

# 4

## Quality of Service

---

### 4.1 Introduction: What is QoS?

Quality of service, or QoS, has been largely ignored in the initial design of IP networks. IP, like other packet network technologies, was built and optimized to transport data files, not voice or video. In this context, the only ‘quality of service’ that was required was that the data should not be corrupted or get lost. Today the improvement of networking technology makes it feasible to transport real-time data, such as voice or video, over an IP network. Therefore, it becomes extremely important to be able to control and characterize the QoS provided by an IP network.

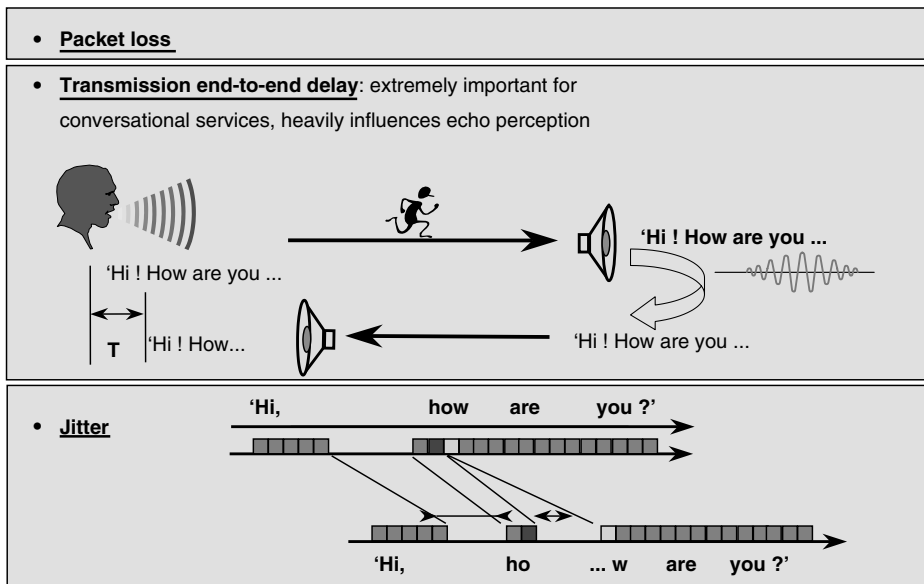
Figure 4.1 summarizes the main parameters characterizing QoS in a packet network:

- The available capacity, also called ‘bandwidth’ in this context (peak, sustained).
- The end-to-end transmission delay (latency) and its variation (jitter).
- Packet loss and desequencing (older packets arriving first).

Bandwidth seems an easy issue to tackle ... just throw more leased line capacity at the problem! In fact, there is more than just providing overall bandwidth: a provider must also ensure that each user of the network gets a fair share of it. This is not a trivial problem, and it is only recently that efficient fair sharing techniques have been deployed.

Latency is by far the most difficult issue of all. A common opinion is to say that IP is simply unsuitable for the transport of latency-controlled data. This is *not* true. Parekh and Gallager found a very useful approach in 1993, leading to a family of queuing algorithms called ‘weighted fair queuing’. These algorithms, although difficult to implement in practice, can guarantee an upper bound on latency for certain flows and enable IP to provide the same guaranteed quality of service as ATM networks.

Jitter is important mainly in real-time applications that need to maintain worst-case buffers to allow for timely delivery of the packets. If there is a lot of jitter, the jitter buffers must be bigger and, therefore, introduce more delays in the end-to-end information path.



**Figure 4.1** Key factors influencing QoS in an IP network.

Packet loss is closely linked to the bandwidth issue and proper use of congestion control at the edge of the network, and within the backbone. Packet loss usually occurs when there is congestion on the packet's path, causing router buffers to overflow. On TCP connections, it will cause a significant drop in the connection's throughput due to the Van Jacobson algorithm. A loss of several packets may switch TCP in slow-start mode and result in a slow connection long after the actual network congestion has stopped. On UDP connections, packet loss will also cause delay (if an acknowledgment and retransmission scheme or a forward error correction method is used) or quality degradation in multimedia applications when loss cannot be recovered due to latency constraints.

Desequencing of packets is mainly influenced by the route stability of the network and efficient queue management in routers with multiple interfaces for a given destination. For most applications desequencing is not a problem *per se* and only causes the same types of delays as jitter.

Telephony is not the only application that poses severe constraints on network QoS: Transaction applications (round trip delays will slow the set-up time of all applications using TCP and may in extreme cases slow the overall transmission speed), interactive applications, such as simulations and games, and some protocol encapsulations, such as SNA, in IP are also very sensitive to network QoS.

## 4.2 Describing a data stream

A data stream, or session, is a sequence of packets that can be grouped logically (e.g., packets coming from a given computer to another computer). In most applications that

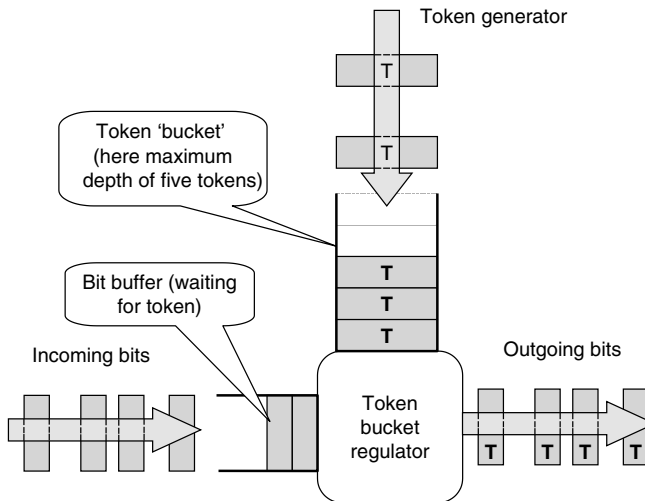
use UDP or TCP packets to communicate between two computers, the packets generated by the application from one computer to the other computer all have the same source IP address and port, and destination IP address and port (software engineers will remark that these elements also characterize a connected ‘socket’). Therefore, these packet properties characterize the group of packets that form the data stream specific to the application session between the two computers.

Protocol, IP address, and port properties can be used to describe the data stream as a group of packets (these properties are often called **filters**), but for QoS engineering it is also important to characterize the timing properties of the sequence of packets: how often a packet is sent, how regularly, etc.

The simplest metaphor that can be used to model a stream is called the fluid model. In this model, packet stream granularity is ignored, just as if it was a continuous stream of bits. A popular fluid model is the **token bucket** regulated stream (or **leaky bucket**) model, as shown in Figure 4.2. The token bucket uses two parameters: the token bucket size  $\sigma$  (in bits) and the incoming traffic long-term average  $\rho$  (in bits/s).

The bits of the incoming traffic must remove a token from the bucket before being forwarded to the output. To regenerate tokens, new ones are created every  $1/\rho$  second until the number of unused tokens stacked in the bucket reaches a depth of  $\sigma$  tokens. When, this limit is reached the bucket is full and the new tokens are rejected. Therefore,  $\sigma$  represents the size of the largest possible traffic bursts. At any point in time the total volume of traffic that gets through the leaky bucket regulator is smaller than  $\sigma + \rho t$ , regardless of the profile of the traffic that has been presented at the input. The original properties of incoming traffic have been ‘shaped’ and can now be represented by the  $\sigma$  and  $\rho$  parameters.

The token bucket model is widely used to represent the timing properties of a data stream. It captures some of the burstiness characteristics of a stream, as well as the stream’s average rate.



**Figure 4.2** The token bucket model.

## 4.3 Queuing techniques for QoS

What causes so much trouble in a packet network is that, at each node, every packet can be received and processed immediately, but it can be forwarded to the next hop only when some capacity is available over the appropriate link. The delay between the reception and the emission of a packet is therefore variable (this variation is called **jitter**): it can be extremely long if the network is congested or if a very long packet is already being emitted on the interface, or very short if the packet always finds available transmission capacity along its forwarding path.

A simple way of reducing delays and jitter for a given packet stream is to prioritize it over all others, but this is not an acceptable solution, as most networks are designed to serve all users equally well. The notion of ‘fairness’ can have many interpretations. The idea is that all flows should be given the same service or a service proportional to their priority, with all flows with the same priority level treated equally. Depending on the exact interpretation of what ‘fairness’ is, several queue management techniques can be used by the nodes of the network. There has been many efforts to design a **scheduling policy** that would minimize transmission delays while still giving each stream a fair share of the available capacity. The scheduling policy decides, on each outgoing queue corresponding to a transmission link, the order of the transmission of packets or eventually their destruction, and seeks to approach for each traffic flow certain goals in terms of capacity, jitter, latency, and packet loss through each node.

Several technologies exist, which can be grouped in two categories. The first category only takes care of packet ordering in the output queues:

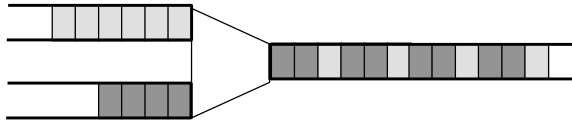
- **FIFO (first in first out)**, also called **first come first served (FCFS)** simply outputs the packets in the order in which they have been received.
- **Class-based queuing** (also called custom queuing by some router vendors).
- **Fair queuing** and **weighted fair queuing** algorithms. Here we will mainly present an algorithm called **PGPS (packet-generalized processor sharing)**, which is the reference fair scheduling algorithm.

These techniques can be combined with any packet loss management technique of the second category:

- Simple overflow.
- **Random early detection (RED)** and **weighted random early detection (WRED)**.

### 4.3.1 Class-based queuing

Class-based queuing (CBQ) sorts data flows into several logical queues according to filtering parameters (e.g., protocol). Arriving packets are sent to one of the appropriate logical queues, and each separate queue works in FIFO mode. Each queue therefore groups a ‘class’ of data streams.



**Figure 4.3** Class-based queuing.

Each queue is usually assigned a priority. A scheduler then picks candidate packets from these logical queues according to the priority of the queue and forwards them to the interface physical transmitter. A weighted round robin scheduler is shown in Figure 4.3. In the example, the scheduler picks two packets (if there are more than two packets waiting to be serviced) in the high-priority queue, then services one packet of the low-priority queue, and this goes on until no packet is left to be serviced.

Class-based queuing is extremely useful. A common configuration is to prioritize UDP (DNS, real-time applications) and interactive (TELNET) traffic. The sorting algorithm can rely on the value of the TOS field (see Section 4.4.1) protocol identifier or sort the packet according to a combination of source address, destination address, and port. It can also be controlled by RSVP (see Section 4.4.2), especially to support the controlled load mode of RSVP.

This is simple and efficient, but does not guarantee any delay to any flow. In addition, since the algorithm is not sensitive to the size of packets, the instantaneous capacity allocated to a given class may vary widely according to the size of packets in the queue at that instant.

#### 4.3.2 Simple fair queuing: bitwise round robin fair queuing algorithm

This algorithm is a refinement of CBQ which takes into account the size of packets. The model of fairness that bitwise round robin fair queuing tries to emulate is a TDM (time division multiplex) link. If each class of flow is allocated a virtual time slot on the TDM link, the service will be equal between all classes. If some priority is needed, then one class could be allocated two or more slots. To emulate TDM on an interface, each queue keeps track of the total received byte count. Initially, all queues start with a byte count of zero, then each arriving packet is assigned a tag with the value of the byte count of the queue just after it arrived. Then the scheduler serves packets in the order of their tags. As described above, TDM emulation will allocate the same capacity share to each flow class. But it is simple to allocate different capacity shares (e.g., if queue 1 needs to be configured to use 50% of the available capacity, queue 2 30% and queue 3 20%, then the tags will not be the byte count, but the byte count divided by 5 for queue 1, divided by 3 for queue 2, and divided by 2 for queue 3).

This TDM model is quite good at allocating different shares of the capacity for each queue in an interface, but it has a major flaw: classes not using capacity accumulate the right to use this capacity later. A flow that has not sent data for a while could potentially send a huge burst and be serviced immediately. During this time, even if other flows have to send data, they will be blocked. In other words, the TDM model does not achieve stream isolation.

It is possible to limit this effect by controlling the maximum amount of data that can get in each queue in any given period (e.g., through a leaky bucket regulator).

### 4.3.3 GPS policy in a node

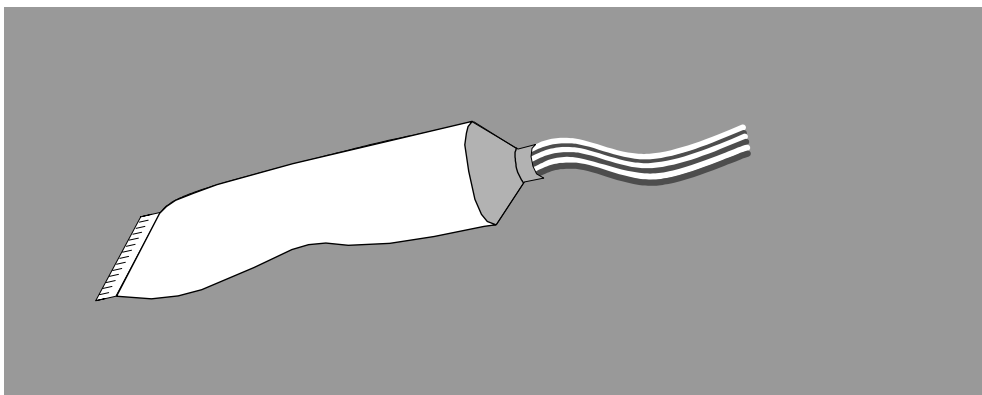
#### 4.3.3.1 Generalized processor sharing (GPS)

**Generalized processor sharing (GPS)** is another view of fairness that is better than TDM. For GPS, the ideal multiplexer node should allocate a share of the available capacity to each stream, proportionally to its priority. However, this share must be *immediately* available as soon as there is some data to be sent, even if other flows happen to be in a burst period at the time. If some flows do not need to send data, then their reserved share of the output capacity is redistributed to other streams proportionally to their respective priorities.

Unfortunately, this policy is only possible if we consider that the data of each stream can be arbitrarily fragmented and that many data elements from different streams can be sent at the same time through the output link of a node. This is called fluid approximation. In the real world of packet data, a packet that is being sent takes all of the bandwidth, even if another packet arrives simultaneously and would need its share of the capacity immediately.

If for a moment we accept that packets can be arbitrarily fragmented, then the best possible multiplexer node looks like a tube of toothpaste (Figure 4.4). If the red toothpaste represents one data stream and the white toothpaste represents another data stream, the output is a mix of the red and white toothpaste, where the “red” stream and the “white” stream each get a fair share of the output.

We can estimate the worst case delay that an element of data belonging to a token bucket-regulated stream (average rate  $\rho$ , maximum burst size  $\sigma$ ) would face going through such a node: it would be  $\sigma/R$ , where  $R$  is the capacity allocated to the stream through the node ( $R > \rho$ ).



**Figure 4.4** An ideal red/white multiplexer.

GPS is a policy where the processing power of the router and the output capacity on each interface are shared among the competing streams. Each stream receives at least a share  $\phi_i$  of the resources ( $\sum_{i=1}^N \phi_i = 1$ ).

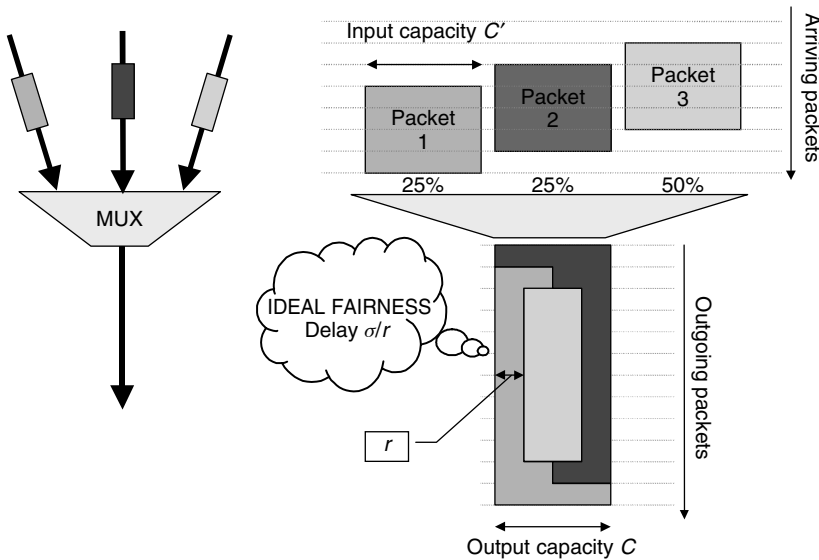
A GPS node is non-idling as long as at least one stream is still queued (which means that the scarce resource, output capacity, is never wasted transmitting nothing). The name of this policy was chosen because there is a good analogy to the sharing of CPU cycles between threads in a multitasking operating system.

In Figure 4.5, when packet 1 arrives, it initially takes all the available output capacity. When packet 2 arrives, it shares the capacity with packet 1, which is still being transmitted. Since both streams have the same priority (25%), the output capacity is shared equally between packet 1 and packet 2. When packet 3 arrives, all packets compete for the output, and the GPS multiplexer node allocates half of the capacity to packet 3 (stream 3 has a priority of 50%) and the rest equally between packet 2 and 3. At some point the transmission of packet 3 is complete, and since there are no other packets from stream 3 in the queue the output capacity is shared again between packets 1 and 2. Note that the order in which the transmission of packets completes is not the same as the order of packet arrival, due to the higher priority of stream 3.

More precisely, if  $S_i(s, t)$  denotes the volume of a stream that has gone through the node between instant  $s$  and  $t$ , then:

$$\frac{S_i(s, t)}{S_j(s, t)} \geq \frac{\phi_i}{\phi_j}$$

for each session  $i$  continuously backlogged between  $s$  and  $t$ . The backlogged active sessions share the resources of inactive sessions (not backlogged) proportionally to their  $\phi_i$ .



**Figure 4.5** Handling three packets using a 'fluid' multiplexer.

In other words, each session  $i$  can use at least the capacity  $\phi_i * C$  at any time, where  $C$  is the total capacity of the output interface considered. If *all* sessions are active, then each session  $i$  uses exactly  $\phi_i * C$ .

The service received by the token bucket-regulated session  $i$  (average rate  $\rho$ , maximum burst size  $\sigma$ ) in a system where  $N$  token bucket-regulated sessions share the scarce resource is always better than the service received by session  $i$  if all other sessions start with their token bucket full, then send the longest allowed traffic burst, and finally keep sending data at the maximum long-term average rate allowed by the token bucket regulator. With this remark, we can calculate:

- The largest delay through the node  $\sigma/\phi_i * C$ .
- The buffer size needed for the worst backlog  $\sigma$ .

GPS policy also has a very important property: the relative order of departure (i.e., last bit of the packet has been output) of two packets  $i$  and  $j$  is independent of future packet arrivals. The reason for this is that if another packet arrives, the transmission speed of  $i$  and  $j$  is changed homogenously (the factor of change is the same for  $i$  and  $j$ ), which preserves the departure time. The exact time of departure of  $i$  and  $j$ , though, is obviously changed.

### 4.3.3.2 PGPS policy in a node

#### 4.3.3.2.1 The PGPS approximation

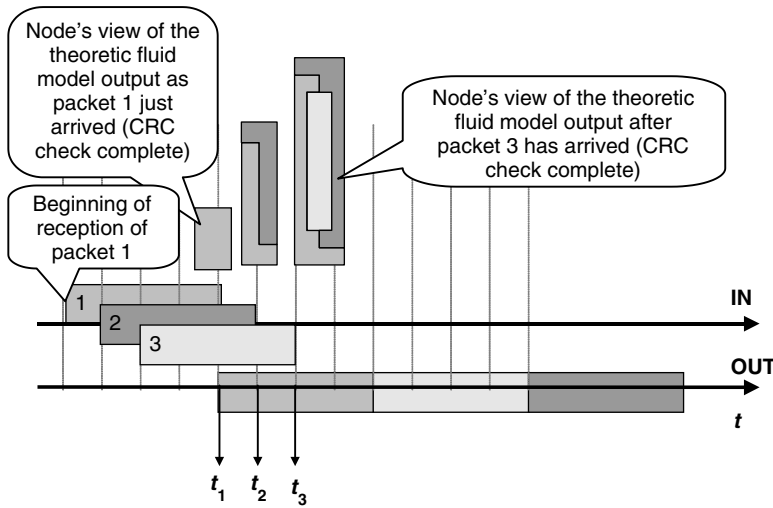
The fluid model used in GPS is not valid for real-world networks, where packets are received as whole entities (i.e., nothing is processed until the final packet CRC is checked) and put in the output queue as blocks of contiguous bits. Taking into account these real-life facts leads to ‘packet by packet GPS’. The idea behind PGPS is to serve the packets in the order in which they would leave under a GPS policy (see Figure 4.6). This order of departure is not changed by future arrivals, so packets that have already been ordered keep the same order, only newly arrived packets may be inserted in this arrangement.

Figure 4.7 is an illustration (PGPS left, GPS right) of the case when the transmission capacity of each input and the output link is 1. The two leftmost streams are given a weight of 0.25, whereas the rightmost stream is given a weight of 0.5. A packet arrives for each stream every  $\frac{1}{4}$  time unit in our case (we take the transmission time of a packet of size 1 to be the time unit):  $a_i$  is packet  $i$ ’s arrival time. In case you wonder why packet 1 is sent first while it only finishes second under GPS, this is because at the time the PGPS node had to choose a packet for output (remember it is non-idling), packet 3 had not yet arrived and so its departure time was not known.

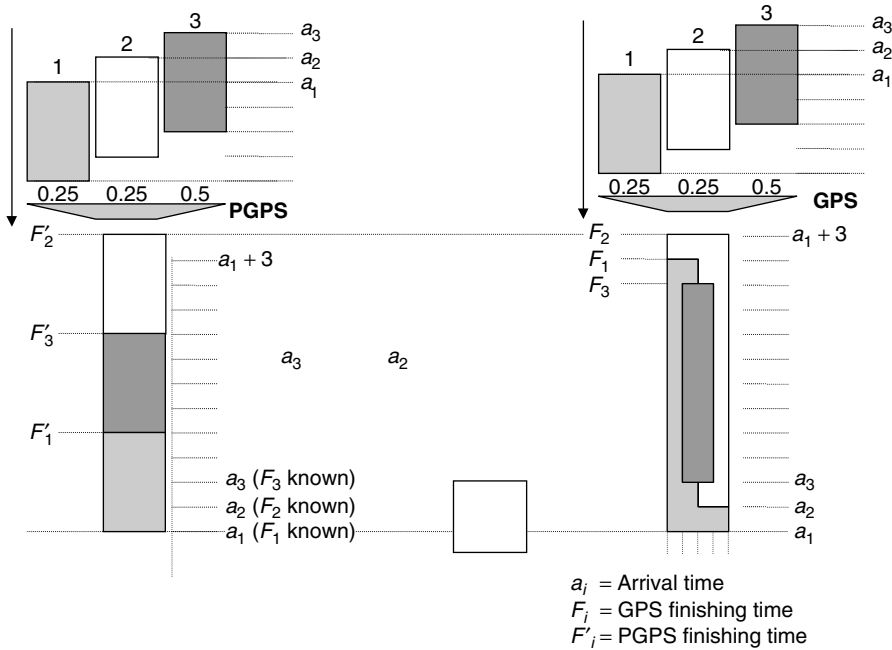
#### 4.3.3.2.2 Assessment of the PGPS approximation

Under PGPS, packets are sent in the order of departure as known when the decision to select a new packet for the output queue is taken. Because we cannot guess what the future will be, this might lead to errors as in Figures 4.6 and 4.7: a new packet (packet 3, Figure 4.6, time  $t_3$ ) arrives just after the selection of the new output packet (packet 1,





**Figure 4.6** Handling three packets using PGPS, according to instantaneous ‘fluid model’ -projected completion order. The router calculates in real time the way packets would be handled under fluid approximation and GPS, in order to find in which order they would finish.



**Figure 4.7** PGPS and GPS packet ordering is not always identical.

which arrived in Figure 4.6 at time  $t_1$ ) and the GPS theoretical departure time  $F_p$  of packet 3 is lower than the GPS departure time of packet 1 currently being sent. The PGPS scheduler has no other choice than waiting until this one finishes. Because a PGPS multiplexer cannot guess the future, in some cases the rule ‘PGPS sends packets in the same order as the GPS finishing order’ is not followed.

Because of this, under the PGPS policy, some packets will have their departure time later than under the ideal GPS policy. Imagine, for instance, a high-priority packet arriving in the multiplexer queue directly after a low-priority packet has been sent to the output transmission line. There is an upper bound to this added delay:

$$F'_p - F_p \leq \frac{L_{\max}}{C}$$

where  $L_{\max}$  is the maximum size of a packet and  $C$  the throughput of the outgoing interface. This limit is approached in the example if we grow stream 3 weight to nearly 1 and we make packets 1, 2, and 3 arrive closer.

The worst delay through the node for stream  $i$  under PGPS becomes ( $L_{\max}$  is the maximum packet size on inbound links, which must be smaller than the **maximum transmission unit, or MTU**):

$$D_i^* \leq \frac{\sigma_i}{r_i} + \frac{L_{\max}}{C} + Tr$$

where  $r_i = C * \phi_i$  is the capacity reserved for stream  $i$ ,  $C$  is the total capacity of the output link, and  $Tr$  is the duration of processing in the router. This result is illustrated in Figure 4.8.

The queue buffer necessary to prevent any overflow for stream  $i$  becomes a bit larger than in GPS because packets may wait longer:

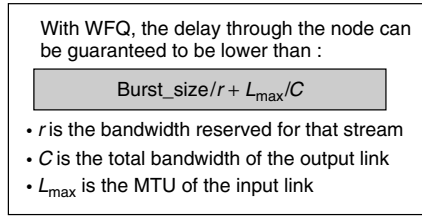
$$Q_i^* \leq \sigma_i + L_{\max}$$

under the stability condition  $\rho_i \leq r_i$ .

It is obvious from these formulas that it is better not to have big packets, which was one of the design goals of ATM. However, as output link capacity increases, this becomes decreasingly important. With current transmission technology in excess of 1 Mbps at the edge (xDSL) and multigigabit in core networks, this explains why ATM is quickly getting displaced by IP.

As proof of this, let us number the packets in the order they are processed by the PGPS server since the last busy session. For any packet  $p_k$  there are two cases:

- All packets  $p_j$  leaving PGPS before  $p_k$  ( $j < k$ ) also leave the GPS server before  $p_k$ . Therefore, at the time  $f_k$  when packet  $p_k$  finishes under GPS, GPS has already served all other packets  $p_j$  ( $j < k$ ). Because PGPS is work-conserving it will have served the same volume as the GPS server between the beginning of the busy session and  $f_k$ . All packets  $p_j$  ( $j \leq k$ ) fit in this volume because GPS has served them, so PGPS has also served them before  $f_k$ , so  $f'_k \leq f_k$ .  
At least one packet  $p_m$  leaving PGPS before  $p_k$  ( $m < k$ ) leaves after packet  $p_k$  under GPS. Among those packets, let us consider the one leaving PGPS last:  $p_M$ . If  $a_i$  denotes



**Figure 4.8** Delay guaranteed by a single PGPS node.

the arrival time of packet  $i$ , then we have:

$$\forall_i \in [M + 1, k], \quad a_i > f'_{M-1}$$

otherwise, if packet  $i$  had arrived before  $f'_{M-1}$  (this is when PGPS must choose which packet will go after  $p_{M-1}$ ), because packet  $i$  finishes before packet  $M$  under GPS (otherwise  $M$  would not be the biggest integer lower than  $k$  having  $f_M > f_k$ )  $p_i$  would have been scheduled next, not  $p_M$ . So we now have that no packet  $p_i$ ,  $i$  between  $M + 1$  and  $k$ , had arrived before  $f_{M-1}$ . But, they have all been served before  $f_k$  (otherwise  $M$  would not be the biggest integer lower than  $k$  having  $f_M > f_k$ ). So we can write:

$$f_k > f'_{M-1} + \sum_{i=M+1}^k \frac{L_i}{C}$$

where  $L$  is the size of the packet and  $C$  the bandwidth of the output. Under PGPS we have exactly:

$$f'_k = f'_{M-1} + \sum_{i=M}^k \frac{L_i}{C}$$

which finally gives:

$$f_k > f'_k - \frac{L_M}{C}$$

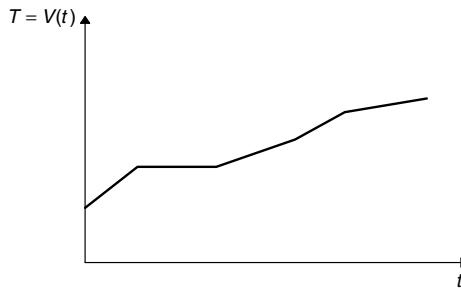
and because  $L_M \leq L_{\max}$  we have our result.

#### 4.3.3.2.3 Computing PGPS packet ordering

A packet has arrived when its last bit has arrived. We call  $a_i^k$  the moment of the arrival time for the  $k$ th packet arriving from flow  $I$ , and the length of the packet is  $L_i^k$ . Let  $s_i^k$  and  $f_i^k$  denote the moment at which packet  $k$  begins and finishes to be processed by the GPS server. Then:

(a)  $s_i^k = \max\{f_i^{k-1}, a_i^k\}$ .

(b)  $f_i^k = s_i^k + t(L_i^k)$ , where  $t$  is the time used by the GPS server to process  $L$  bits.



**Figure 4.9** The  $v(t)$  function.

The difficult part is that the processing speed of the GPS server depends on its load and changes with each packet departure and arrival (called an *event*). If  $t_i$  denotes the time of event  $i$ , the processing speed for session  $i$  is:

$$\frac{\phi_i}{\sum_{j \in B(t)} \phi_j} * r$$

where  $r$  is the total throughput of the outgoing link and  $B$  is the group of buffered sessions at this time. This reflects the fact that a GPS server redistributes unused reserved bandwidth to active sessions according to their precedence.

We call  $v(t)$  the piecewise linear function of  $t$  defined by its slope  $\frac{r}{\sum_{j \in B(t)} \phi_j}$  (Figure 4.9).

We can rewrite (b) as:

$$(b) \int_{s_i^k}^{f_i^k} \phi_i * v'(t) dt = L_i^k.$$

Writing  $S = v(s)$  and  $F = v(f)$ , (a) and (b) become:

$$(a) S_i^k = \max\{F_i^{k-1}, V(a_i^k)\}$$

$$(b) F_i^k = S_i^k + \frac{L_i^k}{\phi_i} \text{ with } F_i^0 = 0, \forall i.$$

This virtual time  $T$  respects the same order relations as  $t$  because function  $v(t)$  is nondecreasing. In order to build a PGPS multiplexer, for each packet it is enough to calculate  $F$ , which is possible as soon as we receive the packet since  $a$  and  $L$  are known.

With this algorithm, we can immediately classify a new packet among the queued packets according to PGPS scheduling. However, this calculation requires that the PGPS multiplexer maintains the parameters necessary for the calculation of  $v(t)$ : the coordinates of the last slope change, the current slope, and the number of backlogged sessions. Since each arrival and departure will change these values and packets can arrive simultaneously, this requires significant processing power on high-capacity links.

#### 4.3.3.2.4 PGPS multiplexers in a network

Along a path going through  $N$  PGPS multiplexers, the maximal end-to-end delay for a token bucket-regulated data flow is:

$$D_i^* \leq \frac{\sigma_i + (N - 1)L_{\max}}{r_i} + \sum_{n=1}^N \left( \frac{L_{\max}}{C_n} + Tr_n \right)$$

where  $r_i$  is the smallest bandwidth amount allocated to stream  $i$  along the path. Both the propagation time and the processing time should be included in term  $Tr$ . The first term shows that the burstiness of the data flow increases through each PGPS multiplexer, due to possible data accumulation while waiting to be served.

The formula is complex, but it is interesting to note that the first term decreases as the reserved bandwidth increases (see Figure 4.10). The guaranteed delay through a set of PGPS multiplexers can be reduced by increasing the reserved bandwidth beyond the average bitrate of the data stream. This is the key result used by RSVP in guaranteed service mode. Now if the stream goes through several WFQ nodes, the end-to-end delay will be lower than shown in Figure 4.10.

There is also a more accurate version of the equation in Figure 4.10. Let us now consider the peak emission rate  $p_i$  of stream  $i$ :

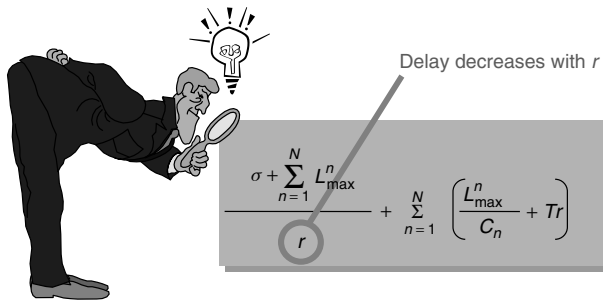
$$D_i^* \leq \frac{\sum_i C_{\text{tot}}}{r_i} + D_{\text{tot}}$$

where:

$$C_{\text{tot}} = \sum_{n=1}^N L_{\max}^n$$

$$D_{\text{tot}} = \sum_{n=1}^N \left( \frac{L_{\max}^n}{C_n} + Tr_n \right)$$

$$\sum_i = L + \frac{(\sigma_i - L)(p_i - r_i)}{(p_i - \rho_i)}$$



**Figure 4.10** End-to-end delay through multiple PGPS nodes.

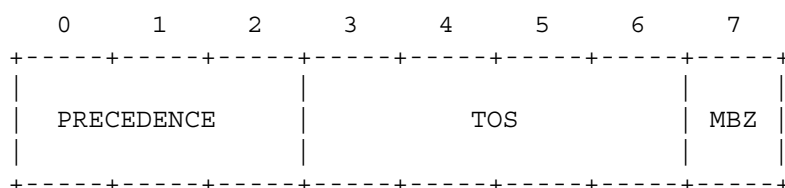
RSVP gives a data flow receiver all the parameters in this formula. The receiver then selects the rate  $r$  that he wants to reserve in order to have an acceptable, guaranteed end-to-end delay.

## 4.4 Signaling QoS requirements

### 4.4.1 The IP TOS octet

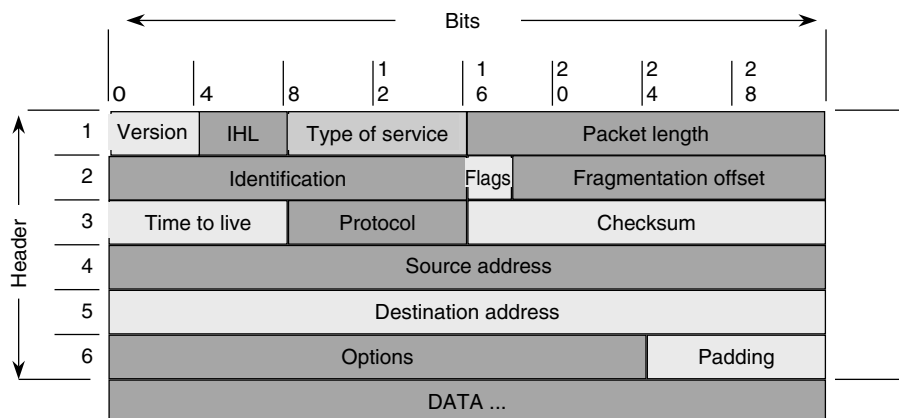
The IPv4 packet **type of service (TOS)** octet, shown in Figure 4.11, captures the parameters describing how this packet should be handled relative to quality of service. IPv6 has a similar octet called the **traffic-class** octet.

The IPv4 TOS octet was traditionally structured as follows:



The IP precedence field uses the first 3 bits, encoding a value between 0 and 7. Packets with a higher IP precedence value should have a higher priority in the network. The traditional meaning of the IP precedence values (RFC 791) values is described in Table 4.1. The vocabulary used reflects the military origin of IP, a ‘flash’ IP packet was supposed to be the electronic equivalent of a flash message.

The following 4 bits form the TOS field and were supposed to be used to mark a desired trade-off between cost, delay, throughput, and reliability, as described in RFC



**Figure 4.11** The IPv4 header.

**Table 4.1** Precedence values in the original RFC 791

Value	Definition	
Network	Match packets with network control precedence	(7)
Internet	Match packets with inter-network control precedence	(6)
Critical	Match packets with critical precedence	(5)
Flash override	Match packets with flash override precedence	(4)
Flash	Match packets with flash precedence	(3)
Immediate	Match packets with immediate precedence	(2)
Priority	Match packets with priority precedence	(1)
Routine	Match packets with routine precedence	(0)

*Note:* Levels 110 (6) and 111 (7) are reserved for routing and some ICMP messages (see RFC 1812 for details). Therefore, only six levels (routine–critical) remain for user applications.

1349. Originally, RFC 791 used only the first 3 bits as the TOS field, and the last 2 bits were part of the MBZ field. RFC 1349 defined some extended values for 4 bits of the TOS field:

- 1000—minimize delay.
- 0100—maximize throughput.
- 0010—maximize reliability.
- 0001—minimize monetary cost.
- 0000—normal service.

The idea is that interactive services like TELNET should require TOS 1000. In the original RFC 971 it was legal to add those values to the required combined properties. RFC 1349 no longer allows this, and value 1100, for instance, could mean anything. RFC 1349 requires the last bit (MBZ) to be 0: the MBZ (‘must be zero’) field is for experimental use and is ignored by routers. RFC 1122 and 1123 (Host requirements) also defined a few rules for setting TOS values for hosts, but they were based on a 5- bit TOS field.

If you are beginning to think that things are not crystal clear, you are right. Things are not clear indeed. When sending traffic over the Internet with a specific IP precedence and TOS value, no one can be really sure of the behavior of routers along the path, unless only one provider was in control of the whole domain and had properly configured the forwarding policies of all routers.

#### **4.4.1.1 Using the IP precedence field**

The most straightforward use of the IP precedence field is to cause interface schedulers to prioritize marked packets. Beyond this, the IP precedence field can be used at the network access level in conjunction with policy routing and priority routing. Packets primarily need to be marked with the proper TOS field. If the sender of the packet does not do it directly, the TOS field can be set by a router: many routers can overwrite the TOS field based on certain filtering criteria.

For example, a router can be configured to set the IP precedence field of TCP traffic on port 80 to critical (this example is for a Cisco router):

```

interface Serial0                                /* interface facing the customer router
ip address 10.0.0.1 255.0.0.0
ip policy route-map test                        /* this activates policy routing on interface
                                                Serial 0
interface Serial1                                /* interface to low-latency backbone
ip address 192.168.1.1 255.255.255.0
interface Serial2                                /* interface to normal latency backbone
ip address 192.168.2.1 255.255.255.0
access-list 101 permit udp any any gt          /* this simple filter will match
1023
                                                RTP traffic (but not only)
route-map test permit 2                        /* defines the default path
set default int serial2
route-map test permit 1                        /* Defines route map 'test' 1
match ip address 101                          /* all ip addresses that pass filter 101
set ip next-hop 192.168.1.5                    /* will go through interface serial1
set ip precedence critical                     /* set TOS field to critical for all traffic
                                                matching access list 101

```

Once this is configured in the customer's access router, the network provider has two options:

- Prioritize the IP traffic in the backbone according to the value of the TOS field. This can be done using priority queuing, class-based queuing or weighted fair queuing.
- Use a separate path in the network (e.g., avoiding satellite links) for IP traffic with critical priority. This ability to bypass the regular routing mechanism to set custom next hops or custom IP precedence to packets is called policy routing. On Cisco routers these features are also enabled by the 'route map' set of commands. In the example the low-latency network can be reached through interface serial1.

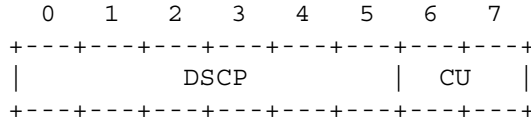
#### **4.4.1.2 (Re)defining the values of the IP TOS octet**

There has been much theoretical work on the behavior of packet networks since the creation of the IP, and people have much more experience on the sophisticated queuing mechanisms used in QoS-enabled equipment. The work done for ATM and frame relay networks has also helped us to get clearer ideas on the QoS-related information that needs to be transported in each packet.

It was high time to review the original meaning(s) of the IP TOS octet and stop wasting 8 bits per IP packet. This revision work became the charter of the IETF DiffServ group, with the intent to 'provide scalable service discrimination in the Internet without the need for per-flow state and signaling at every hop.' The Diffserv group redefined the semantics



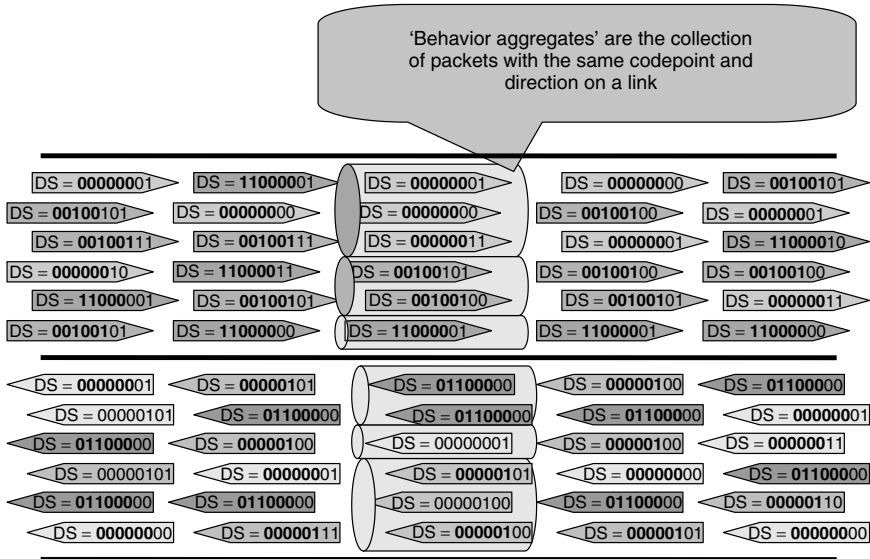
of the IPv4 TOS octet and the IPv6 traffic-class octet in RFC 2474 (December 1998) which made both RFC 1455 and 1349 obsolete. The name TOS itself was changed; this byte is now called the **DS**, or **differentiated services**, byte. The DS byte is subdivided into a 6-bit **DSCP (differentiated services codepoint)** field and a CU (currently unused) field:



The codepoint value should be used as an index to the appropriate packet handler, or **per-hop behavior (PHB)**. A PHB applies a particular forwarding treatment to all packets with a particular DSCP field and direction. This class of packets is called a **behavior aggregate** (Figure 4.12). For instance, each behavior aggregate can be mapped to a particular queue in a weighted round robin CBQ or WFQ scheduler.

The index is based on an exact match on the 6 bits of the DSCP (the 2 CU bits being ignored). Each specified PHB should be assigned a unique default DSCP field among the 64 that could potentially be available with 6 bits. In fact, RFC 2474 has allocated three pools of codepoints:

- xxxxx0 for 'standard actions'. Eight codepoints (yyy000), called **class selector codepoints**, are already allocated for backward compatibility with the IP precedence field of RFC 791. RFC 2474 states that the set of PHBs mapped to these codepoints must offer at least two independent queues, expedite forwarding according to yyy values (the



**Figure 4.12** Behavior aggregates.

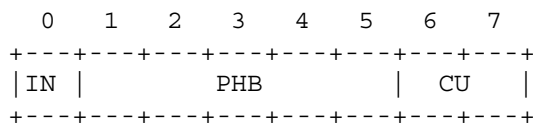
higher the *yyy*, the lower the average queuing delay), and prioritize *yyy* = 110 and 111 (routing traffic) over *yyy* = 000 (best effort traffic). Note that this set of requirements does not imply the use of one particular scheduling algorithm (WFQ, CBQ, or priority queuing could be used). This philosophy will be retained when defining other PHBs.

- xxxx11 for experimental or local use.
- xxxx01 for experimental or local use, or extension of the standard actions pool if it gets fully allocated.

Even if each PHB is allocated a default codepoint, Diffserv nodes are free to assign other codepoints to a particular PHB (except for xxx000 codepoints). In fact, DSCP fields may have a meaning that is only local to a domain. At the boundary of such DiffServ domains, it is very important to control the forwarding of IP packets according to their DSCP fields and, if necessary, to map some codepoints to other values. The configuration of PHB-to-DSCP field mapping is an administrative decision that may vary from domain to domain. Two service providers with service-level agreements must either agree on DSCP field values for each PHB or properly configure DSCP field translation at boundary nodes.

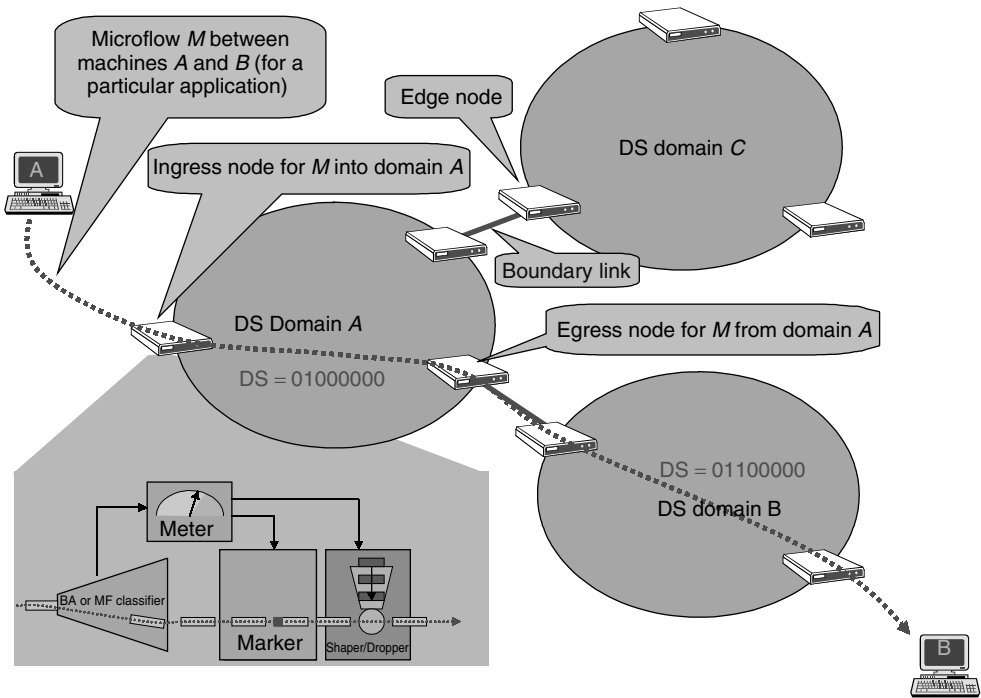
All packets with unknown codepoints (not part of the service-level agreement of a service provider) are assumed to be part of the best effort forwarding behavior. This PHB must always be present in a DS-compliant node. The default codepoint for the best effort PHB is 000000, and this value must be recognized by all DS-compliant nodes. So far only class selector PHBs have been defined (in fact, this was mostly a mapping of the IP precedence semantics of RFC 791), but in the future there could be a 'strict priority queuing PHB' with a different set of requirements.

In order to avoid mixing widely different traffic types into a single queue, providers of DiffServ networks are expected to perform some traffic shaping/regulation at the edge of the network (e.g., with a token bucket). Occasionally, there will be flows exceeding regulator settings. It is not always a good idea to immediately mark such out-of-profile packets for best effort forwarding, since this practice can introduce unnecessary desequencing. Maybe the network still has enough capacity to carry the excess packet in sequence; therefore, the best solution is to mark this packet as being 'out of profile'. A node will forward this packet as if it was 'in profile' if there is enough capacity locally or discard it otherwise (or mark it for best effort forwarding). The solution currently recommended in RFC 2475 for marking in-profile and out-of-profile packets is to use two codepoints. In a previous proposal, the first bit was allocated to mark an out-of-profile packet:



where 'IN' represents in (1) or out (0) of profile. The CU field has yet to be allocated, but could be used for forward/backward congestion notification. This has been very useful in frame relay networks.

A more complete description of the differentiated services architecture has been published in RFC 2475 (December 1998). This RFC mainly introduces a specific DiffServ



**Figure 4.13** The DiffServ ecosystem. The ingress node implements the traffic-conditioning agreement (TCA) used by the provider's service level agreement (SLA).

vocabulary (some of which appears in Figure 4.13), discusses the DiffServ paradigm, and compares it with other architectures supporting service-level agreements.

### 4.4.1.3 Remaining issues with IP TOS/DS octet

#### 4.4.1.3.1 Beware of layer 2!

It is important to remember that IP is used as a layer 3 protocol only and, therefore, any layer 2 multiplexer can potentially ruin the IP-level quality of service. Here are a few common examples:

- *Ethernet:* most PCs and IP phones are connected to Ethernet LANs. While shared coax cable LANs have almost disappeared, most corporations still have Ethernet LANs based on Ethernet hubs. Since hubs typically broadcast any received frame to all connected Ethernet segments, such networks frequently have a high percentage of packets lost due to Ethernet-level collisions. VoIP should never be deployed on such networks. At a minimum, VoIP should be deployed on switched LANs (switches maintain a cache of MAC Ethernet addresses connected on each link and therefore forward Ethernet

frames only to the proper link, reducing unnecessary broadcast traffic). In many cases, however, this is still insufficient if Ethernet concentration links are used where Ethernet frames can get discarded with a probability of over 1% (this can happen even on moderately loaded networks, due to the bursty nature of data communications). A possible enhancement is to use two separate LANs: one for bursty data traffic, one for IP telephony (switched LAN). On recent IP phones and Ethernet switches, the IEEE 802.1Q **VLAN** standard allows network architects to emulate two distinct Ethernet LANs on a single physical network (a 12-bit VLAN tag is appended to the MAC address, as well as 3 bits whose usage is defined in 802.1P). Many IP phones and switches also support the 802.1P QoS standard, which offers eight levels of QoS on an Ethernet LAN: these levels are offered by managing the 3 QoS bits and, in case of collision or congestion, lower precedence frames get discarded. This strict priority-based QoS is sufficient on LANs, because latency is always very low. IP precedence DSCP fields map easily to these eight-layer two-precedence levels.

- *ADSL*: when service providers began deploying ATM DSLAMs (ADSL access concentrators) around 1995, ATM was still regarded by many as the technology of the future for multi-service applications. The Internet and IP technology were still considered a toy. As a consequence, most service providers did not anticipate that *all* future multimedia applications would run exclusively on IP and planned their support for QoS only for native ATM applications. ATM has indeed an extensive support for QoS, based on a connection-per-connection negotiation that is part of the connection set-up process. Unfortunately, service providers quickly discovered that dynamic connection establishment did not scale to large networks, and most ADSL offers are built on ADSL CPEs connected to the core ATM network through a single ATM permanent virtual connection. All IP traffic is therefore routed over that single channel and all IP packets, high priority or not, are sent as fragmented cells. When congestion occurs (e.g., on the ATM concentration links from the DSLAM to the backbone), cells are dropped at random. This has a very negative impact on IP traffic: it is very inefficient, as a single lost cell will prevent the destination router from correctly reconstructing the IP packet, but all the other now-useless cells will still be conveyed by the ATM network. It also ignores IP precedence completely, and high-priority IP traffic is impacted with the same probability as the low-priority IP traffic. This problem has become the hottest issue of many ADSL service providers. All newer DSLAMs are now IP-aware and can properly discard whole packets and prioritize them according to precedence fields. For networks based on older DSLAMs, there can only be work-arounds (e.g., oversizing the ATM network to prevent cell loss) which create two ATM connections per ADSL CPE (one for high-precedence traffic, one for best effort) or use of the CPL ATM cell bit (cell loss priority).
- *Frame relay*: some international backbones still rely on frame relay networks for IP transport. Many of these frame relay networks are still relatively old and were optimized for data only. The only important quality-of-service parameter for such data networks was frame loss; in order to reduce it to the minimum, large buffers have been configured in each frame relay node, thus preventing overflows due to data bursts. Unfortunately, this also creates large, uncontrollable delays (and jitter) and IP packets transported on such layer 2 networks cannot transport voice.

- *Wireless links*: the 802.11 family of wireless standards is very popular. However, at present most vendors do not implement any quality-of-service mechanism that would enable VoIP packets to have precedence over best effort data. The 802.11e prioritization standard is required for any serious implementation of VoIP and data over 802.11 links. Until then, wireless networks can only be used by VoIP if they are dedicated to voice only.

#### 4.4.1.3.2 *Number of traffic classes*

Sorting flows based on the old IP precedence value limits the number of queuing behaviors to eight, of which six are available to end-user applications. This can be further refined by using packet filters based on the protocol number (e.g., to prioritize UDP over TCP) or destination/source addresses and ports.

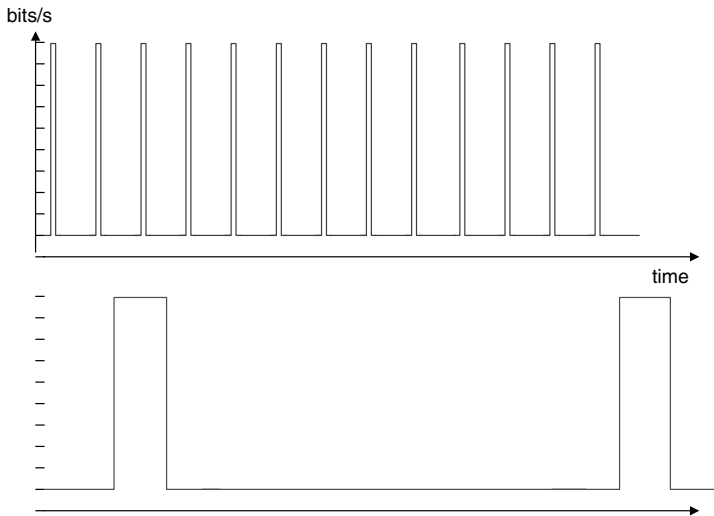
Offering six classes of service to the end-user may seem enough, when thinking only in terms of broad ‘classes’ that should be prioritized, because it is hard to think of more than six very distinct and useful behaviors. However, this is valid only if all sorts of traffic classes that require a specific forwarding behavior can be grouped in the same queues. Unfortunately, very bursty traffic and smooth traffic should not be mixed in the same queues, as this might degrade the properties of smooth traffic (e.g., voice). This requires a traffic-class value for each combination of desired per-hop behavior and category of ‘burstiness’. In addition, as we have seen above, it is useful to mark out-of-profile traffic at the edge of the network, which really requires two identifiers for each traffic class.

The more recent DiffServ framework makes things potentially much better, since up to 32 packet-handling algorithms could be indexed (with the possibility of marking out-of-profile packets, which uses two codepoints for each traffic class). For current applications, this new framework seems to provide enough traffic classes.

#### 4.4.1.3.3 *Identifying data flows that should be mapped to traffic classes*

In an ideal world, all applications would ‘know’ the DSCP codepoint to use when sending IP packets, and no one would try to cheat by using inappropriate codepoints. Unfortunately, in real networks it is frequently necessary to either set or verify the DSCP fields before injecting the packet into the network. Packets that should use a given codepoint can be recognized using filters based on the packet protocol, IP address, etc. However, some application sessions are impossible to prioritize using static filters (e.g., all applications that use a dynamic port negotiation, such as SIP or H.323). If the router has no proxy capability for the application, it has no way of knowing which port to prioritize. The only possibility is to prioritize an entire range of ports or all packets originated from the host. Obviously, in many cases this is not enough. In a shared commercial backbone, this also creates potential security issues: since the prioritization mechanism uses static filters, a devious user can decide to design an application that ‘looks like’ an authorized application but uses many more resources (e.g., if the provider prioritizes UDP in order to speed up small DNS queries, videoconferencing users will also benefit from it, while they obviously use many more resources).

As we will see, RSVP provides a much more powerful solution to negotiate certain QoS levels for a given data stream dynamically. RSVP can also be extended to include



**Figure 4.14** Sessions with same average rate and peak rate, but different burst size.

authentication mechanisms and, therefore, can secure access to the backbone resources for prioritized traffic.

#### 4.4.1.3.4 *Network dimensioning and pricing*

For network dimensioning, it is very useful to know the characteristics of the data streams that are being multiplexed. For instance, let us compare the multiplexing of the sessions illustrated in Figure 4.14:

- Flows having an average rate of 20 kbit/s, a maximum burst of 1 kbit at a constant rate of 100 kbit/s, and a minimum constant bitrate of 10 kbit/s.
- Flows having an average rate of 20 kbit/s, a maximum burst of 100 kbit at a constant rate of 100 kbit/s, and a minimum constant bitrate of 10 kbit/s.

In both cases it is easy to calculate that the high bitrate occurs one-tenth of the time and the low bitrate occurs nine-tenths of the time. But, for the first type of flow the bursts are very short (1/100 s), while in the second case the bursts last 1 s. So, in the first case the provider will be able to fit 1,000 flows in just a little more than  $1,000 \times 20$  kbit/s (20 Mbit/s): the excess traffic during bursts will accumulate in small router buffers, and the resulting delay will not be too large. But, in the second case the bursts last much longer: the required buffers would be too large for a 20-Mbit/s link and the resulting delays unacceptable. So the provider needs to provide significantly more than  $1,000 \times 20$  kbit/s in order to keep the buffer size low in routers.

In general, bursty traffic is much more expensive to carry than smooth traffic. In the next generation IP networks providing QoS, the provider will probably try to isolate these flows and apply a special pricing (such a pricing could be hidden in a 'right to use' the application generating such data streams). In order to identify such flows, it would be useful to have a description of the characteristics of each data stream that a customer

sends in the backbone. Based on this description, the provider can decide which streams can be grouped and how expensive it is to carry them. The TOS octet value alone does not describe the traffic characteristics, and there is no easy way for the provider to sort similar streams together and have higher tariffs for more bursty streams.

We will see that RSVP provides the network with many parameters that characterize the properties of the data stream, which helps each router to decide in which queue the traffic should be sent and eventually to choose a tariff.

#### **4.4.1.3.5 *Programming applications that require QoS***

The internal backbone of a provider will have well-defined TOS values/DS codepoints for each class of service. But different providers will probably use different values for the same class. Even with the current DiffServ RFCs only the relative behaviors of class selector PHBs are defined, but the quality of a particular codepoint could vary widely when changing providers. When designing a DiffServ-aware application, the programmer cannot know in advance which codepoint value must be used and, therefore, the application will probably need some configuration. The average user will have no means of deciding which codepoint is appropriate for each application: What are the implications for delay? Can this application recover from packet loss? Is it sensitive to jitter?

It is much easier for the programmer to be able to ask the network what it needs in terms of bandwidth and delay, and let the network provide the required QoS. This requires some signaling mechanism between the applications and the network, and this is where RSVP can play a key role.

### **4.4.2 RSVP**

#### **4.4.2.1 *RSVP is an enabler of a business-grade Internet***

Many people used to consider IP networks would never need sophisticated QoS mechanisms, because the ever-increasing capacity of backbones would always push back the time when QoS would really be useful. ‘Peer-to-peer’ applications have demonstrated that this assumption was wrong. From the point of view of service providers these applications can be considered as a new generation of viruses which do not attack PCs, but attack best effort IP networks instead: file exchange software automatically uploads files without a need for human intervention, ensuring that whatever capacity is available in the backbone will always be saturated, and peer-to-peer sessions cannot be easily identified (they are even designed to escape most classification attempts). This type of traffic is now jamming most IP networks, and the situation is likely to get much worse when users discover that they can download not only MP3 music, but also high-quality MPEG4 movies.

This situation will soon require legitimate applications to be able to use a level of service beyond ‘best effort’. We saw in Section 4.4.1.3 which difficulties are encountered when we only use IP precedence as a way to signal a need for QoS. Commercial providers need to be able to:

- Promote native support of QoS by IP applications.
- Arrange agreements to support QoS across networks managed by different entities.

- Provide QoS guarantees when needed.
- Bill for the service.

Therefore, they need to:

- Give applications a uniform way to ask for a level of QoS.
- Guarantee a level of QoS to the requesting application.
- Provide authentication.

RSVP is an appropriate answer to all these issues.

#### 4.4.2.2 Services provided by RSVP

RSVP is the key component of the IETF **integrated services** model (**IntServ**) and offers two types of services:

- *The controlled load service*: an application requesting the controlled load service for a stream of given characteristics expects the network to behave as if it was lightly loaded for that stream. The exact meaning of this is not completely defined in RSVP, but the general understanding is that packet loss should be very low or null. The absolute delay is not specified, but jitter should be kept as low as possible since in a lightly loaded network router buffers are empty. This is typically the service that could be requested to distinguish normal web browsing or email applications from peer-to-peer traffic.
- *The guaranteed service*: the guaranteed service not only requests bandwidth, but also a maximum transit delay. RSVP's guaranteed service is built on the PGPS paradigm (see Section 4.3.3.2). In the PGPS formula of the maximum theoretical delay, parameters  $C$  and  $D$  appear as sums along the path of the stream through the network: RSVP is used to calculate these sums and propagate the intermediary results between RSVP routers. The aim is to make  $C$  and  $D$  available to the recipient of the stream, together with the traffic characteristics of the flow, such as maximum burst size  $\sigma$ , average rate  $\rho$ , and peak rate  $p$ . This information allows the recipient to calculate the bandwidth  $r$  he wants to reserve for that stream in order to achieve a particular delay limit: in the formula given in Section 4.3.3.2.4 the maximal delay  $D_i^*$  is a decreasing function of  $r_i$ , so by allocating a greater minimal rate  $r_i$  the recipient can make the transit delay through the network as close as possible to  $D_{\text{tot}}$ , which is the smallest value he can hope for. In our description of PGPS, we emphasized that packets would not arrive later than the calculated PGPS delay limit, but they could also arrive *much* sooner. This means that RSVP cannot be used to specify maximum jitter independently of maximum delay. The jitter guaranteed by RSVP is nothing more than the difference between minimum path latency (propagation delays) and maximum guaranteed delay. The only way to request very low jitter is to request a delay that is very close to minimum path latency: we will see later that this is not very practical since the bandwidth reservation needed to request such a delay is extremely large. Not having strict control over jitter is in fact not very important for most applications: interactive applications need very low round



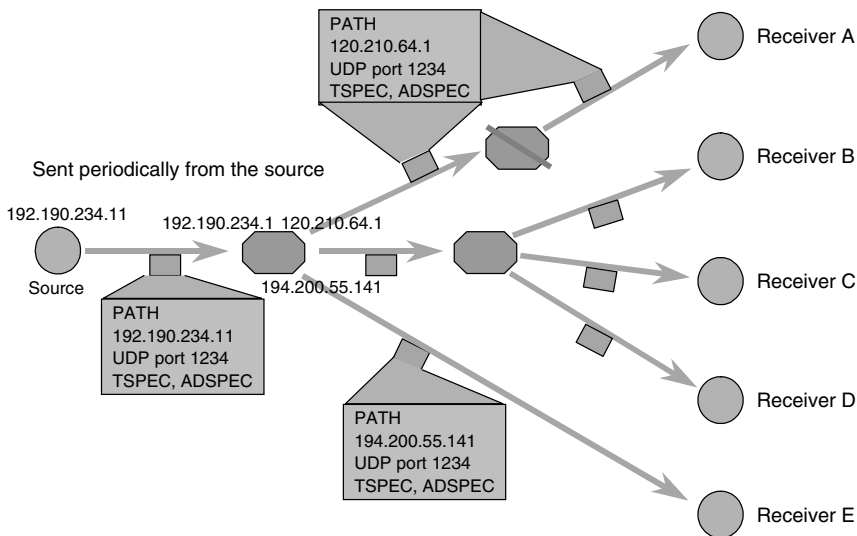
trip delays and can adapt to jitter using jitter buffers and protocols, such as RTP. But this could be a problem for applications using very large bitrate streams, because they would need to allocate a lot of memory for jitter buffers.

There are no clear guidelines in IETF documents for the use of guaranteed service versus controlled load service when writing RSVP-aware applications. The main difference is that controlled load parameters do not include target end-to-end delay values. Since the guaranteed service is more complex to implement, it may not be as readily available as controlled load mode.

### 4.4.2.3 RSVP messages

RSVP mainly uses two types of messages:

- PATH is sent by the source of the stream. This message initially contains data describing the stream (TSPEC); in particular, the bucket parameters  $\sigma$  and  $\rho$ . It follows exactly the same path as the stream itself<sup>1</sup> (including multicast transmission, see Chapter 6), and each router updates the data elements  $C_{tot}$  and  $D_{tot}$  that are also part of that message (ADSPEC). Figure 4.15 describes the propagation of PATH messages for a multicast stream. At each hop, an RSVP router modifies the PATH message to update



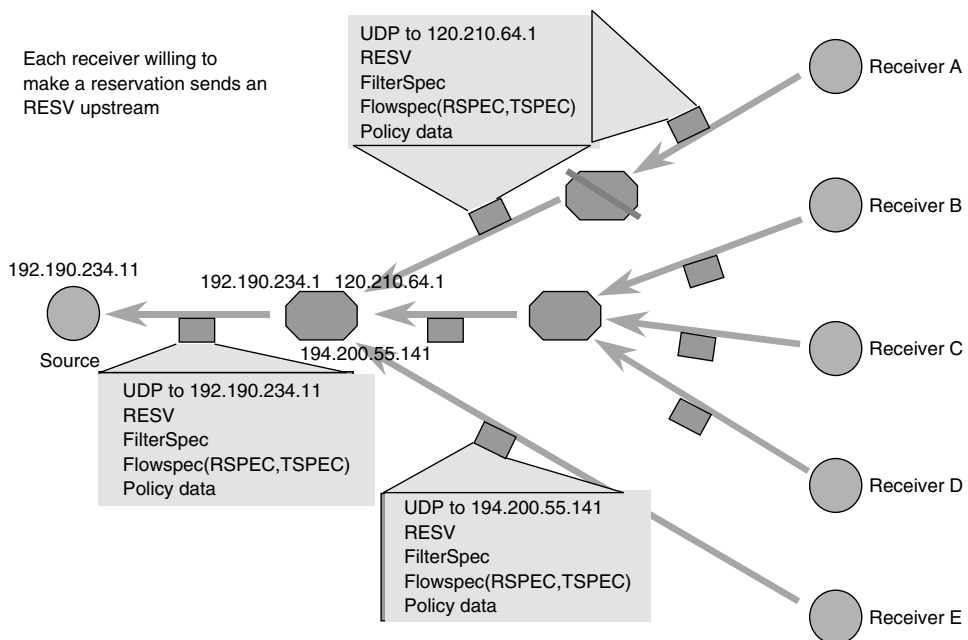
**Figure 4.15** PATH message propagation (multicast example).

<sup>1</sup> RSVP is only useful if the data packet part of the session follows the same path as PATH messages: in order to enforce this for some complex routing algorithms, it is necessary to have a dialog between the RSVP process and the routing process; this is the role of the RSRR (routing support for resource reservation) interface.

the  $C$  and  $D$  parameters, and includes its IP address in the message before forwarding it. It also stores the last hop address that was originally in the PATH message. As illustrated on Figure 4.15, if one of the routers is not capable of handling RSVP, it simply forwards the PATH packet as it received it.

- RESV is sent by the recipient(s) of the stream toward the source following exactly the inverse path of the upstream packets and PATH messages. Each RSVP router, when receiving a RESV message for a flow, forwards it to the last hop address that was previously stored from the PATH message. The RESV message specifies the minimal bandwidth  $r_i$  required for stream  $i$ , calculated from the data contained in the PATH message using PGPS theory, in order to obtain a desired maximum delay. Eventually, it can also specify an error margin on that target delay, if it needs bandwidth but not low delays (e.g., a video-streaming application). The session for which the capacity is reserved is characterized by a filter; so, a single reservation can apply to several streams (e.g., several sources in a conference). This is called a shared reservation.

Figure 4.16 shows how RSVP works even through non-RSVP routers: receiver A used the last hop address which it found in the PATH message as the destination address for the RESV message. This is in fact the address of the last RSVP router along the path: for RSVP, non-RSVP clouds appear between A and B as a direct link between A and B. If there is no congestion or significant delay in the non-RSVP cloud, the end-to-end reservations made by RSVP will still be valid.



**Figure 4.16** RSVP message propagation (multicast case).

Figure 4.16 also shows how multiple reservations for a multicast stream can be merged. If receiver B requires a low delay (large reserved rate) and receiver C is prepared to cope with more delay for the same multicast stream, then only the largest reservation will be forwarded upstream. In the case of multicast streams, reservation requests are initiated by *any* receiver and merged in the network. This is a very powerful feature of RSVP, which so far has no equivalent on ATM networks (although it could be included in ATM UNI 4.0).

During the reservation set-up phase, it is very important to avoid losing either the first PATH message or the RESV message, otherwise the reservation could be delayed by up to 30 s. For instance, over a DiffServ-enabled backbone, PATH and RESV messages could be transmitted over the netctrl (precedence level 7) class of service.

#### 4.4.2.4 Using RSVP to set up a controlled load reservation

The RSVP controlled load service is very simple and can be implemented over custom-queuing routers. If a receiver requests a reservation of 200 kbits/s for a stream with bursts up to 10 kbits, each router can configure its scheduler to allocate an average of 200 kbits/s to the stream: for instance, if the outgoing link is an E1 line (2 Mbits/s), then the scheduler must service the queue allocated to the stream at least 10% of the time.

This is not enough to guarantee a low packet loss if the traffic is bursty: each RSVP router must also make sure that the queue buffer is large enough to accommodate bursts. For instance, in our example, if the scheduler services the stream for 1 ms in every 10 ms, then the worst case is if the burst occurs just after the scheduler has finished to service the stream: the stream traffic will accumulate for 9 ms (i.e., 10 kbits for the burst and  $0.009 * 200$  kbits for the regular flow after the burst). In this case the queue buffer needs to be large enough to accommodate 11.8 kbits of data. The calculation can be refined if we also know the peak rate of the traffic, in which case the initial burst will not be considered instantaneous.

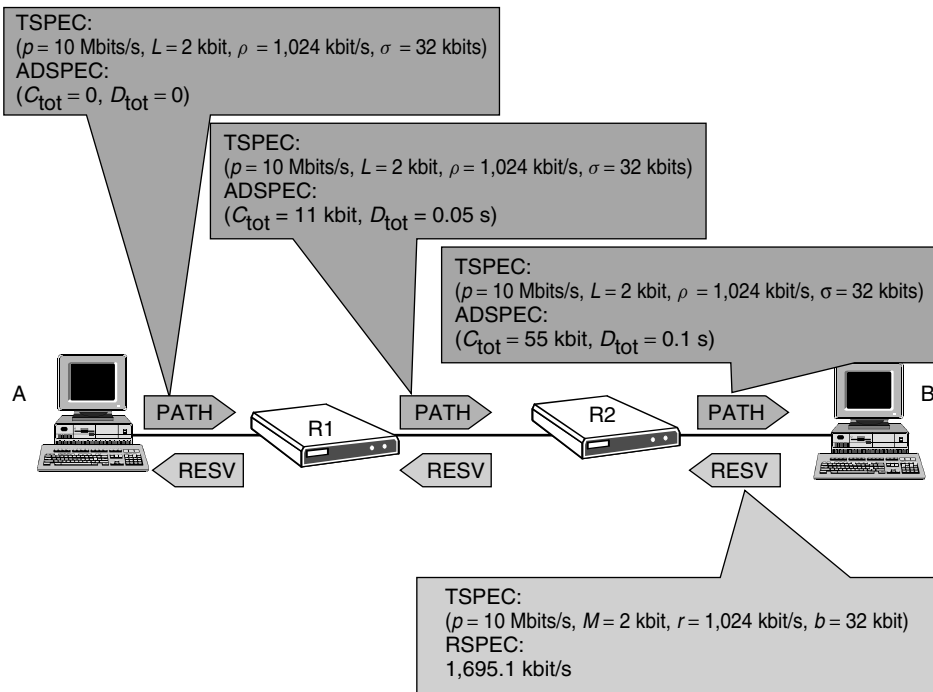
This step is repeated for every RSVP-enabled router on the path. Each router can change the characteristics of the stream, and in general the traffic will become increasingly bursty; so, routers downstream will have to allocate even larger buffers and may choose to reshape the stream. Some routers have low-capacity CPUs and may also become congested because of a lack of CPU power (this is especially true for flows generating small packets, such as IP telephony). The reservation algorithm should also make sure that enough CPU cycles will be saved for processing of the flow.

#### 4.4.2.5 Using RSVP to set up a guaranteed service reservation

##### 4.4.2.5.1 Example

In Figure 4.17, source A sends a stream to B and declares the following stream characteristics in the sender TSPEC and ADSPEC parts of the PATH message:

- *TSPEC*: ( $p = 10$  Mbits/s,  $L = 2$  kbits,  $\rho = 1,024$  kbit/s,  $\sigma = 32$  kbits).



**Figure 4.17** Updates to ADSPEC through the network.

- *ADSPEC*:  $(C_{\text{tot}} = 0, D_{\text{tot}} = 0)$ .

The first RSVP router R1 keeps the TSPEC part of the PATH message unchanged but modifies the ADSPEC part (i.e.,  $C_{\text{tot}} = 11 \text{ Kbit}$ ,  $D_{\text{tot}} = 0.05 \text{ s}$ ). The second RSVP router R2 relays TSPEC as it is and modifies ADSPEC (i.e.,  $C_{\text{tot}} = 55 \text{ Kbit}$ ,  $D_{\text{tot}} = 0.1 \text{ s}$ ). The receiver B chooses the guaranteed QoS service to obtain a specific end-to-end delay. To find which reservation he needs as a function of the desired delay (which will always be greater than  $D_{\text{tot}} = 0.1 \text{ s}$ ), he solves the equation derived from the above results:

$$r = \frac{(p - \rho)(L + C_{\text{tot}}) + (\sigma - L)p}{(D_{\text{desired}} - D_{\text{tot}})(p - \rho) + \sigma - L} \quad \text{with the constraint } r > \rho$$

For  $r = \rho$  the delay is simply  $D = (\sigma + C_{\text{tot}})/\rho + D_{\text{tot}}$  or 0.185 s; so B can choose any delay between 0.1 and 0.184 s. Here are the results obtained with some reservation values:

Desired delay (s)	$r$ to ask (kbits/s)
0.15	1,695.1
0.11	6,777.1
0.101	20,823.9

It is obvious that user B has to be reasonable and pay, because the reserved bandwidth  $R$  ‘explodes’ when the desired delay approaches  $D_{\text{tot}}$ !

B chooses a delay of 0.15 s and decides to request 1,695.1 kbit/s. It sends a RESV message with  $\text{TSPEC} = (p = 10 \text{ Mbits/s}, M = 2 \text{ kbit}, r = 1,024 \text{ kbit/s}, b = 32 \text{ kbit})$  and  $\text{RSPEC} = 1,695.1 \text{ kbit/s}$  toward A along the path followed by the PATH messages.

The receiver can also specify the ‘slack’ he can accept on top of  $D_{\text{tot}}$ . This is useful if B, for instance, ideally would like to have a low delay of 0.15 s but is prepared to accept a delay up to 0.185 s. This slack value is then transmitted in the RSPEC element. It can be used in a node if that node cannot allocate the requested bandwidth: in that case the node will ‘eat’ a part of the slack and pass on a decremented value, instead of rejecting the reservation.

RSVP can ‘work’ over non-RSVP clouds, since these clouds will forward PATH and RESV messages. But, these clouds are seen as direct links by RSVP, and the delay they introduce as well as the congestion state will not be reflected in PATH parameters: therefore, the RESV message will be wrong. This works only when the non-RSVP cloud is non-congested and is a low-delay area compared with the delay in the RSVP region.

#### 4.4.2.5.2 Soft states

It would be a very bad idea to agree to make a reservation in a node if there is the slightest chance that this reservation is not properly terminated. In ATM or PSTN networks, this leads to rather complex, but safe, signaling. The reservations within an IP network are made even more complex because routes can change at any time such as when the network topology changes (e.g., after a link failure). This is generally considered a feature that gives IP a lot of robustness facing network failures. But when you consider a reservation along a given path, this becomes a serious issue. RSVP works around this problem by only making temporary reservations, which must be refreshed from time to time by the receiver of the stream: a reservation is a ‘soft state’ with a timeout.

Since PATH messages follow the path of the stream, they will follow the new routes. Therefore, new RESV messages, which follow the inverse path, will attempt to make reservations along the new route. The old reservations will not be properly terminated through signaling, but they will time out. In some cases it will not be possible to set up the reservation along the new path: for instance, if the network is too congested or if the policy along this path is different, but this should not happen very often in a well-designed network (the same situation could also occur in the PSTN if an important link broke down). However, there could be an adverse interaction between RSVP and dynamic routing algorithms assigning packets to the less loaded link: these algorithms should make sure that existing RSVP sessions remain intact (i.e., not change the route of PATH messages ‘on the fly’).

### 4.4.3 Scaling issues with RSVP

#### 4.4.3.1 CPU limitations

RSVP itself is nothing more than a way to calculate and transmit the parameters that a node needs in order to perform a bandwidth reservation. RSVP is not responsible for

*actually reserving* bandwidth. A router must implement a scheduling or resource-sharing mechanism, such as PGPS. The class of algorithms that can support RSVP is generically called ‘weighted fair queuing’, but this name can apply to PGPS or other simplified schemes, such as SCFQ (self-clocked fair queuing). The resulting performance and actual delay limits obtained vary widely according to what is actually used.

The first difficult task is to be able to sort in real time all incoming flows based on the source address and port, and destination address and port. This function is commonly called a **multi-field classifier** (**MF classifier**). Many implementations are very sensitive to the number of flow filters that need to be recognized and have serious scalability limits when the number of filters exceeds a few dozens. Other implementations use optimized classification trees based on bit-per-bit analysis with a fixed convergence time that does not depend on the number of filters. There are even hardware-based real-time implementations.

The second difficult task is to schedule the packets to be served in an optimal order. A quick glance at the equations of PGPS gives an idea of the complexity of this task, which must be performed for every packet: PGPS, while it leads in theory to the tightest delay limits, requires a lot of processing power.

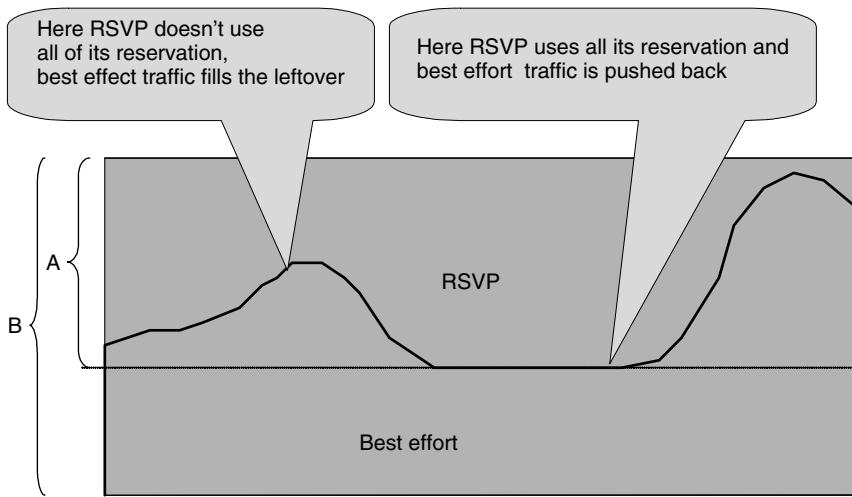
In reality, it is not the ‘complexity’ of RSVP that is the obstacle (actually, RSVP is simpler than many other signaling protocols), but the complexity of PGPS and other WFQ mechanisms. For instance, the most highly tuned kernel-mode Unix implementations of PGPS on a Pentium 166 (Ian Marsch, SICS) achieve a throughput at 90 Mbits/s over ten flows.

It is quite obvious that PGPS cannot be applied stream by stream in a backbone network. Most router vendors are using heuristics that approach the behavior of PGPS but require less CPU power. Still, it is impossible to scale per-stream queuing techniques to the throughput required in modern backbones. Backbones require approached techniques where all streams with similar properties and requiring similar priority handling are grouped and handled by the same queue.

#### 4.4.3.2 Over-provisioning

A more fundamental problem is the fact that the hard-delay limits derived above have nothing to do with what is observed statistically. For a given reservation  $r$ , the delay observed would be *much* lower than the delay guaranteed by PGPS for almost all packets. The theory behind the guaranteed mode of RSVP leads to systematic over-reservations for all applications that can tolerate losing a delayed packet from time to time. It is expected that most applications, knowing this problem, will select the controlled load mode of RSVP and will simply use the  $D$  parameter of RSVP PATH as an indication of what delay they are going to experience (e.g., to set jitter buffer parameters): in this mode, the exact average bandwidth advertised by the source is reserved to avoid packet loss.

However, in the case of a link that carries much more best effort data than real-time data, the over-provisioning required by the guaranteed service is not as bad as it seems. This is a very common situation when we consider that peer-to-peer software agents and other fancy applets on our desktops can eat as much bandwidth as the developers think they need, while we cannot speak more than 24 hours a day. Most scheduling algorithms, including of course PGPS, are able to reallocate the bandwidth not actually used by a



**Figure 4.18** Best effort traffic uses unused reserved bandwidth. A: reserved bandwidth for RSVP streams; B: available bandwidth.

stream to the rest of the traffic: the extra bandwidth reserved but not actually used by RSVP will be used by best effort traffic and there is no waste (as shown in Figure 4.18). On links where real-time traffic is predominant, the network will refuse, based on PGPS theory, reservations that in practice it could have handled; this is indeed a problem. This problem is not specific to RSVP, it also occurs on ATM-based networks much more often, as these networks have mostly been used by professional organizations for applications very sensitive to QoS that have no best effort traffic.

#### 4.4.3.3 State

RSVP is a connection-oriented technology. As such it requires network nodes to maintain state information about each connection in the network. This is a fundamental move from the connectionless paradigm of IP and probably the source of much of the debate around RSVP. Any other connection-oriented technology has the same problem: basically, these techniques do not scale as the number of connections increase through the network. In the optimal case of multicast flows, the amount of state information scales as the number of peripheral end points. In the case of unicast traffic, the amount of state information scales as the square of the number of peripheral end points!

Compared, for instance, with ATM, RSVP has its advantages and drawbacks. The one major advantage is that an IP network implementing RSVP requires state information only about QoS connections; so, if most of the traffic only needs best effort transport, it is only a small subset of the overall traffic. By comparison ATM will use a connection per stream, regardless of whether the stream has requested QoS or not (UBR connections). But ATM is based on persistent states: there needs to be signaling activity only at connection set-up and tear-down. Since RSVP is based on soft states, each connection needs periodic

signaling activity; so, the work required for the same amount of streams is much higher for RSVP than it is for ATM.

In the current state of the specification, it is hard to decide whether RSVP is really better or worse than other connection-oriented QoS techniques. However, there is an interesting perspective on the future of RSVP which could improve its scalability: using it mostly as an edge reservation request mechanism and thus simplify QoS management in the core network. This new layered model is demonstrated in Section 4.4.4, and a practical use of this model is discussed in Section 4.5.4.3.2.

#### **4.4.4 Scaling RSVP with a layered architecture**

The solution that emerges for the use of RSVP on commercial networks is to employ it as a layered architecture that utilizes the fact that RSVP messaging can cross a non-RSVP cloud. This solution focuses on the business requirements for RSVP: in short, use RSVP where it is very useful, and avoid it as much as possible when it is not strictly necessary. The business requirements are:

- (1) To give applications a uniform way to ask for a level of QoS and describe data streams, in order to minimize manual configuration and management tasks.
- (2) To find a way to guarantee a level of QoS, end to end, for each application that has requested it.
- (3) To provide authentication and facilitate billing.
- (4) To provide the necessary statistics in order to help the backbone provider properly dimension its network.

Requirements (1), (3), and (4) are for access nodes only. The only requirement that seems to require RSVP in the backbone is (2). If we can find another way to guarantee QoS in the backbone, then RSVP could be used only at the access network.

With these remarks, it is logical to divide the network into two separate concentric layers:

- In the outer layer, RSVP is used with flow-by-flow WFQ, which is possible because the bandwidth and the number of streams are still low. In addition, access routers deal with security and generation of accounting information for billing.
- In the core network, streams with ‘similar properties’ (see Section 4.4.4.1) and facing similar constraints in terms of delay and required bandwidth are grouped using classes of service. Core routers do not perform any per-flow accounting or policing, and are tuned to achieve a maximal throughput with minimal delay.

##### **4.4.4.1 Flow grouping in the backbone**

Section 4.3.3.2 summarizes results that relate to the delay limits achievable using a separate virtual queue for each stream scheduled using PGPS weighted fair queuing.



However, this scheme is not scalable due to the amount of processing power required. In this section we try to evaluate the impact of grouping several streams together in a single queue in the backbone.

In this section we call ‘similar’ those streams having similar characteristics in terms of burstiness/average rate ratio and maximum packet size. A ‘class of streams’ is composed of all streams  $i$  whose characteristics fall within the following limits:

- Average rate  $k_i(\rho \pm \Delta\rho)$ .
- Maximal burstiness  $k_i(\sigma + \Delta\sigma)$ .
- Maximal packet size  $L_{\max}$  is supposed to be common to all streams in this class.

We now suppose that  $N$  streams fit in this definition (e.g., the rate and burstiness chosen are typical of the G.723.1 sound channels of IP phones). The resulting combined stream will have an average rate of  $(\rho \pm \Delta\rho) \sum_i k_i$  and a burstiness lower than  $(\sigma \pm \Delta\sigma) \sum_i k_i$ . With these results we can calculate a delay bound at each backbone hop, where  $c$  means ‘class’:

$$D_c^* \leq \frac{(\sigma \pm \Delta\sigma) \sum_i k_i}{(\rho \pm \Delta\rho) \sum_i k_i} + \frac{L_{\max}}{C} + Tr = \frac{(\sigma \pm \Delta\sigma)}{(\rho \pm \Delta\rho)} + \frac{L_{\max}}{C} + Tr \quad (A)$$

assuming that the aggregate stream is assigned a portion of total bandwidth  $C$  equivalent to  $(\rho \pm \Delta\rho) \sum_i k_i$  (stability condition).

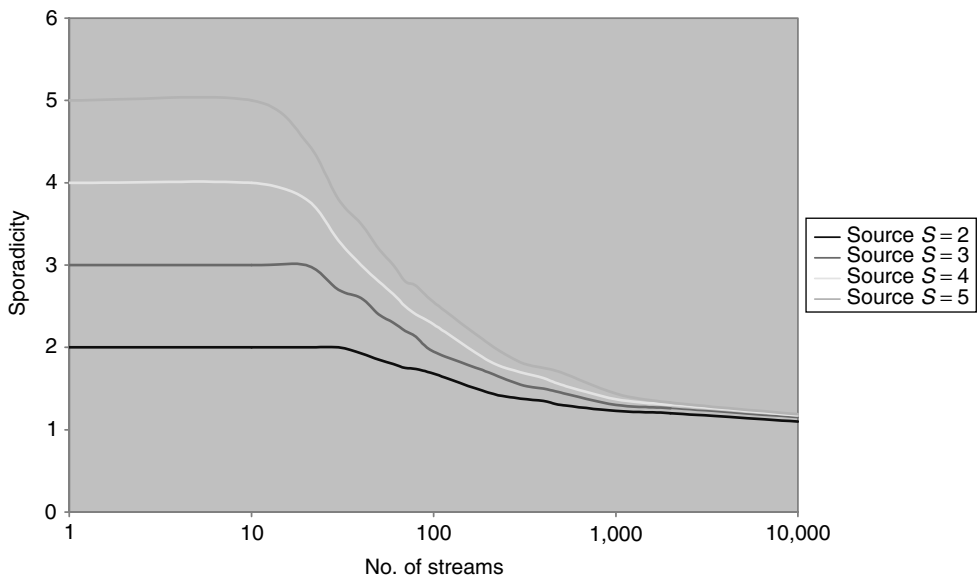
If each individual flow had been assigned a separate queue under PGPS and a capacity equal to the average rate, the result would have been:

$$D_i^* \leq \frac{\sigma_i}{\rho_i} + \frac{L_{\max}}{C} + Tr \quad (B)$$

Equations (A) and (B) give very similar results, which shows that grouping similar streams in the backbone does not cause any significant loss in guaranteed end-to-end delay. Moreover, (A) is really a worst case bound since it assumes burstiness is an additive parameter: in reality, when each individual stream is independent the resulting burstiness is much lower. How much lower depends on the exact nature of the data streams, but for a sum of periodical streams with random phase it can be calculated exactly. In Figure 4.19 the sporadic nature of source  $s$  is defined as  $\text{MaxThroughput}(s)/\text{AverageThroughput}(s)$ , where  $\text{MaxThroughput}(s)$  is defined as the level that will be exceeded by  $\text{Throughput}(s, t)$  with a probability lower than  $10^{-9}$ .

If we now consider end-to-end delay, grouping several ‘similar’ flows is very beneficial. One reason is that the maximum burst size of the aggregate stream is likely to be much lower in proportion to the aggregate bitrate. In addition, in the end-to-end formula given for PGPS through  $H$  hops, one of the components of the delay was  $(H - 1)L_{\max}/r$ , with  $L_{\max}$  the largest datagram size: when grouping several flows, this now becomes  $(H - 1)L_{\max}/Nr$ , which is much less!

If the grouped flows are similar in sporadic nature, less bandwidth is needed to achieve the same delay limit with a very high probability. This suggests that streams in the backbone with a similar sporadic nature at a similar priority level should be grouped.



**Figure 4.19** Sporadicity of aggregate streams.

#### 4.4.4.1.1 Bandwidth management

In the previous section it was assumed that  $N$  streams were grouped. In reality, this number would vary with time, and it would be difficult to adjust dynamically the bandwidth reserved for this traffic class each time  $N$  changes. A possible heuristic for bandwidth reservation could be some over-provisioning of bandwidth, which would be incremented if the class uses more than 90% of the bandwidth or decremented if it uses less than 70% (the threshold given here is arbitrary and should be derived from effective dispersion of used bandwidth within the class). This hysteresis would reduce the number of bandwidth reservation changes in the backbone for that class.

Using class-by-class WFQ with some bandwidth, over-provisioning is an acceptable waste of bandwidth in most cases, since the unused reserved bandwidth is always available for best effort traffic at any time. On most IP backbones, best effort packets represent the vast majority of the traffic.

#### 4.4.4.2 Using DiffServ with RSVP tunneling

The simplest way to simplify the provision of QoS in the backbone and avoid per-flow state is to ignore RSVP completely, relying on the simpler DiffServ architecture. DiffServ is an approach where all flows carried by the backbone are grouped in several classes of service, eliminating most of the scalability issues encountered by per-flow QoS provisioning. There are many ways to implement classes of service in the core:

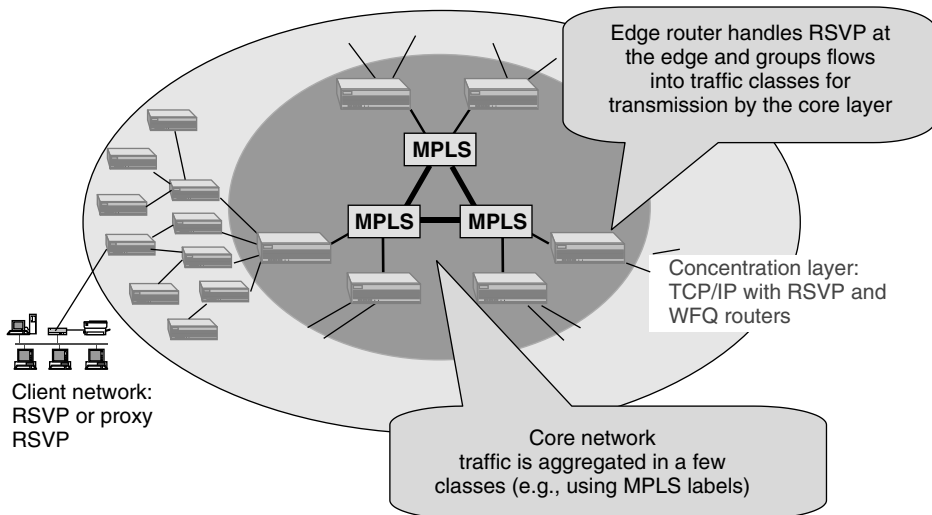
- Using IP TOS/DS: internal routers can use class-by-class WFQ, where each class is determined according to the precedence bits of IP packets.

- Using layer 2 capabilities:
  - Providers with an ATM backbone can open several virtual channels (VCs) between concentration routers with several levels of QoS or simply use the ATM CLP (cell loss priority) bit to define two rough classes of service.
  - Providers with a frame relay backbone can open several VCs between border routers with various levels of QoS parameters or use the DE (discard eligible) bit.
  - Providers with MPLS (multiprotocol label switching) backbones can also define one level of QoS for each label.

These techniques can be combined with congestion control mechanisms, such as RED or WRED, to smooth TCP traffic and improve fairness between the flows within each class of service.

We have seen that RSVP is an end-to-end protocol; so, RSVP messages need to be passed between hosts across the backbone. This is not as simple as it seems. First, RSVP messages must be ignored in the backbone, otherwise we would run into scalability issues. Second, each RSVP packet is marked with a 'router alert' option in order to help routers identify this packet as one needing special treatment (the router alert option should be turned off, or ignored, when passing along the backbone). Some implementations (e.g., ISI) do not rely on the router alert option, but rather on interception of packets with protocol 46. In this case it is also possible to use a new IP protocol number that only access routers would recognize, or tunnel RSVP packets in an IP tunnel through the backbone.

Figure 4.20 shows a layered network architecture in which RSVP is used at the edge for admission control and as a way for applications to declare data flow properties. Data flows are then grouped into traffic classes. Core layer QoS mechanisms only consider these traffic classes.



**Figure 4.20** Scalable QoS using a layered architecture.

#### 4.4.4.3 Handling of RSVP messages through the backbone

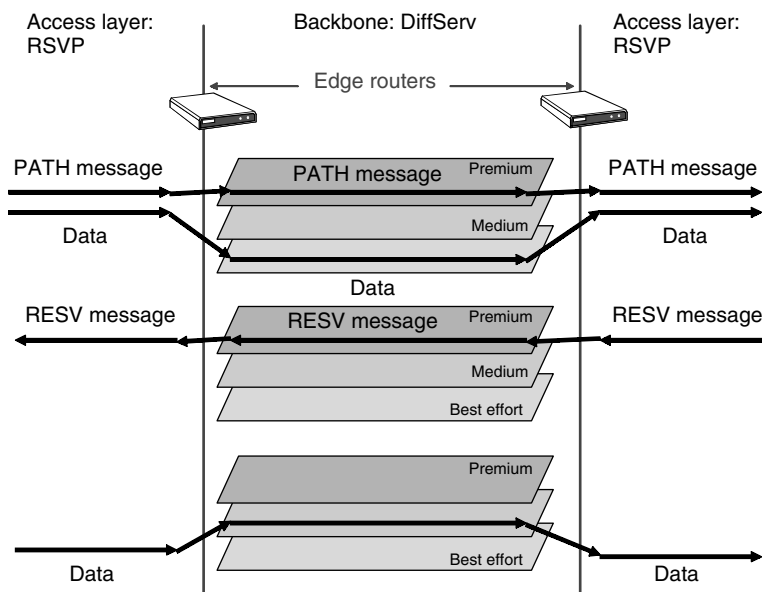
The RSVP and the DiffServ paradigms are based on opposite models: RSVP gives precedence to a stream based on the receiver's wishes, while DiffServ prioritization is controlled by the sender.

In the layered model, an edge router sits between the DiffServ domain and the IntServ (RSVP) domain. This edge router can modify the TOS/DS value of all packets injected in the backbone and, therefore, has complete control over the priority of these packets. In order to emulate receiver-based RSVP behavior, this router must decide which class of service to use for a flow based on the QoS requirements contained in the RESV messages (as shown in Figure 4.21). In this case, after analyzing the first RESV message, the edge router decides that this session must be aggregated into the 'medium' class of service and all the packet parts of this stream are marked accordingly.

##### 4.4.4.3.1 PATH messages

For the end-user application, the network must behave as if it was RSVP-enabled end to end. Therefore, the PATH messages generated by the sender must cross the backbone and reach the receiver(s). Having received a PATH message describing a flow, the receiver may choose to send back a RESV message in order to reserve resources for this flow in the backbone.

However, if an implementation transmits PATH parameters transparently the receiver will have a false view of the backbone, because the parameters within the PATH messages will not have been updated to reflect latency and other characteristics of the backbone.



**Figure 4.21** Handling an RSVP reservation through a DiffServ core.

Therefore, the receiver might be misled, thinking the backbone adds less delay than it actually does, and make a wrong reservation (in ‘guaranteed service’ mode).

If the backbone is built using powerful ‘gigarouters’, then each PATH message can be updated at each hop. On the other hand, if the backbone is built using ATM or MPLS technology, the PATH message needs to be updated only once by an edge router that evaluates the overall transmission delay along the virtual circuit to the destination of that stream, making the backbone appear as a single node for RSVP. A practical approach is to propagate the value of the  $C$  parameter ‘as is’ and update only the  $D$  parameter. In other words, we consider the queuing delay added by the backbone not to be very sensitive to the characteristics of individual streams, but able to be approximated by an absolute delay value that does not depend on the capacity reserved for the individual stream. This approximation is generally valid if the number of aggregated streams is large. A single stream has a negligible influence on backbone transit delay. Therefore, we update  $D$  with a value representing the propagation delays and the average queuing delay through the backbone.

#### 4.4.4.3.2 *RESV messages*

If the backbone uses routers that have enough processing power to handle RESV messages hop by hop, then each RESV message can be propagated normally through the backbone. But, instead of creating a separate queue for the stream as in regular RSVP operation, the router will direct that stream to the queue that is most appropriate for this type of session and the required class of service (as discussed in Section 4.4.4.1). These routers will maintain the bandwidth usage information for each class of service and eventually decide to add more bandwidth for a particular class.

For most backbones, the edge router receiving a RESV message will simply tunnel this RESV message to the edge router used by that stream to enter the backbone. This router will then be responsible for properly choosing the DiffServ DSCP field for packets of this stream. The Diffserv codepoint will be used to choose a specific traffic class for the backbone and may be mapped to layer two QoS mechanisms (e.g., in MPLS tags).

#### 4.4.4.4 *Caveats*

With this layered approach, the ingress edge router will aggregate several flows over each class of service. This will not have a significant impact on the QOS level experienced by each individual flow except in some situations:

- When one of the flows does not conform to its TSPEC: if the provider has dimensioned each virtual link between the ingress and egress routers for an aggregate of flows with well-defined characteristics, one non-conformant flow may be enough to ruin the quality of service experienced by all other flows on the same class.
- When very sporadic flows are merged with smooth flows (e.g., video with voice).

Therefore, the access router must have the ability to check the TSPEC of each flow (and destroy or mark out-of-profile packets), and the service provider must avoid, when defining the classes of service of the backbone, merging sporadic and smooth flows.

## 4.5 The CableLabs® PacketCable™ quality-of-service specification: DQoS

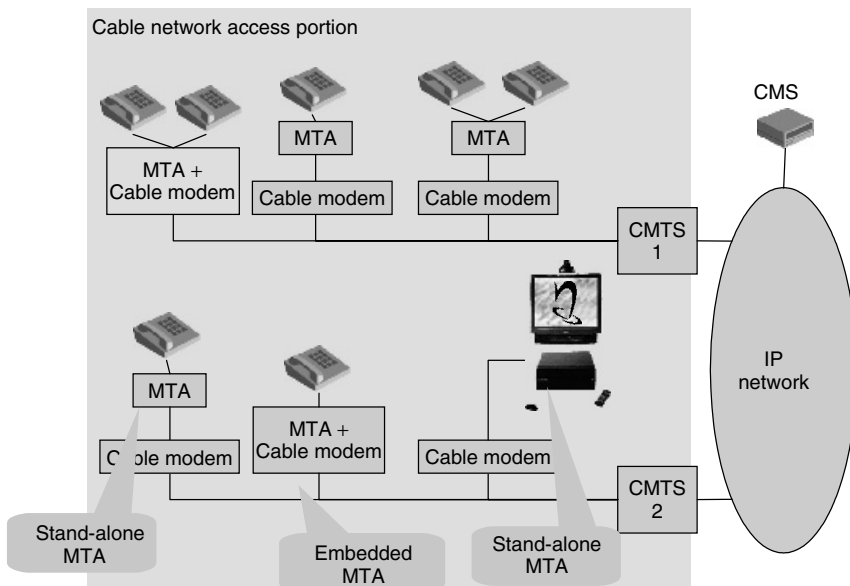
Although simpler than the framework described in Section 4.4.4, the DQoS framework is one of the most advanced examples of a layered network architecture providing tight per-flow QoS control at the edge, while requiring only broad classes of service in the core backbone. Compared with the RSVP tunneling method described above, DQoS differs slightly by offering the option to terminate RSVP locally, in which case RSVP is used only as a local admission control and service-level request protocol, and not for end-to-end QoS provisioning.

### 4.5.1 What is DQoS?

Dynamic quality of service (DQoS) defines an architecture and a set of protocols to provide assured quality of service in the access portion of a cable network. DQoS has been specified within the PacketCable™ project of the CableLabs® consortium. The specification is publicly available on the PacketCable™ website ([www.packetcable.com](http://www.packetcable.com)).

The access portion of a cable network is defined as the portion between the **MTA** and the **CMTS** (Figure 4.22):

- The MTA (multimedia terminal adapter) is the component that generates multimedia streams. It can be a PC, a stand-alone VoIP gateway for analog phones, an IP phone,



**Figure 4.22** The PacketCable® ecosystem.

etc. The MTA is connected to a cable modem (CM). If the MTA and the cable modem are in the same device, it is called an ‘embedded MTA’, otherwise it is a ‘stand-alone MTA’, which can be connected to the cable modem (e.g., using an Ethernet cable or a USB cable).

- The CMTS (cable modem termination system) is responsible for collecting all data streams from cable modems and routing the data either to an external IP network or back to a cable modem. The CMTS is the only trusted entity of the access portion.

The MTAs are controlled by a **call management server (CMS)**, using one of the two call-signaling protocols defined by PacketCable™: **NCS (network-based call signaling)**, a variant of MGCP or **DCS (distributed call signaling)**, a variant of SIP). Today the market is almost completely NCS. The CMS also sets the proper QoS authorizations (**gates**) on the CMTS; this is the **gate controller** function in the DQoS framework.

Between the MTAs and the CMTS, two mechanisms may be used for QoS control:

- A layer 2 mechanism (DOCSIS 1.1 MAC), which is the only mandatory mechanism in DOCSIS 1.1 and is only available for embedded MTAs.
- A layer 4 mechanism based on an extended version of RSVP, called RSVP+, which is available both for embedded or stand-alone MTAs. In PacketCable™ versions after 1.0, RSVP+ support is required for non-embedded MTAs and the CMTS. For embedded MTAs, the RSVP+ or the MAC mechanism must be supported.

DQoS provides quality of service on a segment-per-segment basis; that is, on a call from CMTS1 to CMTS2, quality of service will be performed independently on the CMTS1 DOCSIS segment and on the CMTS2 DOCSIS segment, possibly using different mechanisms (MAC- or RSVP-based). Each service provider and each segment can select its own preferred DQoS mechanisms. Optionally, the backbone segment between the two CMTSs can also implement a QoS mechanism (IntServ or DiffServ), but the protocols used at this level are not within the scope of DQoS.

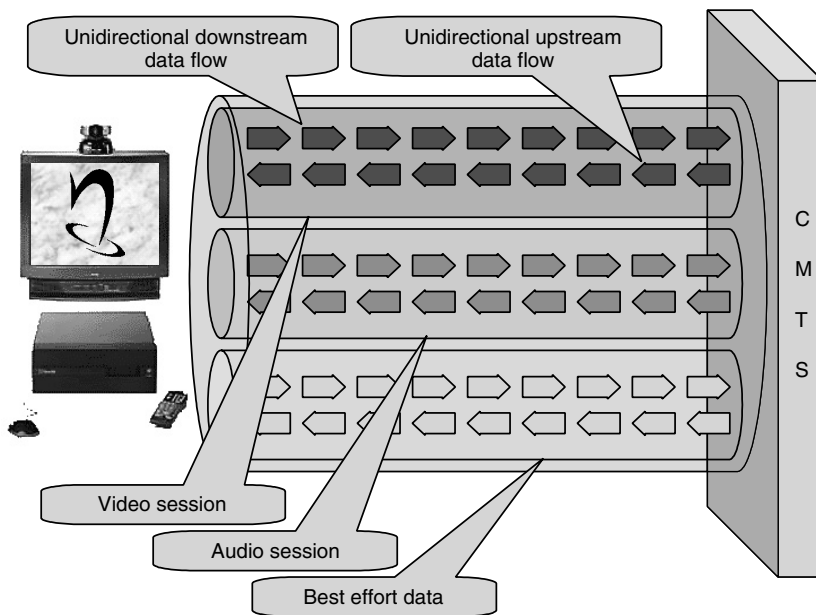
## 4.5.2 Session-per-session QoS reservation

DQoS allocates resources to the flow of data between two applications running on separate endpoints. Bidirectional<sup>2</sup> data communication is called a *session* (Figure 4.23). The QoS is provided individually to both the unidirectional upstream data flow and the unidirectional downstream data flow within a session. The QoS is provided only to authorized sessions, and for each session usage can be monitored: these accounting data enable usage-based charging.

The term ‘dynamic’ is used because the QoS policy category of a user can change (e.g., ‘gold’ to ‘bronze’) without resetting the cable modem. The QoS settings of a session can

---

<sup>2</sup>This terminology is specific to DQoS. In the rest of the chapter the word ‘session’ refers to a monodirectional data stream



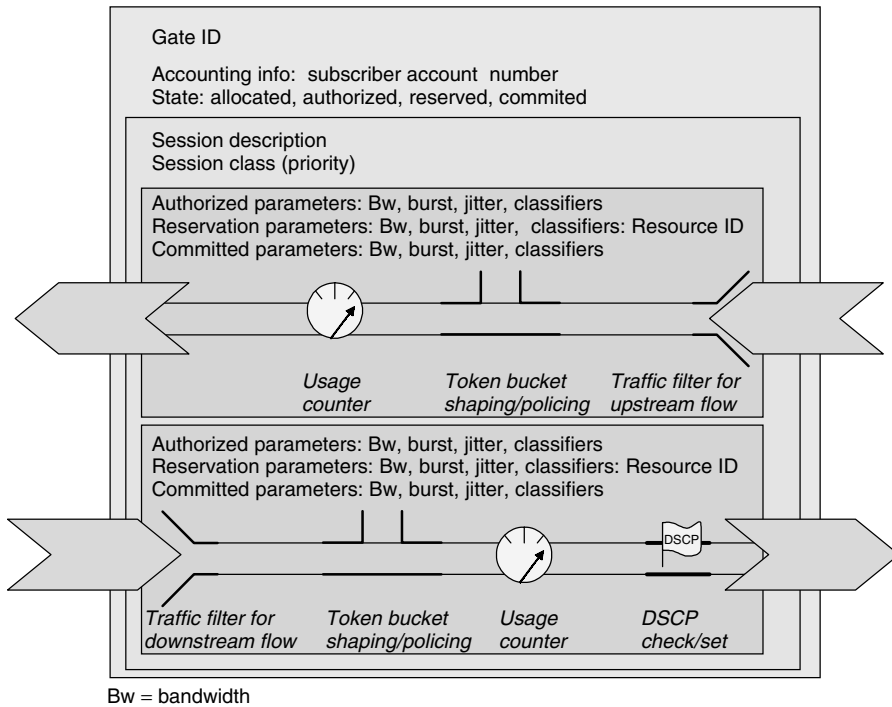
**Figure 4.23** Flows and sessions.

change in the middle of a session. For instance, if there is a codec change from G729 to G711 for a fax, the MTA can dynamically commit more bandwidth if it had already reserved enough capacity or can attempt to reserve more bandwidth if the reserved capacity is not enough.

The CMTS allocates and schedules the bandwidth corresponding to each session reservation, performing the role of the **policy enforcement point (PEP)**, using the terminology of the IETF **Resource Allocation Protocol** framework defined in **RFC 2753**). The CMTS performs this function by implementing a **DQoS Gate** (a set of packet classification and filtering functions, as described in Figure 4.24) for each session between the cable network and the IP backbone or between MTAs on the cable network. Each CMS implements a gate controller function that installs and controls gates on the CMTS: this is the policy decision point (PDP) in the IETF Resource Allocation Protocol framework. If a gate is closed, then the traffic will either be dropped or forwarded in the best effort class, depending on service provider policy.

The CMTS is responsible for providing the QoS requested by cable modems if allowed to do so by the current policy, allocating the proper upstream capacity on the shared medium. At layer 2 of the cable network, the CMTS uses the mechanisms defined in DOCSIS 1.1 at the MAC level (not covered here) to implement the QoS policy. At the IP level, the CMTS also verifies that IP streams sent by cable modems are properly shaped, and verifies and eventually resets the DSCP of the IP packets it sends to the backbone. In the other direction, the CMTS classifies the IP packets coming from the backbone interface, applies traffic shaping, and DSCP marking before forwarding these packets on the cable network.





**Figure 4.24** The DQoS 'gate'.

The cable modem is responsible for implementing DOCSIS 1.1 QoS mechanisms to request and obtain QoS on the cable network. It must also classify upstream IP packets according to the filters declared to the CMTS and shape them properly according to the declared token bucket parameters. Since the cable modem is not trusted, the CMTS will double-check that the flow conforms to the profile declared in the gate for this session.

### 4.5.3 Two-phase reservation mechanism

Figure 4.24 shows that the gate can have several states:

- *Allocated.* The gate has been created at the initiative of the gate controller, but not yet initialized with resource authorization parameters.
- *Authorized.* The gate controller has set the maximum level of resources (envelope) that can be reserved within this gate. This restriction applies to any subsequent reservation request. A gate controller can change an authorization for a given gate, but this only applies to future reservation requests. Since it establishes gates and gate authorizations in advance of a resource request, the gate controller can remain unaware of the state of sessions in progress (stateless model). For voice traffic, a permanent gate is allocated to signaling flows and a per-call gate is established for each call.

- *Reserved.* This state corresponds to admission, the first phase of the QoS reservation mechanism. Reserved resources are available to best effort traffic until they committed, but they are removed from the pool of resources available for admission control. Usually, the MTA is responsible for reserving resources. Reservation is soft state and will expire unless it is refreshed. At the end of reservation, DOCSIS link service flows are in 'admitted state'.
- *Committed.* The gate is in committed state once an MTA has sent a confirmation that the resource is being used. The gate is only open after reservation has been committed, preventing fraud and theft of service. Excess reserved resources beyond committed resources are released unless the MTA periodically refreshes the reservation (e.g., necessary for calls on hold). At the end of the commit phase, DOCSIS link service flows are in 'active' state and usage recording starts. In practice, the IP flows of a session in 'committed' state use a WFQ-prioritized waiting queue, which ensures they will receive a guaranteed level of service. If, however, the effective resource usage of these streams remains below the committed capacity, then the scheduling algorithm of the CMTS, if appropriately designed, will enable best effort traffic to use leftover capacity (this is the case with PGPS scheduling, compare Section 4.3.3.2).

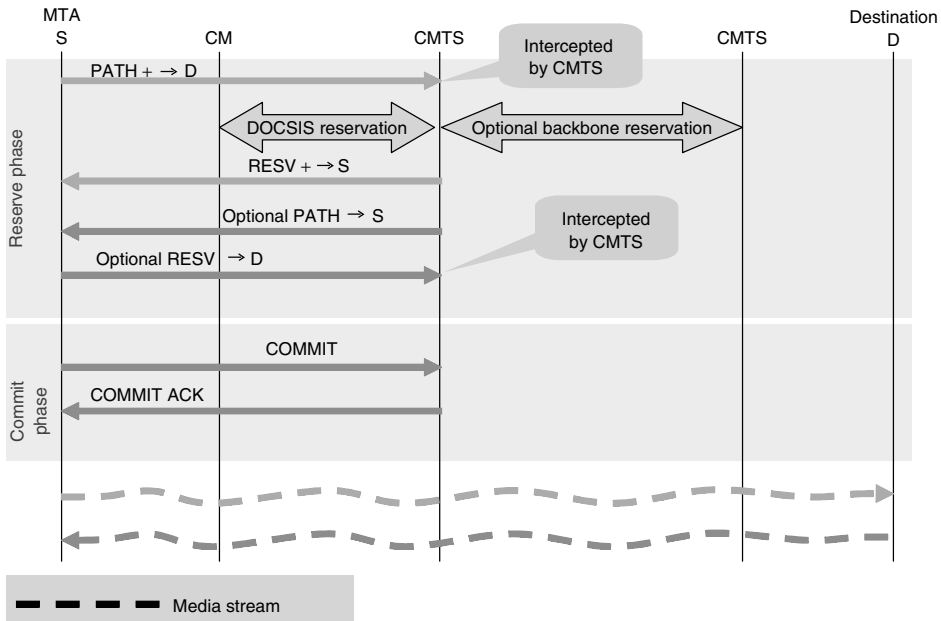
The binding between a reservation and a session is dynamic. For instance, for calls on hold, a reservation will be used for the active call and the associated session switches to the held call when it becomes active.

When reserving resources for a session, a session class is specified, allowing the CMTS to keep resources for high-priority traffic (e.g., emergency calls). Session classes may be overlapping (e.g., 50% is the maximum for normal calls and 70% for emergency calls) and may be pre-emptive. The session class is communicated in the Gate-Set request of the CMS to the CMTS together with the authorized session envelope.

The MTA is not forced to commit resources if there is a reservation or to reserve resources that have been authorized. For instance, if port-to-port calls remain on the same MTA (same IP address), IP packets are not forwarded on the cable network. This situation could be recognized by the CMS, in which case the CMS sends a connection request without a Gate-ID, indicating that the MTA must not reserve or commit resources. But, in the most common case such a situation is not necessarily known at the beginning of the call. If a Gate-ID was communicated to the MTA and the far end was still not known, the MTA must tear down the service flow (reservation and/or commit) once the port-to-port call is recognized.

#### **4.5.3.1 Separate MTA and CM: the MTA to CMTS QoS protocol**

The admission part of the QoS reservation two-phase process uses a superset of RSVP, named RSVP+. The main difference is that, unlike RSVP, a PATH message from the cable modem in RSVP+ requests resources in both directions (upstream and downstream), and a RESV message from the CMTS confirms the reservation request has passed admission control for both directions (Figure 4.25). In RSVP, the PATH message applies only to the stream sent by the PATH sender.



**Figure 4.25** Use of RSVP+.

The PATH message of RSVP+ is sent to the same destination as the data flow, as in RSVP, but it is intercepted by the CMTS and therefore never reaches the destination. In order to preserve compatibility with RSVP routers that may be present between the CM and the MTA, the CMTS may also send a PATH message toward the MTA.

SDP information sent in the call control protocol from the CMS to the MTA contains enough details to allow calculation of an RSVP flowspec for well-known codecs. For other codecs, SDP can include explicit bandwidth information:

```

b:<modifier CT for Conference Total or AS for Application
  Specific>:<bandwidth value including IP/UDP/RTP overhead>
    
```

RSVP+ exchanges also cause the CMTS to initiate DOCSIS MAC-level QoS reservation with the cable modem.

The commit phase is performed by the MTA using a separate COMMIT UDP message (not an RSVP message) sent to an address specified in the RSVP + RESV message. It is acknowledged by a COMMIT ACK.

#### 4.5.3.2 *Embedded MTA: the optional MTA to CMTS protocol – the one used in practice*

An embedded MTA may use the MAC control service interface described in the DOCSIS 1.1 RFI specification, using DSx messages to reserve resources on the cable plant instead of the RSVP+ interface. This mechanism cannot be used if the MTA and the cable modem

are separate entities. In practice, in all cable networks today the MTA and the CM are integrated, in order to simplify the provisioning of telephony lines to customer premises and to ensure that the service will be maintained for emergency calls using the batteries of cable modem. RSVP+ then becomes useless, like the rest of the DQoS framework, because the CM is considered, in practice, a trusted element of the network. Therefore, contrary to what is sometimes heard, DQoS is not a prerequisite for the deployment of telephony over cable networks. It is only required if the cable network operator wishes to provide service to non-integrated MTAs over the end customer LAN. Unfortunately, there are no—to our knowledge—non-integrated MTAs with support for RSVP+ on the market. It seems the market for non-integrated MTAs is too small to justify developing specific versions of analog VoIP gateways for cable networks. This is the reason the DQoS framework remains mainly an interesting theoretical exercise, one that certainly prefigures future developments for next generation broadband networks. But, in reality it is of little use on current cable networks (except as an argument to sell next generation CMTS systems).

#### **4.5.4 CMS to CMTS communications**

The CMS and CMTS communicate using the IETF Common Open Policy Service (COPS) protocol.

##### **4.5.4.1 The COPS protocol**

COPS is defined in RFC 2778. It is designed to convey control communications between one or more policy enforcement points (PEP) and a policy decision point (PDP). These terms are defined by the IETF framework for policy-based admission control (RFC 2753): a PEP is responsible for executing a security policy set by the PDP. PEPs filter and classify IP packets, and may need to mark them with appropriate DSCP codes or to perform shaping on the data streams. They can also accept or reject RSVP QoS requests. The COPS protocol is not completely specified and needs to be profiled for any given application. In the following sections we focus on the DQoS profile.

##### **4.5.4.1.1 Generic COPS message format**

The COPS protocol is based on messages that begin with a specific header and contain multiple encapsulated objects (Figure 4.26). The C-Num octet identifies the type of object from among the 16 types of objects used by COPS, while the C-Type octet identifies the subtypes of a given object. The objects used by the PacketCable™ DQoS are indicated in bold in Table 4.2.

Most of the objects in Table 4.2 are placeholders for information which must be specified separately in a COPS profile. In the case of DQoS, a summary of the profile can be found in Section 4.5.4.2.

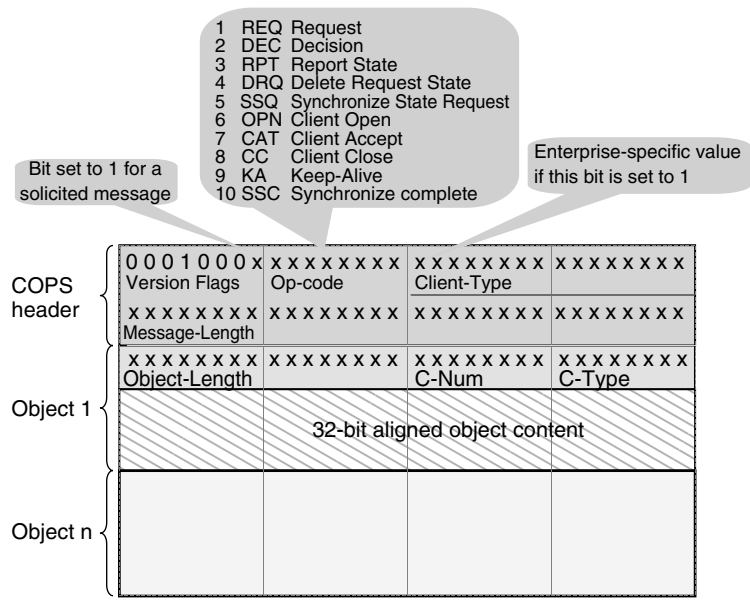


Figure 4.26 COPS message format.

4.5.4.1.2 Generic COPS operations

PEPs initially connect with a PDP using a TCP socket to port number 3288 and then begin the COPS session by sending a Client-Open (OPN) message. The PDP can either redirect the PEP to another PDP (Client-Close message with a <PDPredirAddr> object) or accept the connection by sending a Client-Accept (CAT) message. The integrity of the connection is then checked periodically using Keep-Alive (KA) messages, until the connection is closed with a Client-Close message.

The PEP then sends Request (REQ) messages to the PDP, which installs a QoS-related state (specified in the COPS profile) in the PDP. The PDP replies to each request with a Decision (DEC) message. Each Request is identified by a client handle object, generated by the PEP, which is copied in the PDP DEC message. This handle is associated with the state installed by the request, and the PDP can send new DEC messages regarding that state until the client handle is explicitly closed by the client (using a Delete Request State, or DRQ, message). COPS messages are illustrated in Figure 4.27.

4.5.4.2 COPS profile for DQoS

The DQoS COPS profile is specified in sect. 5.2 of the DQoS specification. In DQoS, the gate controller (PDP) initiates the COPS connection by establishing a TCP connection to the IP address of the CMTS (port 2126). DQoS PEPs do not support PDP redirections in Client-Close messages. The COPS Client-Type for a DQoS PEP is 0x8008.

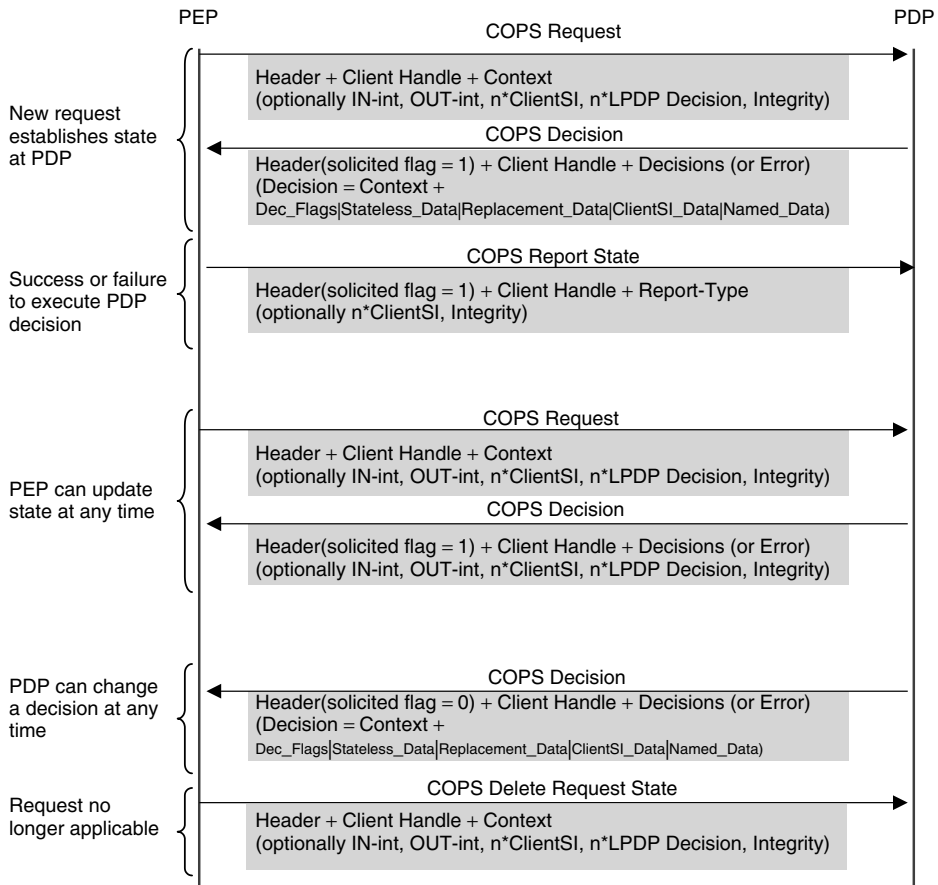
**Table 4.2** COPS specific objects

C-Num	Object type	C-Type and subtypes
<b>1</b>	<b>Handle:</b> unique value that identifies an installed state	1: Client handle
<b>2</b>	<b>Context:</b> the type of event that triggered a query	1: Request-Type/Message-Type
3	In interface: incoming interface on which a request applies	1: IPv4 address + Interface 2: IPv6 address + Interface
4	Out interface: outgoing interface on which a request applies	1: IPv4 address + Interface 2: IPv6 address + Interface
5	Reason code: reason for a delete request message	1
<b>6</b>	<b>Decision:</b> decision made by a PDP	1: Decision flags 2: Stateless data 3: Replacement data 4: Client-specific decision data 5: Named decision data
7	LDPD decision: decision made by a PEP-local PDP	Same as Decision
<b>8</b>	<b>Error:</b> identifies a COPS protocol error	1
<b>9</b>	<b>Client-specific info (SI)</b>	1: Signaled client SI 2: Named client SI
<b>10</b>	<b>Keep-Alive timer</b>	1: KA timer value
<b>11</b>	<b>PEP identification</b>	1: ASCII string
12	Report type	1
13	PDP redirect address	1: IPv4 address + TCP port 2: IPv6 address + TCP port
14	Last PDP address	1: IPv4 address + TCP port 2: IPv6 address + TCP port
15	Accounting timer	1: value
16	Message integrity	1: HMAC digest

As shown in Figure 4.28, DQoS really uses COPS as a transport or tunneling protocol, without really using its semantics: it uses a single COPS request and handle, and then uses COPS DEC messages and RPT messages to exchange DQoS-level messages:

- From the GC to the CMTS: DQoS primitives are placed inside the Decision Object (object C-Num 6, C-Type 4) of a COPS DEC message.
- From the CMTS to the GC: DQoS primitives are placed inside a Signaled Client SI object (C-Num 9, C-Type 1) of a COPS RPT message.

The Context object in the COPS Decision message has the R-Type (Request Type Flag) value set to 0x08 (Configuration Request) and the M-Type set to zero. The Command-Code field in the mandatory Decision-Flags object (C-Num 6, C-Type 1) is set to 1 (Install Configuration).

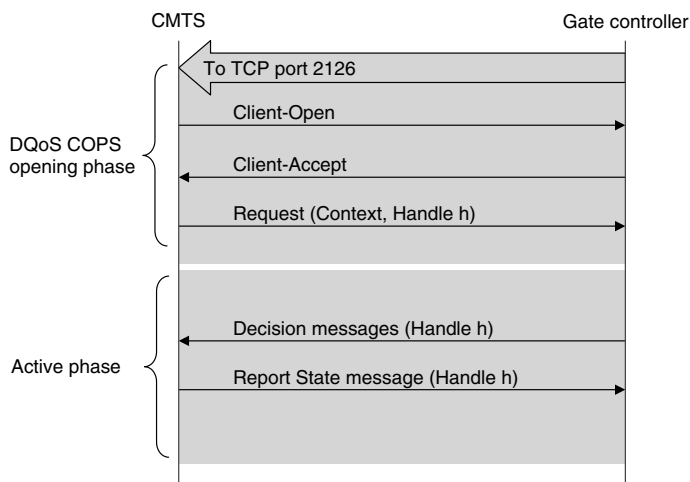


**Figure 4.27** PEP to PDP dialog.

#### 4.5.4.2.1 DQoS commands

The following commands are defined by DQoS (encapsulated DQoS-specific subobjects are shown in *italic*):

- GC-initiated messages to CMTS:
  - **<Gate-Alloc-Command>**=<COPS Header> <Handle> <Context> <Decision-Flags> <Decision-Header> <TransactionID> <Subscriber-ID> [ <Activity-Count> ].  
Gate-Alloc validates the number of simultaneous sessions allowed to be set up from the originating MTA and allocates a Gate-ID to be used for all future messages regarding this gate.
  - **<Gate-Set-Command>**=<COPS Header> <Handle> <Context> <Decision-Flags> <Decision-Header> <Transaction-ID> <Subscriber-ID> [ <Activity-Count> ] [ <Gate-ID> ] [ <Event-Generation-Info> ] [ <Electronic-Surveillance-Parameters> ]



**Figure 4.28** Opening a DQoS connection.

[<Session-Description-Parameters>] <Gate-Spec> [<Gate-Spec>]. Gate-Set initializes and modifies all the policy and traffic parameters for the gate or set of gates, and sets the billing and gate co-ordination information.

- <**Gate-Info-Command**>=<COPS Header><Handle><Context><Decision-Flags><Decision-Header><Transaction-ID><Gate-ID>. GATE-INFO is a mechanism by which the gate controller can discover all the current state and parameter settings of an existing gate or set of gates.
- <**Gate-Delete-Command**>=<COPS Header><Handle><Context><Decision-Flags><Decision-Header><Transaction-ID><Gate-ID><PacketCable-Reason>. Gate-Delete allows a gate controller to delete a recently allocated gate under certain circumstances.
- CMTS to GC responses:
  - <**Gate-Alloc-Ack-Response**>=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Subscriber-ID><Gate-ID><Activity-Count>.
  - <**Gate-Alloc-Err-Response**>=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Subscriber-ID><PacketCable-Error>.
  - <**Gate-Set-Ack-Response**>=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Subscriber-ID><Gate-ID><Activity-Count>.
  - <**Gate-Set-Err-Response**>=<COPS-Common-Header><Handle><Report-Type><ClientSI-Object>.
  - <**Gate-Info-Ack-Response**>=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Subscriber-ID><Gate-ID>[<



*Event-Generation-Info>][<Electronic-Surveillance-Parameters>] [<Session-Description-Parameters>][<Gate-Spec>] [<Gate-Spec>].*

- **<Gate-Info-Err-Response>**=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Gate-ID><PacketCable-Err>.
- **<Gate-Delete-Ack-Response>**=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Gate-ID>.
- **<Gate-Delete-Err-Response>**=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Gate-ID><PacketCable-Err>
- CMTS-initiated messages (use COPS to GC but Radius to remote CMTS) which can also be GC-initiated (using COPS):
  - **<Gate-Open>**=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Gate-ID>.
  - **<Gate-Close>**=<COPS-Common-Header><Handle><Report-Type><ClientSI-Header><Transaction-ID><Gate-ID><PacketCable-Reason>. Gate-Open allows the CMTS to inform the gate controller that gate resources have been committed. Gate-Close allows the CMTS to inform the GC that the gate has been deleted due to MTA interaction or inactivity. These messages provide a feedback path from CMTS to CMS in order to allow for accurate call-state management at the CMS element.

#### 4.5.4.2.2 DQoS subobjects

4.5.4.2.2.1 *Transaction-ID* The Transaction-ID is used by the gate controller to match CMTS responses to previous GC requests.

Length = 8	S-Num = 1	S-Type = 1
2-octet Transaction-ID	Gate Command Type	

The following Gate Command Types are defined:

GATE-ALLOC	1
GATE-ALLOC-ACK	2
GATE-ALLOC-ERR	3
GATE-SET	4
GATE-SET-ACK	5
GATE-SET-ERR	6
GATE-INFO	7

GATE-INFO-ACK	8
GATE-INFO-ERR	9
GATE-DELETE	10
GATE-DELETE-ACK	11
GATE-DELETE-ERR	12
GATE-OPEN	13
GATE-CLOSE	14

4.5.4.2.2.2 *Subscriber-ID* This identifies the subscriber for this service request.

Length = 8	S-Num = 2	S-Type = 1
IPv4 address (for IP-v6, use S-Type = 2)		

4.5.4.2.2.3 *Gate-ID* This specifies the gate referenced in a command message or assigned by the CMTS in a response message.

Length = 8	S-Num = 3	S-Type = 1
Gate-ID		

4.5.4.2.2.4 *Activity count* In a Gate-Alloc message, this represents the maximum number of gates that can be allocated to a Subscriber-ID. In a Gate-Set-Ack or Gate-Alloc-Ack, it indicates the number of gates assigned to a single subscriber.

Length = 8	S-Num = 3	S-Type = 1
32-bit counter		

4.5.4.2.2.5 *Gate-Spec* The Gate-Spec object defines the filters that identify a flow, the direction of the flow, and its token bucket parameters for authorization (the authorization envelope). Filter values of zero (e.g., IP ID = 0) are wildcards (in this case the IP

indicated in the header of the IP packet parts of the flow can be anything). Two Gate-Spec objects must be used for a bidirectional session.

Length = 60		S-Num = 5	S-Type = 1
0:Downstream/1:Upstream	IP ID	Flags	Session-Class
Source IP address (32 bits)—0 for wildcard			
Destination IP address (32 bits)—0 for wildcard			
Source port—0 for wildcard		Destination port—0 for wildcard	
DiffServ DSCP			
Timer T1 value—maximum authorization to commit time (in ms)			
Timer T7 value—maximum duration of a single gate open state when a flow crosses two CMTS (in ms)			
Token Bucket Rate ( $r$ bytes/s)			
Token Bucket Size ( $b$ bytes/s)			
Peak Data Rate ( $p$ bytes/s)			
Minimum Policed Unit, (i.e., smallest IP packet in the stream ( $m$ octets))			
Maximum Packet Size ( $M$ bytes)			
Rate ( $R$ bytes/s)			
Slack Term, (i.e., acceptable delay on top of theoretical delay obtained for $R = r$ (s))			

The flag value 0x01 can be used to tell the CMTS to automatically commit the reserved resources without waiting for an MTA command.

**4.5.4.2.2.6 Event-Generation-Info** This element is included if the CMS wants the CMTS to generate accounting records.

Length = 44	S-Num = 7	S-Type = 4
Primary Record Keeping Server IP		
Primary Record Keeping Server Port	Flags	

Secondary Record Keeping Server IP		
Secondary Record Keeping Server Port	Flags	
Billing Correlation ID (24 bytes)		

*4.5.4.2.2.7 Electronic surveillance parameters* This enables the CMS to ask the CMTS to duplicate call-related events or even call media streams, and to send them to an interception device.

Length = 20	S-Num = 10	S-Type = 1
DF-IP-Address-For-CDC (IP address where call events should be sent)		
DF-Port-For-CDC	Flags: 1 = Send Events to CDC, 2 = Send Content to CCC	
DF-IP-Address-For-CCC (IP address where call content should be sent)		
DF-Port-For-CCC		

### 4.5.4.3 Sample call flow

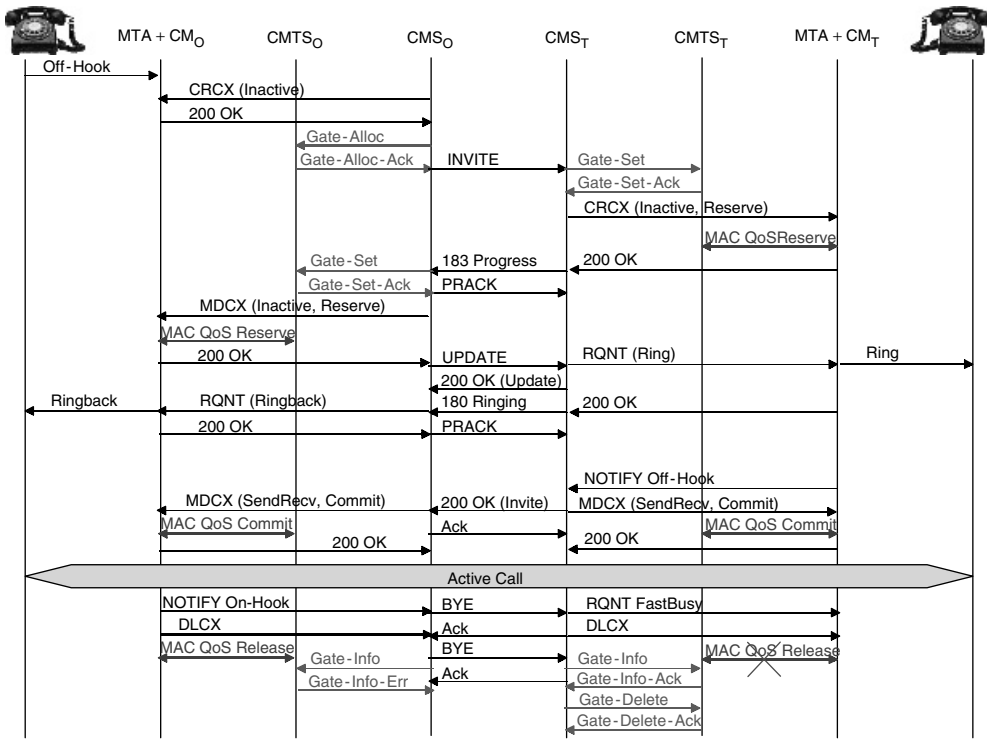
#### 4.5.4.3.1 Embedded MTA, no use of Gate-Open messages

In this first sample call flow (Figure 4.29), the embedded MTAs use layer 2 (DOCSIS) signaling for QoS reservations over the cable plant. The two CMSs use the DCS version of SIP to communicate between them and the NCS version of MGCP to control the MTAs.

CMS<sub>O</sub> (originating) uses the Gate-Alloc message to retrieve a Gate-ID handle from the CMTS. Subsequently, it passes this Gate-ID in all GC commands. The gate is initially in ALLOCATED state.

Note the special use of SIP: the first INVITE does not cause the called phone to ring; instead, CMS<sub>T</sub> (terminating) waits to receive an UPDATE message from CMS<sub>O</sub>. This call flow makes sure that the destination phone does not ring if there are not enough resources end to end:

- The 183 Progress provisional response is sent from CMS<sub>T</sub> to CMS<sub>O</sub> to indicate that sufficient resources have been reserved on the terminating site (and that optionally one-way backbone provisioning from the terminating side to the originating side if RSVP is used in the backbone).
- The UPDATE message from CMS<sub>O</sub> indicates to CMS<sub>T</sub> that sufficient resources have been reserved on the originating side (and that optionally one-way backbone provisioning from the originating side to the terminating side if RSVP is used in the backbone).



**Figure 4.29** EMTA scenario.

In order to secure resources on the cable plant, the CMS first gives an Authorization to the gate indicating certain filters and traffic properties in the Gate-Set message. The gate transitions to AUTHORIZED state. The CMS then indicates to the MTA (via the call control protocol) that the MTA should reserve these resources. MGCP NCS is extended to support the indication of the target Gate-ID (L: dq-gi:<gate-ID>). Using the information provided by the call control, the MTA then reserves the required resources via the MAC-layer protocol and confirms successful reservation via the call control protocol (MGCP 200 OK). If MAC-level reservation takes time, the MTA can send a provisional '100 277 PENDING' response. If the reservation is successful, the gate transitions to RESERVED state.

Once the resources are reserved, the MTA can be sure that a command to commit these resources and begin to use them will not fail. The reserved gate state is soft state, so the reservation should be refreshed if not used quickly. After a commit command from the MTA, the gate is in COMMITTED state.

The CMS also gets involved in checking that the MTA closes the gate properly by releasing resources at the end of the call. In case the MTA fails to do so (as shown for MTA<sub>T</sub>), then the CMS Gate-Info command will succeed because the gate is still in place. If this occurs, the CMS forces the CMTS to close the gate with a Gate-Delete command.

#### 4.5.4.3.2 Stand-alone MTA with RSVP+

If the MTA and the cable modem are physically separate, the MTA cannot use the DOCSIS MAC layer to request resources: so a layer 4 protocol, based on RSVP, is used, which triggers a MAC layer QoS reservation initiated by the CMTS. A sample call flow is described in Figure 4.30.

On the cable plant, full duplex bandwidth is reserved after the first PATH/RESV exchange, because of the extensions of RSVP+. However, the CMTS still sends a PATH toward the MTA because the normal RSVP semantics apply between the cable modem and the MTA; therefore, each PATH/RESV exchange between the CMTS and the MTA only reserve one-way resources in the segment between the cable modem and the MTA.

## 4.6 Improving QoS in the best effort class

While it is easy to understand that a network provider will try to offer low loss rates and low latency to its premium customers, one might wonder what he can do for its best effort customers. This section describes how to improve fairness among best effort customers (each customer must have an equal opportunity to use the capacity of the best effort class) and how to prevent some issues that may occur with TCP.

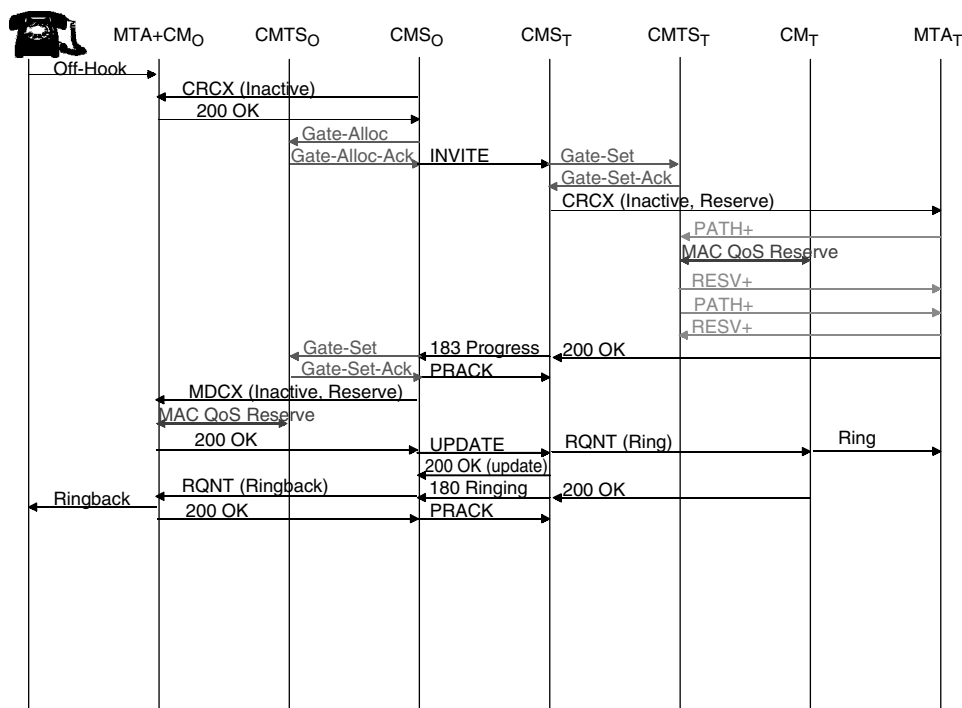


Figure 4.30 Stand-alone MTA scenario with RSVP+.

### 4.6.1 Issues with UDP traffic

UDP traffic has no standard rate-limiting feature in case of congestion. Properly written applications should be able to detect network congestion and react by backing off the rate of UDP traffic (e.g., all applications using RTP/RTCP can detect congestion when RTCP reports increasing packet loss and higher latency). But, on the best effort class where everyone pays a flat fee, it is very tempting to create a UDP application which has a high redundancy scheme and which reacts to congestion not by reducing its bitrate, but by *increasing* it with more redundant and forward error correction packets!

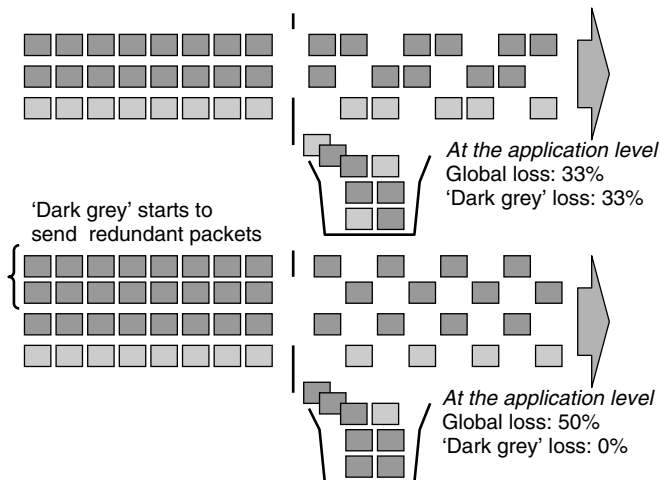
Unfortunately, it will work, because most best effort queues are handled in FIFO mode and packets are dropped when the queue overflows. Statistically, the packets that remain in the queue without being dropped represent a percentage of the offered traffic: so the more you offer traffic to the node, the more packets you get through the node, as shown in Figure 4.31. Such behavior will cause “honest” users to be gradually excluded from the network, while greedy users will enjoy it all.

### 4.6.2 Issues with TCP traffic

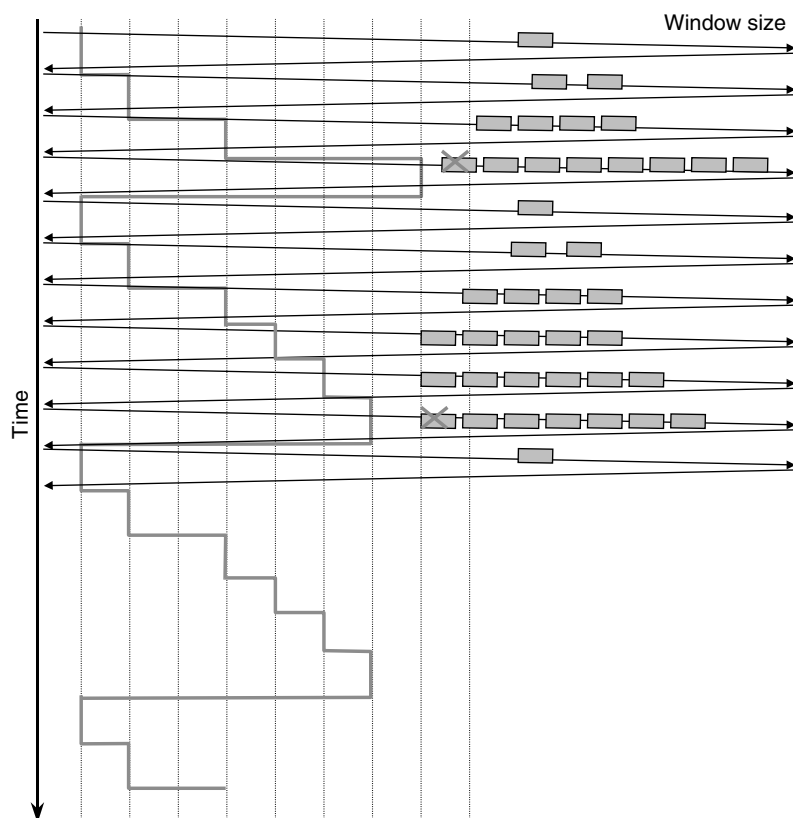
#### 4.6.2.1 TCP congestion avoidance and spontaneous synchronization

All modern TCP stacks implement congestion control algorithms developed by Van Jacobson, as specified in RFC 1122 (and updated in RFC 2001). These algorithms interpret packet loss as an indication that the network is congested and react by slowing down the rate at which TCP injects traffic in the backbone.

The Van Jacobson and Karels congestion control algorithm is one of the most popular. It works by defining, in addition to the usual receiver window size (the maximum number



**Figure 4.31** Greedy application pushing out other sessions.



**Figure 4.32** TCP window size backs off when packet loss is detected.

of bytes that the receiver is prepared to receive and not yet acknowledged), a congestion window. The actual window size used during transmission is the minimum of the receiver window size and the congestion window. Figure 4.32 shows the slow-start mechanism of this algorithm, as well as the TCP stack reaction to packet loss. Congestion window size begins with the size of one segment (512 or 536 bytes unless the other end requests a different size) and gets one segment larger for each acknowledgement received without loss. This exponential growth is characteristic of slow-start mode. If there is a timeout while waiting for an ACK or if there are duplicate ACKs sent by the receiver,<sup>3</sup> the sender deduces that there is some congestion and sets a congestion avoidance threshold to one-half of current window size. Then the sender sets the window size to one segment and increases it in slow-start mode. Once the window size gets larger than the congestion avoidance threshold, the sender increases the window size by  $(\text{segment\_size}/\text{window\_size})$  segments for each ACK, and the window size increases linearly with time.

<sup>3</sup> Modern TCP stacks (like the 4.3 BSD ‘Reno’ stack) implement a specific fast retransmit/fast recovery procedure instead of going into slow start immediately when the receiver sends duplicate ACKs (duplicate ACKs are sent when the receiver receives out-of-sequence packets).



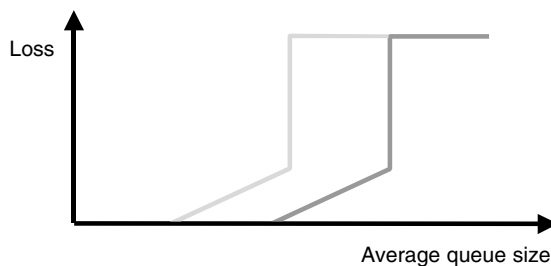
When a FIFO queue overflows, many active TCP sessions are very likely to lose an IP packet *Simultaneously*. These sessions will back off (Figure 4.32) and hopefully the congestion will disappear. Then all these TCP sessions will see that they no longer lose packets and will increase their window size little by little . . . and after a while congestion is back, packets are dropped, and all sessions back off again. This leads to an oscillation of TCP traffic caused by synchronization of VJ algorithms after queue overflow.

### 4.6.3 Using ‘intelligent’ packet discard

In addition to the synchronization problem described above, TCP implementations that incorporate congestion avoidance are network-friendly, while UDP programs or hacked TCP stacks will continue to flood congested links. TCP stacks with congestion avoidance will rapidly back off and progressively let the aggressive traffic use the complete capacity of congested links. Fortunately, it is possible to improve the behavior of the network by discarding packets ‘intelligently’.

**Random early detection (RED)** allows the backlog in routers to be kept as low as possible, while avoiding TCP synchronization. A RED router starts dropping packets at random before the FIFO buffer actually overflows, as shown in Figure 4.33. The session that experiences packet loss should start to reduce its bitrate. Little by little more sessions will experience some random packet loss and reduce the aggregate bitrate smoothly before real congestion occurs. Without this policy, all sessions will experience some synchronized packet loss and reduce their bitrate simultaneously when the buffer overflows, which may cause undesired synchronization and possible oscillations.

With weighted RED (WRED), it is possible to go further and decide to increase packet loss for a class of sessions. **WRED** can be used to do some basic prioritization (e.g., traffic in class ‘premium’ would experience packet loss only after class ‘medium’), but more interestingly it enforces fairness. In Figure 4.31, it is relatively easy to identify the greedy ‘blue’ stream as one getting more bandwidth than it should (e.g., by analyzing lost packets). The normal behavior of a router is to lose packets of a given session proportionally to the traffic offered by the session. We have seen that this actually encourages the use of aggressive redundancy schemes. WRED introduces some nonlinearity (e.g., in Figure 4.31 the router could decide to drop 80% of the ‘blue’ packets).



**Figure 4.33** Probability of packet loss introduced by WRED according to transmission queue size.

## 4.7 Issues with slow links

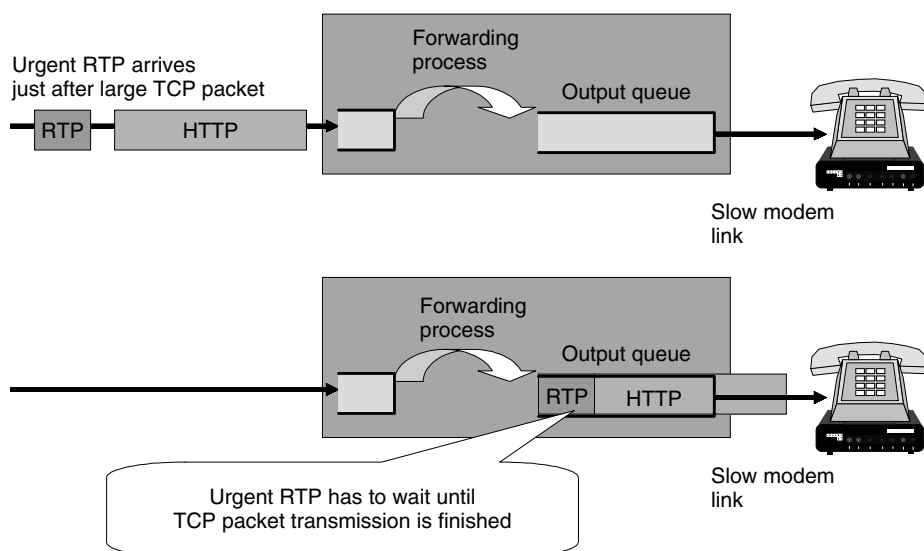
### 4.7.1 Queuing

Slow links are especially challenging for delay-sensitive applications. For instance, a 1,500-octet packet (a common size since it is the MTU over Ethernet) needs over 400 ms to be transmitted over a 28.8-kbit/s PPP link. This would not be an issue if there was only the delay-sensitive application on the link, because this application could use smaller packets. Unfortunately, the line is often shared with other applications using larger packets, such as a web browser, and urgent packets will be delayed if they are queued behind large packets (see Figure 4.34).

### 4.7.2 Overhead

Another issue is that the overhead of the IP suite is quite high (e.g., in each packet of a telephony or video application, IP uses 20 bytes, UDP uses 8 bytes, and RTP uses 12 bytes). The link-layer overhead is also significant: PPP and HDLC take an additional 4 bytes per packet on a modem link, a frame relay link will add 9 bytes per packet. Over ATM the overhead depends on packet size: for an 80-byte packet (40 ms G.729) it adds 25 bytes (this overhead is called the ‘cell tax’).

To see how bad it is, consider what happens with IP phones using the G.723.1 coder. The codec has an audio frame length of 30 ms. In 6.4-kbit/s mode (MP-MLQ), each frame is 24 octets long. When the IP phone stacks only one frame per packet, each IP packet



**Figure 4.34** Latency caused by large unfragmented packets.

is  $24 + 40$  bytes long, and PPP + HDLC will make it 68 bytes long on the PPP link. A packet is sent every 30 ms, and the resulting bitrate is over 18 kbits/s. So only one-third of the bitrate is actual audio information from the G.732.1 codec!

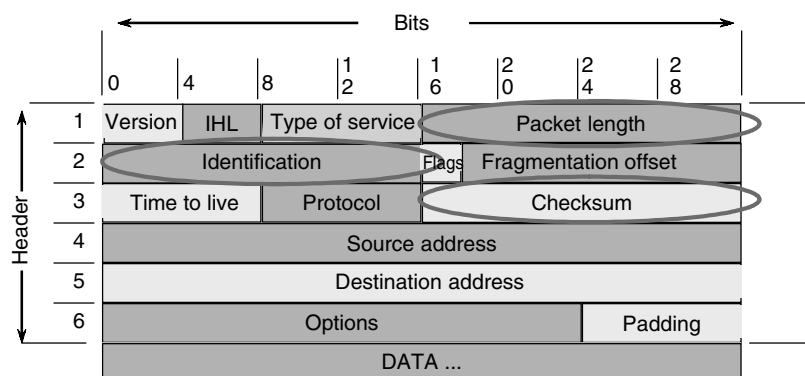
In order to leave some room for data and video over 28.8-bit/s modems, applications such as Microsoft NetMeeting® have to stack several frames per packet. A common choice is to group 4 G.723.1 frames in one packet. This reduces the bitrate to 7.7 kbit/s, but the packetization delay becomes  $4 * 30 = 120$  ms (128 ms including look-ahead). If we assume the best case situation where this packet can be sent immediately, it will need 30 ms to get through the 28.8-bit/s modem link. This results in network delay and again 30 ms at the egress modem ... Unfortunately, this sets conversational delay to a level that is unacceptable for most non-hobby users.

### 4.7.3 Overhead compression

The overhead issue can be solved by using header compression. Header compression is based on a simple idea: since about half of the overhead of an IP/UDP/RTP packet is constant for a given stream (e.g., the source and destination IP addresses and ports are constant), it is therefore possible on a point-to-point link to negotiate a shorter ‘index’ for those constant values when the stream is set up. Moreover, some fields increment by constant values and can be reconstructed at the receiving end if a proper per-stream state is maintained.

Such an IP header compression algorithm has been standardized by M. Engan, S. Casner, and C. Bormann in RFC 2509 ‘IP Header Compression over PPP’ (February 1999), and extended by S. Casner and V. Jacobson for applications using RTP in RFC 2508 ‘Compressing IP/UDP/RTP Headers for Low-speed Serial Links’ (February 1999). This method is also called **compressed RTP (CRTP)** and can be negotiated as part of the serial link protocol negotiation (e.g., as PPP options). More recently there was some work to improve the tolerance of header compression to packet loss: an improved robust header compression (ROHC) has been defined in RFC 3095 mainly for 3G networks. Most of the time, the IP/UDP/RTP headers are compressed to just 2–4 bytes (2 bytes if the UDP checksum is not used) instead of 40 for a full IP/UDP/RTP header. While compression can be done end to end for RTP alone (preserving IP-addressing information), a compression protocol for all three protocols (IP/UDP/RTP) can only work on a link-per-link basis (IP-addressing information is not available in compressed packets).

With CRTP, the sending host replaces the large header with a small index to a session context and differential values for variable fields. The receiving host reverses the operation. In each IPv4 header of a packet stream belonging to a given context (Figure 4.35), only packet length, packet ID, and header checksum will change. The packet length indication is redundant with framing of the link layer and CRC is already provided by link layer: both can be reconstructed. Packet ID changes by one or a small increment for each packet, so only the increment value needs to be in the context. For UDP, the length field is redundant. For RTP the SSRC value is constant for a context and the sequence number is incremented by one unless packets are disordered. The RTP timestamp is incremented by multiples of the frame size for audio (e.g., 2 if there are 2 codec frames in each RTP



**Figure 4.35** Very few fields change between IP packet parts of the same flow.

packet), and for video it will change for each first packet of a video frame description, then remain constant for RTP packets containing the rest of the video frame description. Only the increment size needs to be in the context. The RTP M bit will change each time speech commences but would take too much overhead if compressed (e.g., there would be a need to resynchronize context at each change).

In each CRTP packet, the context is identified by a session **context identifier (CID)** of 1 or 2 octets and a 4-bit serial number is added to detect packet loss on the link. Each host can generate multiple RTP sessions and, therefore, a session context must be associated with each group (source IP address, source port, destination IP address, destination port, and RTP SSRC field). The link layer (e.g., PPP) also needs to convey an indication of the format of the packet (no compression: **FULL\_HEADER**; only IP + UDP header compressed in 6 bytes or just 2 if UDP checksum is disabled: **COMPRESSED\_UDP**; IP + UDP + RTP headers compressed, generally just 2 bytes in total: **COMPRESSED\_RTP**; and packet indicating that a context identifier is invalid: **CONTEXT\_STATE**). A bit mask indicates which fields differ from their prediction and, therefore, conveyed in differential form in the compressed packet. Any packet that cannot be predicted using one of the compressed formats (unusual change) will be sent in **FULL\_HEADER** format. The context contains:

- Full IP/RTP/UDP headers, initialized with values found in the **FULL\_HEADER** packet.
- Increment values of IP packet ID, RTP timestamp, initialized by comparing the values of the first two transmitted stream packets (the increment value for the RTP sequence number is implicitly set to one).
- Current 4-bit sequence number.

The CRTP compression scheme is optimized for duplex links and uses a backward **CONTEXT\_STATE** packet to signal a loss of synchronization between coder and decoder. All transmitted packets are lost as long as CRTP contexts are not resynchronized.

The scheme works best with RTP packets, but since the compression scheme is able to reconstruct the IP packet exactly (lossless), it will not cause any harm to a UDP packet

improperly recognized as an RTP packet. A pair of routers using CRTP can therefore use relatively simple pattern matching to recognize potential RTP packets, even if there is a small probability that some non-RTP UDP packets will match by error. The routers need to keep a cache of contexts (which get invalidated too frequently) in order to avoid using RTP compression for these data streams.

It is interesting to note that IP and TCP header compression has also been introduced in the UMTS standard for transport of IP packets over the UTRAN (Universal Terrestrial Radio Access Network) radio link. The PDCP (Packet Data Convergence Protocol) layer compresses IP headers using RFC 2507 ('IP Header Compression') and TCP/IP headers using RFC 1144 ('Compressing TCP/IP headers for Low-speed Serial Links'). Unfortunately, RTP compression has not yet been defined for PDCP, but PDCP is extensible; therefore, it is likely that RTP compression will also be supported one day, especially if mixed-mode VoIP/WiFi and UMTS 3G terminals become more common.

#### 4.7.4 Packet fragmentation, prioritization over serial links

The queuing issue can be solved by allowing real-time packets to pre-empt non-real time packets. Two techniques can be used for pre-empting:

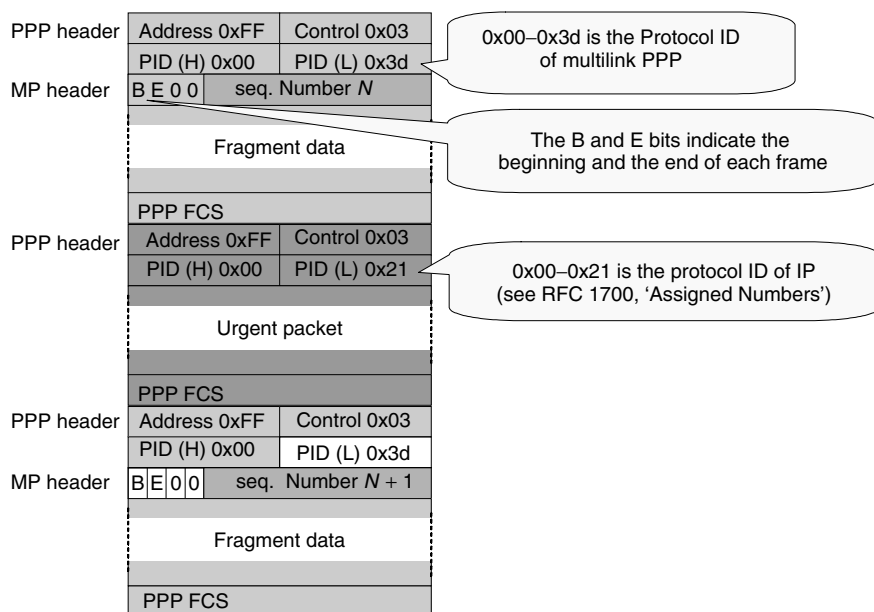
- Using the **multilink PPP (PPP-MP)** protocol (RFC 1990), also called **link fragmentation and interleaving (LFI)**. The original design of multilink PPP was to allow the bundling of several physical or logical links into a single virtual link. Compared with plain PPP, PPP-MP provides additional means to sequence and correlate fragments arriving from several links. Since multilink PPP is able to fragment packets, it can be used to stop transmission of a long packet, send an urgent packet, and resume transmission of the long packet. Figure 4.36 shows the multilink PPP bitstream format, when the multilink short-sequence number fragment format is used (only 12 bits instead of 24, and PPP compression of address, control, and most significant PID bytes assumed active). This simple solution has the advantage of being compatible with existing PPP-MP implementations, but cannot be used for more than one pre-emption level. This is because the PPP-MP specification requires the sequence numbers of fragments of a packet to be monotonically increasing and contiguous.<sup>4</sup> Therefore the only way to send an urgent packet between fragments is to send it without the PPP-MP header,

---

<sup>4</sup> Extract from RFC 1990: "On each member link in a bundle, the sender **MUST** transmit fragments with strictly increasing sequence numbers (modulo the size of the sequence space). This requirement supports a strategy for the receiver to detect lost fragments based on comparing sequence numbers. The sequence number is not reset upon each new PPP packet, and a sequence number is consumed even for those fragments which contain an entire PPP packet, i.e., one in which both the (B)eginning and (E)nding bits are set."

An implementation *must* set the sequence number of the first fragment transmitted on a newly constructed bundle to zero. (Joining a secondary link to an existing bundle is invisible to the protocol, and an implementation *must not* reset the sequence number space in this situation).

The receiver keeps track of the incoming sequence numbers on each link in a bundle and maintains the current minimum of the most recently received sequence number over all the member



**Figure 4.36** Insertion of an urgent packet in the PPP multiplex. Another limitation to using PPP for QoS purposes is that it cannot be used on bundles of multiple serial links, as there is no sequencing information for inserted priority packets, which therefore would not be kept in sequence.

which is allowed by RFC 1990. An urgent packet interleaved between two fragments of a long packet are shown in Figure 4.36 (HDLC 0×7E flags at the beginning and end of each frame are not represented).

- RFC 2686, ‘**The Multi-class Extension to Multi-link PPP**’ (MCML), addresses this limitation and makes it possible to manage a priority class over multiple serial links. Delay-sensitive traffic (‘class 0’) is also fragmented just like pre-empted non-delay-sensitive packets (‘class 1’). All sequence numbers are managed per class, resolving the ambiguities that prevented doing the same on multilink PPP.

Both techniques should be used over links with relatively low packet loss and low round trip delays, which is usually the case with access links. Some enhancements of ML-PPP have been proposed to address these limitations, but they are not widely implemented. ML-PPP and MCML are intended to be used on links with typically less than 2 Mbps of bandwidth; for higher speed links the gain on delay is small and there may be excessive impact on router CPU usage, especially for a service provider aggregation router if it has to manage several customer serial links.

links in the bundle. The receiver detects the end of a packet when it receives a fragment bearing the (E)nding bit. Reassembly of the packet is complete if all sequence numbers up to that fragment have been received.

## 4.8 Conclusion

Quality of service over packet networks is not a trivial issue, regardless of the technology. There has been a lot of theoretical work lately, and a consensus on the deployment strategy needed to build a large, QoS-aware, IP backbone is slowly emerging.

The main factor driving the latency aspect of quality of service is the length of the data packet divided by the transmission link speed. Small packets are preferable because they tend to introduce less latency: this is the background behind the advantage of using native ATM for small to medium links, typically below 1 Mbit/s. However, small packets also come with a big disadvantage: they create a lot of overhead (sometimes called the 'cell tax' in ATM); therefore, if the bandwidth is large enough, large packets become preferable because they are more efficient.

From this point of view, IP is clearly the technology of the future. In most IP backbones, bandwidth is already more than adequate to reduce packet size-related latency to something insignificant. The same is true within corporate LANs, with 10/100/1,000 Mbit/s Ethernet links deployed massively. The last problematic bottleneck in many networks is the local loop. Non-broadband techniques (ISDN, modem) require fragmentation techniques to reduce link latency. Today, first-generation broadband techniques, such as cable or ADSL, offer barely the bandwidth necessary to transmit IP packets without fragmentation in the upstream direction (about 512 kbit/s to 1 Mbit/s maximum) and are often restricted further by service providers. Second generation broadband techniques will offer a more comfortable bandwidth: ADSL2 will offer 10 Mbit/s downstream and 1.2 Mbit/s upstream,<sup>5</sup> thus offering better support for IP-level quality of service over multiple parallel transport channels.

ADSL2+ doubles the downstream bandwidth figure by doubling the occupied frequency spectrum. For non-residential use requiring symmetric bandwidth, SDSL is already more than adequate for transporting low-latency IP packets (Table 4.3).

The deployment of xDSL technologies is a very big success, but they will probably increasingly compete with Ethernet as a first-mile technology, over a fiber to the building infrastructure. These techniques are promoted by the Ethernet First Mile Association (EFMA) and the Metro Ethernet Forum (MEF). Some cities already have a significant coverage of fiber to the building infrastructure, e.g., Milan—Figure 4.37—with a 100,000 homes covered by a combination of fiber provision in the building and Ethernet to the home. Optical fiber in the local loop will remove the last bottleneck for end-to-end quality of service at 100 Mbps and over.

In this chapter, we focused on IPv4. There is sometimes the perception that IP will be able to provide QoS only once IPv6 is deployed. This is clearly wrong. All important QoS mechanisms present in IPv6 are also available in IPv4, and IPv4, given the current state of access loop technology, creates much less overhead. With DiffServ and the ever-improving layer 2 technologies, IPv4 is now ready for widespread deployment of applications requiring quality of service, like VoIP. The last area where significant

---

<sup>5</sup> On average ADSL2 offers 50 kbit/s more bandwidth than ADSL, the 200-kbit increase of the upstream channel is obtained by taking the bandwidth previously reserved for baseband voice ... assuming voice over packet will be used!

Table 4.3 DSL performance table

Technique			Downstream (Mbps)	Upstream (Mbps)	Max bandwidth distance (km)
Assymetric	ADSL with splitter	ADSL/RADSL	<8.2	<1	2.5
		ADSL2 (G.992.3/G.992.4)	<12	<1.2	3
		ADSL2+ (G.992.5)	<26	<1.2	1.5
		VDSL	<26	<9	1
	Splitterless ADSL	UDSL/UADSL/RDSL/CDSL/ DSLlite/ADSLlite/Glite	<1.5	<0.512	≈2
Symmetric	SDSL	1 twisted pair:	<2.3		3
		SDSL/SPDSL/HDSL(G.991.1)/ HDSL2 (ANSI T1418)			
		SHDSL (G.991.2) in single-pair mode	<2.3		4
	Ethernet	IEEE P.802. ah EFM copper (EFMC) Metro Ethernet Forum E-line, user network interface	10		0.75
		IEEE P.802.ah EFM single-mode fiber (EFMF)	100–1,000 ps		10
		IEEE P.802.ah EFMP passive optical network-shared fiber with optical splitters	1,000		20



Figure 4.37 Fiber deployment in Milan (FastWeb).



improvements are still desirable is access control, in order to secure IP backbones against DoS attacks, which become more threatening as the bandwidth of end-user access links increases. This seems to be where IntServ may have a much more pre-eminent role in the future.

## 4.9 References

- A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks, Part I. *IEEE/ACM Transactions on Networking*, **1**(3), pp. 344–357 (June 1993).
- A.K. Parekh and R.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), pp. 137–150 (April 1994).
- S.J. Golestani. *A Self-clocked Fair Queuing Scheme for Broadband Applications*. Bellcore, ATT Research Labs.
- N.R. Figueira and J. Pasquale. An upper bound on delay for the virtual clock service discipline. *IEEE/ACM Transactions on Networking*, **3**(4) (August 1995).
- V. Jacobson. Congestion avoidance and control. *Proceedings of ACM SIGCOMM '88*, pp. 314–329 (August 1988).
- S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, **1**(4), pp. 397–413 (August 1993).
- [RFC 1812] Requirements for IPv4 Routers (F. Baker).
- [RFC 2205] Version 1 Functional Specification of RSVP.
- [RFC 2211] Specification of the Controlled-load Network Element Service.
- [RFC 2212] Specification of the Guaranteed Quality of Service.
- [RFC 2474] Definition of the Differentiated Services Field in the IPv4 and IPv6 Headers.
- [RFC 2475] An Architecture for Differentiated Services.
- [RFC 1812] Requirements for IPv4 Routers.
- [RFC 2508] Compressing IP/UDP/RTP Headers for Low Speed Serial Links (February 1999: S. Casner, V. Jacobson).
- [RFC 1990] The PPP Multilink Protocol (MP) (August 1996: K. Sklower, B. Lloyd, G. McGregor, D. Carr, T. Coradetti).
- [RFC 1661] The Point-to-Point Protocol (PPP) (July 1994: W. Simpson, editor).
- [RFC 1662] PPP in HDLC-like Framing (July 1994: W. Simpson, editor).