit/s*, the other* 9 Mbit/s *allocated to this user are lost. On the other hand, suppose we know that 5 users send traffic at an average of* 1 Mbit/s*, and 5 of them normally reach their* 10 Mbit/s*. There are* 45 Mbit/s *unused. There are* 45 Mbit/s *unused.*

That leaves 45 Mbit/s *unused, so we could allocate that bandwidth to other users so that, on average, we never reach* 100 Mbit/s *of link capacity.*

By definition, fixed bandwidth networks have to operate in the worst case (traffic peaks), while dynamic bandwidth networks operate based on average traffic.

1.4 QoS parameters

Quality of Service parameters are metrics used to determine the performance of different applications on the Internet. The main metrics are bandwidth, volume, loss and delay.

- **Bandwidth (or throughput):** is the amount of link capacity a flow requires. It is measured in bit/s.

  Example 1.2 (Bandwidth management) *One of the things we can do with QoS is to create different queues and put certain types of traffic in different queues, for example, configure the router so that queue one gets 50% of the bandwidth, queue two gets 20% of the bandwidth, and queue three gets the remaining 30% of the bandwidth.*

- **Volume:** is the number of bytes transmitted in a time interval. Volume can also be viewed as the number of bytes consumed in an arbitrary period of time.

  Example 1.3 (Volume SLA) *Think of a service level agreement (SLA) for a smart phone in which the amount of bytes you can transmit for the tariff you pay is fixed (e.g.* 2 GB*). From the consumption established in the contract, you start paying per byte transmitted.*

- **Losses:** is the number of packets lost in a transmission. If you send 100 packets and only 95 arrive at the destination, you have a 5% packet loss. Losses usually occur when there is congestion. The causes of congestion are: i) bandwidth mismatch (e.g. high capacity link with low capacity link, Figure 1.a), ii) aggregation (e.g. multiple links multiplexed on a single link, Figure 1.b), and iii) extra-traffic resulting from faulty hardware/firmware, excess retransmission due to data corruption and high traffic applications or misbehaving hosts (e.g. malware). When queues are full and packets need to be discarded. With QoS, at least we can decide which packets are discarded when this happens. Other causes of packet loss are signal degradation (e.g. attenuation), noise and faulty hardware;

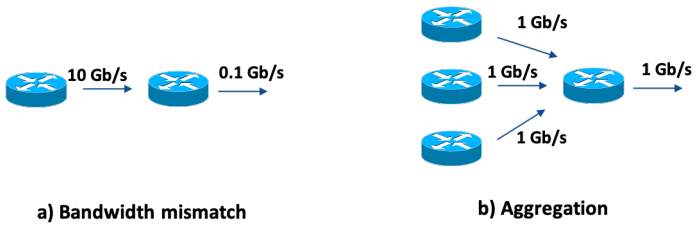

Figure 1: Contribution to congestion, a) bandwidth mismatch, and b) aggregation of traffic.

- **Delay:** is the time it takes for an individual packet to traverse the network. We can define two types of delays:
  - the first type of delay is the **latency (or end-to-end delay or round trip time, RTT)** defined as the time it takes a packet to travel from source to destination, and it is measure in ms. The end-to-end delay is composed of several delays such as *processing delay* (time it takes for a device to perform all tasks required to forward the packet), *queuing delay* (amount of time a packet is waiting in a queue), *serialization (or transmission) delay* (time it takes to send all bits of a frame to the physical interface for transmission), and *propagation delay* (time it takes for bits to cross a physical medium);
  - the second type of delay is **jitter** and is defined as a measure of the variation in packet delay, figure 2. High jitter occurs when there is *congestion*, it is to say, when a large number of packets pass from a faster network link to a slower one (bandwidth mismatch) or when several network links are merged into a single link (aggregation). The router or switch is forced to buffer the packets, which increases the time it takes for packets to traverse the network. Other causes of jitter are retransmission of L4 protocols or dynamic rerouting.

    Example 1.4 (jitter) *Suppose an IP phone sends a constant stream of voice packets 10 ms separated. Due to network congestion, some packets are buffered and therefore delayed. The delay between packet 1 and packet 2 is 20 ms, the delay between packet 2 and packet 3 is 40 ms, the delay between packet 3 and packet 4 is 5 ms, and so on. The receiver of these voice packets must deal with the jitter by ensuring that the packets have a constant delay or it will experience poor voice quality.*

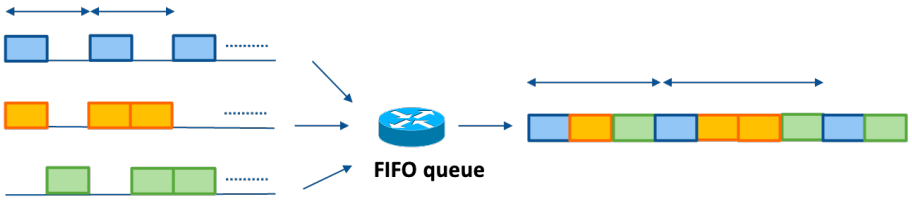

Figure 2: Contribution to Jitter: packets are buffered and the input space between packets is different from the output space.

There has always been much debate as to whether delay can be avoided by increasing the capacity of the devices. End-to-end delay can be minimised by prioritising some of the different components. For example, better CPUs and Hw will improve processing delay, a good queue management scheme will improve queuing delay, and increasing the capacity of a link will improve transmission time. Propagation delay is something we cannot change.

What about jitter? Most people think that jitter can be avoided by increasing link capacity. However, this is not true. Jitter is not related to increasing capacity, and it is possible to have low jitter on low bandwidth links and high jitter on high capacity links, and vice versa. Jitter is related to having high capacity links connected to low capacity links or when several links multiplex traffic onto a single link. In these cases, packets from one connection are queued with others, and the separation between them may increase or decrease, producing variable gaps between packets on the same connection. Traffic shapers can cope with jitter by separating packets of the same connection with the same spacing.

1.5 Types of traffic

Applications produce different types of traffic with different traffic parameters and different QoS parameter requirements. Let us look at some examples.

- **Non-interactive applications:** such applications typically upload or download files. Examples are web transfer or file transfer, among others.

  Example 1.5 (file transfer) *Let's take a 10 MB file transfer with IP payloads of 1500 B as an example. This file transfer involves 10485760 B of transfer, and about 10485760/1460= 7182 IP packets (removing IP and TCP headers). High bandwidth reduces the file transmission time, since a* 10 Mbit/s *throughput connection transmits the file in* 8.38 s *and a* 100 Mbit/s *throughput connection transmits the file in* 0.838 s*.*

  Then, bandwidth is important, since minimizes file transfer times. Packet losses are not important, as TCP will recover them, at the cost of increas-

ing end-to-end delay, and jitter is not important, as we are not interacting with the download, just waiting for it to finish;

- **Interactive applications:** such applications often connect to servers, such as SSH to connect a client to a router or switch or to remotely run processes on a server. These applications do not require bandwidth, as the transfer consists of only a few bytes. The delay is not important, except if you access the server over a satellite link ($500-700$ ms delay), which means there will be a short pause before you see the characters appear on your console. The best we can do with QoS is to prioritise these connections over bandwidth-hungry applications;

- **Voice and video applications:** are very sensitive to delay, jitter and packet loss.

  Example 1.6 (VoIP application) *Let's assume a VoIP (voice) application using a simple G711 codec, which sends packets every* 20 ms *on a 160B data payload. The IP, UDP and RTP headers add 40 bytes of overhead, so the IP packet will be 200 bytes in total. For one second of audio, the phone will create 50 IP packets. 50 IP packets * 200 B =* 10 000 B/s *per second. That is,* 80 kbit/s. *Other codecs, such as G729, reduce the required bandwidth to* 24 kbit/s *at the cost of reduced audio quality.*

  Bandwidth is not an issue in VoIP, but delay is, since a conversation is in real time (you don't want to wait to hear the other person). Jitter is also an issue, because the codec expects a constant stream of IP packets with voice data that it has to convert back into an analogue signal. Losses are not a problem, as long as they are limited. A good VoIP connection should have to guarantee an end-to-end delay ≤ 150 ms, a jitter $\leq 20-30$ ms and a packet loss $\leq 1\%$.

  Video traffic has similar requirements to voice traffic, and a typical video connection should have an end-to-end delay between $200-400$ ms, a jitter between $20-50$ ms and a packet loss between 0.1%-1%.

1.6 End-to-end QoS levels (service models)

Service levels refer to the actual end-to-end QoS capabilities, i.e. the ability of a network to provide the service required by a specific network traffic from end-to-end or edge-to-edge. These QoS levels depend on the **QoS strictness**, which describes the extent to which the service can be limited by specific bandwidth, delay, jitter and loss characteristics. We can consider three end-to-end levels:

- **Best effort:** there is no guarantee for flows in terms of bandwidth, delay, jitter and loss characteristics. It is characterised by being treated with low priority in case buffering systems apply a queuing priority policy, or if

there is no buffer priority scheme such as using a FIFO scheduler (FIFO is sometimes said to provide a best-effort service). In general, the Internet was designed as a best-effort service, however, if other levels of end-to-end QoS are used, any application sending traffic without any requirements is considered to be best-effort traffic, and so the network will try to give the "best" it can or drop out if it cannot (without informing the endpoints);

- **Soft QoS (DiffServ, differentiated services):** some traffic is treated better than others in terms of better management, higher bandwidth, lower delay (end-to-end and jitter) and lower losses. Flows are grouped into classes, and special treatment is given to classes and not to flows. Therefore, this treatment is statistical and no hard guarantee is given to specific flows. This service is achieved by using priority queuing (PQ) schemes, custom queuing (CQ) schemes, queue management (QM) schemes, traffic shaping and monitoring, and scheduling;
- **Hard QoS (IntServ, guaranteed service or integrated service):** a robust quality of service is guaranteed in terms of bandwidth, delays and losses. This is achieved by using a reservation scheme, together with priority queuing (PQ), traffic shaping and monitoring and scheduling schemes.

Flow classification (hard QoS) can be done using ACLs (access lists). ACLs usually work on L3 and L4 headers. However, modern tools such as the CISCO Network Based Application Recognition (NBAR) tool is able to identify (and then classify) flows in terms of application information such as URL's, MIME type of an HTTP packet or user-agent (Mozilla, Opera, etc).

The **class classification** (soft QoS) can be done by modifying the **precedence bits**. The **precedence bits** are part of some field in the packet. For example, in an IP packet, in the 90's, the first three most significant bits (**precedence** field) of the ToS (type of service, 1B) field are used to mark the class to which the flow belongs, figure 3.

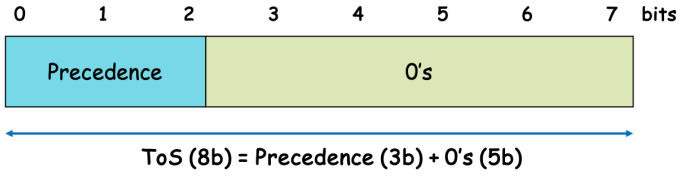

Figure 3: Precedence field.

This field was later extended to 6 bits in DiffServ architectures (DSCP field). The flow is identified by an ACL, the class is established by marking the precedence bits, and then from that router, packets are classified as belonging to a class by identifying the precedence bits.

2 Queueing management

Packets arriving at a router are inspected and routed to an exit interface. However, in this process, packets are queued in a buffer waiting to be served on each interface. We can see in the figure 4 a router with 6 interfaces, and input/output queues on each interface.

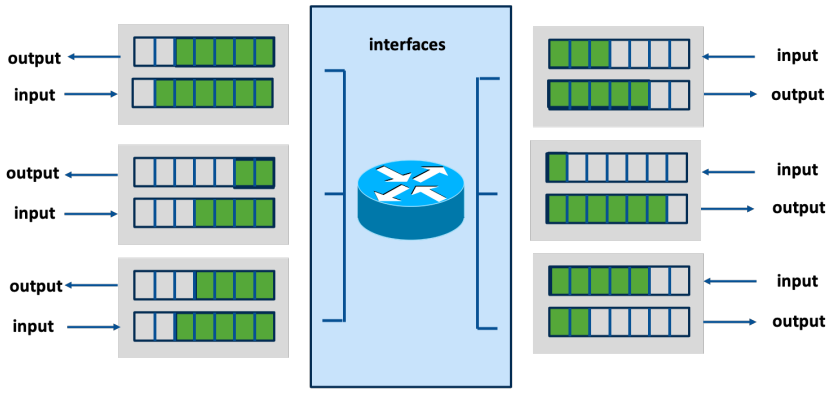

Figure 4: Queues in the interface of a router.

2.1 Explaining congestion in the queue

Suppose a queuing system in which the arrival rate has a Poisson distribution of parameter λ (in packets/s). That is, if we define N(t) as the number of arrivals in the interval (0,t), then N(t) obeys the Poisson distribution (λt):

$$P\{N(t)\} = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \tag{2.1.1}$$

and the inter-arrival times are independent and obey an exponential distribution Exp(λ):

$$P\{\text{interarrival time} > t\} = e^{-\lambda t} \tag{2.1.2}$$

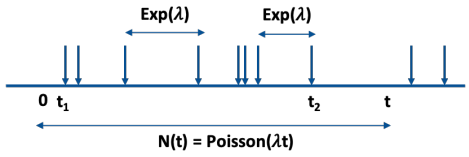

Figure 5: Poisson process (arrival process).

Remember, that an exponential distribution f(x)=$\lambda e^{-\lambda x}$ has average E[x]=$1/\lambda$.

The M/M/1 queuing system is a mathematical framework that models a queuing system whose arrival rate has a Poisson distribution of parameter λ packet/s and the service rate is the number of customers served per unit time, which is exponentially distributed with mean $1/\mu$. The length of the packets is exponentially distributed. Since, if C (bit/s) is the capacity of the link (constant), and the transmission time is given by $T_t = L/C$, saying that the packet length is exponentially distributed, is the same than saying that **the transmission time (service time)** is exponentially distributed. Let us assume that the transmission time (service time) is exponentially distributed with parameter μ. Then, T_t follows the distribution g(t)=$\mu e^{-\mu t}$ with average service time $1/\mu$, i.e., the average transmission time is $T_t = 1/\mu$.

The system load is defined as $\rho = \lambda/\mu$. We can now calculate (not developed in these lecture notes) the following metrics:

- The **mean number of customers in the system (N):**

$$N = \frac{\rho}{(1-\rho)} = \frac{\lambda}{(\mu - \lambda)} \tag{2.1.3}$$

- The **mean delay per customer in the system (also called mean throughput time or system time):** is obtained by Litte's formula where $N = D\lambda$ (it is to say, the mean number of customers in a system is equal to the mean delay multiplied by the number of arrivals). From Little's formula, we can obtain the mean delay,

$$D = \frac{N}{\lambda} = \frac{1}{(\mu - \lambda)} = \frac{1}{\mu(1-\rho)} \tag{2.1.4}$$

- The **mean delay per customer in the queue:** is the delay in the system minus the delay in the service.

$$W_Q = D - T_t = \frac{1}{(\mu - \lambda)} - \frac{1}{\mu} = \frac{\rho}{(\mu - \lambda)} = \frac{\rho}{\mu(1-\rho)} \tag{2.1.5}$$

- The **mean number of customers in the queue (N_Q):**

$$N_Q = \lambda W_Q = \frac{\rho^2}{(1-\rho)} \tag{2.1.6}$$

- The **mean number in the service (N_S):**

$$N_S = N - N_Q = \rho \tag{2.1.7}$$

Example 2.1 (The M/M/1 queue system) *Let us have a communication in which packets arrive Poisson distributed with one packet every* 2.5 ms *(i.e.,*

arrival rate λ= 1 packet/(2.5 ms)= 400 packets/s). Packet transmission times are exponentially distributed with mean 2 ms (i.e. $1/\mu$=2 ms or μ= 500 packets/s). The load is ρ=400/500= 0.8.

Thus, the mean delay per packet in the system is 1/(500-400)= 10^{-2} s= 10 ms, the mean delay per packet in the queue is W= 8 ms. The mean number of packets in the system N=4 packets and the mean number of packets in the queue is N_Q=3.2.

Observe that when the load $\rho \sim 1$, the mean number of customers in the system N and the mean delay per packet in the system increase unbounded ($N \sim \infty$ and $D \sim \infty$). In other words, when the number of arrivals in the systems equals or exceeds the number of services, the queue grows and the delay increases. In this situation, we say that there is **congestion**.

Example 2.2 (Congestion in TCP) *We can observe, figure 6, the throughput and delay in a queueing system using TCP and the effect of increasing ρ. The knee is the point from which throughput increases very slowly and delay increases rapidly, while the cliff is the point from which throughput begins to decrease very rapidly to zero (congestion collapse) and delay approaches infinity. The goal of* **congestion control mechanisms** *is to stay to the left of the cliff, while the goal of* **TCP congestion avoidance mechanisms** *is to stay to the left of the knee. Examples of* **TCP congestion control mechanisms** *are TCP slow-start, back-pressure mechanisms, throttling packets, etc. Examples of* **congestion avoidance mechanisms** *are TCP's congestion avoidance mechanism. A key question is how to detect congestion in the network, e.g., using explicit congestion notification (ECN) mechanims, see section 2.3.2.*

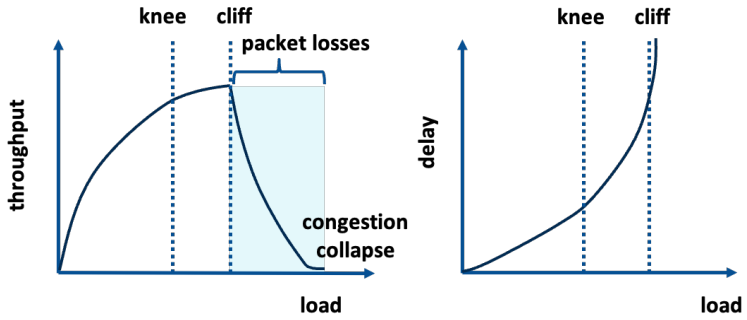

Figure 6: Throughput and delay as a function of the load ρ. Congestion collapse when $\rho \sim 1$.

2.2 Queueing scheduling disciplines

Queue scheduling disciplines define how packets are buffered while waiting to be transmitted. The most common queue scheduling disciplines in networks are FIFO (first in, first out), PQ (priority queuing) and FQ (fair queuing). Queue scheduling disciplines have two main parameters: **bandwidth (scheduling)**, which determines which packet is transmitted next, and **buffer space (buffer management)**, which determines which packet is discarded next (if necessary). Queue scheduling disciplines influence the delay (latency and jitter), throughput or losses of a flow.

2.3 First-in-first-out (FIFO) + drop-tail

This is the classic Internet queuing discipline (best effort). The **drop-tail** mechanism means that arriving packets are discarded when the queue is full, regardless of the flow. The main problem with FIFO is that the service received by one flow is affected by the packet arrivals of all other flows. That is, the flow does not receive a particular service, regardless of the service given to other flows. On the other hand, the management of drop queue buffers forces routers to have a very large buffer size to maintain high utilisation, resulting in stationary queues and, subsequently, long delays. **Steady-state** queues means that, in general, queues take some time to reach the regime predicted by queueing theory (e.g., the M/M/1 queuing system). The reason is that queues do not increase instantaneously to the high levels predicted by high loads, but increase (in the transient period) until the steady state (the equilibrium state) is reached. Transient state means that the state of the queue depends on the time t, while the steady state means that the state of the queue does not depend on the time t. A **heavy steady state** with long delays is reached after some time having a heavy load ($\rho \sim 1$).

In addition, flows suffer from burstiness, synchronisation and lock-out (blocking). **Burst transmission (burstiness)** occurs when hosts send a high-bandwidth transmission (large number of packets) in a short period of time. Bursts cause long delays and queuing losses. Moreover, hosts react in the same way (same algorithm) when packet losses occur in bursts (burstiness) periods. This causes a **synchronisation** effect that causes the same situation to happen to the same flow when new packets arrive in the queue. Then, a side effect of burstiness and synchronisation is that a few flows can monopolise the queue space (the **lock-out effect**).

Different disconnection mechanisms have been proposed to deal with these effects. In general, there are several strategies that can help avoid congestion and reduce queue sizes when overloading occurs. The two main approaches are to drop the packets with some strategy and to mark the packets for later action at some network node or at the endpoints (hosts). Here are some ideas:

- **Synchronization:** can be solved using *random drop* which consists of randomly dropping some packets from the queue;
- **Lock-out:** can be solved using *front drop* which consists of dropping packets at the head of the queue;
- **High steady-state queuing:** can be solved using *early drop* which consists of dropping packets before the queue is full;
- **Burstiness:** can be solved using *early drop*, but taking care of not dropping packets *too early* because the queue may reflect only burstiness and not true overload;
- **Fragile flows:** can be solved using *preference dropping*, identifying (classifying) flows on the fly, and mark critical flows for a specific treatment (not dropping them);
- **Host misbehaving:** can be solved dropping packets proportional to queue occupancy of flow.

Therefore, the goal of a good approach should be to: i) maintain high throughput, ii) maintain low delay, iii) accommodate bursts, iii) queue size should reflect the ability to accept bursts rather than steady state queues, and iv) in case of using TCP/IP, improve the performance of the end-to-end protocol (e.g. TCP) with minimal hardware changes.

2.3.1 Random early detection (RED)

Random early detection (RED) [2] is one of the most popular techniques used on the Internet for congestion avoidance and queue management, acting as a low-pass filter. RED works by monitoring the traffic load at points in the network and stochastically discarding packets if congestion starts to increase. The result of discarding is that the source detects the discarded traffic and slows down its transmission, and is primarily designed to operate on TCP/IP networks. RED starts dropping packets randomly when the average queue size exceeds a threshold value ($\min_{th}$). The packet drop rate increases linearly as the average queue size increases until the average queue size reaches the maximum threshold ($\max_{th}$). Thereafter, a certain fraction - called the mark probability denominator - of packets is dropped, again at random. The minimum threshold must be greater than some minimum value so that packets are not discarded unnecessarily. The difference between the maximum and minimum threshold must be large enough to prevent global synchronisation.

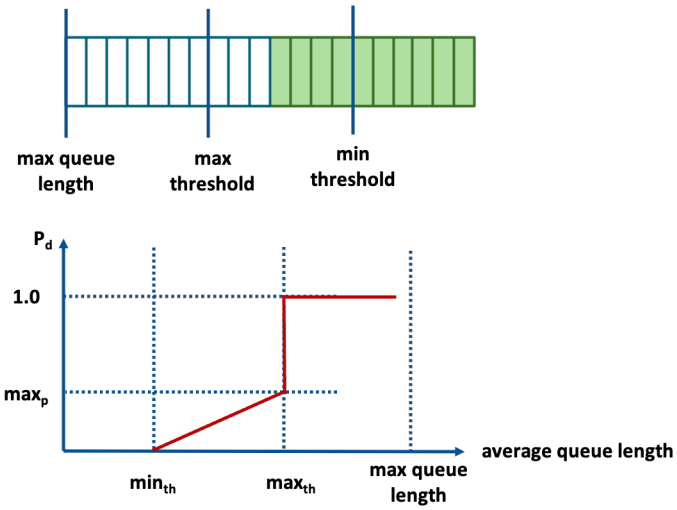

Figure 7: Random early detection (RED).

- On arrival of each packet, calculate the average queue length according to an **exponential weighted moving average (EWMA)** algorithm [2]:

$$avgQ^{new} = (1 - w_q) * avgQ^{old} + w_q * Q_{length}^{current} \quad (2.3.1)$$

  where $avgQ^{new}$ is the new average queue length being calculated at this sample, $avgQ^{old}$ is the old average queue length during the previous sample, $Q_{length}^{current}$ is the instantaneous queue length at the router, w_q is a weight associated to the current queue length (it is usually a small value, e.g., w_q =0.002).

  In CISCO routers, the weight is configured using a value n such that $w_q=1/2^n$. Thus, high values of n (called "exponential weight factor" by CISCO) imply low values of w_q. So, for high values of n, the previous average queue size becomes more important. A large factor smooths out the peaks and lows in queue length. If the value of n gets too high, RED will not react to congestion. Packets will be sent or dropped as if RED were not in effect. If the value of n gets too low, RED will overreact to temporary traffic bursts and drop traffic unnecessarily;

- If $avgQ^{new} \leq \mathbf{min}_{th}$ do nothing (low queueing, send all packets, dropping probability P_d=0);

- If $avgQ^{new} \geq \mathbf{max}_{th}$ drop packets with dropping probability P_d=1 (large queueing, protect from misbehaving sources);

- Else mark (or drop) packet in a manner proportional to queue length to protect against synchronization. For that, calculate dropping probability

P_d according to:

$$P_d = \frac{max_p(avgQ^{new} - min_{th})}{(max_{th} - min_{th})} \tag{2.3.2}$$

where $\max_p$ is the maximum probability of dropout (in the original paper it was set to 0.02, nowadays it is usually set to 0.5);

- in case the queue is measured in bytes and not in packets, we can introduce a normalizing constant:

$$P_d = P_d \frac{Packet\_size}{max\_Packet\_size} \tag{2.3.3}$$

- solving the **bias**: when the average queue length is constant, the number of packets arriving between dropped packets becomes a geometric random variable with P_d [2]. This means that packet drops are not uniformly distributed, since the arrival of packets between two dropped packets is not constant (sometimes few packets arrive and sometimes more arrive). It is undesirable to have too many marked packets close together, and it is also undesirable to have too long an interval between marked packets. Both of these events can result in global synchronization, with several connections reducing their windows at the same time. Therefore, we recalculate the drop probability with a new value that we call P_a, to emulate dropping packet probabilities following a uniform variable distribution:

$$P_a = \frac{P_d}{(1 - count * P_d)} \tag{2.3.4}$$

with *count* the number of packets queued since the last drop. To decide whether the packet is dropped or not, an R value is generated using a uniform random distribution between [0,1], and then the packet is queued if $P_d \leq R$, and dropped otherwise;

- A final observation to consider is what happens if the queue is empty and a new packet arrives. Since RED is invoked on packet arrival, it is possible that the average queue was large, there were no arrivals, the queue is empty, and a new packet arrives. In this case, RED may incorrectly indicate high congestion and drop the packet, even if the queue is empty. To fix this, we add a new condition before calculating the average queue length. We use eq (2.3.1) if the queue is not empty, and use the following equation if the queue is empty:

$$avgQ^{new} = (1 - w_q)^m * avgQ^{old} \tag{2.3.5}$$

where $m = (idle\_stop\_time - idle\_start\_time)/(avgT_{tx})$ with $avgT_{tx}$ is the average transmission time of a packet defined as $avgT_{tx}$=(mean packet size/bandwidth), since packet sizes can be variable in Internet. The idle_start_time is the time when the queue started to be idle, and idle_stop _time is the time when the queue started to be busy again.

The algorithm now is quite simple:

Step 1 If queue is empty calculate $avgQ^{new}$ according to eq (2.3.5), elsewhere calculate $avgQ^{new}$ according to eq (2.3.1);

Step 2 If $avgQ^{new} \leq min_{th}$ enqueue the packet, elseif $min_{th} \leq avgQ^{new} \leq max_{th}$ calculate P_d and P_a according to eq (2.3.2) and (2.3.4);

Step 3 If $P_a \leq R$ enqueue the packet, else drop the packet.

RED advantages are:

1. avoiding congestion;
2. avoiding global synchronization;
3. avoiding lock-out; and
4. maximize **power function** that is the ratio of throughput to delay (implying no losses and low delays).

However, **RED has some drawbacks**, such as:

1. being extremely sensitive to parameter settings (difficult to tune);
2. producing wild queue oscillations upon load changes;
3. failing to prevent buffer overflow as the number of sources increases; and
4. failing to help fragile flows (eg: small window flows or retransmitted packets).

There are variations such as RED with multiple thresholds that refine the discard probabilities between thresholds, or other proposals that maintain a history of packet drops, identifying flows that use disproportionate bandwidth from fragile flows.

One of the authors of RED, Van Jacobson, mentions that RED was distrusted by network operators due to two bugs. Van Jacobson, with Kathy Nicholls and Kedar Poduri proposed an enhancement in a paper called "RED in a different light" in 1999 (`http://mirrors.bufferbloat.net/RelevantPapers/Red_in_a_different_light.pdf`). This paper [3] was never published, and Van Jacobson mentions in an interview in blog `https://gettys.wordpress.com/2010/12/17/red-in-a-different-light/` by Jim Getty that the paper was rejected due to a lack of humour in one of the reviewers in the program committee. For a discussion on this topic read the blog and make a look to the paper for the solution (and guess which is the famous figure).

Currently RED is supported in other RED-based mechanisms, such as **Weighted RED** (**WRED**, $w_q = 2^n$ in CISCO routers) where a single queue may have several different sets of queue thresholds. Each threshold set is associated to a

particular traffic class. Other methods such as **flow-based WRED** (**FRED**), among others watch aggressive UDP flows that use to much queue and discard them. In any case, all of these variations have as based the original RED mechanism.

2.3.2 Explicit congestion notification (ECN)

Explicit congestion notification (ECN, RFC 3168, 2001) is an alternative mechanism used in the Internet to avoid congestion without dropping packets in the queue. The idea is to notify sources to perform a flow control mechanism when congestion is detected on a router. This is why it usually works with a protocol that performs flow control such as TCP. Let's see how it works in TCP/IP networks. When the queue of a router reaches a threshold, the packet is marked (in the case of TCP/IP packets, 2 bits of the ToS field of the IP header are used as the ECN field), figure 8.

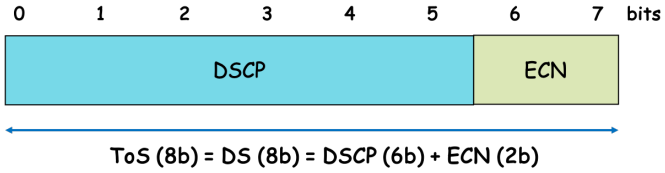

Figure 8: DS field: differentiated service code point (DSCP) and explicit congestion notification (ECN) fields.

When the packet arrives at its destination, TCP marks the ECN field of the ACK packet in order to notify the source that there is congestion on the route to the destination. In this way, the TCP source learns that there is congestion and reduces its congestion window to regulate the traffic flow. The two bits are marked as follows:

- **00** - Not ECN-capable transport, Not-ECT;
- **01** - ECN-capable transport(1), ECT(1);
- **10** - ECN-capable transport(0), ECT(0);
- **11** - Congestion experienced, CE;

when the source or destination does not support ECN, they mark packets with the Not-ECT code point; when the source and destination support ECN, they mark packets with the ECT(0) or ECT(1) fields (routers treat the ECT(0) and ECT(1) codepoints as equivalent). If the router is using RED, experiences

congestion, and the router supports ECN, it can change the code point to CE instead of discarding the packet.

TCP endpoints negotiate ECN support in the 3WHS and use two flags: the destination uses the **ECN-echo** flag to send a flag to the sender indicating that there is congestion on the network, and the source uses the **Congestion Window Reduced (CWR)** flag to send a flag indicating that it has reduced the TCP congestion window.

ECN is good at reducing the number of packets lost in transactional protocols such as HTTP or SQL requests, but not so good in bulky protocols (e.g., FTP) that send large files and are good at forwarding packets when packets are lost. Operating systems such as Linux Ubuntu, Windows Server and Apple support ECN signalling by default.

2.4 Resource allocation in computer networks

Let us define a resource C, and let be c_i user i demand of resource C, y_i user i use of resource C, and x_i user i allocation of resource C. We can observe that it has to be satisfied that:

$$y_i \leq c_i \quad \forall i = 1, \ldots, n \tag{2.4.1}$$

That is, a user uses y_i less or equal to what demands c_i. The point is that there are resource allocation mechanisms that allocate more than what is demanded (inefficient) or allocate less than what is demanded (unfair). In general, if $x_i \leq c_i$, then $y_i = x_i \leq c_i$, and if $x_i \geq c_i$, then $y_i = c_i \leq x_i$. The following definitions are used in resource allocation:

User satisfaction occurs whenever $x_i = c_i$. That is, the user receives what he/she demands.

Efficient allocation (also called **network resource utilization**) is defined as the share of a resource that always causes the entire resource to be used:

$$\sum_{i=1}^{n} y_i = C \tag{2.4.2}$$

Network satisfaction is generally defined as an allocation in which the most demanding users receive more. This is because, the most demanding users pay more for the use of the resource.

Fairness ("equidad" in Spanish) measures whether users or applications or flows are receiving a fair share of the system's resources. In TCP this means that no single flow receives a larger share of the network than others, even if there are other protocols besides TCP. On the Internet, fairness is only achieved if all flows play by the same rules, a situation that is difficult to occur as fairness is poorly defined for short flows, and there are many versions of TCP coexisting,

different transport protocols (TCP vs UDP), different traffic profiles (bulky traffic vs transactional traffic), different link utilisation and traffic demands, different RTT's, etc.

The concept of fairness is closely linked to the allocation of resources in networks. One metric to measure fairness is the Jain index. Suppose n users share a resource. Each user receives x_i amount of the resource, with i=1, ..., n. For example, consider a link with capacity C bit/s and the allocation x_i is an allocation of bandwidth over the link of capacity C. The Jain index is defined as:

$$JI = \frac{(\sum_{i=1}^{n} x_i)^2}{n(\sum_{i=1}^{n} x_i^2)} = \frac{\bar{x}^2}{\overline{x^2}} \tag{2.4.3}$$

2.4.1 Uniform fair allocation

If we perform a **uniform allocation** of x_i=C/n, then

$$JI = \frac{(\sum_{i=1}^{n} x_i)^2}{n(\sum_{i=1}^{n} x_i^2)} = \frac{(\sum_{i=1}^{n} C/n)^2}{n(\sum_{i=1}^{n} (C/n)^2)} = \frac{(nC/n)^2}{n(C/n)^2 n)} = 1$$

Therefore, given that JI=1, the uniform equal allocation is a fair allocation. However, as we can see in the following example, the uniform fair allocation is not an efficient allocation.

Example 2.3 (Uniform fair allocation) *Let us have N=4 users that share a link with capacity* C= 16 Mbit/s *and they have demands of* c_1= 2 Mbit/s, c_2= 4 Mbit/s, c_3= 10 Mbit/s *and* c_4=15 Mbit/s.

If we perform a uniform allocation of x_i=C/N = 4 Mbit/s, *then JI= 1, and it is fair, however it is not efficient since user 1 needs* 2 Mbit/s *and receives* 4 Mbit/s, *thus* 2 Mbit/s *are unused.*

The reason is that uniform allocation does not take demands into account. If a user is allocated $x_i = C/n \geq c_i$, then it is not efficient, even if it is fair (JI=1).

Other allocations are possible. Let us see some examples.

2.4.2 Maximum throughput allocation

Maximum throughput allocation schedules packets to the least demanding flows until the resource (e.g. CPU, capacity, etc.) is consumed.

Example 2.4 (Maximum fair allocation) *With this strategy, example 2.3 will have as allocation x= (2, 4, 10, 0), and*

$$JI = \frac{(\sum_{i=1}^{n} x_i)^2}{n(\sum_{i=1}^{n} x_i^2)} = \frac{(2+4+10+0)^2}{(4*(4+16+100+0))} = \frac{256}{480} = 0.533$$

Thus, we can see that the allocation is efficient, but not fair. The most demanding sources will suffer from "starvation", as they will have to wait for the less demanding sources to finish (e.g. they have no data to transfer).

The *customer satisfaction* is low, as most users suffer from "starvation" (low Jain index). In addition, *network operator satisfaction* is low, as more demanding users tend to pay more, so that the benefit is not maximised. On the other hand, the *resource utilisation (efficiency)* of the network in general is optimal, as resources are fully utilised. In conclusion, fairness is not easy to measure, as it depends on the point of view (user, network operator, resource utilisation).

2.4.3 Max-min fair allocation

So the question is: are there other allocations that are more efficient than a uniform allocation and still fair? The answer is yes if we use linear and non-linear optimisation models. As an example, let's take **max-min fair allocation** where we consider a resource of capacity C, demands c_i (with i=1, ..., n) and we would like to allocate the resource in such a way that $y_i = x_i$; thus:

$$y_i = x_i \leq c_i \tag{2.4.4}$$
$$\sum_{i=1}^{n} x_i \leq C$$

A max-min fair allocation is a vector of allocations $x^* = (x_1^*, \ldots, x_n^*)$ if, 1) it is a feasible set of solutions (i.e. satisfies eq (2.4.4)), and 2) if for each i=1, ..., n, x_i^* can not be increased, while maintaining the feasibility (i.e. satisfies eq (2.4.4)), without decreasing x_i for some i for which $x_i \leq x_i^*$.

The max-min fair allocation gives more priority to smaller flows (i.e., maximize the minimum rate of each flow or in other words), in the sense that if $x_i \leq x_i^*$ then no increase in x_i^* no matter how large, can compensate for any decrease in x_i, no matter how small. In general, max-min provides better fairness than maximum throughput schedulers since shares the resource among all users (avoids starvation).

In general, max-min allocation is hard to calculate mathematically, but there is an easy algorithm to find a max-min allocation. The idea is to *allocate in order of increasing demand*, *no flow gets more than demand*, and *the excess, if any, is equally shared*:

Step 1: label users with demands from lowest to highest;

Step 2: assign x_i = C/N (uniform allocation) bit/s to each user;

Step 3: If x_1 = C/N $\leq c_1$, stop

otherwise; If x_1 = C/N $\geq c_1$, add (x_1-c_1) to the rest of users 2, ..., n, so, they receive x_i = C/N + (x_1-c_1)/(n-1), with i=2, ..., n and x_1=c_1,

Step 4: iterate with the following user i if it receives more than its demand c_i.

Example 2.5 (Max-min fair allocation) *If we repeat the previous example with demands* $c_1 = 2\,\text{Mbit/s}$, $c_2 = 4\,\text{Mbit/s}$, $c_3 = 10\,\text{Mbit/s}$ *and* $c_4 = 15\,\text{Mbit/s}$, *we can achieve an allocation of x=(2, 4, 5, 5)* Mbit/s *with a Jain's index JI=0.914 and a network utilization of 100%.*

Max-min implementation in real queue systems results in **PQ-RR** queue management systems.

2.4.4 Proportionally fair allocation

There are other allocations such as **proportionally fair allocation**. In proportionally fair allocation, what is given to one flow is taken away from others in proportion to their allocation. A proportionally fair allocation is a vector of allocations $x^* = (x_1^*, \ldots, x_n^*)$ if, 1) it is a feasible set of solutions (i.e., it satisfies eq (2.4.4)), and 2) if for any other feasible vector $x = (x_1, \ldots, x_n)$, the aggregate of proportional changes is zero or negative $\sum_{i=1}^{n}(x_i - x_i^*)/x_i^* \leq 0$.

Proportionally fair allocation also gives more priority to smaller flows, but less emphatically. It says that if user i increases its allocation, there will be at least one other user whose rate will decrease and, moreover, the proportion by which it decreases will be greater than the proportion by which user i's rate increases. Proportionally equal allocation (like max-min equal allocation) avoids source starvation.

Proportionally fair allocation implementation in real queue systems results in **WFQ** queue management systems.

2.5 Queueing service stratgies

The objective is to define a set of queues served by a single server using a first-in, first-out (FIFO) scheduling policy. Since the number of flows can be large and it is impractical to define a queue per flow, this mechanism is often used with classes of flows. A class is a set that groups flows in any arbitrary way. The number of classes can therefore be small, e.g. less than 10 (some technologies such as MLPS use 3 bits to define priorities, so that eight classes can be defined).

2.5.1 Round robin (RR)

We assume a system in which flows or classes of flows are assigned a queue and there is a single server. Then, **round robin (RR)** scan class queues serving one from each class. There is not any kind of priority or special treatment.

In the following we assume that there exists a set of queues and a single server scheduler.

2.5.2 Weighted round robin (WRR)

Weighted round robin (RR) scan class queues serving one from each class according to the weight. There is not any kind of priority or special treatment.

Example 2.6 (Weighted round robin (WRR)) *Let's assume 4 queues with weights* $\{w_1, w_2, w_3, w_4\} = \{4, 2, 3, 2\}$. *Then:*

- *first round of scheduling: packet of queues 1, 2, 3, 4;*
- *second round of scheduling: packet of queues 1, 2, 3, 4;*
- *third round of scheduling: packet of queues 1, 3;*
- *four round of scheduling: packet of queue 1;*
- *fith round of scheduling: as the first round and so on.*

2.5.3 Weighted deficit round robin (WDRR)

There are models that accounts for the packet size. For example **Weighted Deficit Round Robin (WDRR)** schedules packets considering the packet length, ensuring that packets are scheduled equally. In WDRR scheduling, a large-sized packet obtains less bandwidth than a small-sized packet.

2.5.4 Priority queueing with round robin (PQ-RR)

Priority queueing with round robin (PQ-RR) scan class queues serving one from each class that has a non-empty queue. As can be seen in figure 9, packets arrive at each queue, and the FIFO round robin server extracts one packet from each queue in each round, always starting with the highest priority queue.

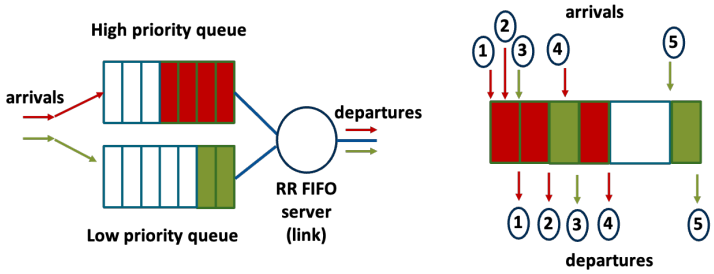

Figure 9: Priority queueing with round robin (PQ-RR).

Example 2.7 (Priority queueing with round robin (PQ-RR)) *Let's take figure 9.b) with transmission times of T_t=1ms, and observe the order of arrivals with labels (1) for the red packets and label (2) for the green packets: $T_1^{(1)}$ = 0ms, $T_2^{(1)}$ = 0.4ms, $T_3^{(1)}$ = 1ms, $T_1^{(2)}$ = 2.6ms and $T_2^{(2)}$ = 6ms. The order of departure using PQ-RR is $T_1^{(1)}$ = 0ms, $T_2^{(1)}$ = 1ms, $T_1^{(2)}$ = 2ms, $T_3^{(1)}$ = 3ms, $T_2^{(2)}$ = 6ms. Note than at time 0 ms there is only one packet at the red queue and nobody at the green queue, and thus only can be scheduled packet $T_1^{(1)}$, while at time 1ms there is one red and one green at the queues, so packets $T_2^{(1)}$ and $T_1^{(2)}$ are scheduled for times 1ms and 2ms. At time 3ms there is only packet $T_3^{(1)}$ for being scheduled, and at time 6ms there is only packet $T_2^{(2)}$ for being scheduled.*

The PQ scheduling mechanism may result in package starvation in low-priority queues. For example, let us assume 8 queues, with queue 7 with highest priority and queue 0 with the lowest priority. If data flows mapped to queue 7 arrive at a 100% link rate in a given period, the scheduler does not process flows in queues 0 to 6.

2.5.5 Priority queueing with weighted round robin (PQ-WRR)

Priority queueing with weighted round robin (PQ-WRR) scheduling integrates the advantages and offsets the disadvantages of both PQ scheduling and WRR scheduling. Packets from queues with lower priorities can obtain the bandwidth by WRR scheduling and short-delay services can be scheduled first by PQ scheduling.

Example 2.8 (PQ-WRR) *Let us assume 8 queues. Queue 7, 6 and 5 serve packets according to PQ, while queues 4 to 0 serves packets according to a weight in a WRR manner. That means that delay sensitive data goes to queues 7-5, while the rest to queues 4-0. Note that the system schedules traffic in other queues in WRR mode only after the traffic in queue 7, queue 6, and queue 5 are scheduled.*

In general, we can say that PQ-RR (or PQ-WRR) protects flows from misbehaving flows that will not affect the performance of well-behaved flows; however, they are more complex than FIFO, since, for example, in PQ-WRR it is necessary to maintain one queue state per flow (or class). In addition, the service is related to the number of packets, regardless of their size (small or large packets count as one packet).

We can say that PQ-RR (or PQ-WRR) implements a **max-min fair allocation**, so, it provides some kind of fair allocation to flows or classes of flows in the Internet.

2.6 Weighted fair queueing (WFQ)

Fair queueing (FQ) (also called processor sharing) [1] (chapter 23) is a scheduler that divides the capacity of the link uniformly among the flows, figure 10. If there are n flows that traverse the link (flows that are in the queue), the capacity of the link (amount of service) given to each flow with non-empty queue is:

$$g_i \geq r_i = \frac{C}{n} \tag{2.6.1}$$

The disadvantage of this strategy is that all flows in the queue receive the same treatment and therefore receive the same amount of service irrespective of the traffic profiles of the flows, the number of packets in the queue per flow, etc.

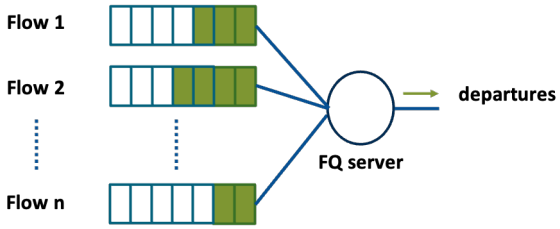

Figure 10: Fair queueing scheduling (FQ).

Weighted fair queueing (WFQ) [1] (chapter 23) is an FQ scheduler that divides the link capacity among the flows, but instead of giving the same share of the capacity (uniformly), it gives it according to the weights assigned to each flow, figure 11. If there are n flows traversing the link (flows that are in the queue), the link capacity (amount of service) given to each flow with non-empty queue is:

$$g_i \geq r_i = \frac{w_i C}{\sum_{i=1}^{n} w_i} \tag{2.6.2}$$

As we can observe, FQ is a WFQ with equal weights in all flows. We will see in example 2.10 that a flow i will always obtain a rate g_i higher or equal than r_i.

The reason, is that the worst case is that all flows are in the queue, and then the flow will obtain the rate g_i=r_i specified by eq (2.6.2). On the other hand, the best case will be that the flow is alone in the queue, and then g_i=C.

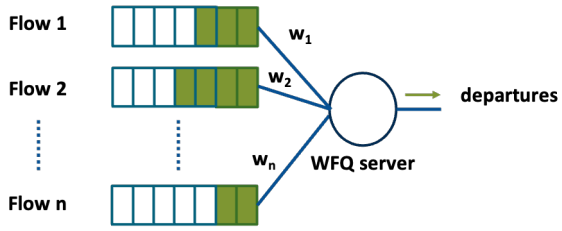

Figure 11: Weighted fair queueing scheduling (WFQ).

Example 2.9 (Weighted fair queueing (WFQ)) *Let's assume five flows with weights* $\{w_1, w_2, w_3, w_4, w_5\}=\{10, 20, 15, 25, 10\}$. *Let us take interval* $[T_1, T_2]$ *in which the five flows are in the queue. Then, the flows will receive* $r_i = \frac{w_i C}{\sum_{i=1}^{n} w_i}$ *b/s. It is to say* $\{r_1, r_2, r_3, r_4, r_5\}=\{0.125, 0.25, 0.1875, 0.3125, 0.125\}$ *of* C *b/s. Let us take interval* $[T_3, T_4]$ *in which only flows* f_1, f_3 *and* f_4 *are in the queue. Then, they will receive* $\{r_1, r_3, r_4\}=\{0.2, 0.3, 0.5\}$ *of* C *b/s.*

2.6.1 General processor sharing (GPS)

WFQ is a scheme that is based on what is called a **processor sharing** scheme. In this scheme it is assumed that flows are served in a fluid manner. A fluid flow approach is one in which the link is assumed to be a pipe and the traffic of each flow is a fluid that shares the pipe (i.e. the link) with other fluids (i.e. flows), and therefore, GPS is a generalization of the WFQ. Let us explain how this works with an example:

Example 2.10 (General processor sharing (GPS)) *Let us assume a queue with two flows and a server with capacity* C=1 Mbit/s. *Flow 1 sends packets at time* 1 , 2 , 3 *and* 11 ms *with sizes* 1000 , 1000 , 2000 *and* 2000 bit. *Flow 2 sends packets at time* 0 , 5 , 9 ms *with sizes* 1000 , 3000 *and* 2000 bit.

Now, let us see how it works with w_1=w_2 *and* w_1=$2 * w_2$.

GPS		**Flow 1**				**Flow 2**		
Arrival time (T_A, ms)		1	2	3	11	0	5	9
Packet size (*1000 bits)		1	1	2	2	3	2	2
w_1=w_2	Departure time (T_S, ms)	3	5	9	13	5	9	11
w_1=$2 * w_2$	Departure time (T_S, ms)	4	5	9	13	4	8	11

Table 1: General Processor Sharing.

To calculate the finishing time of a packet of a given flow, we use the following equation:

$$T_S^{(k)}(j+1) = max\{T_S^{(k)}(j), T_A^{(k)}(j+1)\} + \frac{L^{(k)}(j+1)}{r_k} \tag{2.6.3}$$

where $T_S^{(k)}(j)$ and $T_A^{(k)}(j)$ are the departure and arrival times of packet j at flow k; $L^{(k)}(j)$ is the size of packet j at flow k It is important to note that the service rate has (and thus the departure time) to be re-calculated at each new arrival. Let us see how the departure times are obtained for the case w_1=w_2, figure 12:

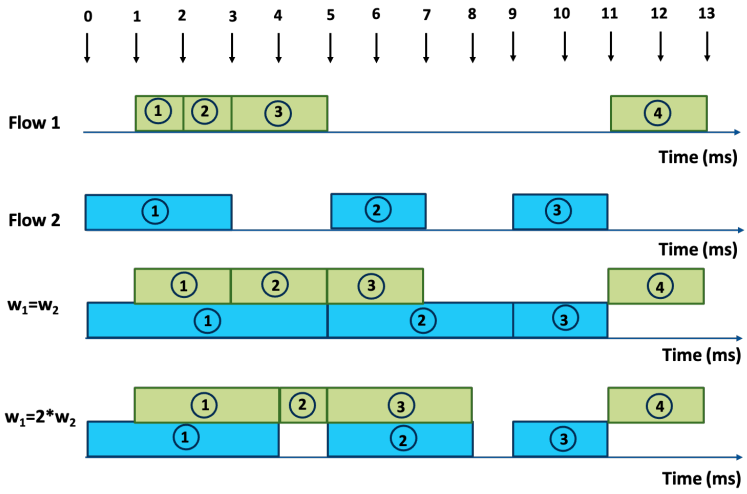

Figure 12: General processor sharing (GPS).

Time 0 ms: we note that r_2=C, and $T_S^{(2)}(1) = max\{0,0\} + 3000/C = 3ms$.

Time 1 ms: we note that r_1=$C/2$ and r_2=$C/2$. The departure times are now $T_S^{(1)}(1)$=max{0,1}+1000/(C/2) = 3 ms. The server has just send 1000 bits from flow 2, so only 2000 bits remain to be sent (we can consider that a packet of 1000 bits has been issued), and $T_S^{(2)}(1)$=max{1,0}+2000/(C/2)= 5 ms.

Time 2 ms: packets 1 of flows 1 and 2 are being served. Packet 2 from flow 1 is queued.

Time 3 ms: packet 1 of flow 1 has just been delivered, packet 2 of flow 1 has to be served, and packet 3 that arrives is queued. We note that r_1=$C/2$ and r_2=$C/2$; $T_S^{(1)}(2)$=max{3,2}+1000/(C/2)= 5 ms. Packet 1 of flow 2 is still being served (expected to leave at time 5 ms).

Time 4 ms: packet 2 of flow 1 is expected at time 5 ms, packet 3 of flow 1 is in the queue, packet 1 of flow 2 is expected at time 5 ms.

Time 5 ms: packet 2 of flow 1 and packet 1 of flow 2 have just leaved. Arrives packet 2 of flow 2. We calculate the departures of packet 3 of flow 1 and packet 2 of flow 2. r_1=$C/2$ and r_2=$C/2$. $T_S^{(1)}(3)$=max{5,3}+2000/(C/2)= 9 ms and $T_S^{(2)}(2)$=max{5,5}+2000/(C/2)= 9 ms.

Time 9 ms: packet 3 of flow 1 and packet 3 of flow 2 have just leaved. Arrives packet 3 of flow 2. r_2=C and $T_S^{(2)}(3)$=max{9,9}+2000/(C)=11 ms.

Time 11 ms: packet 4 of flow 1 arrives and packet 3 of flow 2 have just leaved. r_1=C and $T_S^{(1)}(4)$=max{9,11}+2000/(C)=13 ms.

We leave as exercise to repeat the process for the case w_1=$2 * w_2$.

2.6.2 Weighted fair queueing (WFQ): a virtual time implementation of PGPS (Optional material)

However, GPS cannot be implemented in a real queue, as the server cannot mix bits from different streams. A possible solution would be to implement a general bit-by-bit processor sharing, where the server sends a bit from one stream and then a bit from another stream. However, again, routers are not designed to work (on a receiver basis) on a bit-by-bit basis. Routers work by receiving packets. The solution is a **packet-by-packet general processor sharing** which is the well-known **weighted fair queueing (WFQ)**.

We will adopt the convention that a packet has arrived only after its last bit has arrived. General packet-by-packet processor sharing, or WFQ, [5] calculates the output time of each packet $T_S^{(k)}(j)$ and serves the packets in increasing order of time $T_S^{(k)}$. Assume now that the server becomes free in time τ. The next packet to go out by GPS may not have arrived at time τ and, since the server does not know when it will arrive, there is no way for the server to keep the job and serve the packets in increasing order of $T_S^{(k)}$. The server chooses the first packet that would complete the service in the GPS simulation if no additional packets arrive after the τ time.

The main problem in implementing PGPS is that departure times have to be recalculated at each flow arrival. In addition, packets are far from the same size, and mixing bulk and real-time traffic tends to worsen the size variation. Thus, we will use the concept of *virtual time* to track the progress of GPS leading to a practical implementation of PGPS that we will call WFQ. This "real-time" fair queuing strategy consists of transmitting packets in order of a calculated virtual finishing time, which benefits flows with smaller packets and flows that have not sent packets recently. From now on, we will call **packet-by-packet general processor sharing (PGPS)** as **weighted fair queueing (WFQ)**.

For convenience, sometimes we express w_i as a percentage of C.

Example 2.11 (Normalization of the weights) *Let us assume that we have*

a link of capacity C=2 Mbit/s *and we have 3 flows. We assign to the first flow* 500 kbit/s*, to the second flow* 500 kbit/s *and to the third flow* 1 Mbit/s*, then* w_1=0.25*,* w_2=0.25 *and* w_3=0.5*.*

Let each arrival and departure of the GPS server be denoted as an event, and let t_j, be the time at which the *j-th* event (e.g. a packet arrival/departure) occurs (simultaneous events are arbitrarily ordered) [5]. Let t_1= 0 be the time of the first arrival of a busy period. Now let us observe that, for each j= 2, 3, ..., the set of sessions that are busy in the interval $(t_{j-1},\ t_j)$ is fixed, and we can denote this set as B_j. Define the **virtual time** $V(t)$ as zero when the server is idle (or the beginning of a busy period). Then $V(t)$ evolves as:

$$V(0) = 0 \tag{2.6.4}$$
$$V(t_{j-1} + \tau) = V(t_{j-1}) + \frac{\tau}{\sum_{k \in B_j} w_k} \qquad \tau \leq t_j - t_{j-1},\ \ j = 2, 3, \ldots$$

Then, the rate of change of V is $\partial V(t_j + \tau)/\partial\tau$ is $1/(\sum_{k \in B_j} w_k)$, and each backlogged (queued) flow k receives service at rate $w_k \partial V(t_j + \tau)/\partial\tau$. Thus, V can be interpreted as increasing at the marginal rate at which backlogged sessions receive service.

Now suppose that the j packet of *k-th* flow arrives at time $T_A^{(k)}(j)$; and the packet has length $L_i^{(k)}$. Then, denote **the virtual times** at which this packet begins and completes service in the simulated system as $\hat{T}_A^{(k)}(j)$ and $\hat{T}_S^{(k)}(j)$ respectively. Defining $\hat{T}_S(0)^{(k)} = 0$ for all k, we have

$$\hat{T}_A^{(k)}(j+1) = \max\{\hat{T}_S(j)^{(k)}, V(T_A^{(k)}(j+1))\} \tag{2.6.5}$$
$$\hat{T}_S^{(k)}(j+1) = \hat{T}_A^{(k)}(j+1) + \frac{L^{(k)}(j+1)}{r_k}$$

Observe that $\hat{T}_S^{(k)}(j)$ give us virtual finish times that allow us to order who leaves first in the WFQ system. However, the real finish times $T_S^{(k)}(j)$ are given by the current time plus the transmission time (i.e., quotient between the packet sizes and the real link speed).

There are three attractive properties of the virtual time interpretation from an implementation point of view. First, virtual time finishing times can be determined at the arrival time of packets. Secondly, packets are served in order of virtual time finishing time. Finally, it is only necessary to update the virtual time when events occur in the GPS system. However, the price to pay for these advantages is some overhead in tracking the B_j sets, which is essential in updating the virtual time.

A natural question is how much later packets may depart the system under PGPS relative to GPS. It is easy to proof that if T_S^{PGPS} and T_S^{GPS} are the departure times for PGPS and GPS respectively, then:

$$T_S^{PGPS} - T_S^{GPS} \leq \frac{L_{max}}{C} \tag{2.6.6}$$

where L_{max} is the maximum packet length and C is the capacity of the server.

The above result can easily be misinterpreted to say that the packet WFQ discipline and the fluid GPS discipline provide almost identical service except for a difference of one packet. Contrary to this popular (but incorrect) belief, it has been shown that there could be large discrepancies between the services provided by WFQ and GPS. In fact, what has been shown is that WFQ cannot lag behind GPS by a maximum packet size. However, WFQ can be far ahead of GPS in terms of the number of bits served for a session.

2.6.3 Class-based weighted fair queuing (CBWFQ)

Working with flows is quite difficult due to scalability issues (the number of flows a router has to handle is huge in relation to the time needed to classify and process the flow). The flows are then grouped into classes, which require similar queuing treatment (i.e. have similar QoS requirements), using precedence bits to classify the classes. WFQ is then applied in a per-class queue management system where the weight is applied to the classes and not to the flows. This scheme is called **class-based weighted fair queuing (CBWFQ)**.

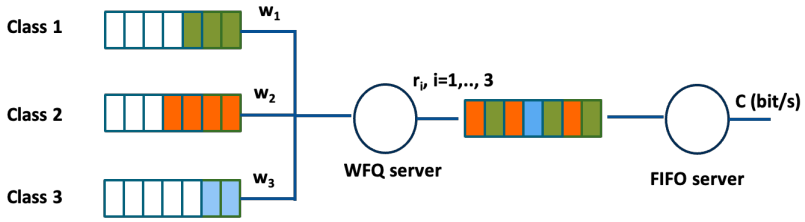

Figure 13: Class-based weighted fair queuing (CBWFQ).

The simplest hierarchy is that flows belonging to the same class are scheduled FIFO, while class packets are scheduled WFQ, figure 13. However, more complex schemes can be envisaged. For example, to ensure fairness between flows of the same class, a per-flow scheduling scheme (e.g. FQ) can be applied to each class. Over time, this will create a hierarchy of scheduling schemes that will determine how the bandwidth of a link is shared, first between higher level classes, then between flows of the same class, or second level (sub)classes of the same higher level class, and so on.

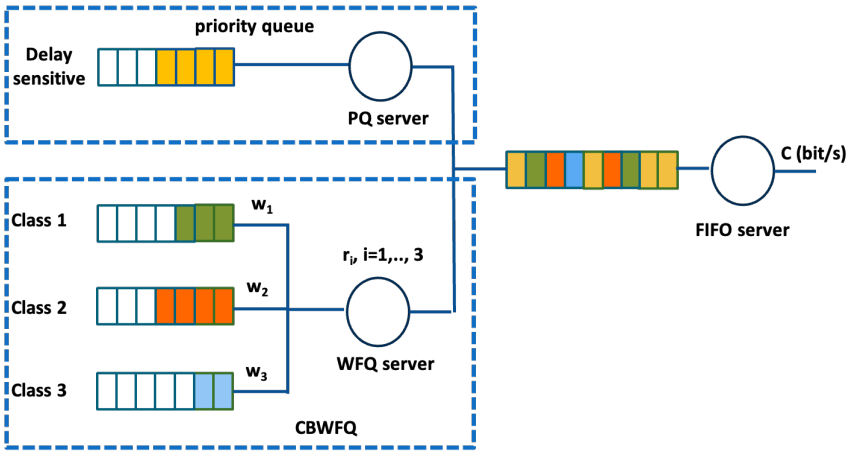

Figure 14: Low latency queueing (LLQ).

2.6.4 Low latency queueing (LLQ)

The **low latency queueing (LLQ)** feature brings strict priority queuing (PQ) to CBWFQ which reduces jitter in voice conversations. Without LLQ, CBWFQ provides WFQ based on defined classes with no strict priority queue available for real-time traffic. Thus, traffic that is delay sensitive such as VoIP, are assigned to a priority queue that is served before the CBWFQ classes, figure 14.

3 Traffic shaping and policing

If in fair queue allocation (FQ/WFQ) a sender is idle, the fair queues distribute that sender's bandwidth among the other senders. The purpose of token bucket mechanisms is to monitor bandwidth, so the bandwidth allocated to a sender is the bandwidth it receives.

Example 3.1 (Allocation with FQ) *Let us assume a link with capacity 1 Gbit/s and packet sizes of 1000 bit, and a connection, let us call it A, to which we allocate 25% of the link (i.e.,* 0.25 Gbit/s*). Since, user A is only paying for this rate of* 0.25 Gbit/s*, we do not want that this connection uses more than* 0.25 Gbit/s*. Using a WFQ, when the queue is idle and not other users are sending packets, user A gets* 1 Gbit/s*. What happens if user A makes use of this knowledge and sends traffic at higher rate than the stipulated rate of* 0.25 Gbit/s*.*

We could use the following strategy: knowing that the transmission time of the link for 1 packet is $T_C = 10^3/10^9 = 1$ µs*, we could wait* 3 µs *after a packet of user A is sent, so the user is receiving a 25% share of the link (1 of every* 4 µs*).*

However, it can happen that a packet of user A is expected at T= 12 µs and arrives at 13 µs, and the following packet arrives at 16 µs. What happen with this last packet? Do we send it? It has arrived at time but the gap with the previous packet is not what it should be since there are 3 µs of inter-packet gap instead of 4 µs (jitter is producing this effect).

The solution is a **token bucket specification** which allows for specification of both an average rate and also a burst capacity. The implemented token-bucket specification is often called a **token-bucket filter**. If a packet does not meet the token-bucket specification, it is **non-compliant** and we can follow one of these actions:

- drop the packet (**called policing**) when the traffic reaches its maximum excess traffic. Bursts are propagated and it does not need to queue packets. It is applied to inbound traffic;
- delay the packet (**called shaping**) until the bucket is ready. Shaping mechanisms queue packets and schedule these packets for later retransmission with the objective of smoothing the traffic. It is applied to outbound traffic and it works in conjunction with schedulers such as PQ, WFQ, CBWFQ, etc;
- mark the packet as non-compliant. The marked packets are sent at lower priority, and are dropped by some downstream router if this is congested.

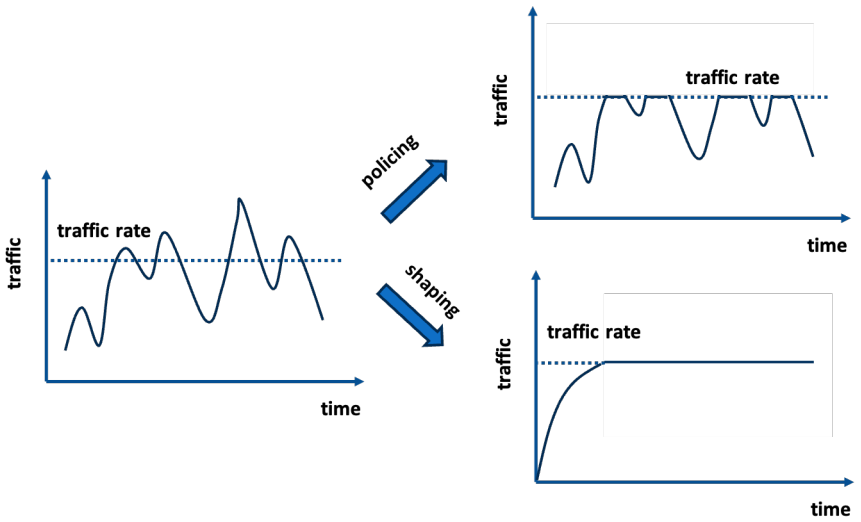

Figure 15: Traffic shaping and policing.

3.1 Token/Leaky bucket algorithms

We define a system with a **bucket capacity** (bucket depth) of B_{max} tokens (e.g., bytes or packets). To send a packet, we need to be able to take a token from the bucket; if the bucket is empty, the packet does not qualify and must receive special treatment as described above (discard, delay, or mark the packet). However, if the bucket is full, the sender can send a burst of packets corresponding to the capacity of the bucket (at which point the bucket will be empty). The bucket is refilled with a **token fill rate** of ρ bit/s.

Then, we will call TB(ρ, B_{max}) the token specification. We will call B(t) the capacity of the bucket at time t. Then, when a packet arrives, if B$\geq$1, the packet is transmitted and B=B-1. If B=0, then the packet is non-compliant and receives a treatment (drop, delay or mark the packet).

The fluid-flow equivalence is called the **leaky bucket** mechanism. In the leaky bucket mechanism, the bucket capacity is measured in bytes and not in tokens (or packets), thus allowing arbitrary sized packets to arrive. Whenever a packet of size L Byte arrives in the system, the packet is transmitted if $B \geq L$.

If $S_i(\tau, t)$ is the amount (in Bytes) of session i flow that leaves the leaky bucket in the interval (τ, t) with token bucket specification LB(ρ_i, B_i), then:

$$S_i(\tau, t) \leq B_i + (t - \tau)\rho_i \tag{3.1.1}$$

Example 3.2 (Simple leaky bucket example) *Let us take an example with a link with capacity C=*1 Mbit/s *with packet size L= 1000 bits. The transmission time is* T_t=1ms. *Let us assume that a user has LB(ρ_i, B_i)= (*0.1 Mbit/s*, 1000 Bytes). Then, every ms, the bucket is filled with* $(t - \tau) * \rho_i$*= 1 ms * 0.1 Mb/s = 100 bits. Thus, the user will be able to send a packet every 10 transmission times, in which the bucket size will reach 1000 bits (1 packet).*

It is possible to limit the maximum delay and queue length (for session i) if the connection is leaky bucket controlled [5]:

$$D_i^{max} \leq \frac{B_i}{\rho_i} \tag{3.1.2}$$

$$Q_i^{max} \leq B_i \tag{3.1.3}$$

There is a slightly difference between a leaky bucket and a token bucket filter. In the **leaky bucket**, when the host has to send a packet, the packet is introduced in the bucket, the bucket leaks at constant rate, meaning that packets leaves the system at the leaky rate. If the bucket is full, the packet is discarded. In the **token bucket**, tokens are generated at regular intervals of time (at the token rate), if there is a packet ready, a token is removed from bucket and the packet

is send. If there is a no token in the bucket, the packet can not be send and waits until there is a token ready.

The leaky bucket provides fairness and stability to users and creates a predictable and constant rate of packets, but does not allow the user to make full use of their bandwidth (send bursts of traffic). On the other hand, the token bucket allows bursts of packets to be sent until the bucket is empty, which provides great flexibility and responsiveness to the user. However, this can cause unfairness and lack of quality of service to other users who are not as bursty (starvation) and can be exploited by malicious or greedy clients.

Example 3.3 (Token bucket) *Let us assume a bucket specification of TB(1/3 packets/ms, 3 packets), and a connection that sends packets (same size with T_t= 1 ms) at times T_i= 0, 1, 2, 3, 4, 6, 7, 8, 10 ms. Let's see which packets are compliant and which ones are not compliant, table 2. We increase the bucket size if arrives a token before decreasing the bucket and checking the compliance of the packet.*

Time (ms)	0	1	2	3	4	5	6	7	8	9	10
Packet arrival	✓	✓	✓	✓	✓		✓	✓	✓		✓
B^-	3	$2\frac{1}{3}$	$1\frac{2}{3}$	1	$0\frac{1}{3}$	$0\frac{2}{3}$	1	$0\frac{1}{3}$	$0\frac{2}{3}$	1	$1\frac{1}{3}$
B^+	2	$1\frac{1}{3}$	$0\frac{2}{3}$	0	$0\frac{1}{3}$	$0\frac{2}{3}$	0	$0\frac{1}{3}$	$0\frac{2}{3}$	1	$0\frac{1}{3}$
Packet compliance	✓	✓	✓	✓	x		✓	x	x		✓

Table 2: Token bucket TB(r, B$_{max}$)= TB(1/3 packets/ms, 3 packets), symbol B^- means bucket size before packet arrival and B^+ means bucket size after packet compliance checking.

3.2 Traffic parameters

- **CIR (committed information rate):** represents the average data rate (bit/s) associated to a service or flow (not an instantaneous data rate);
- **EIR (excess information rate):** average data rate (bit/s) in excess with respect CIR. Sometimes it is specified as the **PIR (peak information rate)**, and PIR=CIR+EIR. But EIR=PIR-CIR, so we can see EIR as an excess;
- **CBS (committed burst size):** size in Bytes of the transmitted information. Then, the CBS is the amount of bytes that can be sent over a period of time T when congestion occurs. Normally is the packet size, e.g., in Internet if the packet size is 1500 B, then CBS=1500 B;
- **EBS (excess burst size):** excess in size in Bytes of the transmitted information. Amount of extra bytes that can sent over the time T that

still remain conformant with the CIR and EIR. If EBS>0, then you can send traffic exceeding the CIR.

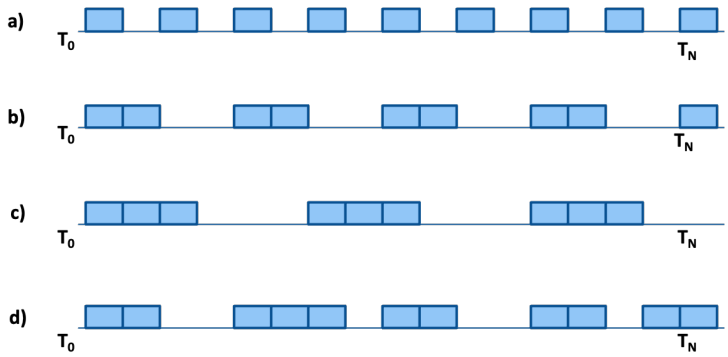

Figure 16: Traffic parameters example.

Example 3.4 (Traffic parameters) *Let us take an example with a link with capacity* C=1 Gbit/s *with packet size* L=*1500 Bytes. The transmission time is* $T_t = \frac{L}{C} = 1.5\,\mu s$. *Let us assume that the flow sends with a traffic profile of 1 packet every* 3.0 µs, *figure 16.a). It is easy to check that the CIR*=0.5 Gbit/s, *CBS*=1500 B *(1 packet) and EBS=0.*

Let us consider the traffic profile of figure 16.b), where the flow sends two packets back-to-back every 6.0 µs. *We can easily see that the CIR*=0.5 Gbit/s, *CBS*=1500 B *(1 packet), EIR*>0 *and EBS*>0.

Let us consider the traffic profile of figure 16.c), where the flow sends three packets back-to-back every 9.0 µs. *We can easily see that the CIR*= 0.5 Gbit/s, *CBS*= 1500 B, *EIR*>0 *and EBS*>0.

Finally, let us consider the traffic profile of figure 16.d), where the flow sends two packets back-to-back and from time to time three packets back-to-back. We can easily see that the CIR=0.5 Gbit/s, *CBS*=1500 B, *EIR*>0 *and EBS*>0.

We can see that in the four cases, the flow sends packets with a CIR of 0.5 Gbit/s *and a CBS of* 1500 B. *However, the impact these traffic profiles have on a router is different. Suppose we have two such flows arriving at a router, each on a different link, and both flows leave on the same outgoing link. All links are with the same capacity of* C=1 Gbit/s. *Let us also assume that both incoming flows are synchronised (packets arrive in the same slot). What is the buffer occupancy in each case? We can observe, figure 17, that although cases a), b) and c) have the same CIR and same CBS, they produce different bursts of packets (different EBS) and thus, have a different impact on the number of packets that the router has to buffer.*

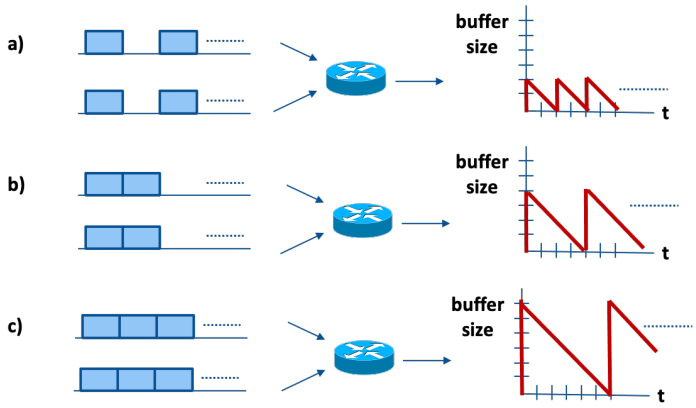

Figure 17: Queue size impact example.

Example 3.5 (Dual Leaky bucket) *Let us assume a system in which we want to control the mean rate (CIR) and the peak rate (PIR). Let p=PIR and r=CIR. We define a bucket specification for the peak rate* $TB_1(p, CBS)$*, and a second bucket specification for the mean rate* $TB(\rho, EBS)$*, where CBS is the maximum packet size and EBS=B (the bucket size). Under these conditions, the allowed number of bytes in the system in the interval (0,t) is:*

$$S(0,t) \leq p \times t + CBS \tag{3.2.1}$$
$$S(0,t) \leq \rho \times t + EBS$$

Example 3.6 (A single rate three color marker - RFC2697 (optional material)) *Also called srTCM meter. The srTCM meters an IP packet stream and marks its packets either green, yellow, or red. Marking is based on CIR (measured in bytes of IP packets per second, including IP headers), CBS (measured in bytes) and EBS (measured in bytes) parameters. A packet is marked green if it doesn't exceed the CBS, yellow if it does exceed the CBS, but not the EBS, and red otherwise. The srTCM is useful, for example, for ingress policing of a service, where only the length, not the peak rate, of the burst determines service eligibility.*

The CBS and EBS must be configured so that at least one of them is larger than 0. It is recommended that when the value of the CBS or the EBS is larger than 0, it is larger than or equal to the size of the largest possible IP packet in the stream. There are 2 token buckets: TB_C*(CIR, CBS) and* TB_E*(CIR, EBS).*

The meter operates in one of two modes. In the **color-blind mode***, the meter assumes that the packet stream is uncolored. In the* **color-aware mode** *the meter assumes that some preceding entity has pre-colored the incoming packet stream so that each packet is either green, yellow, or red.*

When a packet of size L bytes arrives at time t, the following happens if the srTCM is configured to operate in the **color-blind mode***:*

- *if $TB_C(t)$-L≥0, the packet is green and TB_C is decremented by L down to the minimum value of 0, else;*
- *if $TB_E(t)$-L≥0, the packets is yellow and TB_E is decremented by L down to the minimum value of 0, else;*
- *the packet is red and neither TB_C nor TB_E is decremented.*

When a packet of size L bytes arrives at time t, the following happens if the srTCM is configured to operate in the **color-aware mode***:*

- *If the packet has been precolored as green and $TB_C(t)$-L≥0, the packet is green and TB_C is decremented by L down to the minimum value of 0, else;*
- *If the packet has been precolored as green or yellow and if $TB_E(t)$-L≥0, the packets is yellow and TB_E is decremented by L down to the minimum value of 0, else;*
- *the packet is red and neither TB_C nor TB_E is decremented.*

The srTCM can be used to mark a packet stream in a service, where different, decreasing levels of assurances (either absolute or relative) are given to packets which are green, yellow, or red. For example, a service may discard all red packets, because they exceeded both the committed and excess burst sizes, forward yellow packets as best effort, and forward green packets with a low drop probability.

3.3 Leaky bucket algorithm with GPS scheduler

Let us assume a router with a GPS scheduler and a leaky bucket algorithm that controls the traffic entrance, figure 18. Each connection i is assured a rate $r_i \leq C$ by the GPS, and the leaky bucket specification for connection i is LB(ρ_i,B$_i$), with $r_i \geq \rho_i$. The constraint in amount of bytes imposed by the leaky bucket is as follows.

We define as $A_i(\tau, t)$ the amount (in bits or Bytes) of session i flow that enters the system and is queued (in a large enough buffer), and $S_i(\tau, t)$ as the amount (in bits or Bytes) of session i flow that leaves the leaky bucket plus GPS system and enters the network in time interval (τ,t]. Let B$_i$ and ρ_i be the bucket size and bucket rate of the leaky bucket, and r_i be the GPS share of the link capacity C (i.e., $r_i \leq p_i$, with p_i the peak rate) , then we say that A_i conforms to (B$_i$, ρ_i, r_i) or $A_i \sim$(B$_i$, ρ_i, r_i) and:

$$S_i(\tau, t) \leq \min\{(t-\tau)r_i, B_i + (t-\tau)\rho_i\} \qquad (3.3.1)$$

This model is similar to the dual leaky bucket eq (3.2.1), in which the peak and mean rate of the connection is controlled. The difference is that here the capacity of the link is shared with other connections.

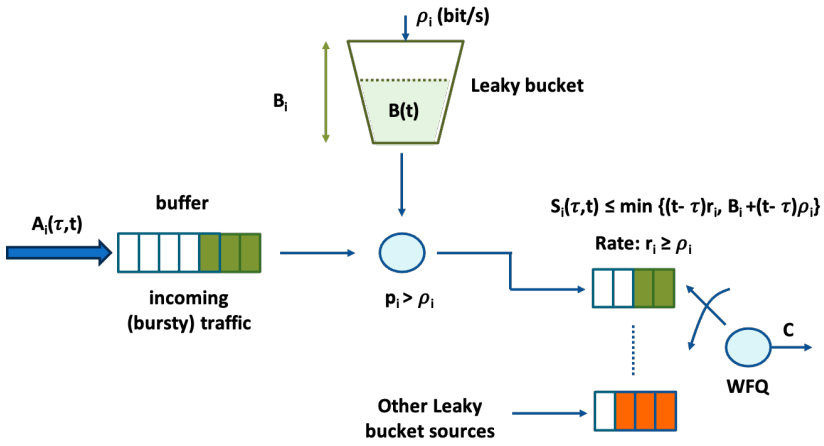

Figure 18: Leaky bucket architecture.

Example 3.7 (A greedy connection) *A greedy connection ($p_i \to \infty$) that tries to transmit as many packets as possible, will consume the bucket at rate r_i (its peak rate), and then it will be limited by the bucket rate ρ_i, it is to say, it will transmit packets at this average rate ρ_i.*

Example 3.8 (A typical configuration for controlling a connection) *A typical configuration is one in which ρ_i=CIR_i, r_i=PIR_i and B_i=EBS_i, and the maximum packet size is the CBS. The connection transmits with $p_i \leq PIR$. When the connection transmits at its peak rate, consumes bytes from the bucket. When the connection slows down to values lower to the CIR, the bucket is replenished, and in average the source transmits at CIR. The maximum that the source can transmit at PIR is the EBS (a burst of packets of size CBS).*

Example 3.9 (Limit of packets in leaky bucket filter) *Let us take example 3.3 and assume a r_i= 1 packet/ms, and calculate the maximum amount of traffic allowed in the leaky bucket filter plus GPS system in a period of 10 ms; it is to say $A_i \sim (B_i, \rho_i, r_i)$= (3 packets, 1/3 packet/ms, 1 packet/ms). Using eq (3.3.1), we get that:*

$$\begin{aligned} S_i(\tau,t) &\leq min\{10ms * 1 \text{ packets/ms}, 3 \text{ packets} + 1/3 \text{ packets/ms} * 10ms\} \\ &\leq min\{10, 3 + 10/3\} \\ &\leq min\{10, 19/3\} \\ &\leq (6 + 1/3) \text{ packets.} \end{aligned}$$

Other metrics are the delay and queue length. The delay and the queue length at time τ, figure 19 are given by [5]:

$$D_i(\tau) = inf\{t \geq \tau; S_i(0,t) = A_i(0,\tau)\} - \tau = t^* - \tau \tag{3.3.2}$$

$$Q_i(\tau) = A_i(0,\tau) - S_i(0,\tau) \tag{3.3.3}$$

We can observe that to calculate the delay, we have to calculate the time t^* in which the input traffic at time τ will leave the system. This time is the one that makes $S_i(0,t^*) = A_i(0,\tau)$.

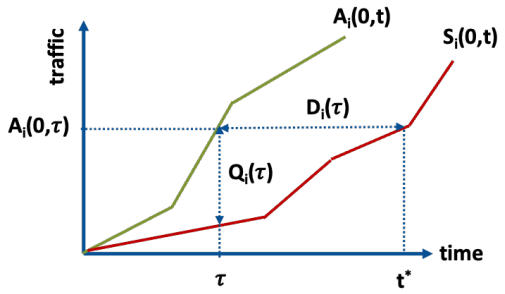

Figure 19: Leaky bucket input (A_i(0,t)), output (S_i(0,t)), queue length (Q_i(t)) and delay (D_i(t)).

It is possible to find bounds on the maximum delay that a connection will get if it is leaky bucket controlled and a GPS scheduler is used. If the connection is greedy and $p \to \infty$ (it is to say, $p \geq C$), then the delay is as in eq (3.1.2):

$$D_i \leq \frac{b_i}{\rho_i} \tag{3.3.4}$$

However, if the connection transmit at a given peak rate (it is to say, $p_i \geq r_i \geq \rho_i$ and $C \geq r_i$), then:

$$D_i \leq \frac{b_i}{(p_i - \rho_i)} \frac{(p_i - r_i)}{r_i} \tag{3.3.5}$$

Finally, we have a third delay boundary for the case of a route path [6], where all routers control the connection with a leaky bucket and a WFQ (i.e. all routers are $A_i \sim$(B$_i$, ρ_i, r_i) conformant). Assuming that the path has K routers, and the maximum packet size is L_{max}, and the connection is greedy ($p_i \geq r_i$) then:

$$D_i \leq \frac{b_i}{\rho_i} + \frac{(K-1)L_{max}}{\rho_i} + \sum_{j=1}^{K} \frac{L_{max}}{C_j} \tag{3.3.6}$$

where, C_j is the link rate for router j=1, ..., K.

4 QoS models

We can consider three models or architectures to send QoS traffic: best effort, integrated services and differentiated services, figure 20.

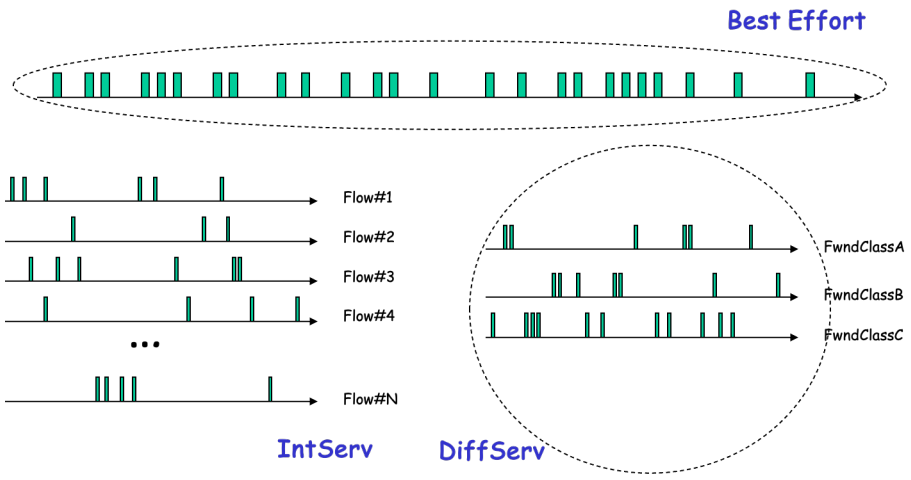

Figure 20: QoS models.

4.1 Best effort

The first so-called **best effort** cannot be considered a QoS model at all, as it does not apply or configure any kind of QoS mechanism. Internet routers do not provide any kind of traffic guarantee, and all packets are treated in the same way. It usually uses a FIFO scheduler. It has the benefits of being the most *scalable*, and it is the easiest and quickest method to implement. It has the disadvantage that it does not guarantee delivery and packets will arrive whenever possible and in any order, if at all. No packet has special treatment, and critical packets (e.g. delay-sensitive packets) receive the same treatment as non-critical packets.

4.2 Integrated services (IntServ)

Integrated services (IntServ) provides a way to deliver end-to-end Qos that real-time applications require by explicitly managing network resources to provide QoS to specific user packet streams (flows). These flows are characterized by a 5-tuple: source IP address (4B), destination IP address (4B), source L4 port (2B), destination L4 port (2B), type of L4 protocol (1B), figure 21.

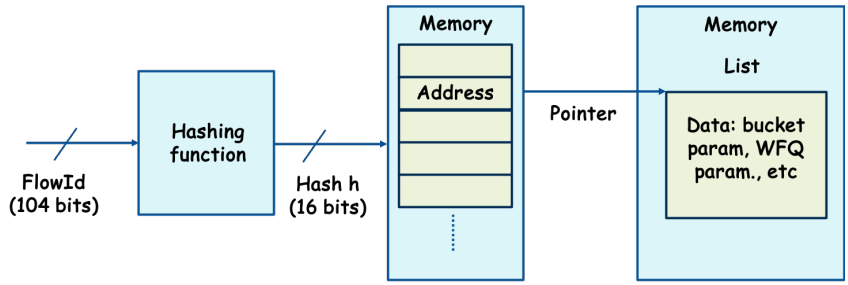

Figure 21: Flow classification.

IntServ is a mechanisms that is connection-oriented (inherit from telephony network design). That means that a flow (or connection) request and reserves hard QoS along the path between source and destination. If some of the routers is not able to provide the QoS requested, the connection is not accepted. The main mechanisms to provide this hard QoS are resource reservation protocols and an admission-control mechanism. In the IntServ model:

- the application requests a specific kind of service from the network before sending the data;
- the application informs the network of its traffic profile and requests a particular kind of service that can encompass its bandwidth and delay requirements;
- IntServ uses the Resource Reservation Protocol (RSVP) to signal the QoS needs of an application's traffic along devices in the end-to-end path through the network. Note that in the case of the reserved resources, these are only used by the requester and are isolated from other traffic;
- if the network devices along the path can reserve the necessary bandwidth, the originating application can begin transmitting – otherwise, no data is sent.

The main drawback on IntServ is that the network (each router) has to maintain per-flow state and support RSVP, admission control, packet classification based on multiple fields of the IP header, and scheduling. The major difficulty lies in the processing capacity of the routers. The higher the switching capacity of the router (and the higher the speed of the link capacity), the less time the router has to inspect and classify the packets figure 22. In a network with millions of flows, this time may be longer than the switching time of an L3 packet (in the order on ns). This **scalability** issue is the main drawback of IntServ.

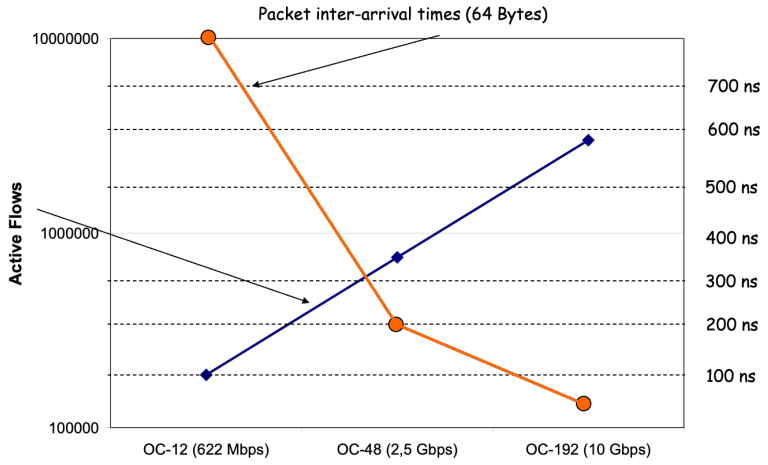

Figure 22: IntServ scalability issues.

If every T seconds, an active flow generates a RSVP message, then the router control plane has to process $L \times N/T$ packets/second, where L is the number of links of the router and N is the number of flows per interface.

Example 4.1 (IntServ scalability issues) *Let us assume a router that has L=16 interfaces, with N= 2 million of packets, and RSVP generates a new reservation every T= 180* s. *Then, every* 5.625 µs *the router control plane has to process a RSVP packet.*

4.3 Differentiated services (DiffServ)

The solution to the scalability problem posed by IntServ is to aggregate traffic (flows) into classes. Classes are identified using 6 bits called **differentiated service code point (DSCP)** (6 bits) in the type of service (ToS) field (now called **differentiated service (8 bits)** field) of the IP header. In this context, service level agreements (SLAs) concern aggregate traffic of the same class and not individual flows. The SLA will specify traffic and QoS characteristics such as token bucket parameters, throughput, delays, losses ratios, priorities, etc (in a per class service).

Example 4.2 (DiffServ class definition) *Let us define a SLA for a class called "VoIP". The Service Provider ISP and the client have the following agreement:*

- *an ingress committed rate (ICR) and a egress committed rate (ECR) are defined. These are the maximum ingress/egress bandwidth for ingress and*

egress traffic for flows in this class. Normally, the traffic will be simetric, so ICR=ECR;

- *as a policy function a Token-Bucket filter will be used with a bucket rate (r) and burst size (B);*
- *in order to have a conforming traffic to the Token-Bucket parameters, the ISP guarantees a maximum delay per each direction (i.e., 15 and 30* ms*), and a loss ratio (i.e., 0.1 %).*

A service is a concept associated with the business of an ISP/operator. The characterisation of the service includes concepts such as performance (losses, delays, etc.), prices, pricing methods, how the service is used, etc. These contracts are long-term (i.e. static) and are not renegotiated when the connection is established. The same service can be offered with different technical solutions (e.g. different per hop behaviour, PHB).

Delay/loss guarantees are obtained through network dimensioning and planning and not through the reservation of resources. The guarantees do not give maximum delay/loss values, but probable values. DiffServ is not an end-to-end architecture. It is defined in a carrier/ISP network. So the market penetration of a specific service will be incremental.

There are two types of nodes, with different responsibilities. Core routers used for intra-connectivity within an administrative domain, and access routers that connect administrative domains to corporate networks or other ISPs.

Access routers has the responsibility of classifying and marking packets in one of the defined forwarding classes. That means that when a packet enters a domain (ISP), it is marked with a DSCP identifier (6 bits), figure 23. Each of the defined classes are called **forwarding class**.

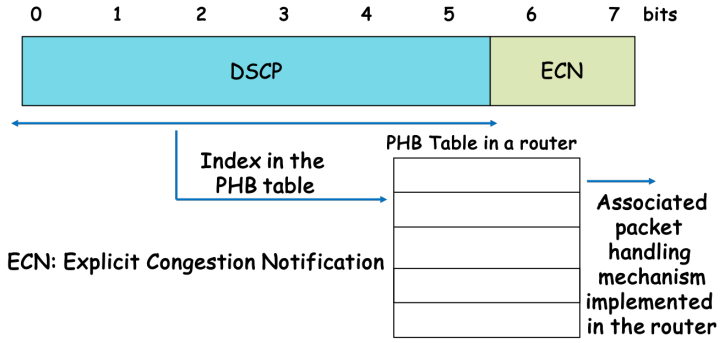

Figure 23: Differentiated service code point (DSCP).

The access router performs the following functionalities, figure 24:

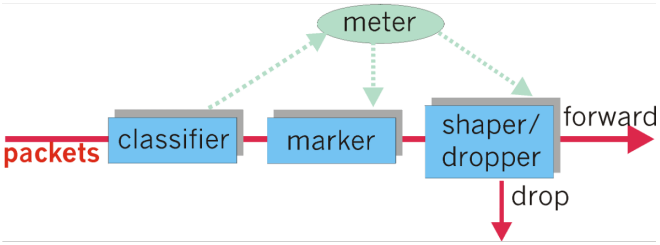

Figure 24: DiffServ access router functionalities.

- **classification:** selects a packet in a traffic stream based on the content of some portion of the packet header;
- **meter:** checks compliance to traffic parameters (i.e., token bucket) and passes results to the marker and shaper/dropper to trigger action for in/out-of-profile packets;
- **marker:** writes/rewrites the DSCP value
- **shaper/droper:** delays (or drops) some packets to be compliant with the profile.

The **core routers**, on the other hand, treat each packet differently depending on the DSCP in the IP header. This treatment is referred to as **per hop behaviour (PHB)**.

Thus, the most expensive functions are implemented at the edge of the network, while in the core network the routers only have to treat the packets differently depending on a small number of classes, as the DSCP field is only 6 bits long (for a maximum of 64 classes). However, the standards recommends four PHB types:

- **Default Forwarding (DF) PH:** the default PHB essentially specifies that a packet marked with a DSCP value of **000000** (recommended) receives the traditional best-effort service from a DS-compliant node (that is, a network node that complies with all of the core DiffServ requirements). Also, if a packet arrives at a DS-compliant node, and the DSCP value is not mapped to any other PHB, the packet will get mapped to the default PHB.
- **Class Selector PHBs:** to preserve backward-compatibility with any IP precedence scheme currently in use on the network, DiffServ has defined a DSCP value in the form xxx**000**, where x is either 0 or 1. These Class-Selector PHBs ensure that DS-compliant nodes can coexist with IP precedence-based nodes (where the first theee bits of the ToS IP header field were used).

Example 4.3 (PBH backward-compatibility) *Let us define packets with a DSCP value of 11000 (the equivalent of the IP precedence-based value of 110) have preferential forwarding treatment (for scheduling, queueing, and so on), as compared to packets with a DSCP value of 100000 (the equivalent of the IP precedence-based value of 100).*

- **Expedited Forwarding (EF) PHB:** dedicated to low-loss, low-latency traffic. The recommended DSCP for EF is 101**110** (value 46).
- **Assured Forwarding (AF) PHB:** gives assurance of delivery under prescribed conditions. The AF PHB defines four AF classes: AF1, AF2, AF3, and AF4. Each class is assigned a specific amount of buffer space and interface bandwidth, according to the SLA with the service provider or policy map. Traffic in the higher class (class 4) is given priority if congestion occurs between classes. Within each AF class, you can specify three **drop precedence (dP)** values, dP=1 for low drop precedence, dP=2 for medium drop precedence, and dP=3 for high drop precedence (where higher precedence means more dropping), figure 25.

Assured Forwarding behavior group

Drop probability	Class 1	Class 2	Class 3	Class 4
Low	AF11 (DSCP 10) 001010	AF21 (DSCP 18) 010010	AF31 (DSCP 26) 011010	AF41 (DSCP 34) 100010
Medium	AF12 (DSCP 12) 001100	AF22 (DSCP 20) 010100	AF32 (DSCP 28) 011100	AF42 (DSCP 36) 100100
High	AF13 (DSCP 14) 001110	AF23 (DSCP 22) 010110	AF33 (DSCP 30) 011110	AF43 (DSCP 38) 100110

Figure 25: DiffServ access AF classes and dropping priorities. Class 4 has the highest priority in the queue.

An AFx class can be denoted by the DSCP value, **xyzab0**, where **xyz** can be 001, 010, 011, or 100, and **ab** represents the dP value.

A good way to remember the AF assurances is to remember AFnm, with n=1, ..., 4 (1=worst, 4=best) queue, and m=1, ..., 4 (1=low, 3=high) dropping priority. Thus AF41 (DSCP=34) is the best queue with best (lowest) dropping priority, and AF13 (DSCP=14) is the worst queue with worst (highest) dropping priority.

Example 4.4 (Expedited Forwarding (EF) PHB) *In instances of network traffic congestion, if packets in a particular AF class (for example, AF1) need to be dropped, packets in the AF1 class will be dropped according to the following guideline:*

$$dP(AFna) \geq dP(AFnb) \geq dP(AFnc) \tag{4.3.1}$$

For example, for class n=1 (AF1), we would have three sub-classes, AF11 (low drop precedence), AF12 (medium drop precedence) and AF13 (high drop precedence). The DSCP values would be: 001010 (AF11), 001100 (AF12)), and 001110 (AF13)).

For class n=2 (AF2), we would have three sub-classes, AF21 (low drop precedence), AF22 (medium drop precedence) and AF23 (high drop precedence). The DSCP values would be: 010010 (AF21), 010100 (AF22)), and 010110 (AF23)), and so on for AF3 and AF4.

Example 4.5 (Precedence compatibility) *To be compatible with the old precedence bits, (first 3 bits of the octect ToS of the IP header, RFC791), DiffServ, defines the following services:*

- **000***00000 DSCP=0 (CS0): compatible with the "best effort" precedence class;*
- **001***00000 DSCP=8 (CS1): compatible with the "priority" precedence class;*
- **010***00000 DSCP=16 (CS2): compatible with the "routine" precedence class;*
- **011***00000 DSCP=24 (CS3): compatible with the "flash" (video or voice signaling) precedence class;*
- **100***00000 DSCP=32 (CS4): compatible with the "flash override" precedence class;*
- **101***00000 DSCP=40 (CS5): compatible with the "critical" (vocie RTP) precedence class;*
- **110***00000 DSCP=48 (CS6): compatible with the "Internet" precedence class;*
- **111***00000 DSCP=56 (CS7): compatible with the "network" precedence class.*

Example 4.6 (DiffServ design) *Traffic classes along with the SLAs for each traffic class in use on the sample DiffServ implementation are described as follows:*

- *Voice is considered premium class. The gold class of traffic consists of TACACS sessions. The silver traffic class consists of SSH, Simple Main Transfer Protocol (SMTP), and FTP sessions. The bronze traffic class consists of web traffic. Anything else is considered as belonging to the "best-effort" traffic class;*

- *The premium class should be forwarded with the lowest delay possible up to a maximum of* 500 kbit/s *during periods of congestion. The gold class should be treated preferentially over the silver class, which in turn should be treated preferentially over the bronze class. The gold, silver, and bronze classes should have 35%, 25%, and 15%, respectively, of the interface bandwidth as the minimum bandwidth guarantees. The bronze class should be shaped to* 320 kbit/s*, and the best-effort class should be policed to* 56 kbit/s*;*

- *To provision for the various traffic classes, the traffic needs to be classified based on DSCP values in a DiffServ domain. So that traffic can be classified based on DSCP values, the traffic should be premarked with the appropriate DSCP values at the time of entering the network.*

An example of DSCP marking would be:

- *Premium (voice): a DSCP=46, that in binary is 000110. This corresponds with a EF (Expedited Forwarding) class;*

- *Gold (TACACS): a DSCP=10, that in binary is 001010. This corresponds with a AF (Assured Forwarding) class of type AF11;*

- *Silver (SSH): a DSCP=18, that in binary is 010010. This corresponds with a AF (Assured Forwarding) class of type AF21;*

- *Silver (SMTP): a DSCP=20, that in binary is 010100. This corresponds with a AF (Assured Forwarding) class of type AF22;*

- *Silver (SSH): a DSCP=22, that in binary is 010110. This corresponds with a AF (Assured Forwarding) class of type AF23;*

- *Bronze (HTTP): a DSCP=26, that in binary is 011010. This corresponds with a AF (Assured Forwarding) class of type AF31;*

- *Best effort: a DSCP=0, that in binary is 000000.*

References

[1] Peter Lars Dordal. An introduction to computer networks. *Loyola University Chicago*, 2014.

[2] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 1(4):397–413, 1993.

[3] Van Jacobson, Kathy Nichols, Kedar Poduri, et al. Red in a different light. *Draft, Cisco Systems*, 1999.

[4] George Ou. Managing broadband networks: a policymaker's guide. *ITIF, December*, 2008.

[5] Abhay K Parekh and Robert G Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM transactions on networking*, 1(3):344–357, 1993.

[6] Abhay K Parekh and Robert G Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM transactions on networking*, 2(2):137–150, 1994.

[7] John Soldatos, Evangelos Vayias, and George Kormentzas. On the building blocks of quality of service in heterogeneous ip networks. *IEEE Communications Surveys & Tutorials*, 7(1):69–88, 2005.