CHAPTER 4

TRAFFIC MANAGEMENT

4.1 QUALITY OF SERVICE

The main task of internetworking is to deliver data from one peer to the other providing a certain minimal level of quality or the quality of service (QoS). QoS can normally be expressed by parameters such as achieved bandwidth, packet delay, and packet loss rates [1]. Achieving QoS guarantee becomes very difficult with the increasing number of users connected to a network, the increasing bandwidth required for applications, the increasing types of QoS requirements, and the increasing usage of multicast. The task becomes even more difficult in a connectionless IP network (the Internet). Congestion control provided by the flow control at the transport layer and to some extent by dynamic routing protocols [such as open shortest path forwarding (OSPF)] is inadequate to provide QoS. Different types of applications have different requirements of how their data should be sent. Some applications require error-free and loss-free transmission without any delay requirements. Some others require timely delivery but can sustain some data loss. Network resources will need to be managed appropriately to deal with these types of traffic.

Traditionally, the Internet has offered a single QoS, best-effort delivery, with available bandwidth and delay characteristics dependent on instantaneous load. Control over the QoS seen by applications is exercised by adequate provisioning of the network infrastructure. In contrast, a network with dynamically controllable QoS allows individual application session to request network packet delivery characteristics according to its perceived needs, and may provide different qualities of service to different applications [1].

The network has been evolving to provide QoS guarantees to the users. For instance, asynchronous transfer mode (ATM) can reserve the bandwidth and buffer size for each virtual connection. Similarly, Internet Integrated Service (IntServ) can also provide QoS for each flow in the IP network, while Internet Differentiated Service (DiffServ) provides

High Performance Switches and Routers, by H. Jonathan Chao and Bin Liu

different treatment for packets to different classes, instead of on flow basis, so that it has better scalability than IntServ.

To maximize network resource utilization while satisfying the individual user's QoS requirements, traffic management should be provided with prioritized access to resources at network nodes. In this chapter, three fundamental traffic management techniques, traffic policing/shaping, packet scheduling and buffer management, are presented in detail.

Traffic policing/shaping is normally performed at the network edge to monitor traffic flows and take appropriate actions when they are not compliant to traffic contract. They are either discarded, marked, and delayed. The control can become challenging when the number of flows is large or the line speed is high. Especially, when traffic shaping is performed by delaying arriving packets in a buffer, choosing a packet that is compliant from hundreds or thousands of packets in the buffer is very difficult.

Packet scheduling specifies the queue service discipline at a node, which determines the transmission order of the queued packets. Since packets of many users may leave for the same output interface, packet scheduling also enforces a set of rules in sharing the link bandwidth. One major concern is how to ensure that the link bandwidth is fairly shared among the connections and to protect the individual user's share from being corrupted by malicious users.

Beside packet scheduling algorithms, buffer management is also very critical to the performance of the network. Buffer management mechanisms are needed in the high speed communication devices such as multiplexers, packet switches, and routers to set up the buffer sharing policy and decide which packet should be discarded when the buffer overflows or is getting filled.

4.1.1 QoS Parameters

QoS parameters represent the QoS to customers. It should be easy for customers to understand the degree of assuring the service. QoS parameters can be different according to the type of services and layers. Generic QoS parameters required in network service include throughput, packet delay, bandwidth, residual error rate, and so on. The definitions of these parameters are as follows [2, 3].

Throughput. Throughput is a connection-mode QoS parameter that has end-to-end significance. It is defined as the total number of bits successfully transferred by a primitive sequence divided by the input/output time, in seconds, for that sequence. Successful transfer of a packet is defined to occur when the packet is delivered to the intended user without error, in proper sequence, and before connection termination by the receiving user.

Packet Delay. Packet delay is the time taken for a packet to travel from a service access point (SAP) to a distant target. It usually includes the transport time and queuing delay. The packet delay distribution occurring in a router can be like the one as shown in Figure 4.1. The maximum packet delay for a flow is the $(1 - a)$ quantile of packet delay.

Bandwidth. Bandwidth means used capacity or available capacity. Service providers usually assure the maximum bandwidth to customers and it is stated clearly in the Service Level Agreement (SLA, see Section 4.3 for detail).

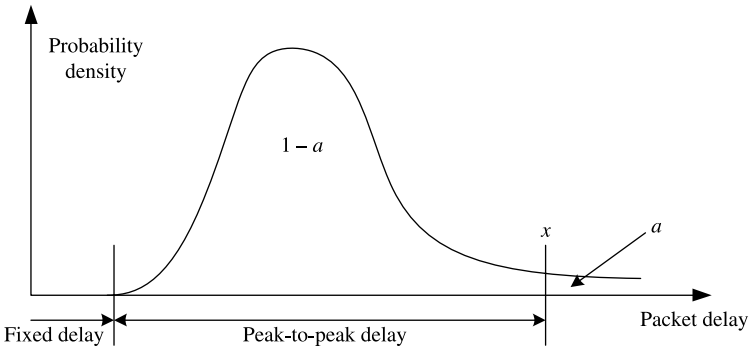


Figure 4.1 Packet delay in a router.

Residual Error Rate. Residual error rate (RER) is the ratio of total incorrect, lost and duplicated packets to the total packets transferred between users during a period of time.

Delay Variation. Delay variation may cause the increasing of TCP retransmit timer and unnecessary packet loss. So delay variation parameter is very important, which can be measured by the dispersion of maximum delay and minimum delay during a short measurement interval.

Loss Rate. Loss rate is the ratio of lost packets to the total packets in transit from source to destination during a specific time interval, expressed in percentages.

Spurious IP Packet Rate. Spurious IP packet rate is defined as the rate of the fraction of the spurious IP packets during a specific time interval (number of spurious IP packets per second).

Availability. Availability is the percentage of the feasibility of service in every particular service request, which means connectivity and functionality in the network management layer. Connectivity is the physical connectivity of network elements and functionality means whether the associated network devices work well or not. Traffic service time can be divided into available time and unavailable time. If packet lose rate is under a defined threshold, it is recognized to be available, otherwise unavailable.

4.1.2 Traffic Parameters

The following traffic parameters are commonly used to categorize a flow [4].

Peak Rate. The peak rate is the highest rate at which a source can generate traffic. The peak rate is limited by the transmission link rate. The peak rate can be calculated from the packet size and the spacing between consecutive packets.

Average Rate. The average rate is the rate averaged over a time interval. The average rate can be calculated in several ways, and the results can be quite different. It is important to know the exact method and the time interval used in the calculation. Typically the average

rate is calculated with a moving time window so that the averaging time interval can start from any point in time.

Burst Size. The burst size is defined as the maximum amount of data that can be injected into the network at the peak rate. The burst size reflects the burstiness of the traffic source. To avoid packet losses, the first hop router may have to allocate a buffer for the source larger than its burst size.

4.2 INTEGRATED SERVICES

Integrated Services (IntServ) is an end-to-end, flow-based mechanism for providing QoS. Routers need to keep track of every flow. IntServ reserves resources along the path of transmission, via resource Reservation Protocol (RSVP). In other words, IntServ sets up a virtual circuit. During the reservation process, a request goes through admission control, which will either grant or deny the request. This is done based on the resources available on that router, and it is necessary to preserve the QoS requirements for other flows currently active in the router.

4.2.1 Integrated Service Classes

In addition to best-effort delivery, IntServ has two additional service classes, and they are:

Guaranteed Service. This service class is used for intolerant playback applications without any distortion like H.323 video conferencing. This class offers perfectly reliable upper bound on delay.

Controlled-Load Service. Also known as predictive service, this service class is more relaxed on the delay bound. It is fairly reliable, but not perfectly reliable in providing the delay bound. It works well when network is lightly loaded. However, if the network is overloaded, some packet loss or delay can be experienced. This is the middle ground between guaranteed and best effort. It is used for tolerant applications like MPEG-II video. This service class is more efficient in utilizing the network resources because it allows other flows to share its resources when it is not in use. For example, an application can ask for 5 Mbps, but the bit rate can vary between 2 and 5 Mbps. Instead of wasting the extra 3 Mbps, this service class allows other flows to use the extra 3 Mbps.

4.2.2 IntServ Architecture

To understand the IntServ model, a reference implementation framework has been specified in RFC 1633. Figure 4.2 shows the key components in the reference model. The model can be logically divided into two planes: the control plane and the data plane. The control plane sets up resource reservation; the data plane forwards data packets based on the reservation state.

Key elements in the reference model are briefly explained below.

RSVP. Connections are established using a reservation setup agent known as RSVP, which makes end-to-end reservations over a connectionless network. RSVP is responsible for changing the traffic control database (used to store classifier and scheduling policies) to

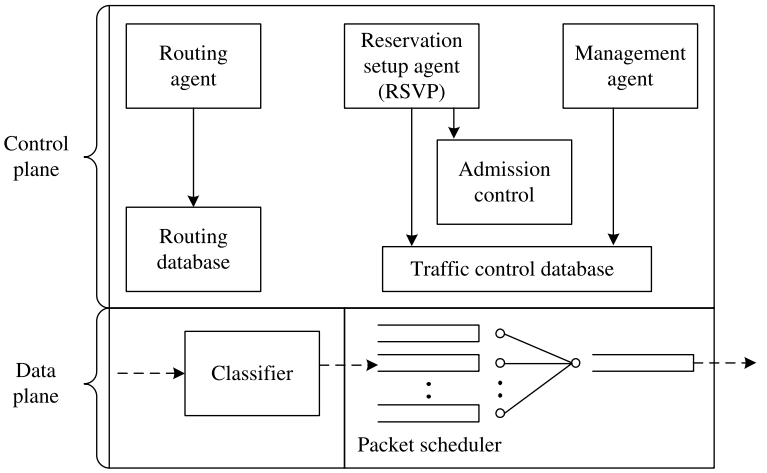


Figure 4.2 IntServ reference model for routers.

accommodate the required QoS. The management agent is used to set up the policies for the classifier and scheduler. More details on RSVP are described in Section 4.2.3.

Classifier. Packets need to be mapped into some class for the process of traffic control. Packets from the same class will get the same treatment from the scheduler. A class can be based on network headers, transport headers, application headers, or any combination of them. For instance, an IP flow is normally identified by the five-tuple in the packet header: source IP address, destination IP address, protocol ID, source port, and destination port, as we discussed in Chapter 3.

Admission Control. Admission control is responsible for permitting or denying flows into the router. Admission control is used to decide whether a new flow can be granted the requested QoS without effecting current guarantees. If the router decides that it does not have the required resources to meet the requested QoS, the flow gets a denied admission. Admission control has two basic functions. The first function is to determine if a new reservation can be set up based on the admission control policies. The second one is to monitor and measure available resources. Parameter based approach and measurement based approach are two basic methods to admission control. In the parameter based approach, a set of parameters is used to precisely characterize traffic flows; the admission control agent then calculates the required resources based on these parameters. Instead of relying on a prior traffic characterization, the measurement based approach measures the actual traffic load and uses that for admission control.

Packet Scheduler. The packet scheduler reorders packet transmission so that certain flows can be served based on their service class and levels. For example, packets that require guaranteed service will be served first before those that require controlled-load service or best-effort service. The packet scheduler is responsible for enforcing resource allocation, which directly affects the delay. The key task of a packet scheduler is to select a packet to transit when the outgoing link is ready. We present several packet scheduling schemes in Section 4.5.

4.2.3 Resource ReSerVation Protocol (RSVP)

A quick summary about RSVP is given below:

- RSVP is a signaling protocol for establishing a guaranteed QoS path between a sender and (a) receiver(s).
- RSVP establishes end-to-end reservations over a connectionless network.
- RSVP is robust when links fail: Traffic is re-routed and new reservations are established.
- RSVP is receiver-initiated and so is designed with multicast in mind.
- RSVP is simplex (reserves for unidirectional data flow) and receiver-oriented.
- Operates in 'soft state', responds to changes in routing, multicast membership.
- Provides transparent operation through routers that do not support RSVP.
- RSVP is independent of the IP protocol versions – RSVP applies equally well to IPv4 and IPv6.

RSVP operations. Figure 4.3 illustrates how RSVP sets up resource reservation between senders (sources) and receivers (destinations). There are two types of messages in RSVP: PATH messages and RESV messages. PATH messages are sent from traffic sources, while RESV messages are sent between RSVP-aware routers hop by hop after receiving the PATH messages. A PATH messages is addressed to the flow's ultimate destination but it actively interacts with all of the RSVP-aware routers as it travels downstream along the path to its destination, setting up control blocks as it goes. After receiving the PATH message, all of the routers in the path will have the information they need to recognize the flow when it appears at one of their ports. The PATH message also carries the traffic characteristics of the flow which is called 'Tspec', so the routers will know how much the flow is asking for to prevent over-reservation and perhaps unnecessary admission control failures. In addition, the PATH messages can contain a package of OPWA (One Pass With Advertising) advertising information, known as an 'Adspec'. The Adspec received in the PATH message is passed to the local traffic control, which returns an updated Adspec; the

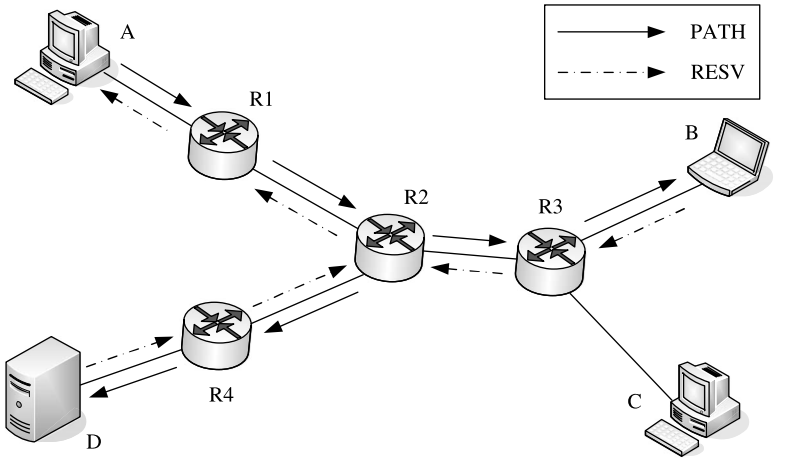


Figure 4.3 RSVP operations.

updated version is then forwarded in the PATH message sent downstream. When the PATH message finally reaches its destination, the destination can then, if it wants, ask to make a QoS reservation for the flow.

The reservation is made by the RESV message, which follows the path of the PATH message backward, upstream to the data source. However, it is well known that IP networks are asymmetrically routed. That is, the transmission path from user A to user B is often totally different from the one which is from user B to user A. When the PATH message travels from the flow source to the destination, it takes care to place a special PHOP (previous hop) address in each control block that it sets up. Therefore, the RESV message will be able to follow the path message's path backward by looking at the PHOP address stored in all the control blocks. Then it will continue upstream, marking changes in all the routers' control blocks as needed to install the designated QoS reservation, until it finally arrives at the flow source. At that time, the flow source will know if it was able to make the reservation successfully to the flow's destination.

The process of RSVP can be concluded as follows. First, traffic sources send PATH messages towards receivers which contain the information of the sender (bandwidth, delay and jitter it needs) and the path characteristics to the receivers. Second, the routers forward the PATH messages. The forwarding process is determined by the unicast and multicast routing protocols. Third, routers determine if reservation can be fulfilled and forward/merge RESV messages upstream toward the source along the exact reverse path of the PATH message. Fourth, after receiving the RESV messages, the sender can start to transmit packets along the reserved path. When transmission is finished, routers tear down the resource reservation.

Unicast and Multicast. For unicast reservation, two applications agree on a specific QoS required to support its service and expect the network to support this QoS. If the network is experiencing congestion, or is on the verge of experiencing it, it may not provide the requested QoS. In that case, the communicating applications will be notified and may choose not to start a session, and instead wait for the network traffic to decrease. They may also choose to start the session, but at a lower QoS.

Multicast transmission is the driving point of RSVP implementation. A multicast transmission can generate an enormous amount of network traffic due to its nature: it could be either a high volume data application (such a video broadcast), or having many scattered subscribers to the same service.

Receiver-Initiated Reservation. The approach to resource reservation used in Frame Relay and ATM consists of the source of the data flow requesting a specific set of resources, which works well for unicast traffic. However, this approach does not work well in a multicast environment. And the main reason is that different destinations in a multicast group may have different resource requirements, which is quite common in the current Internet. If a source transmission flow can be divided into component subflows, then some destination members may want to require only one subflow. For example, some receivers may not have enough processing power to handle a high definition video multicast, and choose to receive only low-definition video. In a word, the QoS requirements of different receivers may differ, depending on their hardware capabilities, processing powers, and link speeds.

This is the big plus of RSVP: The receivers specify the QoS of the transmission. Routers can then aggregate multicast resource reservations to take advantage of shared paths along the distribution tree.

Soft State. RSVP makes use of the concept of a soft state. A connection-oriented scheme takes a hard-state approach, in which the nature of the connection along a fixed route is defined by the state information in each of the switching nodes. The soft state approach is characterized by the fact that the each switching node along the path has cached information that is refreshed after a specified time limit. If the state is not refreshed after this time limit, the router considers the state invalid and discards it. It will then propagate its new status to the next hop. The soft state approach is used because membership of a large multicast group and the resulting multicast tree topology are likely to change with time. The RSVP design assumes that state for RSVP and traffic control is to be built and destroyed incrementally in routers and hosts. Of course, the state can also be deleted by a teardown message.

4.3 DIFFERENTIATED SERVICES

IntServ and RSVP provide a structure for very detailed control of the QoS of the individual flow as it passes through a network of IP routers. Unfortunately, the scalability of IntServ solution, which is mainly caused by its complex per-flow classification and scheduling, is still in doubt. IntServ requires routers to store and process each individual flow that goes through it, and it becomes overwhelming in the Internet. The Differentiated Services (DS or DiffServ) architecture was then developed in response to the need for relatively simple, coarse methods of providing different levels of service for Internet traffic. The advantage of DS is that many traffic streams can be aggregated into one of a small number of behavior aggregates, forwarded using the same PHBs (per hop behavior) at the router, thereby simplifying the processing and associated storage. PHB is a description of the externally observable forwarding behavior of a DS node applied to a particular DS behavior aggregate. In addition, there is no signaling, other than what is carried out in the DSCP (DS Codepoint) of each packet. No other related processing is required in the core of the DS network, since QoS is invoked on a packet-by-packet basis. Before explaining the details, a quick summary of DS is given below:

- Demands from customers for service differentiation.
- Emerging of new multimedia applications for which the best-effort service in today's Internet cannot support.
- Define the meaning of the DS field, that is, Type of Service (TOS) field in IPv4 or Traffic Class field in IPv6, in the IP packet header for each class of services.
- Mark the DS field of packets based on their service classes and process them differently.
- Offering services for traffic on a per-class basis rather than on a per-flow basis at the internal nodes.
- Forcing as much complexity out of internal nodes of a network to boundary nodes, which process lower volumes of traffic and smaller number of flows.
- Pushing per-flow processing and state management to the boundary nodes.

The DiffServ is different from IntServ in many aspects, and the key difference is that DiffServ distinguishes a small number of forwarding classes rather than individual flows. IntServ uses a signaling protocol (RSVP, discussed in Section 4.2.3) to reserve resources for individual flows, while DiffServ allocates resources on a per-class basis which is based on the ToS octet in IPv4 header or the Traffic Class octet in IPv6 header. Each router along

the path examines this octet and makes a QoS decision based on the policies set up on that router. As a result, all the information that the router needs to handle the packet is contained in the packet header; so routers do not need to learn and store information about individual flows. There is no need for soft-state refresh messages. There is no worry about the packets that temporarily don't get their proper QoS handling after they have been rerouted. There is no scalability problem associated with need for the router to handle per-flow packet.

Each router is independent from one another. Therefore to provide consistency in QoS, routers should be configured with similar policies.

4.3.1 Service Level Agreement

A Service Level Agreement (SLA) is a service contract between a customer and a service provider that specifies the forwarding service a customer should receive. A customer may be a user organization (source domain) or another DS domain (upstream domain). Here, a DS domain is a contiguous set of nodes which operates with a common set of service provisioning policies and PHB definitions.

A SLA can be dynamic or static. Static SLAs are negotiated on a regular basis, for example, monthly or yearly. Dynamic SLAs use a signaling protocol to negotiate the service on demand.

The SLA typically contains:

- The type and nature of service to be provided, which includes the description of the service to be provided, such as facilities management, network services, and help desk support.
- The expected performance level of the service, which includes two major aspects: reliability and responsiveness. Reliability includes availability requirements – when is the service available, and what are the bounds on service outages that may be expected. Responsiveness includes how soon the service is performed in the normal course of operations.
- The process for reporting problems with the service, which forms a big part of a typical SLA. It includes information about the person to be contacted for problem resolution, the format in which complaints have to be filed, and the steps to be undertaken to resolve the problem quickly, and so on.
- The time frame for response and problem resolution, which specifies a time limit by which someone would start investigating a problem that was reported, and so on.
- The process for monitoring and reporting the service level, which outlines how performance levels are monitored and reported – that is, who will do the monitoring, what types of statistics will be collected, how often they will be collected, and how past or current statistics may be accessed.
- The credits, charges, or other consequences for the service provider when not meeting its obligation (i.e., failing to provide the agreed-upon service level).
- Escape clauses and constraints, including the consequences if the customer does not meet his or her obligation, which qualifies access to the service level. Escape clauses are conditions under which the service level does not apply, or under which it is considered unreasonable to meet the requisite SLAs. For example, when the service provider's equipment has been damaged in flood, fire, or war. They often also impose some constraints on the behavior by the customer. A network operator may void the SLA if the customer is attempting to breach the security of the network.

4.3.2 Traffic Conditioning Agreement

An SLA also defines Traffic Conditioning Agreement (TCA) which defines the rules used to realize the service – what the client must do to achieve desired service and what the service provider will do to enforce the limits. More specifically, a TCA is an agreement specifying classifier rules and any corresponding traffic profiles and metering, marking, discarding and/or shaping rules which are to apply to the traffic streams selected by the classifier. A TCA encompasses all of the traffic conditioning rules explicitly specified within a SLA along with all of the rules implicit from the relevant service requirements and/or from a DS domain's service provisioning policy.

A traffic profile specifies the temporal properties of a traffic stream selected by a classifier. It provides rules for determining whether a particular packet is in-profile or out-of-profile. For example, a profile based on a token bucket may look like:

$$codepoint = X, \text{ use token-bucket } r, b$$

The above profile indicates that all packets marked with DS codepoint X should be measured against a token bucket meter with rate r and burst size b. In this example out-of-profile packets are those packets in the traffic streams which arrive when insufficient tokens are available in the bucket (details are covered in Section 4.4.3). The concept of in- and out-of-profile can be extended to more than two levels, for example, multiple levels of conformance with a profile may be defined and enforced.

As packets enter the domain, they will be classified into a traffic aggregate based on the specified filter at the domain ingress interface of the border router. The filter must be associated with a traffic profile that specifies committed information rate (CIR) and a description on how it is to be measured. For example, the measurement may be based on a committed burst size (CBS) or an averaging time interval (T1).

The traffic profile may also include other traffic parameters. These parameters may place additional constraints on packets to which the assurance applies or may further differentiate traffic that exceeds the CIR. Such parameters could include: peak information rate (PIR), peak burst size (PBS), excess burst size (EBS), or even a second averaging time interval (T2).

4.3.3 Differentiated Services Network Architecture

DS divides a network into several domains as shown in Figure 4.4. A DS domain [5] is a continuous set of nodes which operates with a common set of resource provisioning policies and PHB definitions. It has a well defined boundary and there are two types of nodes associated with a DS domain: boundary nodes and interior nodes. Boundary nodes connect the DS cloud to other domains. Interior nodes are connected to other interior nodes or boundary nodes, but they must be within the same DS domain. The boundary nodes are assigned the duty of classifying ingress traffic so that incoming packets are marked appropriately to choose one of the PHB groups supported inside the domain. They also enforce the TCA between their own DS domain and the other domains it connects to. The TCA defines the rules used to realize the service, such as metering, marking, and discarding.

Interior nodes map the DS codepoints of each packet into the set of PHBs and perform appropriate forwarding behavior. Any non-DS compliant node inside a DS domain results in unpredictable performance and a loss of end-to-end QoS. A DS domain is generally made up of an organization's intranet or an ISP, that is, networks controlled by a single entity.

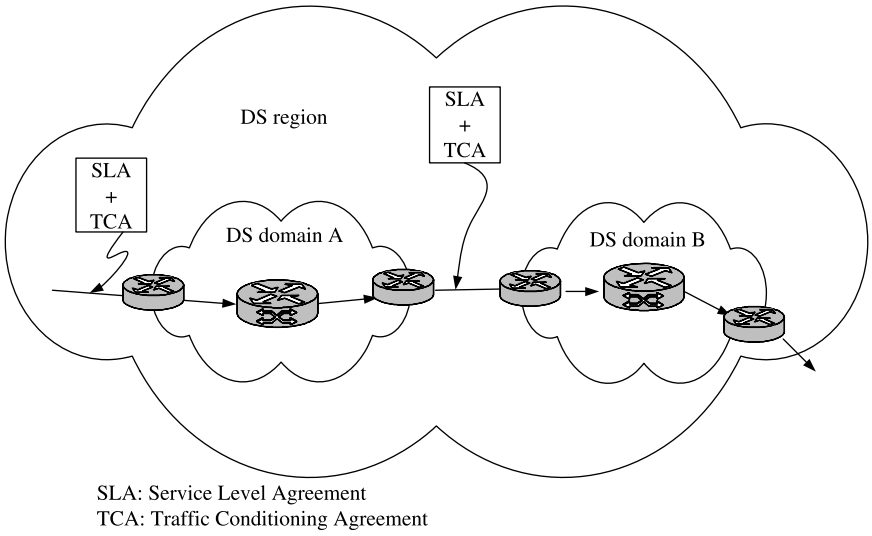


Figure 4.4 Differentiated services (DS) network architecture.

DiffServ is extended across domains by SLA between them. An SLA specifies rules such as traffic remarking, actions to be taken for out-of-profile traffic, and so on. The TCAs between domains are decided from this SLA.

Depending on direction of traffic flow, DS boundary nodes can be both ingress nodes and egress nodes. Traffic enters the DS cloud through an ingress node and exits through an egress node. An ingress node is responsible for enforcing the TCA between the DS domain and the domain of the sender node. An egress node shapes the outgoing traffic to make it compliant with the TCA between its own DS domain and the domain of the receiver node.

Flows are classified by predetermined rules so that they can fit into a limited set of class flows. The boundary routers use the eight-bit ToS field of the IP header, called the DS field, to mark the packet for preferential treatment by the interior routers. Only the boundary routers need to maintain per-flow states and perform the shaping and the policing. This is usually advantageous since the links between the customer and service provider are usually slow, so additional computational delay is not that much of a problem for routers interfacing to these links. Therefore, it is affordable to do the computationally intensive traffic shaping and policing strategies at the boundary routers. But once inside the core of the service providers, packets need to be routed (or forwarded) very quickly and so it must incur minimum computational delay at any router/switch. Since the number of flows at the boundary router is much smaller than that in the core network, it is also advantageous to do flow control at the boundary routers.

4.3.4 Network Boundary Traffic Classification and Conditioning

Traffic conditioners perform various QoS functions and are located at network boundaries. The boundary routers classify or mark traffic by setting the DSCP field and monitor incoming network traffic for profile compliance. The DSCP field indicates what treatment the packet should receive in a DS domain. The QoS functions can be those of packet classifier, DSCP marker, or traffic metering function, with either the shaper or dropper action.

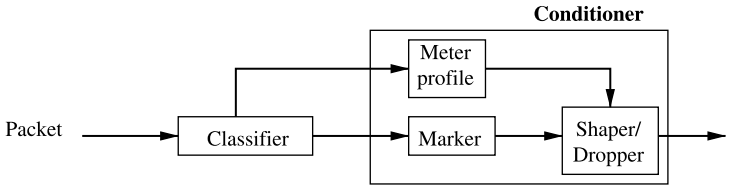


Figure 4.5 Packet classifier and traffic conditioning.

Figure 4.5 presents the logical structure of traffic classification and conditioning functions. The classification of packets can be done in one of two ways, depending on the connectivity of the boundary router. Some boundary routers are connected to customer networks, and some others are connected to other ISPs.

A boundary router that is connected to a customer network uses six fields in an incoming IP packet to determine the PHB that the packet should receive in the core network. These six fields are IP source address, IP destination address, protocol ID, DS field in the incoming packet header, source port, and destination port in the transport header respectively. A rule that maps a packet to a PHB does not need to specify all six fields. We refer to these rules as classificaton rules. When a classification rule does not specify any specific value for a field, that specific field is not used for the purpose of classification.

Boundary routers could use just one field in the incoming IP packet to determine the PHB for their network. This field could be the DS field contained in the incoming packet. A boundary router would simply change the DS field to some other value that corresponds to a specific PHB at the core routers. This type of classification would be the one expected at the exchange points of other ISPs. The neighboring ISP domain may have been using a different set of PHBs or may use different DS field values to represent the same PHB.

A boundary router may limit the total number of packets that can be sent into each class. If it is found that a packet cannot be mapped into any of the PHBs because a limit would be exceeded, it may either be mapped into a different PHB or discarded.

The meters check conformance of traffic streams to certain parameters and pass the results to the marker and shaper/dropper. Traffic meters measure the temporal properties of the stream of packets selected by a classifier against a traffic profile specified in a TCA. A meter passes state information to other conditioning functions to trigger a particular action for each packet which is either in- or out-of-profile.

The marker is responsible for writing/rewriting DSCP values. It can mark packets based on classifier match or based on results from the meters. Packet markers set the DS field of a packet to a particular codepoint, adding the marked packet to a particular DS behavior aggregate. The marker may be configured to mark all packets which are steered to it to a single codepoint, or may be configured to mark a packet to one of a set of codepoints used to select a PHB in a PHB group, according to the state of a meter. When the marker changes the codepoint in a packet, it is said to have 're-marked' the packet.

The shaper/dropper is responsible for shaping the traffic to be compliant with the profile and may also drop packets when congested. Shapers delay some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. A shaper usually has a finite-size buffer, and packets may be discarded if there is no sufficient buffer space to hold the delayed packets. Droppers discard some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. This process is

known as 'policing' the stream. Note that a dropper can be implemented as a special case of a shaper by setting the shaper buffer size to zero (or a few) packets.

4.3.5 Per Hop Behavior (PHB)

According to RFC 2475 [5], PHB is the means by which a node allocates resources to aggregate streams. A given example is that a PHB defines a percentage of the capacity of a link. The interior routers in the DS model only need to forward packets according to the specified PHB.

If only one behavior aggregate occupies a link, the observable forwarding behavior will generally only depend on the congestion of the link. Distinct behavioral patterns are only observed when multiple behavioral aggregates compete for buffer and bandwidth resources on a node as shown in Figure 4.6. There are two flows: 'Source 1 → Destination 1' and 'Source 2 → Destination 1.' They have different classifications and will be treated differently by the interior router, for example, using different scheduling and/or discarding preference. A network node allocates resources to the behavior aggregates with the help of the PHBs. PHBs can be defined either in terms of their resources (e.g., buffer and bandwidth), in terms of their priority relative to other PHBs, or in terms of their relative traffic properties (e.g., delay and loss). Multiple PHBs are lumped together to form a PHB group [5] to ensure consistency. PHBs are implemented at nodes through some buffer management or packet scheduling mechanisms. A particular PHB group can be implemented in a variety of ways because PHBs are defined in terms of behavior characteristics and are not implementation dependent.

The standard for DiffServ describes PHBs as the building blocks for services. The focus is on enforcing an SLA between the user and the service provider. Customers can mark the DS octet of their packets to indicate the desired service, or have them marked by the boundary router based on multifield classification (MF), such as IP destination and source addresses, transport port numbers, protocol ID, and so on. Inside the core, packets are forwarded according to their behavior aggregates. These rules are derived from the SLA. When a packet goes from one domain to another, the DS octet may be rewritten by the new network boundary routers. A PHB for a packet is selected at a node on the basis of its DS codepoint. The four most popularly used PHBs are default behavior, class selector, Assured

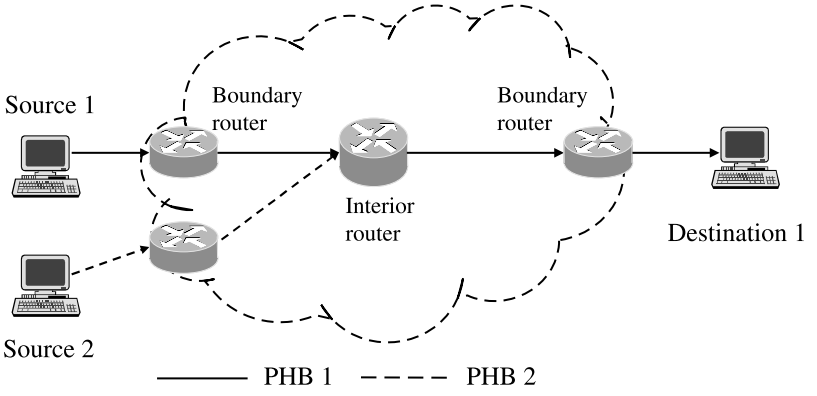


Figure 4.6 Per hop behavior in DS.

Forwarding (AF), and Expedited Forwarding (EF), described in Section 4.3.6. The mapping from DS codepoint to PHB may be 1 to 1 or *N* to 1. All codepoints must have some PHBs associated with them. Otherwise, codepoints are mapped to a default PHB. Examples of the parameters of the forwarding behavior that each traffic class should receive are bandwidth partition and the drop priority. Examples of implementations of these are WFQ (weighted fair queuing) for bandwidth partition and RED (random early detection) for drop priority.

4.3.6 Differentiated Services Field

The DS values are known as the Differentiated Service Code Point (DSCP). Figure 4.7 illustrates the ToS and DSCP octet formats. In IPv4, only the first three bits were used for QoS purposes. In DiffServ, six bits are used. The same octet is now referred to as the DS field. The DS field is broken up into three-bit class selector and three-bit drop precedence. A more complete mapping between the ToS octet and DS octet is presented in Figure 4.8.

Default Behavior. The default or best-effort PHB corresponds to the default best-effort packet forwarding in the traditional IP network. Packets belonging to this PHB could be forwarded in any manner without any restrictions. The recommended codepoint by IETF for best effort PHB is 0×000000.

Class Selector. The left most three bits of the DS field/IP ToS define eight classes. They are the DS5 to DS3 bits in Figure 4.7. A packet with a higher numeric value in the Class Selector field is defined to have a better (or equal) relative priority in the network for forwarding than a packet with a lower numeric value. A router does not need to implement eight different priority levels in the network to support the class selector PHBs. It can claim compliance with the standards by supporting only two priority levels, with the eight numeric values mapping to one of the two classes.

Assured Forwarding (AF). The AF PHB is used to provide Assured Services to the customers, so that the customers will get reliable services even in times of network congestion. Classes 1 to 4 are known as the AF service levels. Because making a decision based on only the class selector is very coarse, the AF PHB was created to provide more granularities in buffer management. This class makes use of the drop precedence bits (DS2 to DS0 in Figure 4.7).

When backlogged packets from an AF forwarding class exceed a specified threshold, packets with the highest drop priority are dropped first and then the packets with the lower drop priority. Drop priorities in AF are specific to the forwarding class; comparing two drop priorities in two different AF classes may not always be meaningful. For example, when a DS node starts to drop the packets with the highest drop priority in one forwarding class,

ToS Octet:	P2	P1	P0	T3	T2	T1	T0	Zero
DS Octet:	DS5	DS4	DS3	DS2	DS1	DS0	ECN1	ECN0
	(Class Selector)			(Drop Precedence)				

Figure 4.7 IPv4 ToS octet and the DS octet.

IP Precedence (3 bits)			DSCP (6 bits)				
Name	Value	Bits	Per-Hop Behavior	Class Selector	Drop Precedence	Codepoint Name	DSCP Bits (decimal)
Routine	0	000	Default			Default	000 000(0)
Priority	1	001	AF	1	1: Low	AF11	001 010(10)
					2: Medium	AF12	001 100(12)
					3: High	AF13	001 110(14)
Immediate	2	010	AF	2	1: Low	AF21	010 010(18)
					2: Medium	AF22	010 100(20)
					3: High	AF23	010 110(22)
Flash	3	011	AF	3	1: Low	AF31	011 010(26)
					2: Medium	AF32	011 100(28)
					3: High	AF33	011 110(30)
Flash Override	4	100	AF	4	1: Low	AF41	100 010(34)
					2: Medium	AF42	100 100(36)
					3: High	AF43	100 110(38)
Critical	5	101	EF	5		EF	101 110(46)
Internetwork Control	6	110	—				(48–55)
Network Control	7	111	—				(56–63)

Figure 4.8 IPv4 ToS to DSCP mapping.

the packets in other forwarding classes may not experience any packet dropping at all. Each forwarding class has its bandwidth allocation. Dropping takes place only in the forwarding class in which traffic exceeds its own resources.

In general, a DS node may reorder packets of different AF classes but should not reorder packets with different drop priorities but in the same class. The boundary nodes should avoid splitting traffic from the same application flow into different classes since it will lead to packet reordering with a microflow in the network.

Expedited Forwarding (EF). Class 5 is known as EF. The EF PHB is used to provide premium service to the customer. It is a low-delay, low-jitter service providing nearly constant bit rate to the customer. The SLA specifies a peak bit rate which customer applications will receive and it is the customers' responsibility not to exceed the rate.

Expedited forwarding PHB is implemented in a variety of ways. For example, if a priority queuing is used, then there must be a upper bound (configured by the network administrator) on the rate of EF traffic that should be allowed. EF traffic exceeding the bound is dropped.

4.3.7 PHB Implementation with Packet Schedulers

This section describes some typical packet schedulers and how they can be used to support the PHBs. Here we only show two implementation examples. Other possible implementations regarding packet scheduling such as WFQ and its variants can be found

in Section 4.5. Implementation regarding buffer management such as RED can be found in Section 4.6.

Static Priority Queues. Consider a static priority queue scheduler with two levels of priorities. Such a scheduler serves packets in the higher-priority queue if they are available and only serves the lower-priority queue if the higher-priority queue is empty.

This scheduler can support the class selector PHBs by mapping packets with the DS fields of 1000000, 101000, 110000, and 111000 into the higher-priority queue and those with the DS fields of 000000, 001000, 010000, and 011000 into the lower-priority queue. Other mappings are also possible. The DS standards require that packets with the DS fields of 110000 and 111000 should map to the higher-priority queue, and packets with the DS field of 000000 should map to the lower-priority queue. Packets with other DS fields may be mapped to either of the two queues. A network administrator should ensure that the mapping is consistent across all the routers within a single administrative domain.

Weighted Round Robin (WRR). A round-robin scheduler with weights assigned to each of multiple queues would be the closest match to implementation of the EF and AF PHBs. The round-robin queues could be assigned weights based on the allocated bandwidth so that they would be able to serve each of the queues proportional to their bandwidth. Class selector PHBs could be supported: packets with the DS field of 110000 and 111000 are mapped to a queue with a larger weight than the queue to which packets with the DS field of 000000 (best effort) are mapped.

If a router were to implement only the EF and the default PHBs, it would need to implement only two queues, which are served round-robin with specific weights assigned to them. If the router were to implement EF, AF, and the default, it would need three queues overall, each with a different assigned weight.

4.4 TRAFFIC POLICING AND SHAPING

Traffic policing and shaping functions are performed in both ATM and IP network. ATM has usage parameter control (UPC) and similar mechanisms are required at the edge of autonomous networks for IntServ and DiffServ. Policing functions monitor traffic flows and take corrective actions when the observed characteristics deviate from a specific traffic contract. The mark action or drop action taken by policing functions are determined by an SLA between the user and the network providers. On the other hand, shaping is to control burst data or traffic rate to send data streams to ensure conformance to a traffic contract. If a flow does not conform to the traffic contract, shaping function can be used to delay nonconforming traffic until they conform to the profile. Therefore, a shaper usually has a finite-size buffer, and packets may be discarded if the buffer space is not enough to hold the delayed packets.

Policers and shapers usually identify traffic descriptor violations in an identical manner. They usually differ, however, in the way they respond to violations [6].

- A policer typically drops packets. For example, a committed access rate (CAR) rate-limiting policer will either drop the packet or rewrite its IP precedence code, resetting the packet header's type of service bits.

- A shaper typically delays excess traffic using a buffer, or queueing mechanism, to hold packets and shape the flow when the data rate of the source is higher than expected. For example, general traffic shaping (GTS) uses a weighted fair queue to delay packets in order to shape the flow, and frame relay traffic shaping (FRTS) uses either a priority queue (PQ), a custom queue (CQ), or a first-in-first-out (FIFO) queue for the same, depending on how one configures it.

4.4.1 Location of Policing and Shaping Functions

Figure 4.9 shows the generic placement of policing and shaping functions in the network. Usually, first node of the network performs policing and the end user performs the shaping, as shown by the solid box in the figure. One network may police the traffic received from another and shape the traffic before sending to downstream network, but it is optional. Properly shaped traffic never fails a policing check when both functions employ the same traffic contract. The downstream network node should perform policing accounting for accumulated impairments, unless the previous network performs the shaping function.

As illustrated in Figure 4.10, a router or switch is a collection of ports interconnected by a switch fabric. Both input side and output side perform traffic management functions. Beginning on the input side, a policing function determines whether the incoming flow is in conformance to the traffic contract or not. Next, buffer management block determines whether the incoming packets should be accepted or not, and how to discard packets. Also a packet scheduling block is placed next in line to determine which packet to serve next. Buffer management and packet scheduling functions are performed also at the output side. Then a shaping function appears before sending traffic to downstream network. Policing and shaping are discussed in this section. Packet scheduling and buffer management will be explained in Sections 4.5 and 4.6, respectively.

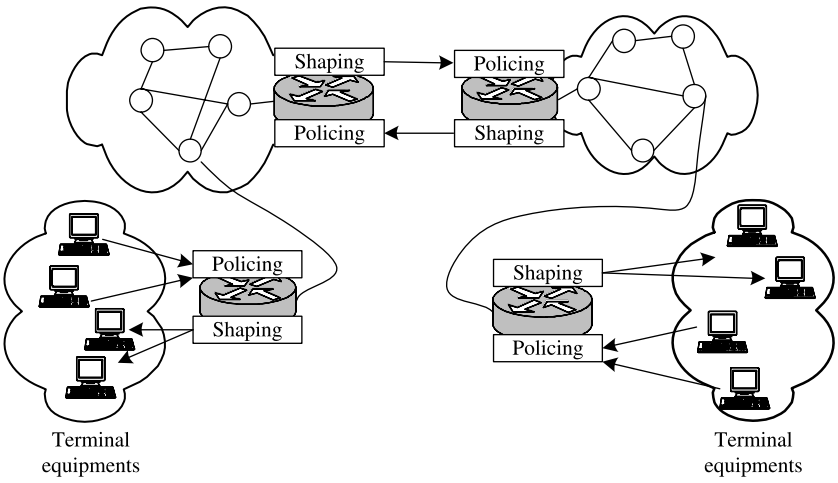


Figure 4.9 Generic location of policing and shaping function.

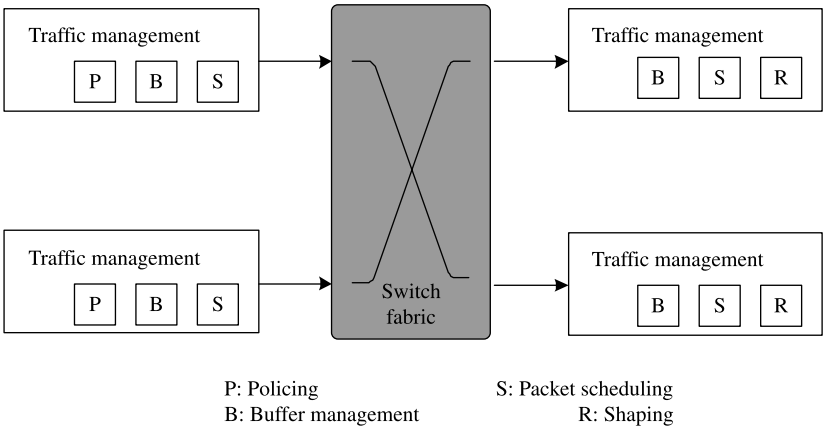


Figure 4.10 Generic block diagram of traffic management in router/switch.

4.4.2 ATM's Leaky Bucket

The leaky bucket algorithm is a general algorithm that can be effectively used to police and shape real time traffic. ATM networks use a continuous state version of the leaky bucket algorithm called the generic cell rate algorithm (GCRA) to police traffic at the entrance to the ATM network.

The leaky bucket algorithm is the key to define the meaning of conformance. The leaky bucket analogy refers to a bucket with a hole in the bottom that causes it to 'leak' at a certain rate corresponding to a traffic cell rate parameter (e.g., PCR: peak cell rate). The 'depth' of the bucket corresponds to a tolerance parameter (e.g., CDVT: cell delay variation tolerance). Each cell arrival creates a 'cup' of fluid flow 'poured' into one or more buckets for use in conformance checking. The cell loss priority (CLP) bit in the ATM cell header determines which bucket(s) the cell arrival fluid pours into.

In the algorithm, a cell counter represents the bucket. This counter is incremented by one for each incoming cell. The 'leak rate' in the algorithm is the decrement rate which reduces the counter value by one at certain intervals. This rate is given by the cell rate under consideration (e.g., 1/PCR) and is governed by the minimum distance between two consecutive cells. The bucket volume is analogous to the cell counter range, which is represented by the permissible time tolerance for the incoming cells. This value is determined through the traffic contract or is set by the network provider and is called CDVT. If the counter exceeds a certain value, the cells are assumed not to conform to the contract. To counteract this, nonconforming cells can now either be marked (CLP-1) or dropped. The algorithm is called 'dual leaky bucket' if several parameters (e.g., PCR and SCR) are monitored at once, or 'single leaky bucket' if only one parameter is monitored.

The GCRA algorithm is characterized by an increment I and a limit L. The increment denotes the average interarrival time between cells. The limit places a bound on the burstiness of the traffic. Figure 4.11 shows the flowchart of GCRA and the equivalent continuous state leaky bucket algorithms.

In GCRA, TAT denotes the theoretical arrival time of the next cell, that is, the time when the next cell is expected to arrive. If the cell arrives after TAT, then the cell is conforming, and TAT is set to the sum of the cell's arrival time and the increment I. If the cell arrives

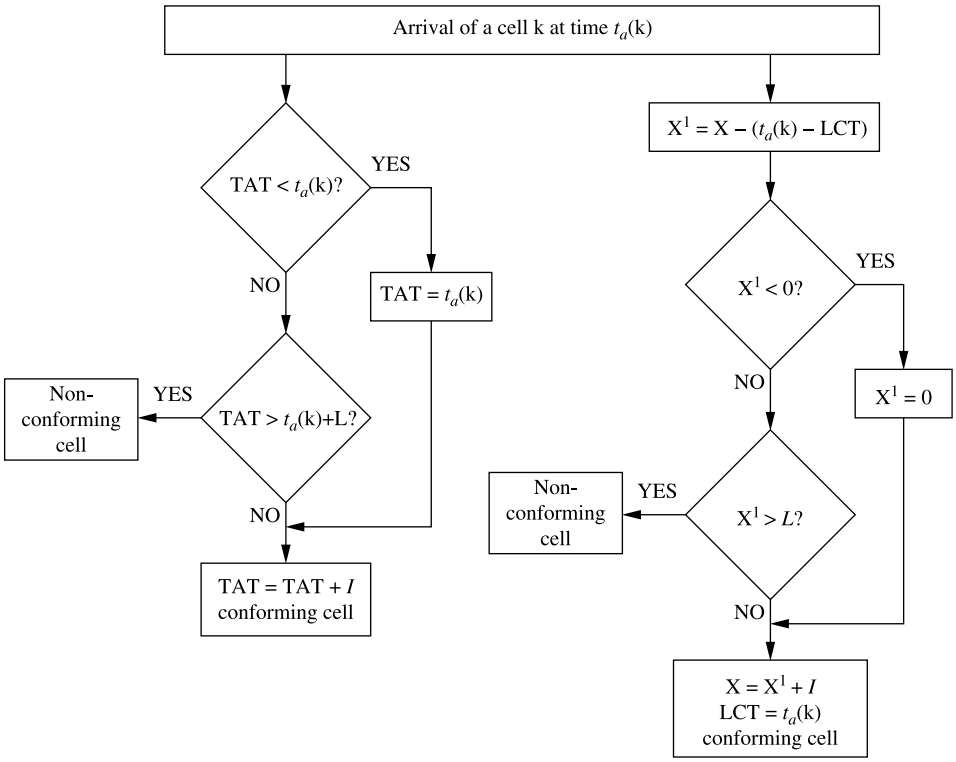


Figure 4.11 Leaky bucket algorithm for generic cell rate algorithm (GCRA).

before TAT, that is, the actual arrival time t_a of the cell is less than TAT, then the cell can arrive within a limited period before TAT. This period is denoted by limit L. If TAT $> t_a + L$ then the cell is marked non-conforming and the TAT is left unchanged. Otherwise, the cell is marked conforming and TAT is incremented by the increment I.

In the continuous state leaky bucket algorithm, a counter leaks from a bucket at each time unit. The bucket initially holds 0 counters. The total capacity of the bucket is L. The arrival of a cell adds I counters to the leaky bucket. When a cell arrives, if the bucket has leaked enough so that the addition of I counters will not cause the bucket to overflow, then the cell is conforming, else the cell is nonconforming.

4.4.3 IP's Token Bucket

In the Internet, a token bucket is used instead of a leaky bucket to control the average transmission rate and burst duration. A token bucket of a traffic regulator has three components:

- Mean rate r. It is measured in bytes of IP datagrams per unit time on average permitted by the token bucket. Values of this parameter may range from 1 byte per second to 40 tera-bytes per second [7].
- Bucket depth b. It specifies in bits per burst that can be sent within a given unit of time without creating scheduling concerns. Values of this parameter may range from 1 byte to 250 gigabytes [7].
- Time interval, also called the measurement interval. It specifies the time quantum in seconds per burst.

In the token bucket metaphor, tokens are put into the bucket at a certain rate. The bucket itself has a specified capacity. If the bucket fills to capacity, newly arriving tokens are discarded. Each token is the permission for the source to send a certain number of bits into the network. To transmit a packet, the regulator must remove from the bucket a number of tokens equal in representation to the packet size.

If not enough tokens are in the bucket to send a packet, the packet either waits until the bucket has enough tokens or the packet is discarded. If the bucket is already full of tokens, incoming tokens overflow and are not available to future packets. Thus, at any time, the largest burst a source can send into the network is roughly proportional to the size of the bucket.

Figure 4.12 shows the operation of the token bucket algorithm. Algorithm adds the tokens to the bucket at a rate of r bytes per second. An incoming packet conforms to the token bucket traffic specification, if the level of tokens in the bucket X equals or exceeds the length of incoming packet. Specifically, the arbiter checks the incoming packet length L and the current bucket level X. If $L \leq X$, then the packet conforms to the token bucket

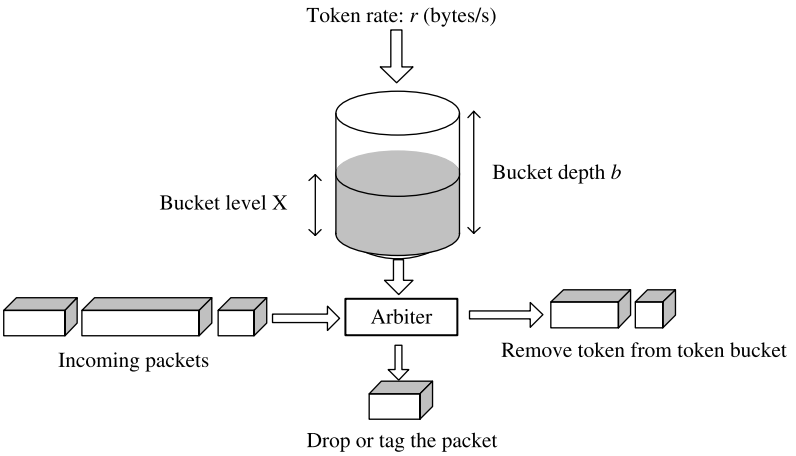


Figure 4.12 Operation of token bucket.

traffic specification and L bytes from token bucket are removed. Otherwise, the packet is nonconforming and no tokens is removed. The relationship between token bucket parameters (r, b) and the amount of data sent over time interval T, $D(T)$, as follows:

$$D(T) \leq rT + b \tag{4.1}$$

This equation limits the maximum amount of data that can arrive over an interval T via a linear function. Therefore, the actual average rate over an interval T can be described as $A(T) = D(T)/T = r + b/T$. Note that, as $T \to \infty$, the actual rate $A(T)$ approaches the desired rate r.

4.4.4 Traffic Policing

The policing mechanism in ATM network is called UPC/NPC (usage parameter control/network parameter control), which enforces traffic contract between users and network or between networks. Without policing, unfair situations where vicious user greedily take up resources can occur. In Internet, policing function is performed similar to the one in ATM.

A leaky bucket or a token bucket is utilized for policing, thus it can pass temporary bursts that exceed the rate limit as long as tokens are available. Once a packet has been classified as conforming or exceeding a particular rate limit, the router performs one of the following actions to the packet:

Transmit. The packet is transmitted.

Drop. The packet is discarded.

Set precedence and transmit. The ToS bits in the packet header are rewritten. The packet is then transmitted. One can use this action to either color (set precedence) or recolor (modify existing packet precedence) the packet.

Continue. The packet is evaluated using the next rate policy in a chain of rate limits. If there is no another rate policy, the packet is transmitted.

Multiple Rate Policies. A single committed access rate (CAR) policy includes information about the rate limit, conform actions, and exceed actions. Each interface can have multiple CAR policies corresponding to different types of traffic. For example, low priority traffic may be limited to a lower rate than high priority traffic. When there are multiple rate policies, the router examines each policy in the order entered until the packet matches. If no match is found, the default action is to transmit. Rate policies can be independent: each rate policy deals with a different type of traffic. Alternatively, rate policies can be cascading: a packet may be compared to multiple different rate policies in succession. Cascading of rate policies allows a series of rate limits to be applied to packets to specify more granular policies. For example, one could limit total traffic on an access link to a specified subrate bandwidth and then limit World Wide Web traffic on the same link to a given proportion of the subrate limit. One could configure CAR to match packets against an ordered sequence of policies until an applicable rate limit is encountered, for instance, rate limiting several MAC addresses with different bandwidth allocations at an exchange point.

4.4.5 Traffic Shaping

Traffic shaping allows one to control the traffic going out of an interface in order to match its flow to the speed of the remote, target interface and to ensure that the traffic conforms to policies contracted for it. Thus, traffic adhering to a particular profile can be shaped to meet downstream requirements, thereby eliminating bottlenecks in topologies with data-rate mismatches.

The primary reasons one would use traffic shaping are to control access to available bandwidth, to ensure that traffic conforms to the policies established for it, and to regulate the flow of traffic in order to avoid congestion that can occur when the transmitted traffic exceeds the access speed of its remote, target interface. Here are some examples:

- Control access to bandwidth when, for example, policy dictates that the rate of a given interface should not, on the average, exceed a certain rate even though the access rate exceeds the speed.
- Configure traffic shaping on an interface if one has a network with differing access rates. Suppose that one end of the link in a frame relay network runs at 256 kbps and the other end of the link runs at 128 kbps. Sending packets at 256 kbps could cause failure of the applications using the link. A similar, more complicated case would be a link-layer network giving indications of congestion that has different access rates on different attached data terminal equipment (DTE); the network may be able to deliver more transit speed to a given DTE at one time than another (this scenario warrants that the token bucket be derived, and then its rate maintained).
- Configure traffic shaping if one offers a subrate service. In this case, traffic shaping enables one to use the router to partition T1 or T3 links into smaller channels.

For traffic shaping, a token bucket permits burstiness but bounds it. It guarantees that the burstiness is bounded so that the flow will never send faster than the token bucket's capacity plus the time interval multiplied by the established rate at which tokens are placed in the bucket. It also guarantees that the long-term transmission rate will not exceed the established rate.

The full RSVP traffic specification starts with the token bucket specification and adds three additional parameters: a minimum-policed unit m, a maximum packet size M, and a peak rate p. The packet size parameters, m and M, include the application data and all protocol headers at or above the IP level (e.g., IP, TCP, UDP, etc.). They exclude the link-level header size.

The minimum-policed unit requires that the device removes at least m token bytes for each conforming packet. The parameter m also allows the device to compute the peak packet-processing rate as b/m. It also bounds the link-level efficiency by $H/(H + m)$ for a link-level header of H bytes. The maximum packet size M determines the largest permissible size of a conforming packet. In other words, any arriving packet with a length greater than M bytes is nonconforming. The measure for the peak traffic rate p is also bytes of IP datagrams per second, with the same range and encoding as the token bucket parameter r. When the peak rate equals the link rate, a node may immediately forward packets that conform to the token bucket parameters. For peak rate values less than the link rate, a leaky bucket shaper following the token bucket conformance checking ensures that the transmitted data $D(t)$ over any interval of time T satisfies the following inequality: $D(T) \leq Min[pT + M, rT + b]$.

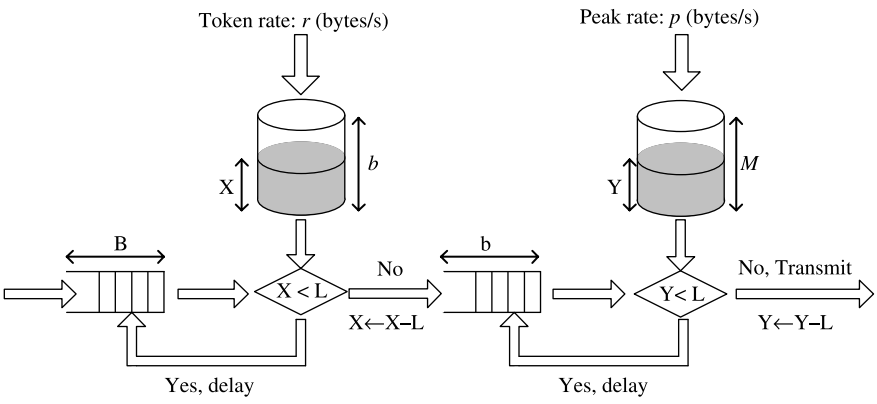


Figure 4.13 Traffic shaping via buffering in conjunction with a token bucket and peak regulate.

Figure 4.13 shows the block diagram model for a token bucket with parameters r and b operating in conjunction with a peak rate shaper with parameters p and M. The shaping buffer must be of size b bytes to admit a burst of packets conforming to the token bucket. The peak rate shaper only operates on packets conforming to the token bucket parameters. The combination of buffers and regulators delays packets until transmission is in conformance with both the token bucket and peak rate parameters. The logic that compares the number of tokens in the bucket with the length of the first packet in the buffer achieves this objective.

4.5 PACKET SCHEDULING

Packet networks allow users to share resources such as buffer and link bandwidth. However, the problem of contention for the shared resources arises inevitably. Given a number of users (flows or connections) multiplexed at the same link, a packet scheduling discipline is needed to determine which packets to serve (or transmit) next. In other words, sophisticated scheduling algorithms are required to prioritize users' traffic to meet various QoS requirements while fully utilizing network resources. For example, real-time traffic is delay-sensitive, while data traffic is loss-sensitive. This section presents several packet scheduling schemes. They are typically evaluated along several dimensions, such as tightness of delay bounds, achievable network utilization, fairness and protection, protocol overhead, computational cost, and robustness. Figure 4.14 illustrates a packet scheduler, which, for instance, can be located at the output of a switch/router. The packet scheduler, based on a scheduling algorithm, determines the departure sequences of the packets stored in the memory. Several packet scheduling schemes are discussed below.

4.5.1 Max-Min Scheduling

One of the goals for packet scheduling is to fairly allocate the available bandwidth among sessions (see Fig. 4.15 for example). The link bandwidth between user A and router R1 is 10 Mb/s, between user B and router R1 is 100 Mb/s, and between R1 and destination C is 1.1 Mb/s. Here comes the question: what is the 'fair' allocation, 0.55 Mb/s and 0.55 Mb/s or 0.1 Mb/s and 1 Mb/s for A and B, individually?

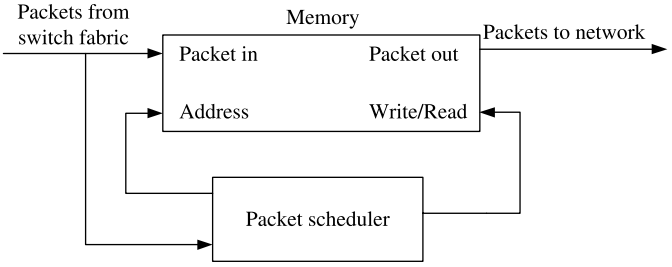


Figure 4.14 Packet scheduler.

The classical principle to this problem is the max-min fairness criteria as discussed, for instance, by Bertsekas and Gallager [8]. One possible way to define fairness for a rate allocation scheme is to require that each user should obtain the same transmission rate. For example, when R users access a single link with capacity C, a fair allocation would give each user a transmission rate equal to C/R. Applying this notion to a network consisting of several links could lead to an inefficient use of link resources [5, 10]. Instead, one could first allocate the same transmission rate to all users, and then share the remaining network bandwidth to fully utilize the network. Or more formally, a max-min fair allocation can be defined as follows.

Considering N flows share a link of rate C. Flow $f_i (1 \leq i \leq N)$ wishes to send at rate $W(f_i)$, and is allocated rate $R(f_i)$. We call a rate allocation $\{R(f_1), \ldots, R(f_N)\}$ is feasible, when $\sum_{i=1}^{N} R(f_i) \leq C$. We call a feasible allocation $\{R(f_1), \ldots, R(f_N)\}$ is max-min fair, when it is impossible to increase the rate of a flow p without losing feasibility or reducing the rate of another flow q with a rate $R(f_p) \leq R(f_q)$. Roughly, this definition states that a max-min fair allocation gives the most poorly treated user (i.e., the user who receives the lowest rate) the largest possible share, while not wasting any network resources.

A common way to allocate flows can be described as follows. It is illustrated by an example shown in Figure 4.16, where four flows share a link from router R1.

Step 1. Pick the flow f_j from set $\{R(f_n)\}$ $(1 \leq j \leq N)$, with the smallest requested rate.
Step 2. If $W(f_j) \leq C/N$, then set $R(f_j) = W(f_j)$.
Step 3. If $W(f_j) > C/N$, then set $R(f_j) = C/N$.
Step 4. Set $N = N - 1$, $C = C - R(f_j)$, remove flow f_j from set $\{R(f_n)\}$.
Step 5. If $N > 0$ goto Step 1.

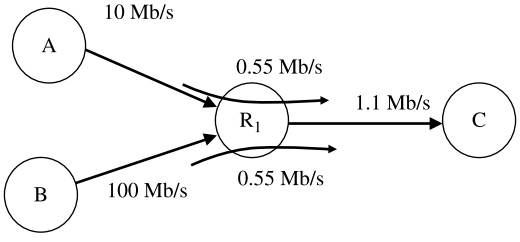
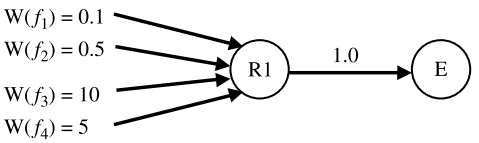


Figure 4.15 What is the 'fair' allocation: (0.55 Mb/s, 0.55 Mb/s), or (0.1 Mb/s, 1 Mb/s)? [9].



Round 1: Set R(f_1) = 0.1
Round 2: Set R(f_2) = 0.9/3 = 0.3
Round 3: Set R(f_3) = 0.6/2 = 0.3
Round 4: Set R(f_4) = 0.3/1 = 0.3

Figure 4.16 Max-min fairness example [9].

4.5.2 Round-Robin Service

In a round-robin (RR) scheduler, newly arriving packets are stored in different queues based on their flow IDs or service classes. The server polls each queue in a cyclic order and serves a packet from any nonempty queue encountered. A misbehaving user may overflow its own queue but will not affect others'. The RR scheduler attempts to treat all users equally and provide each of them an equal share of the link capacity (in other words, providing isolation among the users). It performs reasonably well when all users have equal weights and all packets have the same size (like cells in ATM networks).

When the number of queues is large (e.g., a few thousands or a few hundreds of thousands) and the link rate is high, it is impractical to poll every queue in a round-robin manner to determine which queue has a cell to send. One practical way to implement the RR scheduler is to use a so-called departure queue (DQ) to store the indices of the queues that are nonempty, as shown in Figure 4.17. Here, we consider fixed-length service unit (cell). The packet scheduler at the output of the switch has M flow queues (FQ) and one DQ. There can be up to N cells from the switch fabric arriving at the scheduler in each cell time slot (assuming the switch size of N inputs and N outputs). They are stored in a cell memory (not shown in the figure) and their addresses in the memory (e.g., A_i, A_j, and A_k) are stored in the corresponding FQs. For every nonempty FQ, its FQ index (1 to M) will be stored in the DQ (only one copy from each FQ).

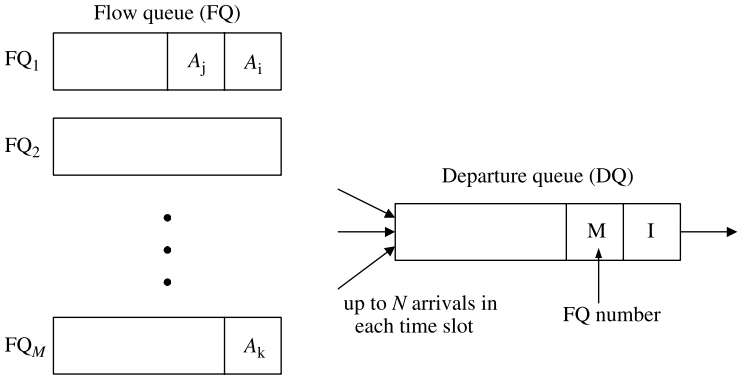


Figure 4.17 Implementation of round-robin service without a system bottleneck.

In each time slot, the DQ chooses the head-of-line (HOL) index and transmits its HOL cell. If the FQ is nonempty after sending the HOL cell, its index is written back to the tail of the DQ. If the FQ becomes empty, do nothing. When a cell arriving at an empty FQ, its FQ index is immediately inserted to the tail of the DQ.

With this approach, in each time slot there may be up to N cells written to the memory, N memory addresses written to the FQs, one access to the DQ to get the HOL FQ index, and one memory access to transmit the cell. It is independent of the number of FQs (M), which is usually much larger than the switch size (e.g., a few tens in general). Thus, it is very scalable. One thing worthy to notice is that the output cell sequence may not be completely identical with the one from the RR service. But, they both achieve the same fairness in the long run.

4.5.3 Weighted Round-Robin Service

Consider a weighted round robin (WRR) system in which every connection i has an integer weight w_i associated with it. When the server polls connection i, it will serve up to w_i cells before moving to the next connection in the round. This scheme is flexible since one can assign the weights corresponding to the amount of service required. The scheme attempts to give connection i a rate of

$$g_i = \frac{w_i}{\sum_j w_j} \tag{4.2}$$

For instance, a RR cell system in which there are two users, A and B with their weights $w_A = 3$ and $w_B = 7$ cells, respectively. The scheme attempts to give 30 percent $[= w_A/(w_A + w_B)]$ share of the link capacity to the user A and 70 percent $[= w_B/(w_A + w_B)]$ share to the user B. One possible outgoing cell sequence in a round is *AAABBBBBBB*. A better implementation of WRR operates according to a frame structure and gives *ABBABBABBBA*. That is, user A does not need to wait for seven cell time slots before having its cell sent.

The WRR service can be implemented by using a register storing weight (w_i) and a counter (C_i) for each FQ (FQ_i) as shown in Figure 4.18. The counter keeps track the

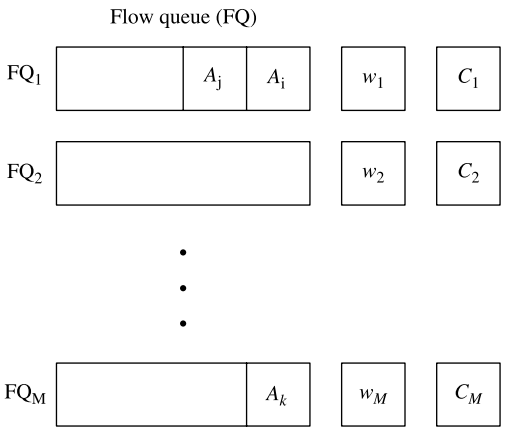


Figure 4.18 Traditional way of implementing the WRR service.

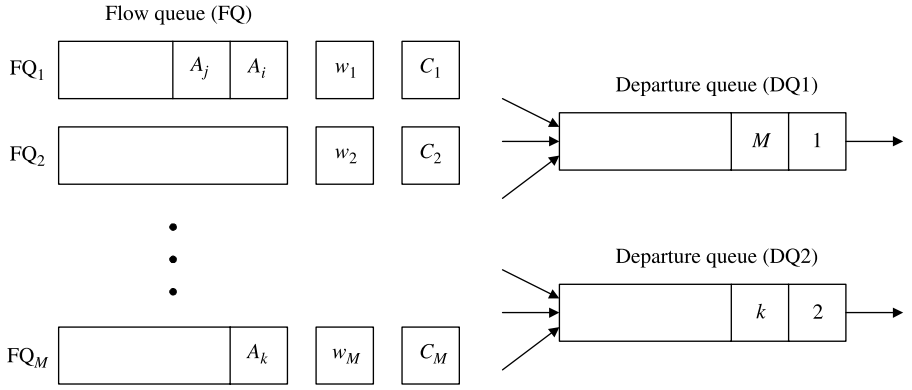


Figure 4.19 Implementation of WRR service without a system bottleneck.

number of cells that can be sent in a frame, whose size is the sum of all weights. Let us assume the frame size is F ($= \sum w_i$). A FQ is said to be eligible if it has cells to send and its counter value is not zero. At the beginning of a frame, each counter (C_i) is loaded with its weight (w_i). Each eligible FQ is served in RR manner. Upon a cell being served from a FQ, its C_i value is reduced by one. After F cells have been served, a new frame starts. However, if prior to F cells being served, no eligible VOQ can be found, a new frame starts. This scheme has a bottleneck from searching an eligible FQ when there are many of them.

To implement the WRR without a system bottleneck, we use two DQs as shown in Figure 4.19. One is active, responsible for transmitting cells, and the other is standby, denoted as DQ_a and DQ_s, respectively. The way of storing arriving cells to the memory and their addresses to the FQs is similar to the RR case. DQ_a chooses its HOL FQ to transmit its cell until it becomes empty. When that happens, the role of DQ_a and DQ_s swap. In the system initialization, each counter (C_i) is loaded with its weight (w_i). When a cell arrives at a FQ (say FQ_i), if it is a HOL cell and C_i is nonzero, it joins DQ_a (i.e., its index is inserted to the tail of DQ_a). If it is not a HOL cell, it simply joins the tail of its FQ. If it is a HOL cell but its C_i is zero, it joins DQ_s. Upon a cell being served from a FQ (say FQ_i), its C_i value is reduced by one. FQ_i is then checked. If it is nonempty and $C_i \neq 0$, insert i to the tail of DQ_a. If it is non-empty but $C_i = 0$, insert i to the tail of DQ_s. If FQ_i is empty, do nothing.

4.5.4 Deficit Round-Robin Service

Deficit round robin (DRR) [11] modifies the WRR to allow it to handle variable-sized packets in a fair manner. The basic idea is to use the RR service discipline with a quantum of service assigned to each queue. The only difference from traditional RR is that if a queue was not able to send a packet in the previous round because its packet size was too large, the remainder from the quantum is added to the quantum for the next round. Thus, deficits are recorded. Queues that were shortchanged in a round are compensated in the next round. The detailed algorithm is as follows.

Assume that each flow i is allocated Q_i worth of bits in each round; there is an associated state variable DC_i (Deficit Counter) recording the deficits. Since the algorithm works in rounds, we can measure time in terms of rounds. A round is one RR iteration over the queues that are backlogged. To avoid examining empty queues, an auxiliary list *ActiveList* is kept

and consists of a list of indices of queues that contain at least one packet. Packets coming in on different flows are stored in different queues. Let the number of bytes of HOL packet for queue i in round k be $\text{bytes}_i\ (k)$. Whenever a packet arrives at a previously empty queue i, i is added to the end of *ActiveList*. Whenever index i is at the head of *ActiveList*, say, in the round k, the algorithm computes $DC_i \leftarrow DC_i + Q_i$, sends out queue is HOL packet subject to the restriction that $\text{bytes}_i\ (k) \leq DC_i$ (and updates $DC_i \leftarrow DC_i - \text{bytes}_i\ (k)$ if the condition is true). If, at the end of this service opportunity, queue i still has packets to send, the index i is moved to the end of *ActiveList*; otherwise, DC_i is reset to zero and index i is removed from *ActiveList*. The DRR, however, is fair only over time scales longer than a round time. At a shorter time scale, some users may get more service (in terms of bits sent) than others.

An example showing how the DRR works is illustrated in Figure 4.20. At the beginning the counter variables DC_i are initialized to zero. The round robin pointer points to the top of the active list. When the first queue is served the quantum value of 500 is added to the DC_i value. The remainder after servicing the queue is left in the DC_i variable. After sending out a packet of size 200, the queue had 300 bytes of its quantum left. It cannot be served in the current round, since the next packet in the queue is 750 bytes. Therefore, the amount 300 will carry over to the next round when it can send packets of size totaling $300 + 500 = 800$ (deficit from previous round $+ Q_i$). The pointer now moves to FQ #2, where its DC is added with 500 bytes, allowing one HOL packet to be sent out.

4.5.5 Generalized Processor Sharing (GPS)

Generalized Processor Sharing (GPS) [12, 13] is an ideal scheduling policy in that it provides an exact max-min fair share allocation. The GPS is fair in the sense that it allocates the whole outgoing capacity to all backlogged sessions proportional to their minimum rate (bandwidth) requirements. Basically, the algorithm is based on an idealized fluid-flow model. That is, we assume that a GPS scheduler is able to serve all backlogged sessions instantaneously and the capacity of the outgoing link can be infinitesimally split and allocated to these sessions. However, in real systems only one session can be serviced at each time and packets cannot be split into smaller units. An important class of so-called packet fair queueing (PFQ) algorithms can then be defined in which the schedulers try to schedule the backlogged packets by approximating the GPS scheduler, such as weighted fair queueing (WFQ), virtual clock, and self-clock fair queueing (SCFQ), which are discussed in the following sections. We first introduce the ideal GPS algorithm.

Before getting into the detailed discussion of the GPS, let us use a couple of examples to briefly explain the concept. In Figure 4.21, four flows with equal weights share the link bandwidth. Their rate are 0.1, 0.3, 0.3, and 0.3, respectively. With GPS scheduling, the order of service for the four flows is: $f_1, f_2, f_3, f_4, f_2, f_3, f_4, f_2, f_3, f_4, f_1, \ldots$, where one bit is served at each time. In practice, the link shall serve packet by packet. Figure 4.22 illustrates how packet-mode GPS works, which is called WFQ.

Assume there are four queues labeled A, B, C, and D, sharing the link bandwidth of 4 bps with equal weight. Figure 4.22*a* shows the initial stage of the scheduling. There are one packet of four bits in queue A (A1), one packet of three bits in queue B (B1), two packets of one bit in queue C (C1 and C2), and two packets of one bit and two bits in queue D (D1 and D2). In round one, the first bit of each queue departs and two packets, A2 (two bits) and C3 (three bits), arrive (Fig. 4.22*b*). As C1 and D1 has only one bit, they depart in round one. Figures 4.22*c* and 4.22*d* show that the GPS scheduler goes on transmitting each

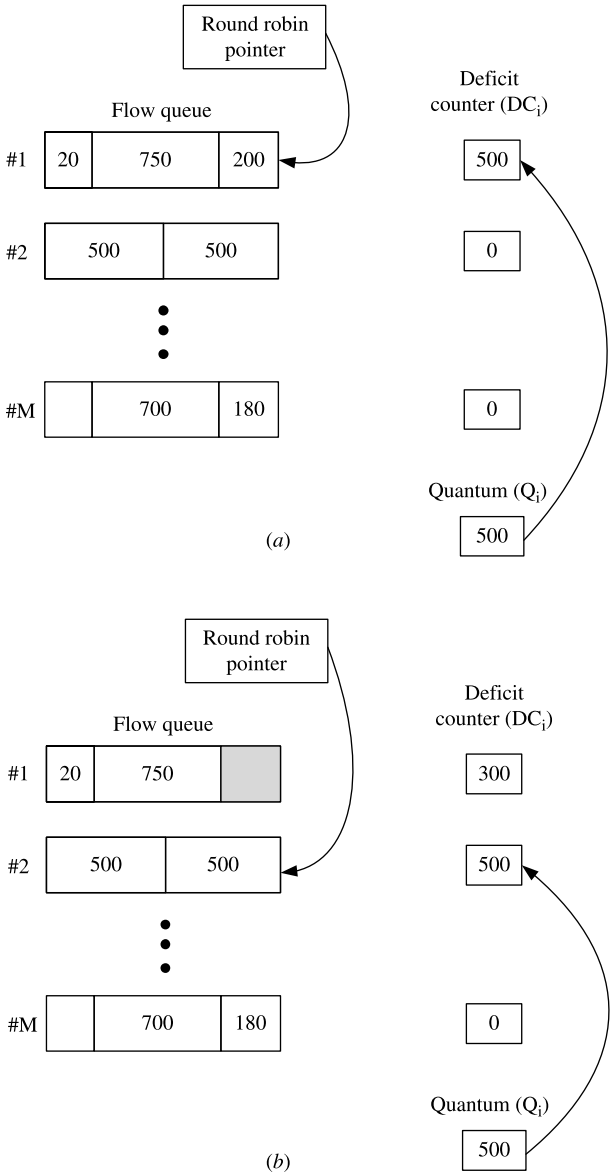


Figure 4.20 Example for DRR (*a*) The first step; (*b*) The second step.

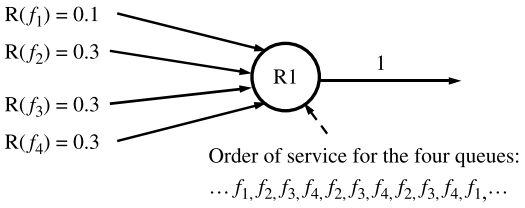


Figure 4.21 Weighted bit-by-bit fair queuing (GPS).

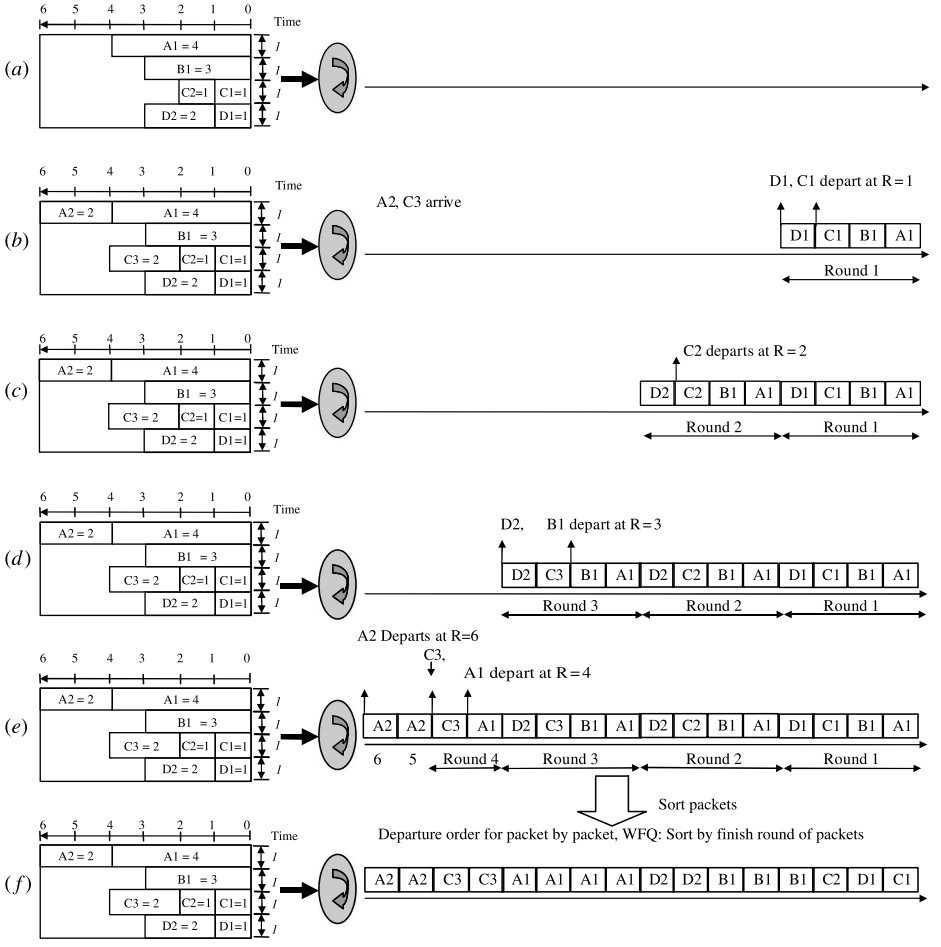


Figure 4.22 Illustrated example for packet-mode GPS with four queues, sharing 4 bits/sec bandwidth, and equal weights [9].

queue with one bit in each of the second and third rounds. C2 departs in round two, while B1 and D2 depart in round three. After three rounds scheduling, queues B and D become empty, and queues A and C are scheduled during round four as shown in Figure 4.22*e*. At the end of fourth round, only queue A is nonempty. So, A2 is scheduled during round 5 and round 6. Figure 4.22*f* shows the actual departure order for all queued packets, sorted by their finishing rounds.

Assume that a set of N sessions (connections), labeled $1, 2, \ldots, N$, share the common outgoing link of a GPS server. For $i \in \{1, 2, \ldots, N\}$, let r_i denote the minimum allocated rate for session i. The associated admission policy should guarantee that

$$\sum_{i=1}^{N} r_i \leq r \tag{4.3}$$

where r is the capacity of the outgoing link.

Let $B(t)$ denote the set of backlogged sessions at time t. According to the GPS, the backlogged session i will be allocated a service rate $g_i(t)$ at time t such that,

$$g_i(t) = \frac{r_i}{\sum_{j \in B(t)} r_j} \times r \tag{4.4}$$

We will use an example to illustrate the service rate allocation principle of GPS servers. Let $A_i(\tau, t)$ be the arrivals of session i during the interval $(\tau, t]$, $W_i(\tau, t)$ the amount of service received by session i during the same interval, and $Q_i(t)$ the amount of session i traffic queued in the server at time t, that is,

$$Q_i(t) = A_i(\tau, t) - W_i(\tau, t)$$

Note that, whenever the system becomes idle, all parameters can be reset to zero.

Definition 4.1. A *system busy period* is a maximal interval of time during which the server is always busy with transmitting packets.

Definition 4.2. A *backlogged period for session i* is any period of time during which packets of session i are continuously queued in the system.

Definition 4.3. *A session i busy period* is a maximal interval of time $(\tau_1, \tau_2]$ such that for any $t \in (\tau_1, \tau_2]$, packets of session i arrive with the rate greater than or equal to r_i, that is,

$$A_i(\tau_1, t) \geq r_i(t - \tau_1), \quad \text{for } t \in (\tau_1, \tau_2]$$

With reference to Figure 4.23, assume that the capacity of the server $r = 1$, and three connections, labeled 1, 2, and 3, share the same outgoing link of the server, where $r_1 = \frac{1}{6}$, $r_2 = \frac{1}{3}$, and $r_3 = \frac{1}{2}$.

Suppose that each packet has a fixed length that needs exactly one unit time to transmit. At time $t = 0$, session 1 starts a session busy period in which packets from session 1 arrive at the server with a rate one packet per unit time. At $t = 1$, packets from session 2 also start to arrive at the server with the same arrival rate. Session 3 starts a session busy period at $t = 3$ with the same packet arrival rate. The arrival curves of these three sessions are given in Fig. 4.24*a*.

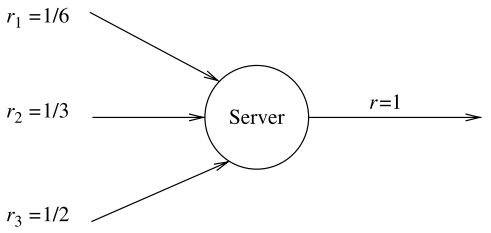


Figure 4.23 GPS server with three incoming sessions.

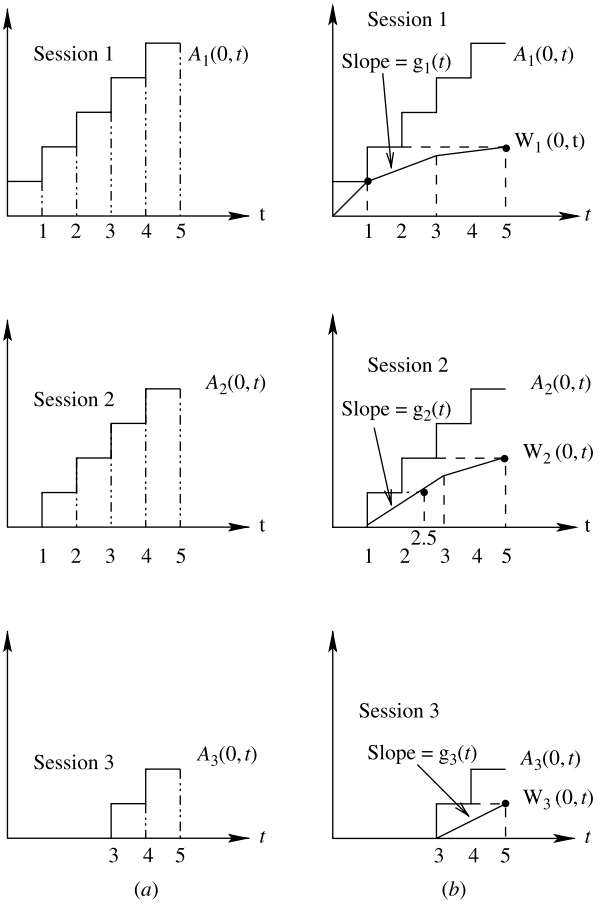


Figure 4.24 Arrival and service curves (*a*) Arrival curves; (*b*) Service curves.

The GPS server will allocate a service rate to each session as follows:

$$g_1(t) = \begin{cases} 1: & 0 < t \leq 1 \\ \frac{1}{3}: & 1 < t \leq 3 \\ \frac{1}{6}: & t > 3 \end{cases}$$

$$g_2(t) = \begin{cases} 0: & 0 < t \leq 1 \\ \frac{2}{3}: & 1 < t \leq 3 \\ \frac{1}{3}: & t > 3 \end{cases}$$

$$g_3(t) = \begin{cases} 0: & 0 < t \leq 3 \\ \frac{1}{2}: & t > 3 \end{cases}$$

The service curves are shown in Figure 4.24*b*. Note that $g_i(t)$ is also the slope of the service curve of session i at time t. Furthermore, at any time during the system busy period,

$\sum_{i=1}^{N} g_i(t) = r$ because of the work-conserving property.[1] The departure times of the first packet of sessions 1, 2, and 3 are 1, 2.5, and 5, respectively.

The *fairness index* of backlogged session i can be defined as $W_i(\tau_1, \tau_2)/r_i$. That is, during any time interval (τ_1, τ_2), for any two backlogged sessions i and j, the scheduler is said to be *perfectly fair* if and only if it always satisfies that

$$\frac{W_i(\tau_1, \tau_2)}{r_i} = \frac{W_j(\tau_1, \tau_2)}{r_j}$$

The GPS server is perfectly fair.

The GPS is an ideal PFQ service policy based on a fluid-flow model. However, because the fluid GPS is not practical, a class of PFQ algorithms has been proposed to emulate the fluid GPS to achieve the desired performance. The objective is to design an efficient and scalable architecture that can support hundreds of thousands of sessions in a cost-effective manner. All of them are based on maintaining a global function, referred to as either *system virtual time* or *system potential*, which tracks the progress of the GPS. This global function is used to compute a *virtual finish time* (or *time stamp*) for each packet or the head-of-line (HOL) packet of each session in the system. The time stamp of a packet is the sum of its *virtual start time* and the time needed to transmit this packet at its reserved bandwidth. Packets are served in an increasing order of their time stamps.

The implementation cost of a PFQ algorithm is determined by two components: (1) Computing the system virtual time function; and (2) Maintaining the relative ordering of the packets via their time stamps in a priority queue mechanism. Several PFQ algorithms are introduced in the following sections.

4.5.6 Weighted Fair Queuing (WFQ)

Although the GPS service principle is perfectly fair, the idealized fluid-flow model is not practical to implement. However, we can simulate the GPS server and then schedule the backlogged packets in accordance with the packet behavior of the simulated GPS server. A WFQ (also called Packetized GPS) system is defined with respect to its corresponding GPS system. Let d_p^{GPS} be the time at which packet p will depart (finish service) under GPS. A good approximation of GPS would be a scheme that serves packets in an increasing order of d_p^{GPS}. However, this is not always possible without causing the discipline to be nonwork-conserving. This is because when the packet system is ready to choose the next packet to transmit, the next packet to depart under GPS may not have arrived at the packet system yet. Waiting for it requires the knowledge of the future and also causes the system to be nonwork-conserving. In WFQ, the server simply assigns the departure time of a packet in the simulated GPS server as the time stamp of that packet, and then the server transmits packets in an increasing order of these time stamps. When the server is ready to transmit the next packet at time τ, it picks the first packet that would complete service in the corresponding GPS system as if no additional packets were to arrive after time τ.

Weighted fair queuing [12] uses the concept of *virtual time* to track the progress of GPS that will lead to a practical implementation of packet-by-packet GPS. Denote as an *event* each arrival and departure of sessions from the GPS server, and let t_j be the time at which

[1] A server is work-conserving if it is never idle whenever there are packets to be transmitted. Otherwise, it is nonwork-conserving.

the jth event occurs (simultaneous events are ordered arbitrarily). Let the time of the first arrival of a busy period be denoted as $t_1 = 0$. Now observe that, for each $j = 2, 3, \ldots$, the set of sessions that are busy in the interval (t_{j-1}, t_j) is fixed. We denote this set as B_j. Virtual time $V(t)$ is defined to be zero for all times when the server is idle. Consider any busy period, and let the time that it begins be time zero. Then, $V(t)$ evolves as follows:

$$\begin{aligned} V(0) &= 0, \\ V(t_{j-1} + \tau) &= V(t_{j-1}) + \frac{r\tau}{\sum_{i \in B_j} r_i}, \qquad \text{for } \tau \le t_j - t_{j-1},\ j = 2, 3, \ldots \end{aligned} \tag{4.5}$$

The rate of change of V, namely $dV(t_j + \tau)/d\tau$, is $r/\sum_{i \in B_j} r_i$, and each backlogged session i receives service at rate $r_i[dV(t_j + \tau)]/[d\tau]$, that is, $g_i(t_j + \tau)$ according to Eq. (4.4). Thus, V can be interpreted as increasing at the marginal rate at which backlogged sessions receive service.

Now suppose that the kth packet from session i arrives at time $a_{i,k}$ and has length $L_{i,k}$. Then, denote the virtual times at which this packet begins and completes service as $S_{i,k}$ (also called *virtual start time* [12] or *start potential* [14]) and $F_{i,k}$ (*virtual finish time* [12] or *finish potential* [14]), respectively. Defining $F_{i,0} = 0$ for all i, we have

$$\begin{aligned} S_{i,k} &= \max\{F_{i,k-1}, V(a_{i,k})\}, \\ F_{i,k} &= S_{i,k} + \frac{L_{i,k}}{r_i}. \end{aligned} \tag{4.6}$$

The role of $V(a_{i,k})$ is to reset the value of $S_{i,k}$ when queue i becomes active (i.e., receives one packet after being empty for a while) to account for the service it missed [14, 15]. Therefore, the start time of each backlogged queue can stay close to each other (they are the same in a GPS server).

For the above example, let the kth packet from backlogged session i be labeled by $(i, d_{i,k}^{GPS})$, where $d_{i,k}^{GPS}$ is the departure time of this packet in the simulated GPS server. Figure 4.25 shows the service curves and the departure order of packets in the WFQ server for the previous example. As shown in the right-hand side of Figure 4.25, packets depart according to the departure time in the GPS system. For those packets that have the same departure time, they are served arbitrarily.

On the other hand, let each packet be labeled by $(i, F_{i,k})$. Figure 4.26 shows the virtual time $V(t)$ according to [Eq. (4.5)]. Figure 4.27 shows the virtual finish time curves and the departure order of packets for the previous example. $F_i(t)$ denotes a stair-wise function of time t with $F_i(a_{i,k}) = F_{i,k}$, $a_{i,k}$ is the arrival time of the kth packet of session i and is calculated according to Eq. (4.6). Note that the departure order is the same with that in Figure 4.25.

In [12], Parekh establishes the following relationships between the GPS system and its corresponding packet WFQ system:

$$d_{i,k}^{\text{WFQ}} - d_{i,k}^{\text{GPS}} \le \frac{L_{\max}}{r}, \quad \forall i, k \tag{4.7}$$

$$W_i^{\text{GPS}}(0, \tau) - W_i^{\text{WFQ}}(0, \tau) \le L_{\max}, \quad \forall i, \tau \tag{4.8}$$

where $d_{i,k}^{\text{WFQ}}$ and $d_{i,k}^{\text{GPS}}$ are the times at which the kth packet on session i departs under WFQ and GPS, respectively, $W_i^{\text{WFQ}}(0, \tau)$ and $W_i^{\text{GPS}}(0, \tau)$ are the total amounts of service

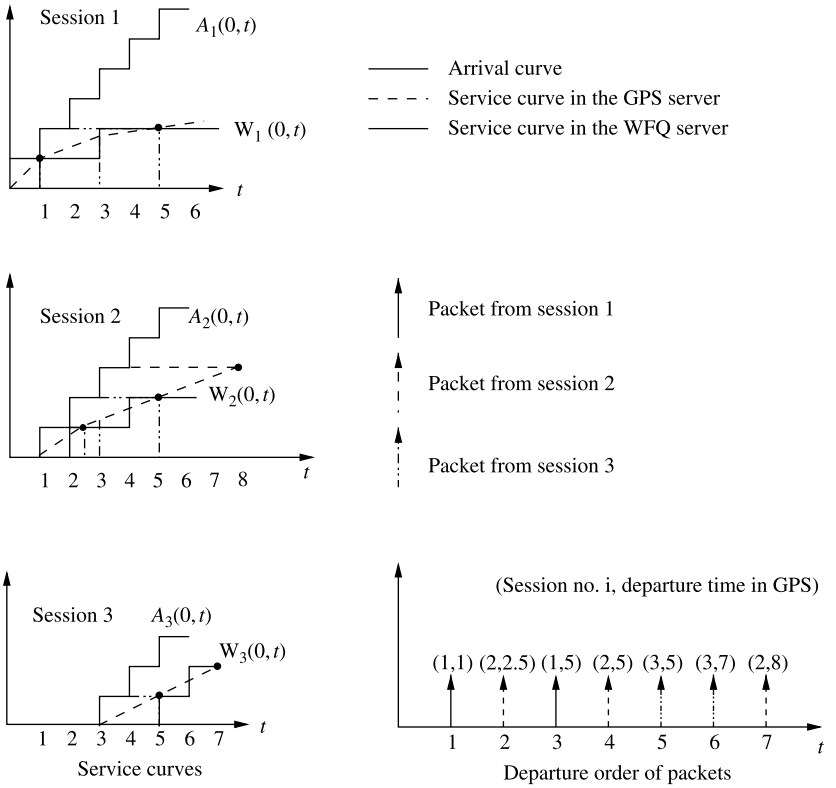


Figure 4.25 Service curves and the departure order of packets in the WFQ server. Packets depart according to the depart times as if the packets are served by a GPS server.

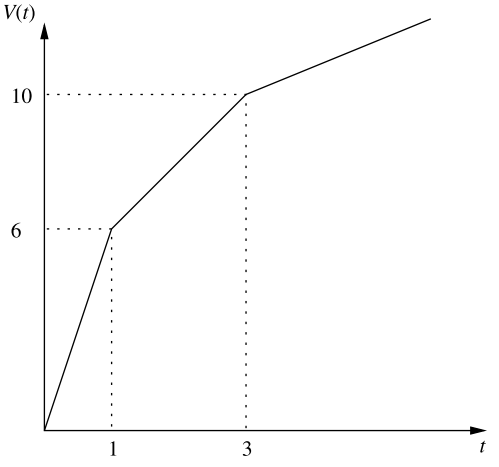


Figure 4.26 Virtual time function $V(t)$.

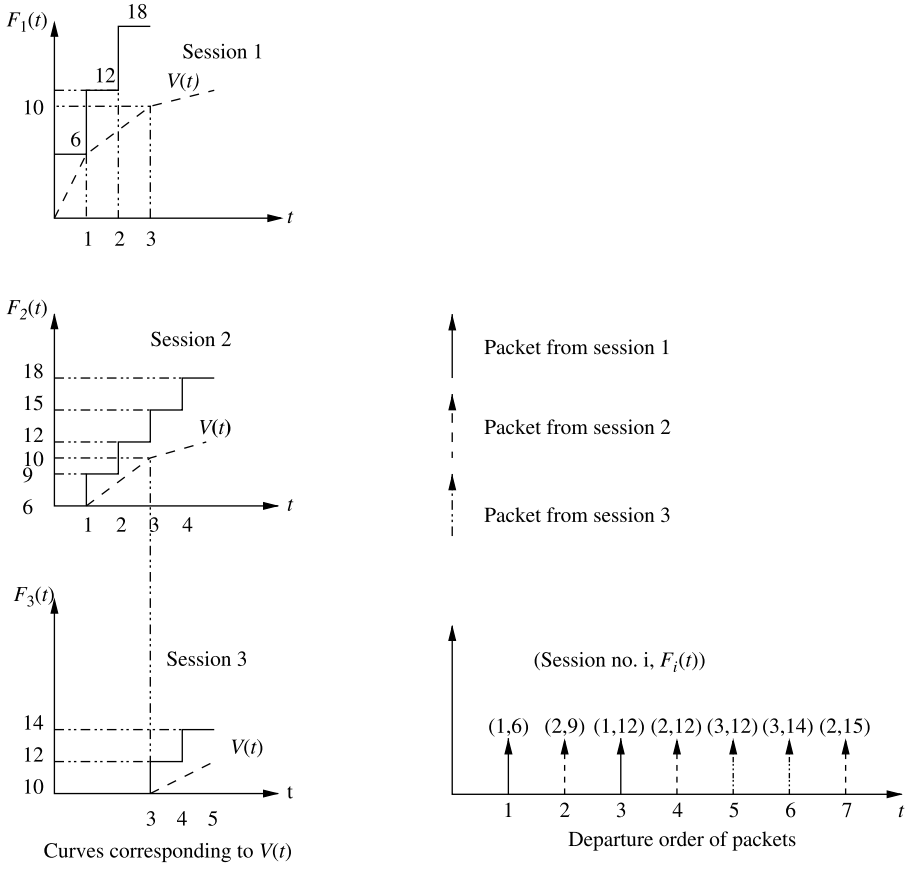


Figure 4.27 Virtual finish time stamps, $F(t)$, and the departure order of packets. $F(t)$ is calculated with the knowledge of virtual time, $V(t)$. Packets depart according to their $F(t)$ values.

received by session i (the number of session i bits transmitted) by time τ under WFQ and GPS, respectively, and $L_{\max}$ is the maximum packet length among all the sessions.

Another parameter called *latency* [14] can be defined and used to compare the performances of the WFQ and the GPS servers.

Definition 4.4. The *latency* of a server $\mathcal{S}$, $\Theta_i^{\mathcal{S}}$, is the minimum non-negative number that satisfies

$$W_{i,j}^{\mathcal{S}}(\tau, t) \geq \max\{0, r_i(t - \tau - \Theta_i^{\mathcal{S}})\}. \tag{4.9}$$

for any time t after time τ when the jth busy period started and until the packets that arrived during this period are served, where $W_{i,j}^{\mathcal{S}}$ is the service received by session i corresponding jth busy period.

With reference to Figure 4.28, the inequality (4.9) defines an envelope to bound the minimum service offered to session i during a busy period. It is easy to show that the

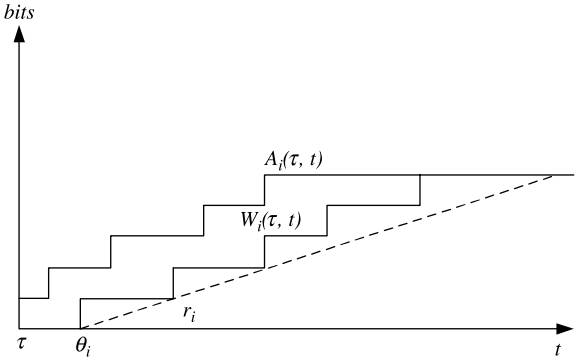


Figure 4.28 Latency.

latency Θ_i^S represents the worst-case delay seen by the first packet of a busy period of session i.

In the GPS server, the newly backlogged session can get served immediately with a rate equal to or greater than its required transmission rate. As a result, the latency is zero.

In the WFQ server, however, the worst-case delay of the first packet of a backlogged period for session i is $d_{i,1}^{\text{WFQ}} - a_{i,1}$, where $a_{i,1}$ is the arrival time of that packet. From inequality (4.7), we have,

$$d_{i,1}^{\text{WFQ}} - a_{i,1} \leq d_{i,1}^{\text{GPS}} + \frac{L_{\max}}{r} - a_{i,1} \leq \frac{L_i}{r_i} + \frac{L_{\max}}{r}$$

where L_i is the maximum packet size of session i. Thus, we can conclude that the latency of session i in the WFQ server is bounded by $(L_i/r_i) + (L_{\max}/r)$.

The WFQ algorithm has a time complexity of $O(N)$ because of the overhead in keeping track of sets B_j, which is essential in the updating of virtual time, where N is the maximum number of backlogged sessions in the server. Other functions of time have been proposed to approach the virtual time function such that the computation complexity of the scheduling algorithm can be further reduced. As shown in the following sections, all PFQ algorithms use a similar priority queue mechanism that schedules packet transmission in an increasing order of their time stamps, but differ in choices of system virtual time function and packet selection policies [14, 15].

4.5.7 Virtual Clock

The virtual clock (VC) scheduler uses a real time function to approach the virtual time function. That is, the scheduler assigns

$$V^{\text{VC}}(t) = t, \quad \text{for } t \geq 0. \tag{4.10}$$

The kth packet from session i will be assigned a time-stamp $F_{i,k}$ from Eq. (4.6) and Eq. (4.10), that is,

$$F_{i,k} = \max\{F_{i,k-1}, a_{i,k}\} + \frac{L_{i,k}}{r_i}, \tag{4.11}$$

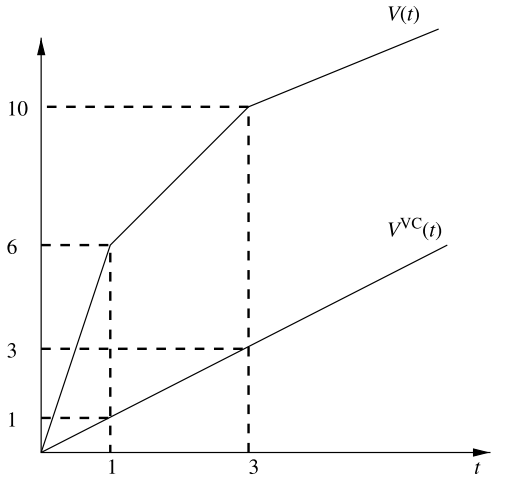


Figure 4.29 VC scheduler uses real time to approach the virtual time.

where $a_{i,k}$ is the arrival time of the kth packet of session i. Figure 4.29 shows the curves $V(t)$ and $V^{\mathrm{VC}}(t)$, and Figure 4.30 shows the service curves and the departure order of packets for the previous example.

Since the real time is always less than or equal to the virtual time, the VC scheduler can always provide the newly backlogged session with a latency smaller than or equal to that provided by the WFQ server [16].

However, the VC service discipline is defined with reference to the static time system and the calculation of the time stamp is independent of the behaviors of other sessions. As a result, if a connection has sent more packets than specified, it may be punished by the VC, regardless if such misbehavior affects the performance of other connections. For example, suppose there are two sessions 1 and 2 as shown in Figure 4.31.

All packets from both sessions are the same size. The link capacity is normalized as one packet per time slot. Let $r_1 = r_2 = 0.5$ packet/slot, so the time stamp for a session will be advanced by two slots each time its HOL packet is sent based on Eq. (4.11). Initially, at time 0 (in unit of slot), $F_{1,0} = F_{2,0} = 0$. Session 1 source continuously generates packets from time 0, while session 2 source starts to continuously send packet from time 900, as illustrated in Figure 4.31. Up to time 900, 900 packets from source 1 have been transmitted by the VC scheduler, and based on Eq. (4.11), $F_{1,901} = 1802$ at time 900, while $F_{2,1} = 902$. Therefore, the 901th packet arriving from session 1 at time 900 (with its stamp 1802) cannot get any service until the 449th packet from session 2 (arriving at time 1349 and stamped with 1800) finished its service. In other words, session 1 packets that arrived in the interval [900, 1349) are being punished since the session used the server exclusively in the interval [0, 900). But note that this exclusive use of the server was not at the expense of any session 2 packets.

Figure 4.30 also shows that session 1 is punished by the Virtual Clock scheduler when compared with the departure order in the Figure 4.27. In this case, old backlogged sessions must wait for the server to serve the newly backlogged session until its HOL packet has a time stamp larger than or equal to the time stamps of those old backlogged sessions. As a result, there is no fairness bound for the VC scheduling algorithm because there is no bound for $|W_i(\tau_1, \tau_2)/r_i - W_j(\tau_1, \tau_2)/r_j|$ when both sessions i and j are backlogged.

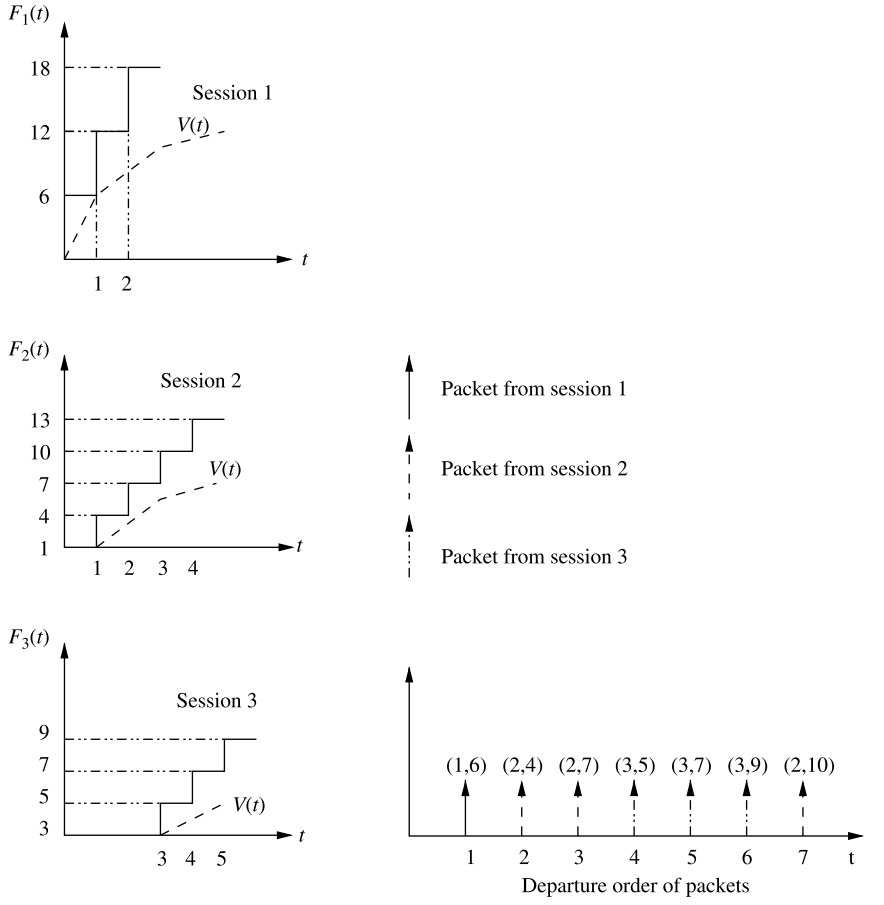


Figure 4.30 Virtual finish time stamps and the departure order of packets in the VC scheduler.

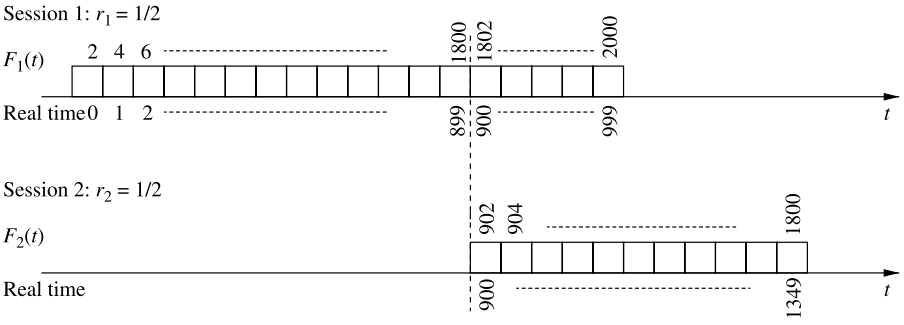


Figure 4.31 Example showing unfairness of the VC.

4.5.8 Self-Clocked Fair Queuing

The self-clocked fair queuing (SCFQ) [17] scheduler updates its virtual time function only when a packet departs, and the assigned value is equal to the time stamp of that packet. That is, the scheduler assigns

$$V^{\text{SCFQ}}(t) = F_{j,l}, \text{ if the } l\text{th packet of session } j \text{ departs at time } t \geq 0. \tag{4.12}$$

Similarly, the kth packet from session i will be assigned a time stamp $F_{i,k}$ from Eq. (4.6) and Eq. (4.12), that is,

$$F_{i,k} = \max\{F_{i,k-1}, V^{\text{SCFQ}}(a_{i,k})\} + \frac{L_{i,k}}{r_i}. \tag{4.13}$$

Figure 4.32 shows the curves corresponding to $V(t)$ and $V^{\text{SCFQ}}(t)$, respectively, while Figure 4.33 shows the time stamps and the departure order of packets for the previous example.

Figure 4.34 demonstrates that how the SCFQ is able to provide fairness guarantee for the same situation given in Figure 4.31 where the VC fails to do so. With reference to Figure 4.34, we can see that under the SCFQ, packets from both sessions are served in a round robin fashion according to their time stamps after session 2 becomes active at time 900.

Compared with the VC scheduler, the SCFQ can approach WFQ more accurately. However, the problem is that $V^{\text{SCFQ}}(t)$ can be larger than $V(t)$ as shown in Figure 4.32, and thus the latency can be very large. Consider a worst case situation where $(N-1)$ sessions have backlogged and their F values are the same. Assume where one packet is completely transmitted at τ, the virtual time is updated to the departed packet's virtual finish time F, say $V^{\text{SCFQ}}(\tau)$. Also assume session i becomes backlogged at time τ, $(N-2)$ HOL packets from other backlogged sessions have the same time stamp value as $V^{\text{SCFQ}}(\tau)$. Since the first packet of the newly backlogged session has a time stamp with a minimum

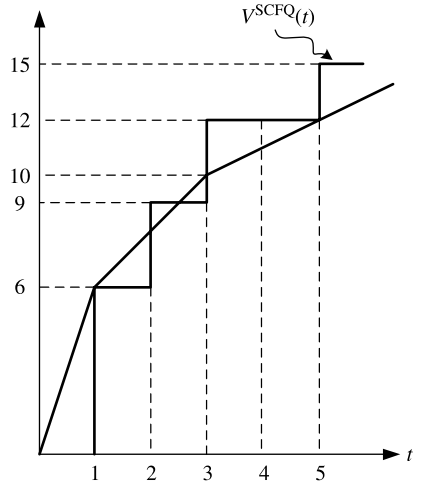


Figure 4.32 The SCFQ scheduler uses the time stamp of the last departed packet to approach the virtual time.

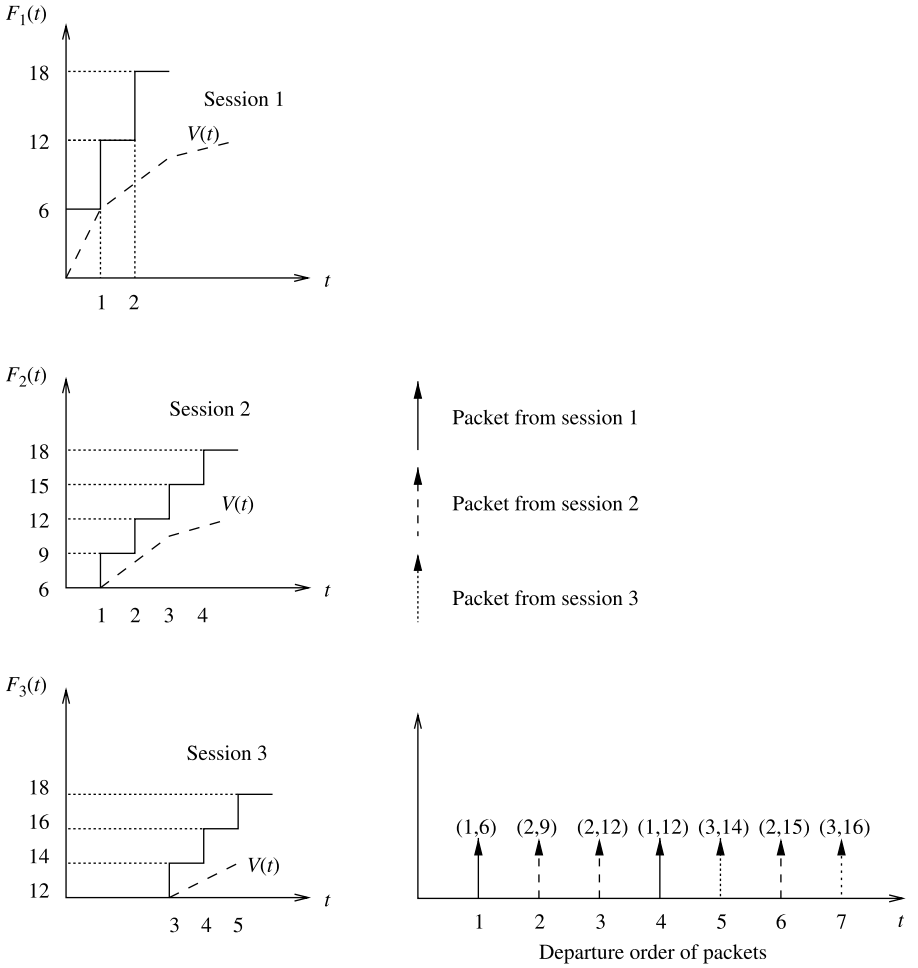


Figure 4.33 Virtual finish time stamps and the departure order of packets in the SCFQ scheduler.

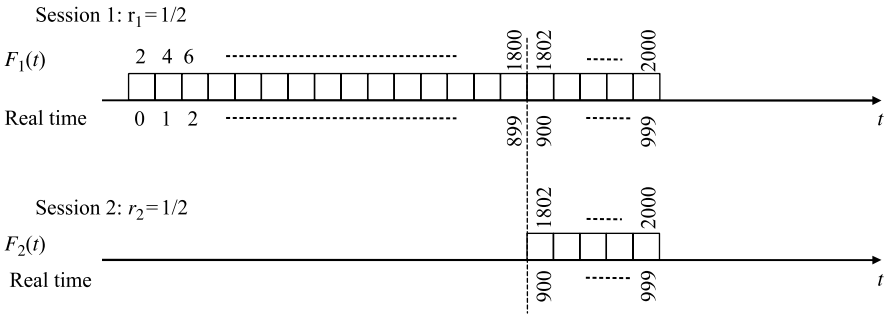


Figure 4.34 Example showing fairness of the SCFQ.

value $V^{\text{SCFQ}}(\tau) + (L_{i,1}/r_i)$, it may experience the worst-case delay time (i.e., latency) $(L_{i,1}/r_i) + (N-1)(L_{\max}/r)$ [14, 18]. The first $(N-2)(L_{\max}/r)$ is for those $N-2$ HOL packets being transmitted, and the last $(L_{i,1}/r_i) + (L_{\max}/r)$ is the same latency as in WFQ. As a result, the latency of SCFQ scheduler is $(L_{i,1}/r_i) + (N-1)(L_{\max}/r)$.

4.5.9 Worst-Case Fair Weighted Fair Queuing (WF^{2}Q)

The results given by Eq. (4.7) and Eq. (4.8) can be easily interpreted to be that WFQ and GPS provide almost identical service except the difference of one packet. What Parekh has proven is that WFQ cannot fall behind GPS with respect to service given to a session by one maximum size packet. However, packets can leave much earlier in a WFQ system than in a GPS system, which means that WFQ can be far ahead of GPS in terms of the number of bits served for a session.

Consider the example illustrated in Figure 4.35*a* where there are 11 sessions sharing the same link [19]. The horizontal axis shows the time line and the vertical axis shows the sample path of each session. For simplicity, assume all packets have the same size of 1 and the link speed is 1. Also, let the guaranteed rate for session 1 be 0.5, and the guaranteed rate for each of the other 10 sessions be 0.05.

In the example, session 1 sends 11 back-to-back packets starting at time 0 while each of the other 10 sessions sends only one packet at time 0. If the server is GPS, it will take two time units to service a session 1 packet and 20 time units to service a packet from another session. This is illustrated in Figure 4.35*b*. If the server is WFQ, at time 0, all 11 sessions have packets backlogged. Since packet $p_{1,1}$ (i.e., the first session 1 packet) finishes at time 2 while all other $p_{i,1}$ $(i = 2, \ldots, 11)$ packets finish at time 20 in the GPS system, WFQ will service $p_{1,1}$ first. In fact, the first 10 packets on session 1 all have finishing times smaller than packets belonging to any other session, which means that 10 packets on session 1 will be served back-to-back before packets on other sessions can be transmitted. This is shown in Figure 4.35*c*. After the burst the next packet on session 1, $p_{1,11}$, will have a larger finishing time in the GPS system than the 10 packets at the head of other sessions' queues. Therefore, it will not be serviced until all the other 10 packets are transmitted, at which time, another 10 packets from session 1 will be served back-to-back. This cycle of bursting 10 packets and going silent for 10 packet times can continue indefinitely. With more sessions, the length of the period between bursting and silence can be larger.

Such oscillation is undesirable for flow and congestion control in data communication networks. To quantify the discrepancy between the services provided by a packet discipline and the fluid GPS discipline, we consider the notion of worst-case packet fair as defined below [19].

Definition 4.5. A service discipline s is called *worst-case fair for session i* if for any time τ, the delay of a packet arriving at τ is bounded above by $[Q_i^s(\tau)/r_i] + C_i^s$, that is,

$$d_{i,k}^s < a_{i,k} + \frac{Q_i^s(a_{i,k})}{r_i} + C_i^s, \tag{4.14}$$

where r_i is the minimum bandwidth guaranteed to session i, $Q_i^s(a_{i,k})$ is the queue size of session i at time $a_{i,k}$ when the kth packet of session i arrives, and C_i^s is a constant independent of the queues of the other sessions multiplexed at the server.

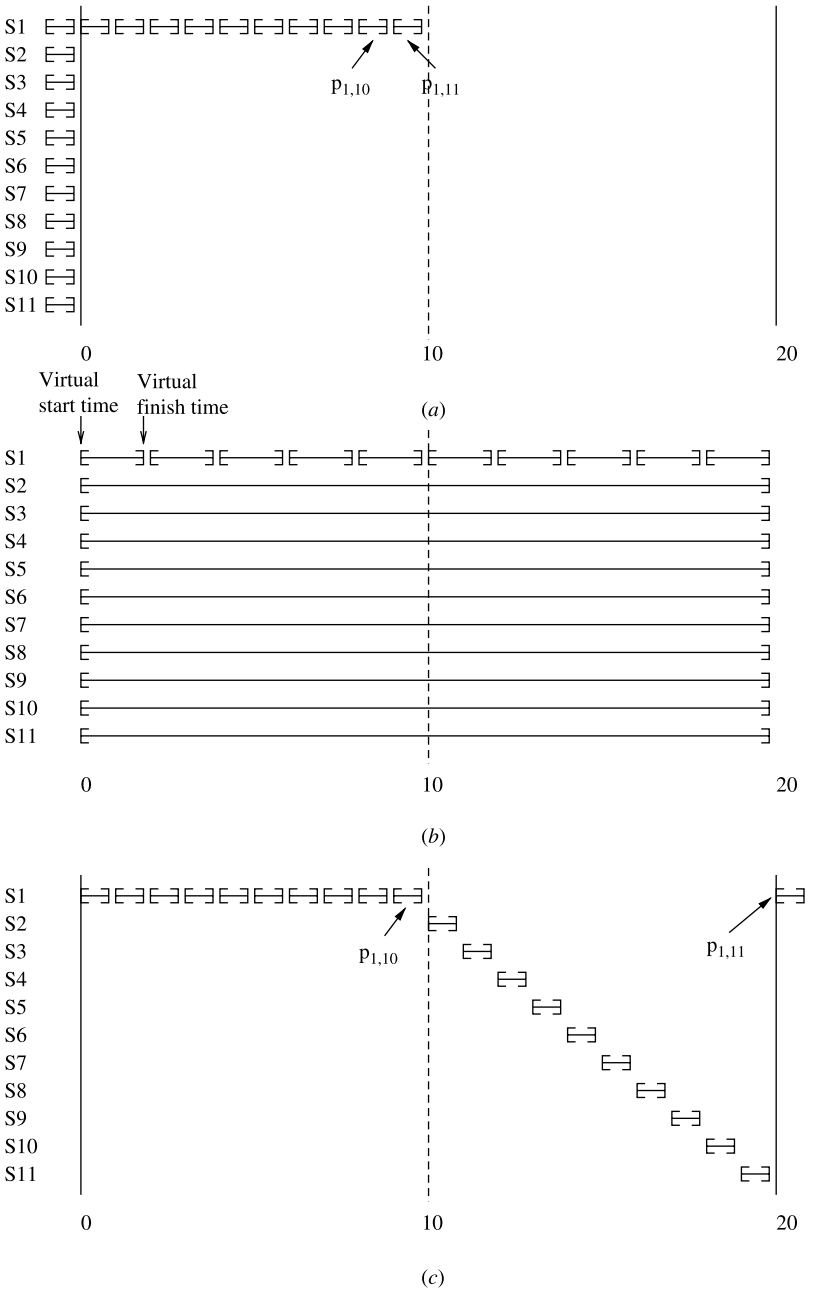


Figure 4.35 Example of GPS and WFQ service (*a*) Packet arrivals; (*b*) GPS service order and (*c*) WFQ service order (©1996 IEEE).

Definition 4.6. A service discipline is called *worst-case fair* if it is worst-case fair for all sessions.

Definition 4.7. C_i^s is called the *worst-case fair index* (WFI) for session i at server s.

Since C_i^s is measured in absolute time, it is not suitable for comparing C_i^ss of sessions with different r_is. To perform such a comparison, the normalized WFI for session i at server s can be defined as

$$c_i^s = \frac{r_i C_i^s}{r}. \tag{4.15}$$

For a server that is worst-case fair, we define its normalized WFI to be

$$c^s = \max_i \{c_i^s\}. \tag{4.16}$$

Notice that GPS is worst-case fair with $c^{\text{GPS}} = 0$. Thus, we can use c^s as the metric to quantify the service discrepancy between a packet discipline s and GPS. It has been shown in [19] that c^{WFQ} may increase linearly as a function of number of sessions N.

To minimize the difference between a packet system and the fluid GPS system, another class of scheduling algorithms called shaper-schedulers [19–23] has been proposed to achieve minimum WFI and have better worst-case fairness properties. With these algorithms, when the server is picking the next packet to transmit, it chooses, among all the *eligible* packets, the one with the smallest time stamp. A packet is eligible if its virtual start time is *no greater than* the current system virtual time. This is called the *eligibility test* or *smallest eligible virtual finish time first* (SEFF) policy [19, 20].

Worst-case fair weighted fair queuing or WF^{2}Q [19] is one such example. Recall that in a WFQ system, when the server chooses the next packet for transmission at time τ, it selects, among all the packets that are backlogged at τ, the first packet that would complete service in the corresponding GPS system. In a WF^{2}Q system, when the next packet is chosen for service at time τ, rather than selecting it from among all the packets at the server as in WFQ, the server only considers the set of packets that have started (and possibly finished) receiving service in the corresponding GPS system at time τ, and selects the packet among them that would complete service first in the corresponding GPS system.

Now consider again the example discussed in Figure 4.35 but in light of WF^{2}Q policy. At time 0, all packets at the head of each session's queue, $p_{i,1}$, $i = 1, \ldots, 11$, have started service in the GPS system [Figure 4.35*a*]. Among them, $p_{1,1}$ has the smallest finish time in GPS, so it will be served first in WF^{2}Q. At time 1, there are still 11 packets at the head of the queues: $p_{1,2}$ and $p_{i,1}$, $i = 2, \ldots, 11$. Although $p_{1,2}$ has the smallest finish time, it will not start service in the GPS system until time 2; therefore, it won't be *eligible* for transmission at time 1. The other 10 packets have all started service at time 0 at the GPS system; thus, they are eligible. Since they all finish at the same time in the GPS system [Figure 4.35(*b*)], the tie-breaking rule of giving highest priority to the session with the smallest number yields $p_{2,1}$ as the next packet for service. In contrast, if a WFQ server is used, rather than selecting the next packet from among the 10 packets that have started service in the GPS system, it would pick the packet among all 11 packets, which results in packet $p_{1,2}$. At time 3, $p_{1,2}$ becomes eligible and has the smallest finish time among all backlogged packets, thus it starts service next. The rest of the sample path for the WF^{2}Q system is shown in Figure 4.36 [19].

Therefore, even in the case when session 1 is sending back-to-back packets, its output from the WF^{2}Q system is rather smooth as opposed to the bursty output under a WFQ system. The following theorem summarizes some of the most important properties of WF^{2}Q [19].

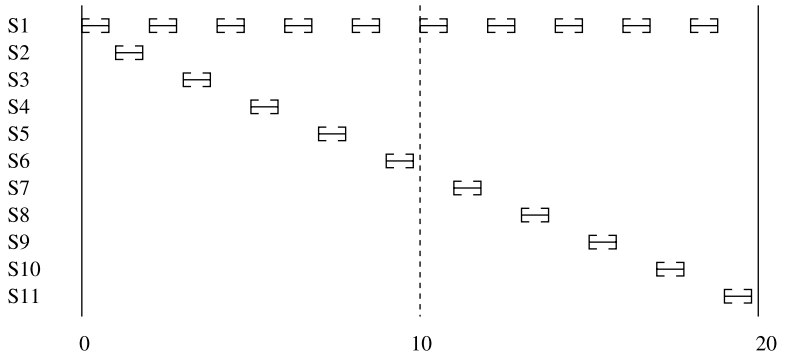


Figure 4.36 An example of WF^{2}Q service order (©1996 IEEE).

Theorem 4.1. Given a WF^{2}Q system and a corresponding GPS system, the following properties hold for any i, k, τ:

$$d_{i,k}^{\mathrm{WF^2Q}} - d_{i,k}^{\mathrm{GPS}} \leq \frac{L_{\max}}{r}; \tag{4.17}$$

$$W_i^{\mathrm{GPS}}(0,\tau) - W_i^{\mathrm{WF^2Q}}(0,\tau) \leq L_{\max}; \tag{4.18}$$

$$W_i^{\mathrm{WF^2Q}}(0,\tau) - W_i^{\mathrm{GPS}}(0,\tau) \leq \left(1 - \frac{r_i}{r}\right) L_i. \tag{4.19}$$

4.5.10 WF^{2}Q+

While WF^{2}Q provides the tightest delay bound and smallest WFI among all PFQ algorithms, it has the same worst-case time complexity of $O(N)$ as WFQ because they both need to compute the virtual time or the system virtual time $V(t)$ by tracing the fluid GPS system.

WF^{2}Q+ [22] and starting-potential fair queuing (SPFQ) [14] have been shown to have worst-case fairness properties similar to WF^{2}Q but is simpler to implement by introducing the following system virtual time function.

$$V(t+\tau) = \max\left\{V(t) + \tau, \min_{i \in \hat{B}(t)}\{S_i(t)\}\right\}, \tag{4.20}$$

where $\hat{B}(t)$ is the set of sessions that are backlogged in the system at time t, and $S_i(t)$ is the virtual start time of the HOL packet of backlogged session i. Let $W(t, t+\tau)$ be the total amount of service provided by the server or the bits that have been transmitted during a time interval $(t, t+\tau]$. In the special case of a fixed rate server, $\tau = W(t, t+\tau)/r$, where r is the link capacity. The time complexity is reduced to $O(\log N)$, attributed to the operations of searching for the minimum start time value among all N sessions.

To approximate the GPS, a PFQ algorithm, such as WF^{2}Q+ and SPFQ, maintains a system virtual time function $V(t)$, a virtual start time $S_i(t)$, and a virtual finish time (or time stamp) $F_i(t)$ for each queue i. $S_i(t)$ and $F_i(t)$ are updated on the arrival of the HOL packet for each queue. A packet departure occurs when its last bit is sent out, while an HOL packet arrival occurs in either of two cases: (1) a previously empty queue has an incoming

packet that immediately becomes the HOL; or (2) the packet next to the previous HOL packet in a non-empty queue immediately becomes the HOL after its predecessor departs. Obviously, a packet departure and a packet arrival in case II could happen at the same time. Therefore,

$$S_i(t) = \begin{cases} \max\{V(t), F_i(t^-)\} & \text{for packet arrival in case I} \\ F_i(t^-) & \text{for packet arrival in case II} \end{cases} \tag{4.21}$$

$$F_i(t) = S_i(t) + \frac{L_i^{\text{HOL}}}{r_i}, \tag{4.22}$$

where $F_i(t^-)$ is the finish time of queue i before the update, and L_i^{HOL} is the length of the HOL packet for queue i. The way of determining $V(t)$ is the major distinction among proposed PFQ algorithms [14, 15].

4.5.11 Comparison

WFQ or packet GPS (PGPS) is probably the first PFQ algorithm [12], in which the state of the GPS is precisely tracked. Although, in terms of the number of bits served for a session, the WFQ has proven that it will not fall behind the GPS by one maximum size packet; it can be far ahead of the GPS. In other words, it is not *worst-case fair*, as indicated with a large worst-case fair index (WFI) [19]. Motivated by this, an eligibility test was introduced in the WF^{2}Q [19] (also SPFQ [14]). In this test, when the next packet is chosen for service, it is selected from those 'eligible' packets whose start times are not greater than the system virtual time. It has been proven that the WF^{2}Q can provide almost identical service to that of the GPS, differing by no more than one maximum size packet. However, a serious limitation to the WF^{2}Q (and WFQ) is its computational complexity arising from the simulation of the GPS. A maximum of N events may be triggered in the simulation during the transmission of a packet. Thus, the time for completing a scheduling decision is $O(N)$.

Table 4.1 summarizes the latency, fairness measures, and time complexity of several scheduling algorithms [14]. Note that the $O(\log N)$ complexity of most of the sorted-priority algorithms arises from the complexity of priority queue operations, as explained in detail in the following section.

TABLE 4.1 Latency, Fairness, and Time Complexity of Several Scheduling Algorithms

Scheduler	Reference	Latency	WFI	Complexity
GPS	[12]	0[a]	0	—
WFQ	[12]	$(L_i/r_i) + (L_{\max}/r)$	$O(N)$	$O(N)$
SCFQ	[17]	$(L_i/r_i) + (L_{\max}/r)(N-1)$	—	$O(\log N)$
VC	[16]	$(L_i/r_i) + (L_{\max}/r)$	∞	$O(\log N)$
DRR	[11]	$(3F - 2\phi_i)/r$	$O(N)$	$O(1)$
WF^{2}Q	[19]	$(L_i/r_i) + (L_{\max}/r)$	$O(L_i/r_i)$	$O(N)$
WF^{2}Q+	[22]	$(L_i/r_i) + (L_{\max}/r)$	$O(L_i/r_i)$	$O(\log N)$

[a] In DRR, F is the frame size and ϕ_i is the weighting factor of session i in bandwidth allocation.

4.5.12 Priorities Sorting Using a Sequencer

This section describes an architecture to implement the packet schedulers. It uses a sorting device, called sequencer, to arrange the departure order of the cells/packets based on the time stamp values, as shown in Figure 4.37 [24]. The example used to illustrate the architecture has P priority levels, N inputs, and one output, but it can be generally applied to more outputs. The write/read controllers generate proper control signals for all other functional blocks. As shown in Figure 4.37, the cells are time-division multiplexed first and then written into the cell pool with idle addresses stored in a FIFO. A pair of a cell's priority field and its corresponding address, denoted as PA, is stored in the sequencer in such a way that higher priority pairs (e.g., smaller time stamp values) are always at the right of lower priority ones so they will be accessed sooner by the read controller. Once the pair has been accessed, the address is used to read out the corresponding cell in the cell pool.

The concept of implementing the Sequencer is very simple, as illustrated in Figure 4.38. Assume that the value of P_n is less than that of P_{n+1} and has a higher priority. When a new cell with priority P_n arrives, all pairs on the right of A_k, including the A_k itself, remain at their positions while others are shifted to the left. The vacant position is replaced with a pair of the new cell's priority field (P_n) and address (A_n).

When the cell pool is full (i.e., the idle-address FIFO is empty), the priority field at the left-most position of the sequencer (e.g., P_z) is compared to that of the newly arrived cell (P_n). If P_n is smaller than P_z, the pair of P_z and A_z is pushed away from the sequencer as the new pair P_nA_n is inserted in the sequencer. Meanwhile, the cell with address A_z in the pool is overwritten with the new cell. However, if $P_n \geq P_z$, the new cell is discarded instead.

Both the traffic shaper's architecture and the queue manager's architecture require a sequencer to sort the cells' departure times (DTs) or departure sequences (DSs) in descending order [25]. In [25], Chao and Uzun implemented the sequencer with a VLSI chip, which is essentially a 256-word sorting-memory chip. Figure 4.39 shows the building block of the

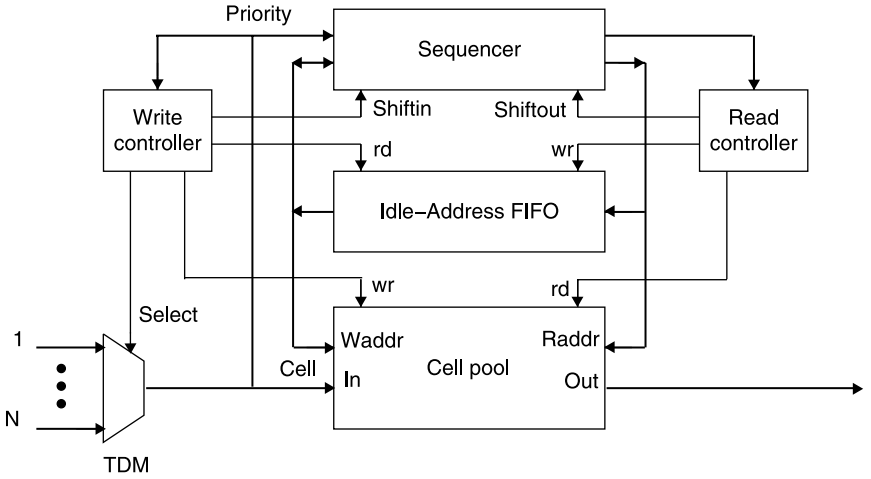


Figure 4.37 Sorting cell's departure with a sequencer. The sequencer determines the departure sequence for the cells that are stored in the cell pool (©1992 IEEE).

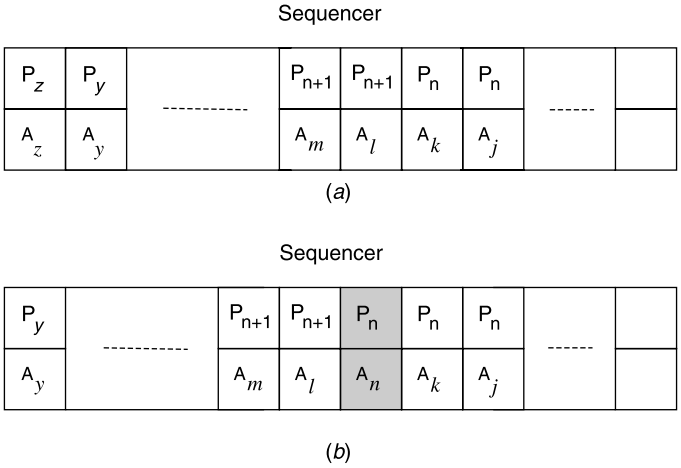


Figure 4.38 Operations of the sequencer. As a pair of new priority and address is inserted, all pairs with lower priority are pushed to the left.

chip, where the circuit in the dashed box is a module and is repeated 256 times in the chip. Each module has a 24-bit register, which stores the 14-bit DT/DS values and the 10-bit address. A single chip can accommodate a cell pool capacity of up to 256 cells and DT/DS values (or the number of priority levels in some applications) up to $2^{14} - 1$. This provides

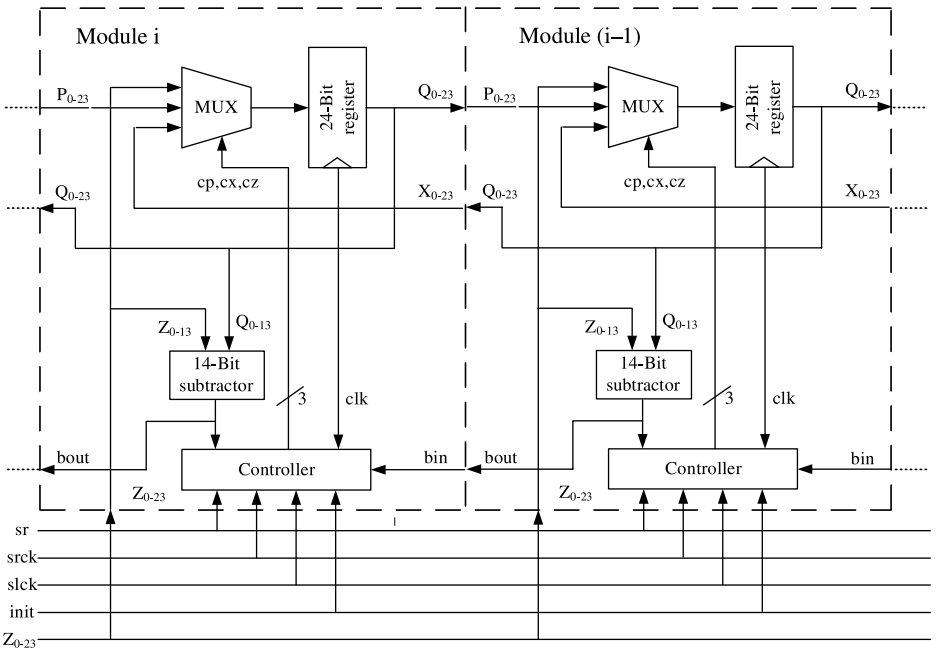


Figure 4.39 Block diagram of the sequencer chip (©1991 IEEE).

TABLE 4.2 Three Possible Actions Performed by the Controller

Cases	b_{out}	b_{in}	Action Performed by the Controller
(*a*) $X_{0-13} \le Z_{0-13} < Q_{0-13}$	1	0	Module i shifts its contents to the left, and $Q_{0-23} = Z_{0-23}$
(*b*) $Z_{0-13} < X_{0-13} \le Q_{0-13}$	1	1	Both Modules i and $(i-1)$ shift their contents left, and $Q_{0-23} = X_{0-23}$
(*c*) $X_{0-13} \le Q_{0-13} \le Z_{0-13}$	0	0	Retain the Q_{0-23}

the DT/DS ranging from 0 to 4095. By cascading multiple Sequencer chips in series or in parallel, a larger cell pool (e.g., a few thousand cells) or a larger DT/DS value can be supported.

Since every module is identical, let us examine the operations of an arbitrary module, say Module i. When a new pair of the DT/DS and the address field, denoted by Z_{0-23}, is to be inserted into the sequencer, it is first broadcast to every module. By comparing the DT/DS values (Q_{0-13}) of Module $(i-1)$ and Module i, and the new broadcast value (Z_{0-13}), the controller generates signals, *cp*, *cx*, *cz*, and *clk*, to shift the broadcast value (Z_{0-23}) into the 24-bit register in Module i, shift Module $(i-1)$s Q_{0-23} to the register, or retain the register's original value. Table 4.2 lists these three possible actions performed by the controller, where X_{0-13} is the Module $(i-1)$s Q_{0-13}. The b_{out} is the borrow-out of $(Z_{0-13} - Q_{0-13})$, and the b_{in} is the borrow-out of $(Z_{0-13} - X_{0-13})$. Since the smaller DT/DS is always on the right of the larger one, the case where $Q_{0-13} \le Z_{0-13} < X_{0-13}$, or $b_{out}b_{in} = 01$, will not happen.

When a cell with the smallest DT/DS value is to be transmitted, its corresponding address will be shifted out from the sequencer chip, and the data of all registers will be shifted one position to the right. For instance, the Q_{0-23} in Module i will be shifted to the register in Module $(i-1)$. Figure 4.40 shows the connection of signals between two cascaded sequencer chips.

Note that the P^l_{0-23} of the left sequencer chip is connected to all 1s; X^r_{0-23} and b^l_{in} of the right sequencer chip are all connected to 0s. The superscripts of l and r indicate, respectively, the module at the left-most and the right-most of the sequencer chip. At the initialization, all the registers inside the chip are loaded with the largest DT/DS values, that is, all 1s, so that new arrival cells with DT/DS values between 0 and $2^{14} - 1$ can be

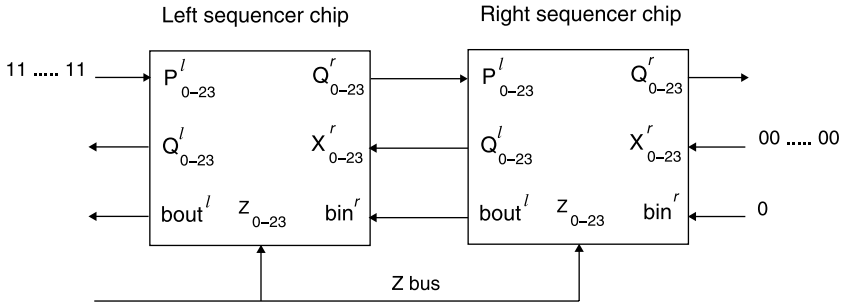


Figure 4.40 Interconnection signals of two cascaded sequencer chips (©1991 IEEE).

inserted into the sequencer. The initialization is done by asserting the *init* and *srck* signals and setting Z_{0-23} to 11 . . . 11.

4.6 BUFFER MANAGEMENT

The Internet is based on a connectionless end-to-end service with the advantages of flexibility and robustness. However, careful design is required to provide good service under heavy load situation. In fact, lack of attention to the dynamics of packet forwarding can result in severe service degradation or 'Internet meltdown.' This phenomenon was first observed during the early growth phase of the Internet in mid 1980s [26], and is technically called 'congestion collapse.'

The original fix for Internet meltdown was provided by Van Jacobson. Beginning in 1986, Jacobson developed the congestion avoidance mechanisms that are now implemented in TCP [27, 28]. These mechanisms operate in the hosts to make TCP connections to 'back off' during congestion. We say that TCP flows are 'responsive' to congestion signals (i.e., packets being dropped) from the network. It is primarily these TCP congestion avoidance algorithms that prevent today's Internet from congestion collapse.

However, that is not the end of the story. Considerable research has been done on Internet dynamics since 1988. It has become clear that the TCP congestion avoidance mechanisms [29], while necessary and powerful, are not sufficient to provide good service in all circumstances. Basically, there is a limit to how much control can be accomplished from the edges of the network. Beside packet scheduling algorithms, buffer management mechanisms are needed in the routers to complement the endpoint congestion avoidance mechanisms. Below we present some examples of the buffer management in the Internet.

4.6.1 Tail Drop

The traditional technique for managing router queue lengths is tail drop (TD), which sets a maximum length (in terms of packets) for each queue, accepts packets for the queue until the maximum length is reached (i.e., we say the queue is full), and then drops subsequent incoming packets until the queue decreases because a packet from the queue has been transmitted.

Tail drop is very simple but it has two important drawbacks. First, in some situations tail drop allows a single connection or a few flows to monopolize queue space, preventing other connections from getting room in the queue. This 'lock-out' phenomenon is often the result of synchronization or other timing effects.

Second, tail drop allows queues to maintain a full (or, almost full) status for long periods of time, since tail drop signals congestion (via a packet drop) only when the queue has become full. It is important to reduce the steady-state queue size for queue management, because even though TCP constrains a flow's window size, packets often arrive at routers in bursts [30]. If the queue is full or almost full, an arriving burst will cause multiple packets to be dropped. This can result in a global synchronization of flows throttling back, followed by a sustained period of lowered link utilization, reducing overall throughput.

The point of buffering in the network is to absorb data bursts and to transmit them during the ensuing bursts of silence [31]. This is essential to permit the transmission of bursty data. According to [31], queue limits should not reflect the steady state queues we

want maintained in the network; instead, they should reflect the size of bursts we need to absorb.

4.6.2 Drop on Full

Besides tail drop, there are two alternative drop on full disciplines: *random drop on full* or *drop front on full*. Under the random drop on full discipline, a router drops a randomly selected packet from the queue when the queue is full and a new packet arrives. While, under the drop front on full discipline [32], the router drops the packet at the front of the queue when the queue is full and a new packet arrives. This facilitates TCP with faster response to network congestion, resulting in a higher throughput. Both of these solve the lock-out problem, but neither solves the full-queues problem described above.

4.6.3 Random Early Detection (RED)

In the current Internet, dropped packets serve as a critical mechanism of congestion notification to end nodes. The solution to the full-queues problem is for routers to drop packets before a queue becomes full, so that end nodes can respond to congestion before buffers overflow. This proactive approach is called 'active queue management' according to [31]. Random Early Detection (RED) [33] drops arriving packets probabilistically. The probability of drop increases as the estimated average queue size grows. Thus, if the queue has been mostly empty in the 'recent past,' RED will not tend to drop packets unless the queue overflows. On the other hand, if the queue has recently been relatively full, indicating persistent congestion, newly arriving packets are more likely to be dropped [31, 33].

The RED algorithm itself consists of two main parts: estimation of the average queue size and the decision of whether or not to drop an incoming packet. The RED calculates the average queue size *avg*, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, a minimum threshold min_{th} and a maximum threshold max_{th}. When the average queue size is less than the minimum threshold, no packets are marked. When the average queue size is greater than the maximum threshold, every arriving packet is marked. If marked packets are in fact dropped, or if all source nodes are cooperative, this ensures that the average queue size does not significantly exceed the maximum threshold.

When the average queue size is between the minimum and the maximum threshold, each arriving packet is marked with probability p_a, where p_a is a function of the average queue size *avg*. Each time a packet is marked, the probability that a packet is marked from a particular connection is roughly proportional to that connection's share of the bandwidth at the router. The general RED algorithm is given in Figure 4.41.

The RED router has two separate algorithms. The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the router queue. The algorithm for calculating the packet-marking probability determines how frequently the router marks packets, given the current level of congestion. The goal is for the router to mark packets at fairly evenly spaced intervals to avoid biases and global synchronization, and to mark packets sufficiently frequently to control the average queue size.

The detailed algorithm for the RED is given in Figure 4.42. The router calculates *avg* at each packet arrival using

$$avg \leftarrow avg + w \cdot (q - avg)$$

```
for each packet arrival
calculate the average queue size avg
if min_th ≤ avg < max_th
    calculate probability p_a
    with probability p_a:
        mark the arriving packet
else if max_th ≤ avg
    mark the arriving packet
else
    admit the arriving packet in the buffer.
```

Figure 4.41 General algorithm for RED routers.

When the queue is empty (the idle period), it considers this period by estimating the number m of small packets that *could* have been transmitted during this idle period,

$$avg \leftarrow (1 - w)^m avg,$$

where m is equal to the queue idle time ($time - q\_time$) divided by the small packet transmission time (s), as shown in Figure 4.42. That is, after the idle period, the router computes the average queue size as if m packets had arrived to an empty queue during that period.

As avg varies from min_{th} to max_{th}, the packet-marking probability (or drop probability) p_b varies linearly from 0 to $P_{\max}$:

$$p_b \leftarrow P_{\max}(avg - min_{th})/(max_{th} - min_{th}),$$

as illustrated in Figure 4.43.

The final packet-marking probability p_a increases slowly as the count (the number of packets) increases since the last marked packet:

$$p_a \leftarrow p_b/(1 - count \cdot p_b),$$

which ensures that the router does not wait too long before marking a packet. The larger the count, the higher the marking probability. The router marks each packet that arrives at the router when the average queue size avg exceeds max_{th}.

One option for the RED router is to measure the queue in bytes rather than in packets. With this option, the average queue size accurately reflects the average delay at the router. When this option is used, the algorithm would be modified to ensure that the probability that a packet is marked is proportional to the packet size in bytes:

$$p_b \leftarrow P_{\max}(avg - min_{th})/(max_{th} - min_{th})$$

$$p_b \leftarrow p_b \cdot \text{PacketSize}/\text{MaximumPacketSize}$$

$$p_a \leftarrow p_b/(1 - \text{count} \cdot p_b)$$

In this case, a large FTP packet is more likely to be marked than is a small TELNET packet.

The queue weight w is determined by the size and duration of bursts in queue size that are allowed at the router. The minimum and maximum thresholds min_{th} and max_{th} are determined by the desired average queue size. The average queue size that makes the

```
Saved Variables:
    avg:            average queue size
    q_time:         start of the queue idle time
    count:          number of packets since last dropped packet
Fixed Parameters:
    w:              queue weight
    min_th:         minimum queue length threshold
    max_th:         maximum queue length threshold
    P_max:          maximum value for p_b
    s:              typical transmission time (of a small packet)
Others:
    q:              current queue size
    p_a:            current packet-marking probability
    time:           current time

Initialization:
    count ← −1
    avg ← 0
for each packet arrival
    calculate the new average queue size avg:
        if the queue is non-empty
            avg ← avg + w · (q − avg)
        else
            m = (time − q_time)/s
            avg ← (1 − w)^m avg
    if min_th ≤ avg < max_th
        increment count
        calculate probability p_a:
            p_b ← P_max(avg − min_th)/(max_th − min_th)
            p_a ← p_b/(1 − count · p_b)
        with probability p_a:
            mark the arriving packet
            count ← 0
    else if max_th ≤ avg
        mark the arriving packet
        count ← 0
    else count ← −1
when queue becomes empty
    q_time ← time
```

Figure 4.42 Detailed algorithm for RED routers.

desired tradeoffs (such as the tradeoff between maximizing throughput and minimizing delay) depends on network characteristics [33].

The RED mechanism relies on *random* dropping decisions when the buffer content exceeds a given threshold, so that heavy flows experience a larger number of dropped packets in case of congestion. Hence, the RED aims at penalizing flows in proportion to the amount of traffic they contribute, and prevents any of them from grabbing a disproportionate amount of resources.

The probability that a packet from a particular flow is dropped is roughly proportional to the flow's share of the link bandwidth. Thus, a flow that is utilizing a larger share of the link

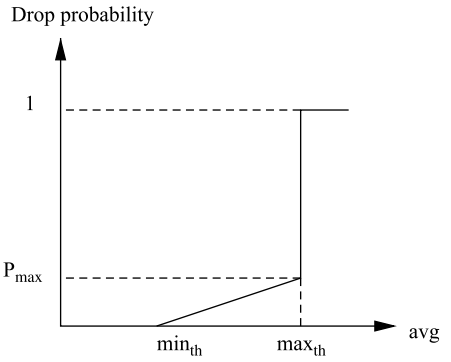


Figure 4.43 RED algorithm.

bandwidth is forced to reduce its rate rather quickly. One of the main advantages of RED is that it does not require per flow state to be maintained in the router. As a result, RED is relatively simple to implement and can be used in conjunction with the simple first-in-first-out (FIFO) scheduler to reduce congestion in the network. This can be of significance in the Internet backbone, where there may be hundreds of thousands of flows on a given link.

RED effectively controls the average queue size while still accommodating bursts of packets without loss. RED's use of randomness breaks up synchronized processes that lead to lock-out phenomena. There have been several implementations of RED in routers, and papers have been published reporting on experience with these implementations [34–36]. For example, weighted RED (WRED) [34] combines the capabilities of the RED algorithm with IP precedence (a three-bit field in the IP packet header). This combination provides for preferential traffic handling for high-priority packets. It can selectively discard lower-priority traffic when the router interface starts to get congested, and can provide differentiated performance characteristics for different classes of service. All available empirical evidence shows that the deployment of active queue management mechanisms in the Internet would have substantial performance benefits [31].

4.6.4 Differential Dropping: RIO

RIO stands for RED routers with `In/Out` bit [37, 38]. It is designed to support Assured Service in the Differentiated Services Internet [5]. The general approach of this service mechanism is to define a *service allocation profile* for each user, and to design a mechanism (e.g., RIO) in the router that favors traffic that is within those service allocation profiles. The basic idea is to monitor the traffic of each user as it enters the network, and tag packets as either `In` or `Out` of their service allocation profiles, then at each congested router, preferentially drop packets that are tagged as being `Out`. The idea of using `In/Out` is the same as used with the cell loss priority (CLP) bit in ATM networks.

Inside the network, at the routers, there is no separation of traffic from different users into different flows or queues. The packets of all users are aggregated into one queue, just as they are today. Different users can have very different profiles, which will result in different users having different quantities of `In` packets in the service queue. A router can treat these packets as a single common pool. This attribute of the scheme makes it very easy to implement.

RIO uses the same mechanism as RED, but is configured with two sets of parameters, one for `In` packets and the other for `Out` packets. By choosing the parameters for respective algorithms differently, RIO is able to discriminate against `Out` packets in times of congestion and preferentially drop `Out` packets.

In particular, upon each packet arrival at the router, the router checks whether the packet is tagged as `In` or `Out`. If it is an `In` packet, the router calculates avg_{in}, the average queue size for the `In` packets; if it is an `Out` packet, the router calculates avg_{total}, the average queue size for all (both `In` and `Out`) arriving packets. The probability of dropping an `In` packet depends on avg_{in}, and the probability of dropping an `Out` packet depends on avg_{total}.

As shown in Figure 4.44, there are three parameters for each of the twin algorithms. The three parameters, min_{in}, max_{in}, and $P_{max_{in}}$, define the normal operation $[0, min_{in})$, congestion avoidance $[min_{in}, max_{in})$, and congestion control $[max_{in}, \infty)$ phases for `In` packets. Similarly, min_{out}, max_{out}, and $P_{max_{out}}$ define the corresponding phases for `Out` packets.

The discrimination against `Out` packets in RIO is created by carefully choosing the parameters. First, by choosing $min_{out} < min_{in}$ the RIO router drops `Out` packets much earlier than it drops `In` packets. Second, in the congestion avoidance phase, it drops `Out` packets with a larger probability, by setting $P_{max_{out}} > P_{max_{in}}$. Third, it goes into congestion control phase for the `Out` packets much earlier than for the `In` packets, by choosing $max_{out} \ll max_{in}$. In essence, RIO drops `Out` packets first when it detects incipient congestion, and drops all `Out` packets if the congestion persists. Only when the router is flooded with `In` packets, as a last resort, it drops `In` packets in the hope of controlling congestion. In a well-provisioned network, this should never happen. If it does, it is a clear indication that the network is underprovisioned.

Figure 4.45 shows the RIO algorithm. By using *avg_total* to determine the probability of dropping an `Out` packet, routers can maintain short queue length and high throughput no matter what kind of traffic mix it has. The `Out` packets represent opportunistic traffic, and there is no valid indication of what amount of `Out` packets is proper. Simply using the average `Out` packet queue size to control the dropping of `Out` packets would not cover the case where the total queue is growing due to arriving `In` packets.

4.6.5 Fair Random Early Detection (FRED)

As pointed out in [39], RED does not always ensure all flows a fair share of bandwidth. In fact, RED is unfair to low speed TCP flows. This is because RED randomly drops

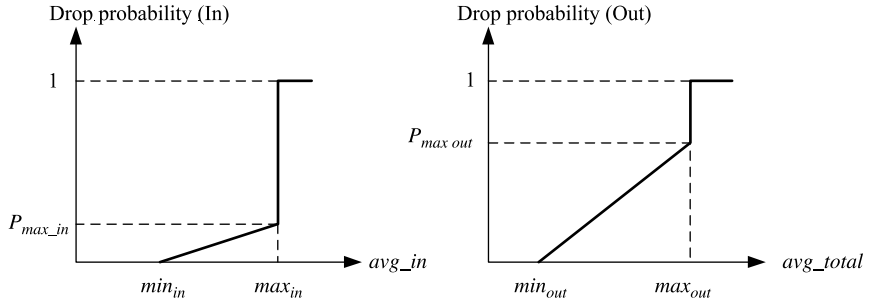


Figure 4.44 Twin RED algorithms in RIO.

```
For each packet arrival
  if it is an In packet
    calculate the average In queue size avg_in;
  else calculate the average queue size avg_total;

If it is an In packet
  if min_in ≤ avg_in < max_in
    calculate probability P_in;
    with probability P_in drop this packet;
  else if max_in ≤ avg_in
    drop this packet.

If it is an Out packet
  if min_out ≤ avg_total < max_out
    calculate probability P_out;
    with probability P_out drop this packet;
  else if max_out ≤ avg_total
    drop this packet.
```

Figure 4.45 RIO algorithm.

packets when the maximum threshold is crossed and it is possible that one of these packets belongs to a flow that is currently using less than its fair share of bandwidth. Since TCP reacts rather strongly to packet loss, the lost packet will force further reduction in the congestion window resulting in an even lower rate. The fair random early detection (FRED) mechanism, presented in [39] as a modification to RED, intends to reduce some of its unfairness. Basically, FRED generates selective feedback to a filtered set of connections that have a large number of packets queued.

In brief, FRED acts just like RED, but with the following additions. FRED introduces the parameters $\min_q$ and $\max_q$, which are the minimum and maximum number of packets each flow should be allowed to buffer. FRED introduces the global variable *avgcq*, an estimate of the average per-flow buffer count; flows with fewer than *avgcq* packets queued are favored over flows with more. FRED maintains a count of buffered packets *qlen* for each flow that currently has any packets buffered. FRED maintains a variable *strike* for each flow, which counts the number of times the flow has failed to respond to congestion notification; FRED penalizes flows with high *strike* values.

FRED allows each connection to buffer $\min_q$ packets without loss. All additional packets are subject to RED's random drop. An incoming packet is always accepted if the connection has fewer than $\min_q$ packets buffered and the average buffer size is less than max_{th}. Normally, a TCP connection sends no more than three packets back-to-back: two because of delayed ACK, and one more due to a window increase. Therefore, $\min_q$ is set to two to four packets.

When the number of active connections is small ($N \ll (\min_{th}/\min_q)$), FRED allows each connection to buffer min_q number of packets without dropping. It also dynamically raises min_q to the average per-connection queue length (*avgcq*). For simplicity, it calculates this value by dividing the average queue length (*avg*) by the current number of active connections. A connection is active when it has packets buffered, and is inactive otherwise.

FRED never lets a flow buffer more than $\max_q$ packets, and counts the number of times each flow tries to exceed $\max_q$ in the per-flow *strike* variable. Flows with high *strike* values are not allowed to queue more than *avgcq* packets; that is, they are not allowed to use more

packets than the average flow. This allows adaptive flows to send bursts of packets, but prevents nonadaptive flaws from consistently monopolizing the buffer space.

The original RED estimates the average queue length at each packet arrival. In FRED, the averaging is done at arrival and departure. Therefore, the sampling frequency is the maximum of the input and output rate, which helps reflect the queue variation accurately. In addition, FRED does not modify the average if the incoming packet is dropped unless the instantaneous queue length is zero. Without this change, the same queue length could be sampled multiple times when the input rate is substantially higher than the output link rate. This change also prevents an abusive user from defeating the purpose of the low pass filter, even if all his/her packets are dropped. Interested readers are referred to [39] for further details of the FRED algorithm.

4.6.6 Stabilized Random Early Detection (SRED)

Similar to RED, stabilized RED (SRED) [40] preemptively discards packets with a load-dependent probability when a buffer in a router seems congested. SRED has an additional feature, that, over a wide range of load levels, helps it stabilize its buffer occupation at a level independent of the number of active connections. SRED does this by estimating the number of active connections or flows. This estimate is obtained without collecting or analyzing state information on individual flows, as FRED does [39].

The main idea is to compare, whenever a packet arrives at some buffer, the arriving packet with a randomly chosen packet that recently preceded it into the buffer. When the two packets are of the same flow, we declare a *hit*. The sequence of hits is used in two ways, and with two different objectives in mind:

- To estimate the number of active flows.
- To find candidates for misbehaving flow.

The definition of *hit* can be flexible. The strongest plausible requirement is to declare a *hit* only when the two packets indeed are of the same flow: same destination and source addresses, same destination and source port numbers, and same protocol identifiers. Alternatively, one could use a more lax definition of *hit*, for example only the same source address. We may also choose not to check for a hit for every arriving packet. Instead, we can test for hits for only a random or deterministic subsequence of arriving packets. Also, we can compare the arriving packet with not one, but with some $K > 1$ randomly chosen packets from the recent past. That would give information of the type 'J out of K' hits, which can be used to more accurately estimate the number of flows.

Rather than maintaining per-flow state, a small cache is used to store a list of M recently seen flows, with the following extra information for each flow in the list: a *Count* and a *time stamp*. The list is called *zombie list* according to [40] and the flows in the list *zombies*.

The zombie list starts out empty. As packets arrive, as long as the list is not full, for every arriving packet, the packet flow identifier (source address, destination address, etc.) is added to the list, the *Count* of that zombie is set to zero, and its time stamp is set to the arrival time of the packet.

Once the zombie list is full, it works as follows: Whenever a packet arrives, it is compared with a randomly chosen zombie in the zombie list.

Hit. If the arriving packet's flow matches the zombie we declare a *hit*. In that case, the *Count* of the zombie is increased by one, and the time stamp is reset to the arrival time of the packet in the buffer.

No hit. If the two are not of the same flow, we declare a *no hit*. In that case, with probability p the flow identifier of the packet is overwritten over the zombie chosen for comparison. The *Count* of the zombie is set to zero, and the time stamp is set to the arrival time at the buffer. With probability $1 - p$ there is no change to the zombie list.

Irrespective of whether there was a hit or not, the packet may be dropped if the buffer occupancy is such that the system is in random drop mode. The drop probability may depend on whether there was a hit or not.

Define $P(t)$ to be an estimate for the hit frequency around the time of the arrival of the tth packet at the buffer. For the tth packet, let

$$h(t) = \begin{cases} 0 & \text{if no hit,} \\ 1 & \text{if hit,} \end{cases} \tag{4.23}$$

and let

$$P(t) = (1 - \alpha)P(t - 1) + \alpha h(t), \tag{4.24}$$

with $0 < \alpha < 1$. It has been shown in [40] that $P(t)^{-1}$ is a good estimate for the effective number of active flows in the time shortly before the arrival of packet t.

To reduce comparison overhead, it is allowable to update $P(t)$ not after every packet, but say after every L packets or at predetermined epochs. If H hits are got out of L packets, a possible update rule is

$$P(\text{new}) = (1 - L\alpha)P(\text{old}) + \alpha H. \tag{4.25}$$

As long as $0 \leq L\alpha \ll 1$, this has practically the same effect as updating after every packet [40].

Let us denote the packet drop probability function by p_z. According to [40], a function $q(x)$ is defined as follows:

$$q(x) = \begin{cases} p_{\max} & \text{if } \dfrac{B}{3} \leq x < B, \\ \dfrac{p_{\max}}{4} & \text{if } \dfrac{B}{6} \leq x < \dfrac{B}{3}, \\ 0 & \text{if } 0 \leq x < \dfrac{B}{6}, \end{cases} \tag{4.26}$$

where B is the buffer size, x is the backlog, and $p_{\max}$ is a parameter ranging between 0 and 1. The range of interest for $p_{\max}$ is $(0.09, 0.15)$ according to [40]. Higher values for $p_{\max}$ merely serve to drive too many TCP flows into time-out, while much lower values allow relatively large congestion windows.

When packet t arrives at the buffer, SRED first updates $P(t)$ from Eq. (4.24). If at the arrival instant the buffer contains x bytes, SRED drops the packet with probability p_z which equals

$$p_z = q(x) \times \min\left\{1, \frac{1}{[\beta P(t)]^2}\right\} \times \left[1 + \frac{h(t)}{P(t)}\right], \tag{4.27}$$

where β is a parameter with the suggested value 256 according to [40].

Note that, unlike RED, SRED does not compute average queue length (this operation can easily be added if needed). In [40], it has been shown that using an averaged buffer occupation does not improve performance. The motivation for choosing (4.26) and (4.27) is multifold. First, the buffer occupancy can vary considerably because of widely varying round-trip times, flows using different maximum segment sizes (MSSs), and transients caused by new flows before they reach the equilibrium of TCP flow and congestion control window. By making the drop probability depend on the buffer occupancy, which is the role of q in (4.27), SRED ensures that the drop probability increases when the buffer occupancy increases, even when the estimate $P(t)$ remains the same.

The ratio 4 in Eq. (4.26) is chosen such that TCP connections reach the new equilibrium after a single packet loss. When $0 \leq P(t) < 1/\beta$, SRED uses $p_z = q$ according to Eq. (4.27). This is for two reasons. First, if the drop probability becomes too large, TCP flows spend much or most of their time in time-out. So further increasing p_z is not sensible. Second, when $P(t)$ becomes small (when hits are rare), estimating $P(t)$ becomes unreliable.

SRED uses hits directly in the dropping probabilities, as indicated by the last term $[1 + (h(t)/P(t))]$ in (4.27). This is based on the idea that misbehaving flows are likely to generate more hits. There are two reasons. First, misbehaving flows by definition have more packet arrivals than other flows and so trigger more comparisons. Second, they are more likely to be present in the zombie list. This increases the drop probability for overactive flows and can also reduce TCP's bias in favor of flows with short RTTs. Interested readers are referred to [40] for further details on the SRED algorithm.

4.6.7 Longest Queue Drop (LQD)

Motivated by the fact that if connections are given equal weights, then connections that use the link more (get a higher share of the bandwidth unused by other connections) tend to have longer queues, longest queue drop (LQD) was proposed in [41] for buffer management. Biasing the packet drops such that connections with longer queues have higher drop rates should make the bandwidth sharing more fair. In addition, the LQD policy offers some flow isolation and protection since if one connection misbehaves consistently, only this connection experiences an increased loss rate.

The LQD requires searching through the backlogged queues to determine which is the longest queue. Ref. [41] proposes a variant of LQD that is particularly easy to implement, approximated longest queue drop (ALQD). A register holds the length and identity of the longest queue as determined at the previous queuing operation (queue, dequeue, drop). On every queuing event (including enqueuing and dequeuing), the current queue length is compared with the longest queue identified in the register. If it is the same queue, the queue length in the register is adjusted. If the current queue is longer, its identity and length are now stored in the register. A similar scheme called Quasi-Pushout is proposed in [42]. Below we briefly introduce the ALQD algorithm and present an implementation architecture.

```
for i = 0 to N − 1
/* N is the number of queues. Each queue may correspond to each port of a switch */
{
   if (queue i has an arriving packet) {
      if (data memory full) {
         push out HOL packet of current Qmax;
         decrement length of current Qmax by length of the dropped packet;
      }
      store cell-by-cell the incoming packet;
      increment length of queue i by length of the incoming packet;
      if (current Qmax is shorter than queue i)
         assign queue i as the new Qmax;
   }
   if (queue j has a departing packet) {
      send out the packet;
      decrement its queue length by length of the outgoing packet;
   }
   if (current Qmax is shorter than queue j)
      assign queue j as the new Qmax;
}
```

Figure 4.46 ALQD scheme.

Instead of sorting out the real longest queue among N flow queues, the ALQD [41] tracks the quasi-longest queue ($Q_{\max}$) by using three comparisons only. One is on the arrival of a packet: the queue length of the destined queue is increased and compared with that of the $Q_{\max}$. The other two are on the departure or dropping of a packet: the queue length of the selected queue is decreased and compared with that of the $Q_{\max}$. If the new length of the destined/selected queue is greater than that of the $Q_{\max}$, the flow identifier (FID) and length of the $Q_{\max}$ are replaced with that of the particular queue. That is, the destined/selected queue becomes the new $Q_{\max}$.

The ALQD drops the HOL packet of the currently longest queue when the data memory is full, because dropping from the front can trigger TCP's fast retransmit/recovery feature faster and, hence, increase throughput [41]. Figure 4.46 shows the ALQD algorithm.

Since packets are of variable length, they are divided into a number of fixed-length segments (or cells) to fully use the data memory in routers. The above algorithm actually checks on each packet arrival whether the data memory is full. Therefore, dropping a packet will guarantee sufficient memory space for the incoming cell. It is not necessary to double check the memory status after dropping a packet. A packet arrival event is defined as when its last cell has arrived at the system. Similarly, a packet departure event is when its last cell has left the system. The queue length updates are performed on packet arrival/departure/dropping events.

ALQD requires only $O(1)$ complexity in time and space. However, its state does not reflect exactly the state of the system. So optimal behavior at all times cannot be ensured especially when scheduling weights vary over a very wide range. A scenario could be constructed where ALQD cannot free enough memory and some incoming packets would have to be dropped, thereby temporarily breaching the strict flow isolation property of LQD. However, the degradation is such that it makes the complexity-performance tradeoff worthwhile [41].

REFERENCES

[1] S. Shenker and J. Wroclawski, *Network element service specification template*, RFC 2216 (Informational), Sept. 1997. [Online]. Available at: http://www.ietf.org/rfc/rfc2216.txt

[2] "Data link provider interface (DLPI)," The Open Group, Jan. 2000.

[3] H. J. Lee, M. S. Kim, W. K. Hong, and G. H. Lee, "QoS parameters to network performance metrics mapping for SLA monitoring," *KNOM Review*, vol. 5, no. 2 (Dec. 2002).

[4] Z. Wang, *Internet QoS: Architectures and Mechanisms for Quality of Service*, Morgan Kaufmann, San Francisco, California, 2001.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An architecture for differentiated service*, RFC 2475 (Informational), Dec. 1998, updated by RFC 3260. [Online]. Available at: http://www.ietf.org/rfc/rfc2475.txt

[6] Cisco. [Online]. Available at: http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/qos_c/qcpart4/qcpolts.htm

[7] S. Shenker and J. Wroclawski, *General characterization parameters for integrated service network elements*, RFC 2215 (Proposed Standard), Sept. 1997. [Online]. Available at: http://www.ietf.org/rfc/rfc2215.txt

[8] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed., Prentice-Hall, Englewood Cliff, NJ, 1992.

[9] N. McKeown, *Packet switch architectures class*. [Online]. Available at: http://klamath.stanford.edu/∼nickm

[10] P. Marbach, "Priority service and max-min fairness," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 733–746 (Oct. 2003).

[11] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, issue 3, pp. 375–385 (June 1996).

[12] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single node case," *IEEE/ACM Transactions on Networking*, vol. 1, issue 2, pp. 344–357 (June 1993).

[13] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," *IEEE/ACM Transactions on Networking*, vol. 2, issue 2, pp. 137–150 (Apr. 1994).

[14] D. Stiliadis, "Traffic scheduling in packet-switched networks: analysis, design, and implementation," Ph.D. Dissertation, University of California, Santa Cruz, June 1996.

[15] J. Bennett, D. C. Stephens, and H. Zhang, "High speed, scalable, and accurate implementation of fair queueing algorithms in ATM networks," in *Proc. IEEE ICNP'97*, Atlanta, Georgia, pp. 7–14 (1997).

[16] L. Zhang, "Virtual clock: a new traffic control algorithm for packet switching networks," in *Proc. ACM SIGCOMM*, Philadelphia, Pennsylvania, pp. 19–29 (Sept. 1990).

[17] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM*, Toronto, Canada, pp. 636–646 (June 1994).

[18] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," in *Proc. IEEE INFOCOM*, San Francisco, California, vol. 1, pp. 111–119 (Mar. 1996).

[19] J. C. Bennett and H. Zhang, "WF^2Q: worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, San Francisco, California, vol. 1, pp. 120–128 (Mar. 1996).

[20] D. Stiliadis and A. Varma, "A general methodology for design efficient traffic scheduling and shaping algortihms," in *Proc. IEEE INFOCOM'97*, Kobe, Japan, vol. 1, pp. 326–335 (Apr. 1997).

[21] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," *IEEE/ACM Transactions on Networking*, vol. 6, issue 2, pp. 175–185 (Apr. 1998).

[22] J. Bennett and H. Zhang, "Hierarchical packet fair queuing algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689 (Oct. 1997).

[23] S. Suri, G. Varghese, and G. Chandranmenon, "Leap forward virtual clock: a new fair queueing scheme with guaranteed delays and throughput fairness," in *Proc. IEEE INFOCOM*, Kobe, Japan, vol. 2, pp. 557–565 (Apr. 1997).

[24] H. Chao, "A novel architecture for queue management in the ATM network," *IEEE Journal on Selected Areas in Communications*, vol. 9, issue 7, pp. 1110–1118 (Sept. 1991).

[25] H. Chao and N. Uzun, "A VLSI sequencer chip for ATM traffic shaper and queue manager," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 11, pp. 1634–1643 (Nov. 1992).

[26] J. Nagle, *Congestion control in IP/TCP internetworks*, RFC 896, Jan. 1984. [Online]. Available at: http://www.ietf.org/rfc/rfc896.txt

[27] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM '88*, Stanford, California, vol. 18, no. 4, pp. 314–329 (Aug. 1988).

[28] R. Braden, *Requirements for Internet Hosts – Communication Layers*, RFC 1122 (Standard), Oct. 1989. [Online.] Available at: http://www.ietf.org/rfc/rfc1122.txt

[29] W. Stevens, *TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms*, RFC 2001 (Proposed Standard), Jan. 1997, obsoleted by RFC 2581. [Online]. Available at: http://www.ietf.org/rfc/rfc2001.txt

[30] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, vol. 2, issue 1, pp. 1–15 (Feb. 1994).

[31] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, *Recommendations on queue management and congestion avoidance in the Internet*, RFC 2309 (Informational), Apr. 1998. [Online]. Available at: http://www.ietf.org/rfc/rfc2309.txt

[32] T. V. Lakshman, A. Neidhardt, and T. Ott, "The drop from front strategy in TCP over ATM and its interworking with other control features," in *Proc. IEEE INFOCOM*, San Francisco, California, pp. 1242–1250 (Mar. 1996).

[33] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 2, no. 4, pp. 397–413 (Aug. 1993).

[34] *Quality of Service (QoS) Networking*, Cisco Systems, June 1999, white paper.

[35] C. Villamizar and C. Song, "High performance TCP in ANSNET," *Computer Communications Review*, vol. 24, issue 5, pp. 45–60 (Oct. 1994).

[36] M. Gaynor, *Proactive packet dropping methods for TCP gateways*, Oct. 1996. [Online]. Available at: http://www.eecs.harvard.edu/~gaynor/final.ps

[37] D. Clark and J. Wroclawski, "An approach to service allocation in the internet," July 1997, Internet Draft: draft-diff-svc-alloc-00.txt

[38] K. Nichols, V. Jacobson, and L. Zhang, "A two-bit differentiated services architecture for the internet," Nov. 1997, Internet Draft: draft-nichols-diff-svc-arch-00.txt

[39] D. Lin and R. Morris, "Dynamics of random early detection," in *Proc. SIGCOMM*, Cannes, France, pp. 127–137 (Sept. 1997).

[40] T. Ott, T. Lakshman, and L. Wong, "SRED: stabilized RED," in *Proc. IEEE INFOCOM*, New York, NY, vol. 3, pp. 1346–1355 (Mar. 1999).

[41] B. Suter, T. V. Lakshman, D. Stiliadis, and A. K. Choudhury, "Design considerations for supporting TCP with per-flow queueing," in *Proc. IEEE INFOCOM*, San Francisco, California, vol. 1, pp. 299–306 (Mar. 1998).

[42] Y. S. Lin and C. B. Shung, "Quasi-pushout cell discarding," *IEEE Communications Letters*, vol. 1, issue 5, pp. 146–148 (Sept. 1997).