Packet Handling

Seminar on Transport of Multimedia Streams in Wireless Internet

Davide Astuti
Department of Computer Science
University of Helsinki
Helsinki, Finland
davide.astuti@cs.helsinki.fi

Abstract **This paper describes a number of packet handling mechanisms on a network, related to classification of packets, admission control, traffic shaping, queuing and scheduling disciplines and packet discard techniques. An overview about packet handling in wireless networks is provided.**

Keywords: IP QoS, FIFO, PQ, FQ, WFQ, CBQ, HTB, RED

I. Introduction

The current Internet is based on IP protocol and supports only best effort services. With the exponential growth of Internet during the last years, IP networks are expected to support not only typical services like ftp and email, but also real-time services and video streaming application. The traffic characteristics of these applications require a certain Quality of Service (QoS) from the network in terms of bandwidth and delay requirements.

The biggest problems in a network are related to the allocation of network resources, as buffers and link bandwidth, to different users. A limited amount of resources has to be shared among many different competing traffic flows in an efficient way in order to maximize the performance and the use of the network resources. The behavior of routers in terms of packet handling can be controlled to achieve different kind of services.

This paper analyzes a number of packet handling mechanisms, which are related to:

- Classification of packets
- Admission Control and Traffic Shaping
- Queuing and scheduling disciplines
- Packet discard techniques

Packet classification indicates the process of categorizing packets into flows in a router. All packets belonging to the same flow or the same class of service are ruled in a predefined manner and are processed in a corresponding way by the router. For example, all packets belonging to a certain application or related to a specific protocol may be defined to form a traffic flow. Packet classification process is needed for those services that require the distinction between different kinds of traffic.

Once packets are classified, admission control functions are performed to check whether enough network resources are available according to the packet service classification. If resources are available, packets are handled accorging to their classification. Traffic shaping functions are used to control the volume of traffic entering in the networks and the rate at which packets are sent. This paper describes two methods for achieving Traffic shaping: leaky bucket and token bucket.

Afterwards, packets are scheduled into queues. Queues are managed in a way to ensure each queue gets the level of services required for its class. A number of queue disciplines are described: First-In-First-Out (FIFO), Priority Queuing (PQ), Fair Queuing (FQ), Weighted Fair Queuing (WFQ), Class Based Queueing (CBQ). A comparison between queue disciplines is provided, giving advantages and disadvantages of each discipline and the area of applicability. The Hierarchical Token Bucket (HTB) method to provide QoS in Linux is outlined.

When network congestion is experienced, packets in excess must be discarded. Packet discard cannot be avoided but can be controlled. Several techniques for handling packet discard are showed, such as Tail Drop, Early Drop, Drop Preferece and RED.

Finally, an overview of how packet handling is achieved in wireless environment is provided. When not otherwise specified, the used reference is [Huston00]. This paper is structured as follows. Packet classification is introduced is section II. Admission control and Traffic Shaping is described in section III. The queueing disciplines are described in Section IV. In Section V, packet discard methods are presented. Section VI gives an overview of packet handling in wireless environment. Finally, a summary of the paper content is provided in Section VII.

II. Packet Classification

Today, Internet routers usually provide only best-effort services and packets are treated in a classical first-in-first-out manner. The need for QoS requires new mechanisms of packet handling. First of all, the router has to distinguish packets belonging to different class of services. Classes of services are specified by rules applied to incoming packets. A collection of rules is called classifier.

By inspecting the packet header, the packet is classified to several service classes and consequently is handled accordingly. The packet header determines also the congestion management of the packet. For example, a packet may be marked as low priority packet so that it may be discarded in case of congestion.

III. Admission Control and Traffic Shaping

Admission control is a mechamism to estimate the level of QoS that a new traffic flow needs and to check whether enough resources are available to serve it. If network resources are available, packets are "admitted" to flow into the network.

Admission control is not mandatory. In many cases, the network accepts the traffic from a new user session without any check about the available resources. However, this behaviour may lead to an overload in the network as uncontrolled use of network services is allowed. Without admission control, the network cannot provide different service classes in an efficient way because it is not possible to prevent a service class from using more than the maximum amount of network resources reserved for that class.

A network providing differentiated services [RFC2475] must accomplish admission control. Once the packet is classified, it is passed through an admission filter to check whether enough network resources are available according to the packet service classification. The admission control monitors and enforces the use of network resources according to a certain policy. The policy criteria can be various and depends on the environment; it may include identifying users/applications or selecting traffic based on the source, destination, traffic type and so on. [RFC2753] describes a policy-based control framework for admission control.

An easy technique to provide admission control is to service flows based on arrival time. This is called CAC (capacity-based admission control). With this method, new flows are admitted until capacity is exhausted (First Come First Admitted) without any additional check. New flows are not admitted until network resources are available.

Another method to achieve admission control consists of allowing some flows to preempt others according to certain priority settings. This method has the advantage of providing an efficient premium service as the amount of resources used by the low priority traffic is controlled and limited. On the other hand, adding high priority traffic can affect the performance of lower priority traffic, so priority queued systems must use sophisticated admission control algorithms. [RFC2751] and [RFC2815] describe admission control in a priority-queued system.

Once the traffic is admitted to the network (through admimission control functions), traffic shaping practices are used to control the volume of traffic entering in the networks and the rate at which packets are transmitted. In this way, the flow of packets is smoothed according to the configured profile of traffic load on the network.

Traffic shaping can be implemented using a leaky bucket filter. The bucket is essentially a regulated queue. A traffic source can produce packets at quite unsteady and unconstant rate, which trickles into the bucket. Packets flowing into the bucket are paced out in steady way at a specified rate. We can imagine that the bucket has an imaginary hole in the bottom and packets can trickle into it, in analogy with a real bucket full of water with a hole. Water trikles out of the bucket little by little, independently of how quickly the bucket is filled up. Using a leaky bucket filter does not allow sending burst of packets, but only at a specified rate. If there is no traffic for a certain period of time, the amount of unused bandwidth cannot be used for later packets; this is achieved by using a token bucket filter.

A. Token Bucket

A finer way to implement traffic shaping uses a token bucket filter, which allows bursts of packets to be sent every now and then. The tocken bucket filter is a way to implement both admission control and traffic shaping. The philosophy is the same than the leaky bucket's one, however the tocken bucket allows having permission to send bursts of packets. Figure 1 shows the token bucket scheme. The algorithm is that every certain period of time – which determines the rate packets are served – a token is saved (tokens are marked with "T" in the figure). Each token can be associated to one packet (blue packets in the figure) or, in general, to a defined quantity of data. When a packet is sent, one of these tokens is removed. If there are no packets to send, tokens can be retained and used to send new packets later on. In the figure, four tokens are retained in the bucket; hence, a sudden burst of four packets can be served and allowed to go through the network.

When a packets arrives and there are not enough tokens retained in the bucket, the action can be various according to how the Token bucket filter is configured: the packet can be buffered and not discarded until new tokens are available; the packet can be marked and, hence, treated in particular way; the packet can be discarded.

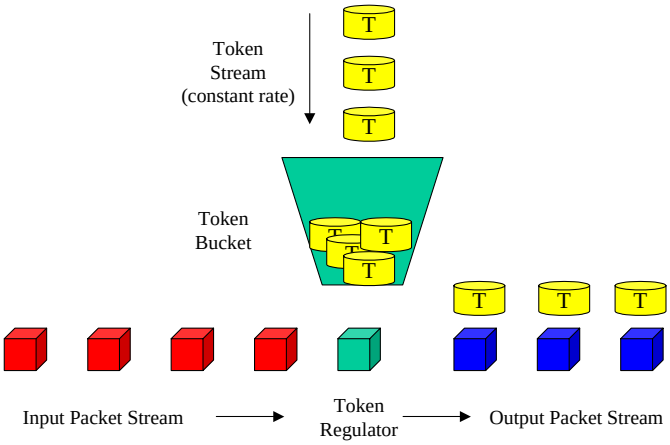

Figure 1. Token Bucket

We can see why the Token Bucket filter implements both Admission control and Traffic Shaping functions. If packets arrive at a rate exceeding the maximum allowed rate for their service class, only the number of packets matching the number of retained tokens is "admitted" to flow into the network. At the same time, the rate at which packets are sent is "shaped" as tokens are accumulated at a specified rate.

IV. Queuing and Scheduling Disciplines

In a network, packets are accumulated and queued into memory buffers of routers and switches. The most common way to arrange packets is called first-in-first-out (FIFO), but various methods may be used to prioritize packets or ensure

that all packets are handled in fair manner, rather than allowing one source to use more than its share of resources.

Packets may arrive at routers in bursts from multiple devices, and a node may sometimes receive more packets than it can process. Buffers hold packets until the router can handle them. If the router is overloaded, buffers fill up and overflow. The most common method applied in this case is called tail drop and consists of dropping the new incoming packets that does not fit in the buffer. This is not the best possible way for handling queues, but it is fast.

This paragraph analyzes the operation, benefits and drawbacks for a number of queue discipline mechanisms. The analyzed queue disciplines schemes include:

- First-in-first-out (FIFO)
- Priority Queuing (PQ)
- Fair Queuing (FQ)
- Weighted Fair Queuing (WFQ)
- Class Based Queuing (CBQ)

A.FIFO

The FIFO discipline is the basic first-in, first-out queuing method. It is very simple: the first packet in the queue is the first packet that is served. In FIFO, all packets are treated in the same way: they are placed in a single queue and are served in the same order they were placed. When the queue becomes full, congestion occurs and the new incoming packets are dropped. Figure 2 shows an example of FIFO. Packets from different flows arrive at the router, which multiplexes them into the same FIFO queue in the same arrival order (1,2,3,4,5,6).

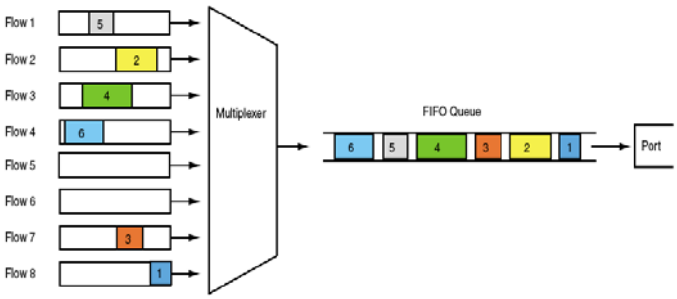

Figure 2. First-In-First-Out (FIFO) queuing [Semeria].

The main benefits of FIFO are described as follows. The FIFO technique is extremely simple when compared with other more elaborated queue disciplines. Consequently, it is a good solution for software-based routers. FIFO does not make packet reordering and the maximum queue delay can be easily calculated: it is determined by taking into account the maximum length of the queue. As long as there is no congestion and the FIFO queue is not full, the contention resolution for network resources is provided in a fast way because of the simplicity of this technique.

On the other hand, FIFO has drawbacks. Since all packets are buffered in the same queue, FIFO does not allow handling packets of different classes in different way. Therefore, it is not possible to provide different Classes of Service. The queuing delay increases as congestion increases and such additional delay affects all queued packets. Consequently, real-time applications or, in general, applications with strict delay and jitter constraints, can suffer and experience bad performance.

Moreover, during network congestion FIFO benefits non-congestion-responsive flows (UDP) over congestion-responsive flows (TCP). When a TCP packet is lost, TCP infers that the network is congested and reduces its sending rate. On the other hand, when a UDP packet is lost, UDP does not respond to such an event and continue the transmission at its normal sending rate. Consequently, FIFO queuing can result in a form of bandwidth starvation for the congestion-responsive flow (TCP) [FF97]. The most severe example of this situation is where an UDP flow transmits packets at a sending rate higher than the available bandwidth. Under these load conditions, the FIFO queue will be always full, apart one empty slot at the tail of the queue every now and then. Such a slot is the unique one a TCP flow can use since it adapts and reduces its sending rate according the estimated network congestion [FF97]. FIFO does not help to resolve this problem of fair allocation of network resources.

B.PQ

Priority Queuing (PQ) is a method to provide in a simple way a differentiated service. This technique uses multiple queues. Queues are serviced with different levels of priority and queues with higher priority are serviced first. Packets are placed in one of the queues according to their classification. Packets are scheduled from a certain queue only if the higher priority queues are empty. Packets are scheduled in FIFO order within each priority queue. In case of congestion, packets are dropped from lower priority queues.

Figure 3 shows an example of Priority Queuing. In this case, three priority queues are used. Flow 3 is classified as highest priority flow, flows 2 and 7 as middle priority flows and the other flows as lowest priority flows. Firstly, packets from the highest priority queue are served in FIFO order. Only once the highest priority queue is empty, the remaining queues will be served using the same principle.

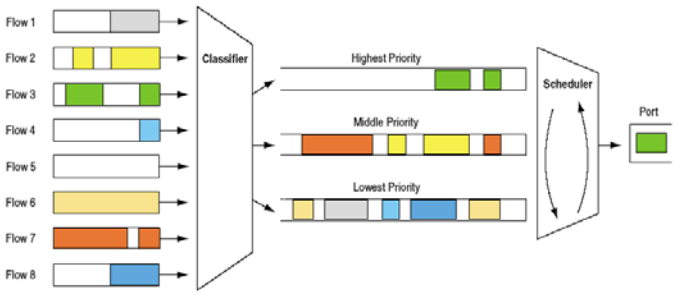

Figure 3. Priority Queuing (PQ) [Semeria].

The main benefit of PQ is that packets of different classes can be managed using different queues and, consequently, a certain class of traffic can be handled in different way than another one. Another advantage is that the PQ technique is relatively simple when compared with other more elaborated queue disciplines.

The drawbacks of PQ are outlined as follows. If the amount of highest priority traffic is excessive then the lower priority queue may not get any service until the highest priority traffic is served completely. During this while, the queues allocated to lower priority traffic may overflow. As a result, the lower priority traffic may experience a large delay or, in the worst case, a complete resource starvation. A solution to this important problem consists of limiting the maximum amount of highest priority traffic (in general, limiting all the priority queues) that can be accepted using appropriate admission control policies.

The FIFO drawback regarding the interaction between TCP and UDP flows (bandwidth starvation for the TCP flows) is not resolved using PQ. A simple use of PQ to cope with such a problem would be to map TCP flows into a higher priority queue than UDP flows in order to compensate the non-congestion-responsive behavior of UDP. This is not a good solution because the TCP algorithm will try to use all the available bandwidth causing a bandwidth starvation for the lower priority queues.

C.FQ

Fair Queuing (FQ) has been designed to avoid the problem of PQ discipline, where one flow can monopolize and use all the available bandwidth of the network and, hence, lead the lower priority flows to bandwidth starvation. In FQ, arriving packets are classified into different flows and stored into a queue dedicated to that particular flow. The round-robin algorithm is used to service all the queues, so that queues are served in a fair way and one source cannot use more than its share of network bandwidth. Figure 4 illustrates an example of FQ. Each flow is redirected to its own queue. Queues are scheduled in round-robin way.

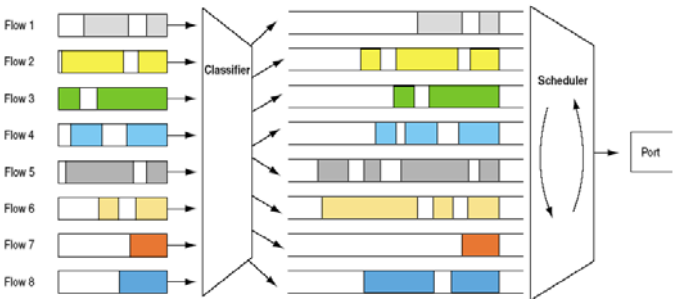

Figure 4. Fair Queuing (FQ) [Semeria].

The main advantage of FQ is that bursty flows do not affect the overall performance. If a flow is too bursty, then only its queue will be filled up, without any interference with any other flow.

The drawbacks of FQ are outlined as follows. The scheduler takes a packet at a time from each queue regardless of the packet length. For example, if a queue contains bigger packets than other queues, then that queue will use a larger portion of the total network bandwidth and it will take more time to be served. Furthermore, the FQ algorithm is more complicated than FIFO and PQ; hence a software implementation (the most common) may result in performance problems. In general, FQ is good to share the same portion of bandwidth among many flows, but it cannot be used for handling different flows bandwidth requirements. Moreover, there is no easy way to provide real-time services.

A variant of FQ is Stochastic Fair Queuing (SFQ) [McKe90]. The difference between FQ and SFQ is in the way the arriving flows are mapped into queues. The number of queues is limited and a hash function is used to separate traffic flows into separate FIFO queues, which are served in a round-robin way. The hash function takes into account the source/destination address, the port number and other protocol parameters. If the number of incoming flows is larger than the number of queues, multiple flows will get hashed to the same queue. It may happen that the hash function maps many flows in the same queue meanwhile there are queues without any data or few data to be sent. In order to avoid such an unfairness situation, the hashing algorithm is perturbed at predefined time intervals so that the mapping between flows and queues periodically changes. Perturbation may however cause packet reordering to occur. SFQ is one of the disciplines available in Linux Traffic Control [LARTC].

D.WFQ

The idea of the Weighted Fair Queuing (WFQ) algorithm can be easily explained as a combination of PQ and FQ algorithms. As in the FQ method, all queues are served so that there is no bandwidth starvation, but some queues have more weight in a sense that they receive more service. In other words, a weight is given to each queue to assign different priorities to the queues. Packets are stored into the appropriate queue according to their classification.

The above discussion gives a general idea of the algorithm; the complete algorithm is a little bit more complicated and it is an approximation of the Generalized Processor Sharing (GPS) system [Keshav97]. The algorithm is explained as follows. A finish time is assigned to each packet taking into account the link bandwidth, the number of queues, the weight of queues and the packets length. Afterwards, the scheduler serves the queue where the packet with the minimum finish time is stored. The finish time is used to build the order in which the packet will be transmitted through the link.

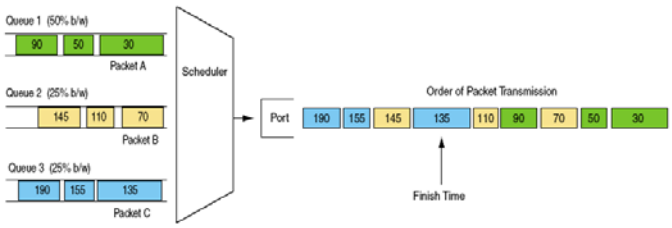

Figure 5. Weighted Fair Queuing (WFQ) [Semeria].

Figure 5 shows an example of the WFQ algorithm. The assigned finish times are only for a qualitative use. At the beginning, three packets are given to the scheduler. The queue 1 has a weight of 50%, queue 2 and 3 have a weight of 25%. Hence, the finish time for one packet in queue 1 is less than for one packet of the same length in the other queues with lower weight. In this example, the first two packets in queue 1 get

finish time of 30 and 50 respectively; therefore they are forwarded before the packet in queue 2, labeled with a finish time of 70. As Figure 5 shows, giving a high weight to a queue allows the scheduler to take more than one packet from that queue. The order of packets transmission is shown on the right part of the figure.

The main benefit of WFQ is that it provides a minimum level of network resources to each configured service class. On the other hand, WFQ is a complex algorithm requiring a per-service class state and iterative checking of all states for each packet arrival and departure. Consequently, the WFQ presents scalability problems and cannot be used in environments with high volume of traffic requiring many classes of service. One possible application of WFQ is where the number of service classes is limited to a small number, so that the computational load is bounded.

Since WFQ was invented, many improvements have been proposed, which include Class-based WFQ, SCFQ, WF2Q. The interested reader can refer to [Semeria] for more information.

E.CBQ

The main idea behind Class-Based Queuing (CBQ) is scheduling packets in the queues and guaranteeing a certain transmission rate. If there are no packets in a queue, its bandwidth is made available to other queues. The strength of this method is that it allows to cope among flows with considerably different bandwidth requirements. This is done by assigning a specific percentage of the link bandwidth to each queue. CBQ also avoids the bandwidth starvation problem of the PQ method as at least one packet is served from each queue during each service round. CBQ and WFQ may look similar as they try to achieve the benefits of PQ and FQ. One difference is that CBQ scheduling is packet-oriented and does not take into account the packet length.

The algorithm is outlined as follows. Firstly, packets are classified into a service class according certain criteria (addresses, protocol, users, etc…) and stored in the corresponding queue. Each queue is served in round robin order. A different amount of bandwidth can be assigned for each queue in two different ways: allowing a queue to send more than one packet for each service round or allowing a queue to send only one packet for each service round but the same queue is visited more than one time during the same service round.

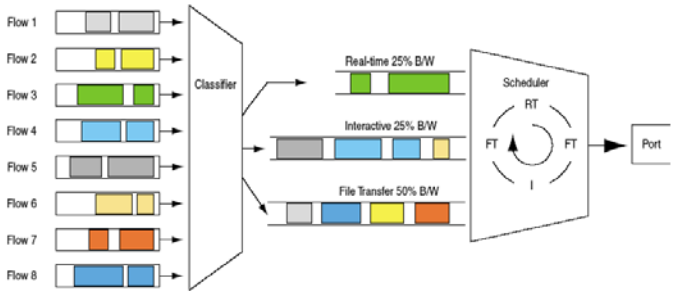

Figure 6. Class-based queuing (CBQ) [Semeria].

Figure 6 shows an example of CBQ. Incoming traffic is classified in three service classes: Real-Time (RT), Interactive (I) and File Transfer (FT). In this example, half of the available bandwidth is reserved for the file transfer service class; the remaining bandwidth is equally shared between real-time and interactive class. Using these weights (25%,25%,50%), the File Transfer queue is visited two times during one service round.

One of the main benefit of CBQ is clearly shown in this example, where we can see why the bandwidth starvation problem does not rise using the CBQ method: each service class receives a certain amount of service for each service round and the case present in PQ, where low-priority queues do not receive any service if higher priority queues contain packets, never happens.

CBQ gives control over the amount of bandwidth each service class can use and it can be easily implemented in hardware, hence there are no the performance limitations of other queue disciplines.

The main drawback of CBQ is that it can provide fair allocation of the bandwidth only if packets of all queues have the same (or comparable) size. If one service class contains longer packets than the other ones do, the service class with bigger packet size will take more bandwidth than the configured value. The interested reader can found a complete research on CBQ in [CBQ].

F.A Comparison between queue disciplines

In this paragraph, a comparison between the outlined queue disciplines is introduced as each method presents benefits and drawbacks. In addition, suitable areas of applications for each queue method are reported. Table 1 summarizes advantages and disadvantages for each queue discipline.

Queue discipline	Advantages	Disadvantages
FIFO	Extremely simple. No packet reordering. Maximum delay easy to predict.	No service class supported. Queuing delay increases as congestion increases. No fairness (bandwidth starvation).
PQ	Supports different service classes. Relatively simple.	Bandwidth starvation problem. No fairness.
FQ	Support different service classes. Bursty flows do not affect other flows. Allows share the same bandwidth among many flows.	Packets must have the same size for efficient scheduling. No way to provide real time services. No way to serve flows with different bandwidth requirements. High complexity.
WFQ	Gives a minimum amount of resources to each service class. Bursty flows do not affect other flows. Allows sharing the same bandwidth among many flows.	High complexity.
CBQ	Gives a minimum amount of resources to each service class. Limited complexity.	Packets must have the same size for fair scheduling.

Table 1. Pro and cons of queuing disciplines

FIFO discipline is very simple and therefore can be easily scaled in large queues and used in fast-switching networks. FIFO behavior is easy to predict and no packet reordering is performed. However, FIFO is not fair among different flows, hence bursty traffic sources may monopolize the link bandwidth and there is no way to handle different traffic priorities.

Even though FIFO has important drawbacks, it can be used both in networks without QoS and network with QoS enabled. In the former case, FIFO is worth to be used as it is very efficient and fast. In the latter case, FIFO can be used in those QoS-enabled networks where the traffic admission control functions are performed at the edge of the network and the core network has enough resources to handle the admitted traffic in the edge. The main requirement is that admission control functions must be able to efficiently limit the amount of traffic admitted in the network so that it can flow into the core network. This kind of configuration is beneficial for very large networks. In this way, services requiring high computational power are performed at the edge of the network where traffic load is generally low and the core network can use FIFO queues since its high-speed requirements.

PQ allows handling different service classes and offers a low-jitter, low-loss service to the highest-priority traffic. On the other hand, if the amount of highest priority traffic is excessive then flows classified with a lower priority may not get any service until the highest priority traffic is served completely. Consequently, PQ can be applicable in those environments where a single high-priority traffic source (or a small number compared with the total traffic) is present. A proper use of admission control functions can limit the maximum amount of highest priority traffic; hence, PQ can be used in those environments where Admission Control is configured to compensate the drawbacks of the PQ discipline.

The main advantage of FQ is that bursty flows do not affect the performance of other flows. FQ tries to share a fair portion of bandwidth among many flows, but it cannot serve flows with different bandwidth requirements. The typical use of FQ is at the edge of the network, where users connect to their service provider. Implementations of FQ typically classifies packets into 256, 512 or 1024 queues using a hash function that is calculated across source/destination addresses, UDP/TCP port numbers and IP ToS byte [Semeria]. At the edge of the network, a user usually generates a limited number of flows; using FQ, each flow or a small number of flows can be assigned to each queue, providing isolation between flows and reducing the impact a bursty flow can have on the other flows.

WFQ tries to share the available bandwidth among all the traffic flows in fair way, by taking into account the weight each queue has. In this way, bursty traffic sources do not affect performance of other flows. However, the complexity of WFQ is high and grows up in proportion to the number of active flows. This is a strong limitation for applying WFQ in high-speed networks. On the other hand, WFQ can be effectively used in medium-low speed access networks.

CBQ has no the drawback of PQ, where low-priority queues do not receive any service if packets are in higher priority queues, and gives to every service class a certain service level. Since the number of queues in CBQ is statically configured, the service granularity that CBQ can provide is limited to the number of configured queues. Network environments where packet sizes are very variable are not suitable for CBQ, as CBQ does not provide fair allocation of bandwidth when packets with a variable length have to be handled.

Both CBQ and WFQ are meant to overcome the limitations of PQ and FQ algorithms. In this sense, WFQ offer the best performance, however its high complexity limits the range of applications it can be used. Both WFQ and CBQ give a minimum amount of resources to each service class, but CBQ needs packets of the same length to provide fair scheduling. On the other hand, CBQ is simpler than WFQ.

Finally, we can state that there is not a queuing mechanism suitable for all environments. The choice depends on the specific characteristics of the network environment. In order to provide efficient packet service, queuing disciplines cannot be used alone but in conjunction with a proper set of packet classification functions, admission control, and packet discard techniques.

G.HTB in Linux

Hierarchical Token Bucket (HTB) is a method to have bandwidth distribution among several service classes in a very flexible way. HTB is actually the most used technique in Linux Traffic Control [HTB] to provide QoS. HTB is composed of an arbitrary number of token buckets filters arranged in hierarchical way. We explain how HTB works using an example. Let us suppose we want to share the link among several service classes. Figure 7 shows a simple example where five service classes shares the same link, each one using a maximum amount of the total link bandwidth. The mail service class shares only 0% of the total bandwidth in case of congestion, which means that mail is serviced only if there is bandwidth available from other services. The CBQ queue discipline may be used to achieve this kind of structure.

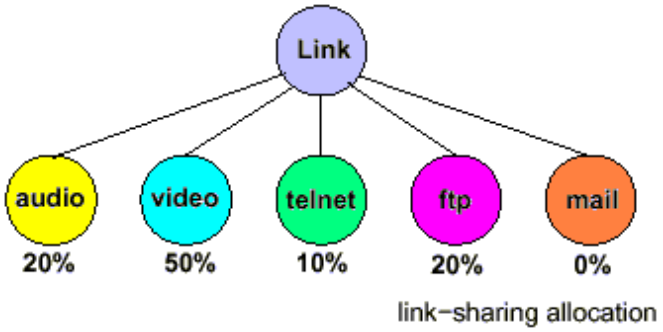

Figure 7. An example of link sharing between service classes.

HTB allows a more dynamic structure of link sharing. Not only it allows sharing the bandwidth among service classes, but permits link sharing between organizations, protocol families, and individual connections inside a service class. Figure 8 shows an example of hierarchical link sharing structure. In this example, we share the same link among three agencies, using respectively 50%, 40% and 10% of the total bandwidth. Inside the service class “Agency A”, we can create a hierarchy of other service classes like real-time, telnet and ftp. Each “sub service class” shares the bandwidth portion defined at the higher layer (real time uses 30%, telnet and ftp uses 10% each one). If the telnet class for Agency A has nothing to send, its assigned portion will be used from real-time and ftp classes on the same hierarchy level. At higher level, if Agency A does not use all its assigned portion of the total bandwidth, it will be distributed among the other classes.

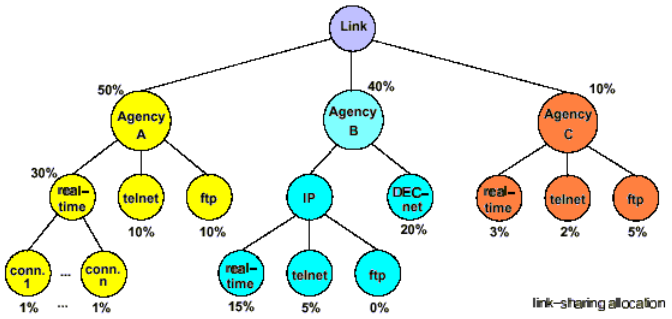

Figure 8. An example of hierarchical link sharing structure (HTB).

As we can see, the HTB structure is much more flexible than the "flat" one showed in Figure 7 because we have multiple levels of bandwidth distributions, not only one. One drawback of HTB is its complexity. Performance limitations may arise when a complex hierarchical link sharing structure is used.

V. Packet Discard Techniques

When network congestion is experienced, the number of arriving packets at a router overwhelms the available bandwidth of the link and the router queues try to cope with such a situation. As queues are finite memory buffers, they will fill up and, consequently, packet discard is necessary. Packet discard events can be controlled by choosing appropriate size of queues. However, huge queue sizes may cause excessive packet delay, as the current packet needs to wait all its predecessors before being served.

Packet discard should not be used as a response for a congestion situation, as any discarded packet has already used a certain amount of network resources up to the point it is discarded. However, packet discard cannot be completely avoided as queues may always overflow. Packets should be discarded in fair way, which means that not only the same source gets its packets discarded but the discarded packets belong to all the present traffic sources. In this paragraph, different packet discard methods are described, each one with advantages and disadvantages, taking a look at the fairness they provide. The described methods include Tail Drop, Early Drop, Drop Preference and RED.

A.Tail Drop

Tail drop is the most simple and common packet discard algorithm and it is commonly used with the FIFO queue discipline. Once the queue gets full, the new incoming packets are dropped. Packets continue to be dropped until new slots are available in the queue.

The main drawback of the Tail Drop algorithm is its low fairness. A sudden traffic burst from one source may fill up all the available slots of the queue and the new incoming traffic would be discarded. A solution to this problem is placing many service classes within the same queue. Each service class has a maximum amount of slots in the queue. When such a limit is overcome, packets of that service class are discarded. In this way, there is still space for other service classes even if one of them exceeded its available space. The drawback of this solution is that packet discarding happens even if there is available free memory for accepting new packets in the queue.

B.Early Drop

Early Drop is a technique meant to give a form of fair resource sharing among different flows. The main idea is to discard a priori bursty flows before the queue overflows, so that there is still space for other flows. According to such a definition, placing many service classes in the same queue (see the previous paragraph about Tail Drop) is one instance of the Early Drop method.

A way of implementing Early Drop consists of monitoring each flow and its traffic characteristics. When the queue length grows up to a certain limit, packets belonging to flows arriving at high rate or bursty flows can be discarded. This is a precise way to achieving Early Drop. However, it may present scalability problems in a loaded system with thousands of distinguished flows. In such an environment, selecting packets to be discarded in random way is usually preferred as in a congested environment the packet that will be chosen would probably belong to one of the flows with the highest bandwidth requirements.

TCP is one of the protocols that respond to packet drops as it supposes that packet drops are a clear signal of network congestion [TCP]. This is why Early Drop can also be used to signal the flow source that congestion is happening and, indirectly, reducing its sending rate.

C.Drop Preference

Another method for choosing which packet to drop consists of using external information for discarding the packet. When congestion arises and the queue length reaches a predefined limit, discarded packets are not the ones in the tail of the queue, but packets marked with a "drop preference" flag (or low priority flag). Such a flag can be set by the source application or by the admission control functions.

In this way, when congestion happens, the first flows to be discarded are the "less important" flows, whereas the most critical traffic is preserved and forwarded. In IP networks, the Drop Preference mechanism is associated with admission control functions. For example, all the packets belonging to a certain service class exceeding the maximum allowed bandwidth for that class are marked so that they are forwarded if bandwidth is available; otherwise they are the first packets to be discarded in case of congestion.

D.Random Early Drop (RED)

Random Early Drop [RED] is a method for discarding packets to cope with the network congestion problem. The philosophy of RED consists of assist and help congestion-responsive protocols, like TCP, to prevent and avoid network congestion. TCP [TCP] assumes that a packet drop is clear indication of congestion and reduces accordingly its sending rate, going back into slow start or reducing its congestion window size. With other packet discard methods, packets are dropped when congestion is already in act (buffers are full); the idea of RED is to start packet discarding before the queues are

full so that congestion is prevented and TCP adjusts its sending rate before congestion happens.

RED uses statistical methods to drop packets before the router queue overflows. It keeps track of the average queue size and discards packets when the average queue size overcomes a certain threshold. In more detail, the average queue size is calculated every time a new packet arrives and is compared to two thresholds: a *minimum* and a *maximum* threshold. When the average queue size is less than the minimum threshold, nothing happens. When the average queue size is greater than the maximum threshold, the arriving packet is dropped. If the average queue size is between the minimum and the maximum threshold, the arriving packet is dropped with a certain probability *p*. The RED algorithm is summarized as follows:

for each packet arrival

calculate the average queue size *avg*

if *min thresh* <= *avg* < *max thresh*

drop the arriving packet with probability *p*

else if max thresh <= avg

drop the arriving packet

The average queue size is calculated using an exponential weighted moving average [FJ93], meaning that if the queue has recently been empty for the most of the time, RED will not react to a sudden burst of packets as if a major congestion event were happened. However, if the average queue length remains near to the maximum threshold, RED will assume congestion and start dropping packets at a higher rate. [FF93] shows a complete analysis of RED behaviour when bursty traffic arrives.

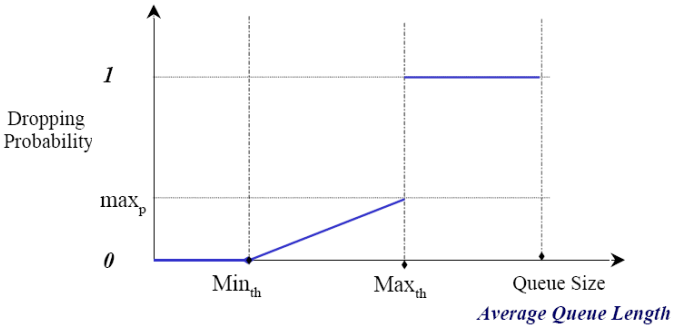

Figure 9. Dropping probability vs Average Queue Length

Figure 9 shows the dropping probability *p* in function of the average queue length *avg*. As we can see, *p* is zero while the average queue length is less then the minimum threshold and it grows up to a max_p value as the average queue length increases. The max_p parameter may be set up to one, depending on the configuration.

Many research studies [RED] shows that RED is an effective mechanism for providing congestion avoidance at routers in cooperation with TCP. Moreover, RED discards packets in more fair way when compared to other packet discard techniques.

A variant of RED consists of not dropping packets but simply marking them when the average queue size overcomes the minimum threshold. In this case, RED is used in conjunction with Explicit Congestion Notification (ECN) [RFC2481]. The router sets a congestion notification bit in the packet and forwards the packet to the receiver. At this point, the receiver can inform the sender to slow down its sending rate.

One critical issue of RED consists of tuning its parameters, like the minimum and maximum threshold and the weight to calculate the average queue size. It is difficult to find a set of parameters that offers good performance in all kinds of traffic profiles. Depending on the traffic source behaviour and the set of chosen parameters, RED may start to drop packets too early or too late, nullifying the reasons of why it has been developed.

A lot of variants have been proposed to improve RED, like Adaptive version of RED (ARED) and Weighted RED. ARED tries to set the RED parameters dynamically in order to better respond to different congestion situations. WRED is a technique of dropping packets based on the type of traffic, on the destination or other criteria. The interested reader can find a complete survey of RED and its variants in [RED].

VI. Packet Handling in Wireless Environment

The characteristics of mobile wireless networks make providing and implementing QoS and, in general, packet handling in an efficent manner more challenging. Wireless links present higher packet loss rate than wired links, variable and higher latency. Moreover, when a mobile node moves from one cell to another one, resources must be allocated in the new point of attachment to provide a continuos service to the user. The lack of resources in the new point of attachment, the time needed to renegotiate the session characteristics and to change the routing path may lead to a fluctuation in the QoS level provided to the user.

In a wireless environment, the IP layer and the link layer should interact and cooperate in order to provide an efficient packet handling as the knowledge of the amount of available radio resources and in general the current status of the wireless medium are needed to provide QoS.

In most cases, the interface between the network layer and the data link layer is a simple FIFO queue with a limited amount of space to temporarily store packets until they are transmitted over the link. In order to cope with the characteristics of wireless links (higher loss rate, higher and variable latency) and to provide QoS, the link layer may use different queues for scheduling the packets received from the IP layer. For example, packets requiring a reliable service may be stored in a special queue at the data link layer so that error-recovery mechanisms (FEC, ARQ) are achieved to recover packet errors.

In order to provide various QoS levels, the data link layer may use multiple queues and a scheduler, in analogy with the IP layer. The same queue and scheduling disciplines outlined in paragraph IV may be used at the data link layer. Using multiple

queues at the link layer may be necessary to provide a better service. However, this is a layer function violation as packet scheduling should be done at the IP layer.

Figure 10 shows four different approches for providing QoS with the assistance of the link layer [MKLLSR01]. The network-controlled approach (Figure 10a) is the most common model in which the interface between the network layer and the link layer is a simple FIFO queue. Packet classification, queuing and scheduling are performed at the IP layer. This model is suitable with low-latency links, but presents the head-of-line blocking problem. Such a problem is exacerbated with a high-latency link. For example, if one packet is lost and the link layer retransmits it, all other packets must wait (behind the head of the line) until the retransmission reaches the receiver or the sender gives up and discards the packet. With high-latency links, this may cause a significant delay for delivering the remaining packets.

Figure 10b shows the link-controlled approach, where the network layer has only one queue and the data link layer incorporates a packet classifier, multiple queues and a scheduler. This model does not present the head-of-line blocking problem. On the other hand, the link layer should perform functions, such as packet classification, belonging to the IP layer and thus leading to a significant function layer violations. Moreover, if IP layer security [RFC2401] is used then the link layer packet classifier may not work properly as the transport/application headers of IP packets may be encrypted and thus unreadable at the link layer [MKLLSR01].

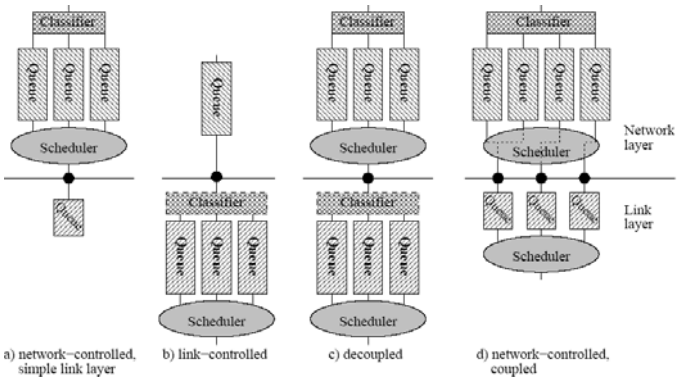

Figure 10. Four scheduling models [MKLLSR01]

Figure 10c shows the decoupled approach, which combines the approachs (a) and (b) and performs packet classification, queueing and scheduling both at the IP and data link layer. This model does not present the head-of-line blocking problem, but requires an accurate configuration in order to avoid bad scheduling interactions between IP and link layer. For instance, a high priority packet forwarded first by the IP layer may be delayed and transmitted later on after other packets by the link layer. In other words, the order of packets transmission over the link may change due to the interactions between the IP and the link layer schedulers. Moreover, the duplication of all the packet handling functions makes the system inefficient.

Figure 10d shows the network-controlled, coupled approach, which means that the IP layer classifies, schedules and forwards each packet to the correct link layer queue. This is done by defining a special interface between the IP and the link layer – called convergence layer – which allows the IP layer to have the information about the status of link layer queues and to control the link layer queueing and scheduling. In this way, the IP layer can store the scheduled packets into the appropriate link layer queue according to their class of service; it can calculate how long a packet can wait in a particular data link queue and decide what to do with those packets exceeding the maximum tolerable delay. Using such a approach, the harmful scheduling interactions between IP and data link layer outlined previosly for the case (c) are avoided and the link layer scheduling can handle packets in a more efficient way as it is controlled by the IP layer.

VII. Summary

This paper describes a number of basic methods for packet handling. The main steps for providing different levels of QoS are described: Packet Classification, Admission Control, Traffic Shaping, Queue disciplines, Packet Discard. Several queue disciplines are analyzed, like FIFO, PQ, FQ, WFQ and CBQ. Advantages and disadvantages for each methods are outlined and they are summarized in Table 1. The HTB discipline used in Linux Traffic Control is described. Packet Discard methods like Tail Drop, Early Drop, Drop Preference and RED are analyzed. Finally, methods for handling packets in wireless environment are described.

After such a survey, we can conclude that there is not a unique set of packet classification, admission control, traffic shaping, queuing disciplines and packet discard methods to handle any kind of network environment. In order to give an efficient packet service, the use of all these techniques must be harmonized and configured so that the specific characteristics of the used network environment, the traffic workload and the needed level of QoS are taken into account.

References

[CBQ] http://www.icir.org/floyd/cbq.html. Refenrences on CBQ (Class-Based Queuing).

[FF97] S. Floyd, k. Fall, “Router Mechanisms to Support End-to-End Congestion Control”, 1997.

[FJ93] S. Floyd, V. Jacobson. “Random Early Detection Gateways for Congestion Avoidance”, August 1993.

[HTB] Martin Devera, “HTB Linux queuing discipline manual”, http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm.

[Huston00] Huston, Geoff, “Internet Performance Survival Guide: QoS strategies for Multiservice networks”. John Wiley & Sons, Febraury 2000.

[Keshav97] S. Keshav, “An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network”, Addison-Wesley, 1997.

[LARTC] Linux Advanced Routing & Traffic Control, http://lartc.org/.

[McKe90] P. McKenney "Stochastic Fairness Queuing", IEEE INFOCOMM'90 Proceedings, San Francisco, 1990.

[MKLLSR01] J. Manner, M. Kojo, A. Laukkanen, M. Liljeberg, T. Suihko, K. Raatikainen, “Exploitation of Link Layer QoS Mechanisms in IP QoS Architectures”, ITCom 2001, Denver, Colorado, 20 - 24 August, 2001.

[RED] http://www.icir.org/floyd/red.html. References on RED Queue Management.

[RFC2401] S. Kent, R. Atkinson, “Security Architecture for the Internet Protocol”, IETF, November 1998.

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, “An Architecture for Differentiated Services”, December 1998.

[RFC2481] K. Ramakrishnan, S. Floyd, “A Proposal to add Explicit Congestion Notification (ECN) to IP”, January 1999.

[RFC2753] R. Yavatkar, D. Pendarakis, R. Guerin, “A Framework for Policy-based Admission Control”, January 2000.

[Semeria] Chuck Semeria, “Supporting Differentiated Service Classes: Queue Scheduling Disciplines”, Juniper Networks.

[TCP] Richard Stevens, “The protocols (TCP/IP illustrated, Volume 1)”, Addison Wesley.