CHAPTER 5

Queuing and Scheduling

A multi-service ATM network provides support for varieties of services with differing QoS requirements to be carried on the same switching node and links. Multiple services share the network resources (e.g., link bandwidth, buffer space etc.) and may try to access a resource simultaneously. Therefore, resource contention arises because of this sharing and a queuing structure is required to temporarily store cells of each service category. The point at which this resource contention occurs is generally referred to as a congestion or contention point. In such a network, the QoS objectives should be met for each connection and service class, independent of others. A scheduling mechanism is implemented at each queuing structure which appropriately selects the order in which cells should be served to meet the QoS objectives. A switching node can be implemented with one or more queuing structures. The term *scheduling* refers to the mechanism which determines what queue is given an opportunity to transmit a cell. A queuing structure and the corresponding scheduling algorithm attempt to achieve the following objectives:

Flexibility: To support different services, and easily evolve to support new services (see Chapter 2).

Scalability: The implementation should be simple to allow scalability to large number of connections and allow for cost-effective implementation.

Efficiency: In maximizing the network link utilization (i.e. maximize the throughput).

Guaranteed QoS: To provide low jitter and end-to-end delay bounds for real-time traffic. Allow implementation of simple CAC functions (see Chapter 4).

Isolation: To reduce interference between service classes and connections.

Fairness: To allow fast and fair re-distribution of bandwidth that is dynamically available. The fairness can be defined by a flexible policy.

This chapter focuses on the design of queuing structures and scheduling algorithms at a contention point within a multi-service ATM switch. Although scheduling a queue may also be tied to a queuing structure, these two are considered separately. After a brief introduction to ATM switch architectures, the queuing structures are discussed followed by the scheduling schemes.

ATM SWITCH ARCHITECTURES – A BRIEF INTRODUCTION

A general “folded” diagram of switch architecture is shown in Figure 5.1. An $N \times N$ switch consists of N input (or ingress) ports and N output (or egress) ports. Since all the ports are bi-directional, generally, the ingress and egress are co-located as shown in the diagram. Due to the fixed cell size (53 bytes), ATM switches (or ingress/egress interfaces) can operate in a time-synchronized manner with a slot time equals to the cell transmission time (called cell-slot) on a given link. The ATM switch concentrates the cells arriving on various incoming links at an ingress point.

The switch core (or fabric) is used to route cells from ingress points to appropriate egress points. Each ingress point may consist of many input links and thus a queuing structure may be needed to store cells before the switch core accepts them. Similarly, in a cell-slot, there can be many cells destined to a given egress from many ingress ports. Thus, a queuing structure may also be needed at an egress. The switch core may also use a queuing structure to store cells temporarily before they are switched to an egress.

There can also be speed mismatch at various queuing points. For example, the egress port link rate can be smaller than the rate at which the switch core sends (internal rate of the switch) cells to an egress. Thus, resource contention may arise either due to the speed mismatch or simultaneous arrivals of cells within a slot at various points in an ATM switch. Therefore, buffering becomes necessary at these contention points.

The amount of buffer needed depends upon the type and architecture of the switch itself. ATM switch fabrics can be classified as “blocking” or “non-blocking” in a broad sense. A blocking switch fabric is the one in which “internal” blocking occurs. Internal blocking can be defined as the situation, when the switching fabric cannot find an internal path to route cells arriving on different ingress ports which are destined to completely different egress ports. Here, it is assumed that cells are destined for a single output port (i.e., no broadcast or multi-cast operation). However, there could be multiple cells destined to a given output port from various input ports in a given cell switching slot. This effect can be referred to as *output conflict* or *resource conflict*. Cell buffering becomes necessary to alleviate this resource conflict.

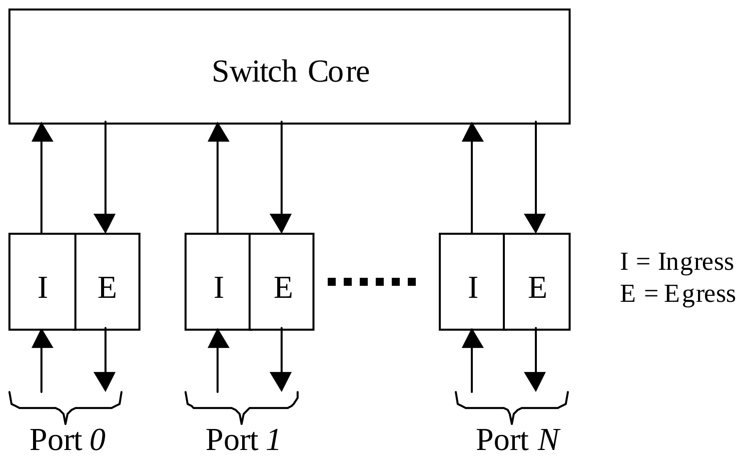

Figure 5-1 Folded Diagram of an ATM switch

The design of switch core is a topic by itself. In general, the switch fabrics are classified into three general categories [Tob90]:

Shared memory A shared memory switch consists of a single dual-ported memory, which is shared by all inputs and outputs. The memory is partitioned (or grouped) per output port. Cells arriving on input streams are multiplexed onto a single stream, which feeds the common memory. Retrieving cells from the output queues sequentially forms output stream of cells.

Shared medium A shared-medium switch uses a common high-speed medium (such as a parallel bus). Cells arriving on various input streams are multiplexed onto this medium. Each output port uses an address filter and a queue to retrieve cells destined for that port.

Space division A space-division switch uses many concurrent spatial paths from each input to a given output, each path operating at the same data rate. However, depending upon the type of switch fabric used and resources available, it may not be possible to setup paths to all required outputs simultaneously, even though paths are setup between completely different input and output pairs. This effect is called “internal blocking” of the switch, because of which buffering becomes necessary at points where path conflict occurs.

In all three types, the technology used in implementing the switch, places limitations on the size and speed of the ports. To build larger switches (e.g., more than 64 ports), many modules are interconnected in a multistage configuration. Since this chapter does not focus on the switching architectures, readers are referred to [Tob90] [AD89] for more information on this topic.

Since it is difficult to provide guaranteed QoS for traffic classes with blocking switches, non-blocking type switch fabrics are preferred for the ATM switches. Even with non-blocking fabrics, the performance depends upon how the buffering is provided at ingress/egress due to port contention that may arise when multiple cells from different ingress ports are destined to a given egress port. The following buffering scenarios are applicable for non-blocking switches [HK88]. Some of these scenarios are depicted in Figure 5.2.

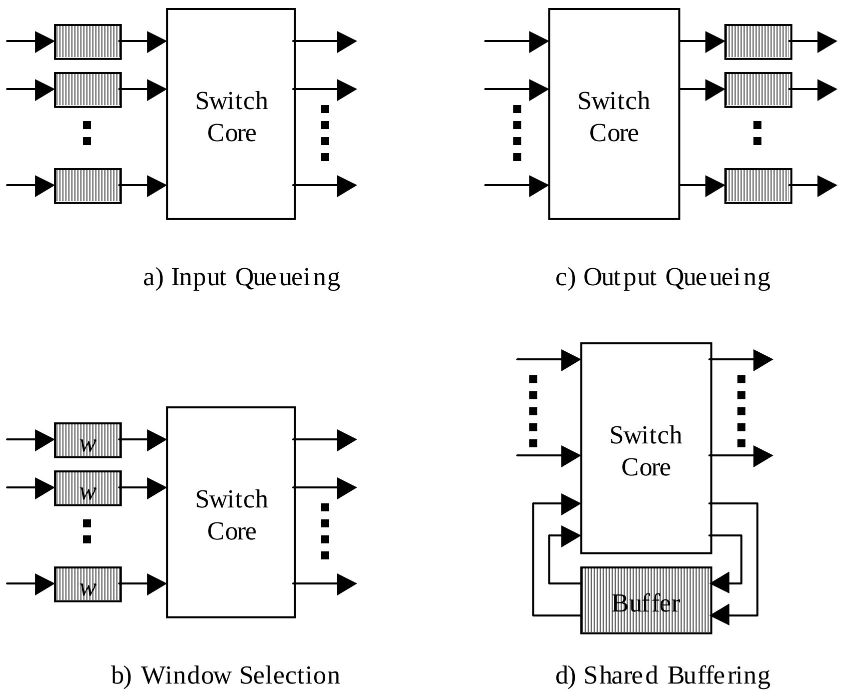

Figure 5-2 Queuing options for non-blocking switches

Depending upon the type of switch architecture used it is also possible to place buffers within the switching core itself. For example, buffering can be provided at each of the contention points in a multi-stage switch or at each cross-point in a crossbar switch. Such buffer placement can improve the switch performance.

Non-Blocking Switch without Buffers

For a non-blocking switch which does not have any buffers, when an output port contention occurs, only one cell is successfully transferred to the destination port in a given cell time and remaining contending cells are dropped from the switch. It is assumed that the speed of the switch is equal to that of input ports, i.e., the switch transfers at most one cell per slot from each of the N inputs. When $N \rightarrow \infty$, the probability of success *p(success)* that a cell wins the output port contention is given by *p(success)*$=(1-e^{-\rho})/\rho$, where ρ is the input traffic load, assuming a Bernoulli arrival process. That is, a cell arrives in a given slot with a probability ρ. The value $(1-e^{-\rho})$ is the throughput of the switch and attains a maximum value of 0.632 when $\rho=1$. This value is attained at the expense of cell loss, that is, 36.8% of incoming cells are dropped. Such high loss cannot be tolerated in high-speed ATM switching. Therefore, the use of buffers is necessary to limit cell loss.

Non-Blocking Switch with FIFO Input Buffers

When buffers are used at the input ports of a switch (Figure 5.2a), cells that did not win the output port contention as well as any new cells that arrived are stored in a buffer. When the switch is operating at the same speed as the ports (assuming input and output ports have the same speed), buffering is only needed at the input ports. The switch transfers at most one cell from each input port in a given slot. Buffers at output ports are not necessary in this case. When a First-In-First-Out (FIFO) [KLE75] service discipline is assumed for the buffers, the maximum switch throughput for $N\rightarrow\infty$ is shown to be 0.586 [HK88] when the inputs are saturated, i.e., each input queue has always a cell to transmit. A FIFO discipline services the cells in the order of their arrival. The performance of this type of switch is summarized in Table 5-1. The maximum throughput of a non-blocking switch with FIFO input buffer is lower than that of the switch, which does not have any buffers because of the effect of Head-Of-Line (HOL) blocking. When a cell at the head of the line in a buffer is blocked due to output port contention, remaining cells in the same input buffer are also blocked due to FIFO service discipline. Even though there could be some cells destined to other ports, they cannot be routed by the switch. A number of techniques are possible to improve this throughput. One approach is to increase the switching capacity (also referred to as *speed-up* of the switch). Another approach is to use a non-FIFO service discipline. That is, when a cell is blocked due to HOL, any cells behind not destined for the same output port can be routed instead. This enhancement reduces HOL blocking and improves performance. When $N\rightarrow\infty$, Hui [JH87] obtained an expression for the upper bound on the cell loss probability at the input buffers, assuming a Bernoulli arrival process as:

$$P_{loss} < \frac{p(2-p)}{2(1-p)}\left[\frac{p^2}{2(1-p)^2}\right]^B \qquad (5.1)$$

Here, p is the probability that a cell arrives in a given slot and B is the input buffer size**.** The selection policy for contending cells also has impact on the performance of the switch. A random selection policy is the most common. That is, one input port is randomly chosen to transmit a cell among all the contending inputs for a given output. This behavior may not always be desirable. A priority selection was proposed for the knock-out switch by Eng [Eng88] in which newly arriving cells are given lower priority over previously blocked cells. When output contention occurs, a cell from the high priority class (if any) is selected first and transferred to the destination. Cells that are lost due to contention are stored in the input buffers. If a cell loses due to contention twice, it is dropped from the input buffer. This scheme has lower buffer requirements than the random selection policy.

Non-Blocking Switches with non-FIFO Input Buffers

Servicing the cells in the input buffers using FIFO discipline leads to HOL blocking which limits the throughput of a switch to 58.6%, assuming no speed-up. The second alternative to improve the throughput of the switch is to implement a non-FIFO service of the input discipline, called the “window selection discipline” or “look-ahead contention resolution”.

Table 5-1 Switch Throughput for FIFO and Window Discipline [HK88].

Size	FIFO	Window Size (*w*)							
N		1	2	3	4	5	6	7	8
2	75.0 %	75 %	84%	89%	92%	93%	94%	95%	96%
4	65.5 %	66 %	76%	81%	85%	87%	89%	94%	92%
8	61.8 %	62 %	72%	78%	82%	85%	87%	88%	89%
16		60 %	71%	77%	81%	84%	86%	87%	88%

32		59%	70%	76%	80%	83%	85%	87%	88%
64		59%	70%	76%	80%	83%	85%	86%	88%
∞	58.6%								

In this servicing scheme (Figure 5.2b), when output contention occurs, the first "*w*" cells in each of the input buffers sequentially contend for the output ports again. The cells at the head of the buffer contend first for access to output. The contention process repeats up to "*w*" times at the beginning of each time slot, sequentially allowing *w* cells in an input buffer's window to contend for any remaining idle outputs, until the input is selected to transmit a cell. The variable *w* is called the "window size". Different names have been used for this service discipline such as: priority scheme, window policy, by-pass queuing, input smoothing etc. Table 5-1 shows the maximum throughput of a switch with various window sizes. The throughput that is indicated in the "FIFO" column is derived using an input saturation analysis. That is, it is assumed that inputs always have a packet to send. The throughput indicated for various window sizes is obtained from simulations. The window discipline is most effective when *N* is small and *w* is large. For example, when *N*=2 and *w*=8, a maximum throughput of 96% is possible. Parallel buffers at inputs can implement the window selection discipline at the expense of additional hardware.

Non-Blocking Switch with Input and Output Buffers (Switch Fabric Speed-Up)

Here the effect of switching fabric speed-up is considered. It is assumed that up to L $(1 \leq L \leq N)$ contending cells are accepted at an output port per cell slot. The variable L is referred to as the *speed-up* or *switching capacity* of the switch. Such a switch can also transfer up to a maximum of L cells from input ports to each output port. In this type of switch, the cells at the head of input buffers contend for output ports and only one cell out of the contending cells is sent to a given output port. This process is repeated L times per slot in FIFO basis. That is, for each repetition, the contention process is tried again with cells that lost the previous contention round(s) plus any new cells that moved to the head position. Therefore, buffers are needed at both input and output ports.

The cell loss probability is given by replacing *p* with *p/L* in equation (5.1). Reference [YO89] summarizes some of the past research done in this area. When $L=N$, cell queuing does not occur at inputs but occurs at the outputs. The switch attains a maximum throughput of 1 in this case. This case offers maximum performance in the sense that even if *N* cells arrive at inputs that are destined to the same output port, these *N* cells can be switched at once. However, it is difficult to implement such switches as *N* becomes very large. So the case of $1 \leq L << N$ is more practical. When $N \rightarrow \infty$, the switch throughput improvement is as follows [YO89]: 58.6%(for $L=1$), 88.5% (for $L=2$), 97.6% (for $L=3$), 99.6% (for $L=4$), 99.9%(for $L=5$).

Non-Blocking Switch with Output Buffers

With output queuing (Figure 5.2c), all queuing is done at the outputs of the switch, with a separate *B* cell FIFO provided for each output. It can be seen that the switch operates *N* times as fast as inputs and outputs, so that if *k* $(k=1,2,\ldots,N)$ cells arrive in a time slot on different inputs all addressed to the same output port, all *k* cells can be routed through the switch fabric to the proper output FIFO in the same time slot. However, only one cell can be removed from the output buffer during a time slot and remaining cells wait in the output buffer. Note that there is no HOL blocking introduced by the output queuing, but there can be cell loss due to congestion caused by simultaneous arrival of multiple cells from different inputs. The waiting time performance for output queuing is the best of all the methods discussed here. The memory available in the switch can be shared in many ways among the output ports (see Chapter 7). In complete partitioning, memory is divided among the N output ports while in complete sharing the buffers are pooled in and shared among all the output ports. Other sharing techniques are sharing with maximum queue depth, sharing with minimum allocation and sharing with maximum queue depth and minimum queue allocation. Each of these strategies present different trade-off in terms of the cell-loss performance with finite switch buffers.

Non-Blocking Switch with Shared Buffers

In the shared buffer switch (Figure 5.2d), there is no separate buffering at the ingress or egress. Newly arriving cells are immediately injected into the switch. When output contention occurs, the winning cell goes through the switch and other cells are stored in a shared buffer for re-entry. Since cells that lost contention are stored in a shared buffer, a queue is not formed at the inputs and HOL blocking does not occur. In addition to the newly arriving cells, cells in the shared buffer can also access the switch resulting in slightly better performance than that of window selection discipline. For a switch of size $N \times N$ and with a buffer of capacity $N \times B$, $N \times B$ additional ports are added internally to the switch for reentry of the packets from the shared buffer. Thus, an $N \times N$ switch should have a fabric size of $N(B+1) \times N(B+1)$. By approximating the steady state distribution of queue length to N fold convolution of an $M/D/1$ queue length [HK88] showed that lower bound on buffer size required to satisfy a given cell loss probability is given by the average queue length in a $M/D/1$ system. This lower bound is given by $\rho^2/2(1-\rho)$ where ρ is the input traffic load. It is also shown that switch with large shared buffer attains the maximum throughput close to 100%. The buffer size required in a shared buffer switch is much less than that of output buffer switch for the same cell loss probability, but the size of switching fabric is bigger.

QUEUING STRUCTURES FOR QOS GUARANTEES

The previous section dealt with how and where queuing becomes necessary in case of non-blocking switches. This section addresses how individual services are queued up for service at a resource contention point. ATM networks provide a platform for differentiated services with guaranteed QoS for each service. These services share the same transmission links as well as switch resources. Therefore, to individually guarantee different QoS for each service, a distinction needs to be maintained between the service categories with respect to the way they are serviced. In general, queuing structures can be organized as:

1. per-Group queuing
2. per-VC/VP queuing

With per-Group queuing, many connections share the same queue in a FIFO arrangement and with per-VC/VP queuing, the traffic of each VC or VP is queued independently. These two methods are described below. It is to be noted that at a congestion point, depending upon the implementation, a combination of both per-Group and per-VC queuing is also possible.

Per-Group queuing

In this queuing structure, the connections are categorized into *groups*. The traffic from each different group is queued up separately. The definition of a group can vary from one implementation to another. Typical groups consist of connections belonging to the same:

- service category
- service class
- conformance definition

Group as per Service Category

When connections are grouped as per service category, then traffic from each service category is separately queued in its own buffer (see Figure 5.3). Through this method, one can achieve per-service isolation, partition of the link capacity and hence separate QoS guarantees to each service. Ideally, the number of queues in such a structure should be equal to the number of service categories.

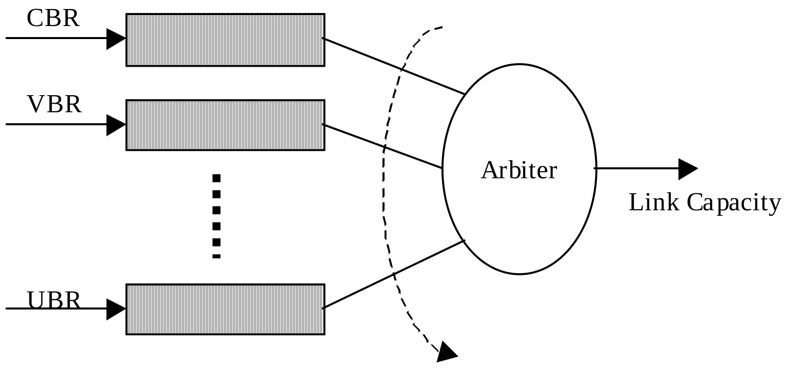

Figure 5-3 Per-Group Queuing: A group is defined as per service category

However, as can be seen latter in this chapter, an increase in the number of queues significantly increases the complexity of the scheduling function. The initial designs of ATM switches consisted of only a few queues at a contention point. Thus, many switches in the market may mix some of the service categories together. In general, when traffic from various service categories is mixed in the same queue, it is difficult to provide QoS guarantees for each service category independently as the traffic from each category interferes with the other. The key to provide efficient service guarantees using such a queuing structure is the appropriate dimensioning of buffers and setting of the discard thresholds such as CLP1, EPD, PPD, EFCI etc. (see Chapter 7), as well as selecting the correct scheduling disciplines. Through setting of some of these thresholds, one can achieve partial service isolation.

Group as per Service Class

ATM networks can provide one or more classes of service (see Chapter 3) within each service category. For example, since CBR service provides CLR and delay guarantees, an ATM network can provide multiple classes of CBR service, say with CTD=*250μSec* and CTD=*2.5mSec*. Or, it can provide a CLR of 10^{-10}, a premium service and 10^{-7}, a normal service for the same CTD.

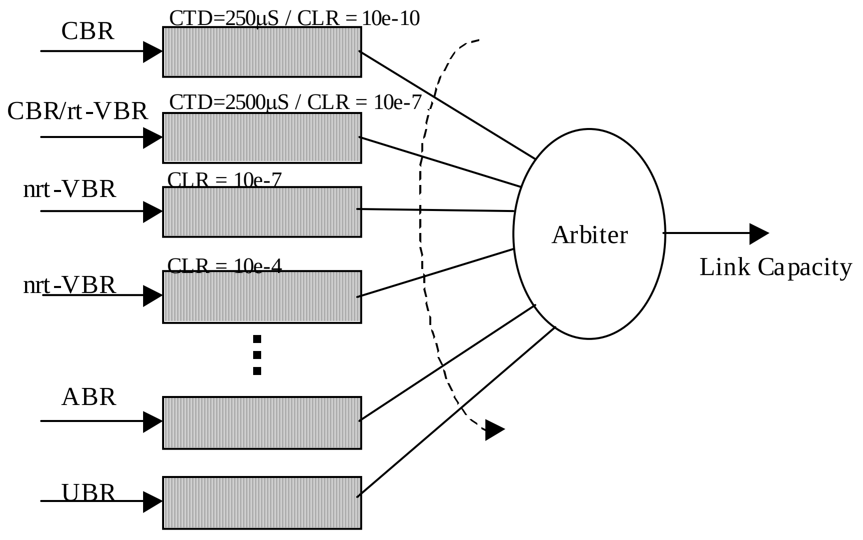

Figure 5-4 Per-Group Queuing: A group is defined as per service class

There are many ways to accomplish different CLR classes for a given service category. One method is to set cell discard thresholds on a given queue, whereby cells of low QoS connections are dropped when the queue depth reaches the discard threshold. Another method is to use a separate queue for each class. Proper connection admission control and bandwidth allocation procedures are required when a single queue is used with multiple discard thresholds. See Chapter 7 on congestion control for other methods of selective cell discard policies such as buffer push-out to provide multiple QoS subclasses within a service category.

Since cell delay is directly proportional to the buffer size for a given service rate, multiple delay classes can be obtained by using separate queues for each class in a given service category. It is also possible to obtain multiple delay classes by using admission thresholds on the buffers. In either case, proper buffer dimensioning, bandwidth allocation and priority servicing of the delay classes are necessary to achieve required delay guarantees. An example of per-Group queuing, where a group is defined as a service class is shown in Figure 5.4.

Group as per Conformance Definition

Another way of grouping the connections is by their conformance definition. For example [see Chapter 3], VBR services (both real-time and non real-time) are classified into VBR.1, VBR.2 and VBR.3. Figure 5.5 demonstrates a queuing structure, which uses the conformance definition as a group. The main advantage with this structure is easy handling of CLP transparency issue, which is discussed in the section labeled "Other Issues" at the end of the chapter.

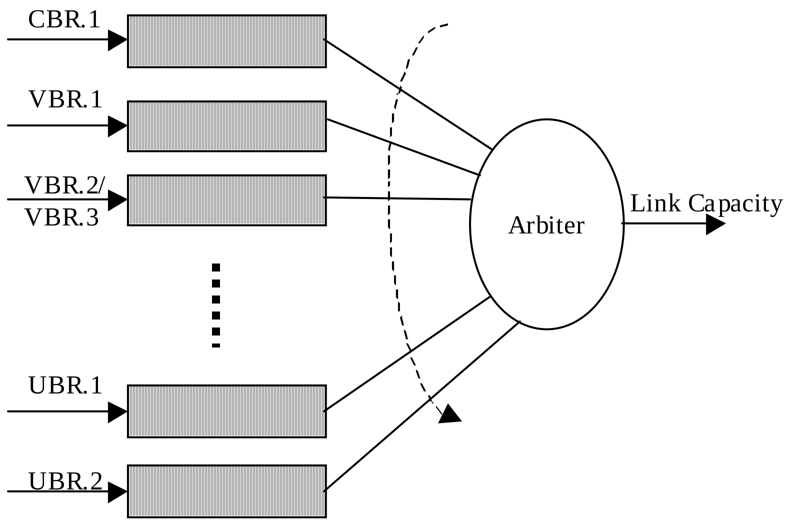

Figure 5-5 Per-Group Queuing: A group is defined as per conformance definition

The main problem with per-Group queuing is the difficulty of achieving per-connection isolation and QoS guarantees. This system offers isolation per group. It cannot prevent a traffic source from interfering with others within a given group. Another major disadvantage is that it cannot achieve per-VC QoS. That is, all connections traversing through a given group queue get the same QoS. For example, each ABR connection needs a different Minimum Cell Rate (MCR) guarantee. If all of the ABR connections are multiplexed together in one single queue, and if there are misbehaving ABR connections, it is only possible to guarantee aggregate MCR ($\sum MCR_i$) for the connections. Aggregate MCR guarantee does not necessarily mean the guarantee of MCR_i for each VC_i. One possible way to achieve per-VC isolation with such queuing structures is to use “per-VC accounting” [see Chapter 7]. That is, keep a record of the cell statistics and perform intelligent discards when congestion occurs.

Various cell scheduling mechanisms that can be implemented for the per-Group queuing structures are Priority Scheduling, Round Robin (RR), Weighted Round Robin (WRR) and Aggregate Traffic Shaping (TS) or combinations thereof. These schemes are discussed in detail latter in the scheduling mechanism section.

Per-VC / VP queuing

With per-VC/VP queuing, the traffic from each VC or VP is queued separately in its own buffer. Per-VC (or VP) isolation can be easily achieved using a per-VC (or VP) queuing structure, which is depicted in Figure 5.6. However, per-VC queuing is very complex, expensive to implement and does not scale well to a large number of connections. This architecture builds firewalls (or achieves flow isolation) between the traffic flows. Therefore a misbehaving connection cannot affect the QoS of other connections. Besides isolation, the major advantage of per-VC queuing is that the system can almost instantaneously achieve QoS guarantees for each connection individually. Taking the example of ABR, each ABR connection can measure its MCR over a very short period of time.

Some of the scheduling mechanisms, which apply to the per-VC/VP queuing structure, are the Priority Scheduling, Round Robin, and Weighted Round Robin, per-VC/VP traffic shaping or combinations thereof.

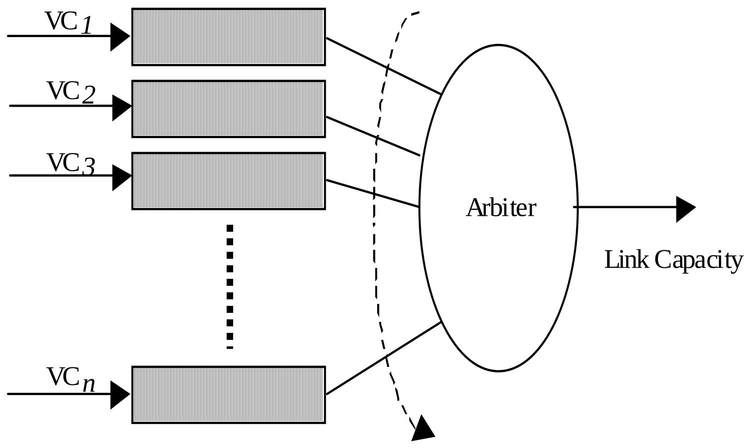

Figure 5-6 Per-VC Queuing Structure

Both per-Group and per-VC structures need buffer management in terms of how the physical memory is shared among the queues. That is, the available buffer can be either completely partitioned or shared among the queues. This particular topic is discussed in Chapter 7. For the purpose of this chapter, it is assumed that the queues are properly engineered or dimensioned.

SCHEDULING MECHANISMS

As described in the above section, a contention point can have a number of queues. An arbitration function is therefore needed to extract cells from these queues and transmit (or serve) them appropriately to meet the QoS objectives for each connection. An *arbiter* or *arbitration function* or *scheduler* is responsible for this scheduling and allocation of bandwidth to the queues. An arbiter implements a *scheduling algorithm* or *scheduling mechanism*. The queues at a contention point can be divided into logical sets, each of which is served by an arbiter. Depending upon how the queues and arbiters are organized, scheduling schemes can be divided into *flat (or single)-level scheduling* and *hierarchical scheduling*.

In a flat (or single)-level arbitration, a contention point uses a flat arbitration scheme with one single arbiter serving all the queues. The arbiter uses a single scheduling function amongst all the queues as shown in Figure 5.7.

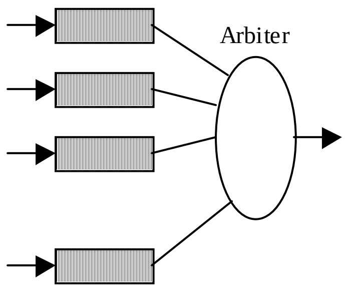

Figure 5.7. Flat Scheduling

However, in order to divide the bandwidth in a more flexible and accurate way, the arbitration can be performed in multiple levels or *hierarchically*, as exemplified in Figure 5.8.

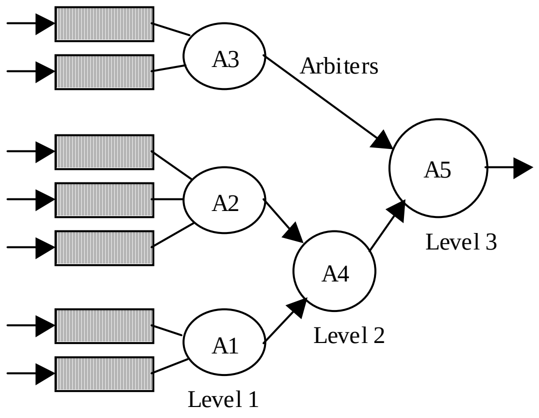

Figure 5.8. Hierarchical Scheduling

Hierarchical scheduling uses multiple arbitration functions. One arbiter is used to arbitrate between a set of queues and a few other arbitration functions are used in a hierarchical fashion to arbitrate between the arbiters. Each of the arbiters in the hierarchy can implement a different scheduling algorithm to achieve specific bandwidth partitioning and control.

Hierarchical scheduling is useful to divide the bandwidth on a link between different sets of queues (e.g., queues belonging to a given customer, to a given service etc.). With hierarchical scheduling, the bandwidth left over by a connection is redistributed first between the connections within the same set. A flat arbitration scheme cannot achieve this type of bandwidth redistribution. Using a hierarchical scheduler also simplifies the weight recalculation in the case of weighted scheduling (see section below on weighted fair queuing), since only weights within a set need to be recalculated at connection set-up or tear down.

The queue scheduling algorithms that an arbiter implements can be classified into four types:

1. Multiple-priority scheduling
2. Wok-Conserving Fair-share scheduling
3. Non Work-Conserving Fair-share scheduling
4. Traffic shaping.

Multiple-Priority Scheduling

A multiple-priority scheduler assigns a priority to each queue and serves them in the order of priority. A lower-priority queue is served only when there is no cell waiting for service in any higher priority queue. That is, cells belonging to higher priority are served first, even though there may be cells of low priority waiting in their respective queues which arrived before. Thus, the transmission opportunity of the low priority queue depends on the traffic load of the high priority queue that may vary dynamically. Therefore, it is difficult to support multiple services with guaranteed QoS. Since CBR and rt-VBR services have stringent delay requirements, they are in general assigned higher priority. nrt-VBR services have a low cell loss requirement but do not require any delay guarantees. So these services can be assigned a priority order lower than CBR and rt-VBR. The ABR and GFR services expect a long-term MCR guarantee with a reasonably low cell loss. Therefore ABR and GFR connections are assigned the next priority level. Since UBR service does not require any cell loss or delay guarantees, it can be served as the lowest priority

The priority scheduling can be applied to both per-Group queuing as well as per-VC queuing. In case of per-VC queuing, all the VC queues belonging to a particular service category can be viewed as at one priority level. If CBR service is assigned highest priority and UBR the lowest, then the CBR service gets the best possible delay and cell loss while the UBR gets the worst. If higher priority queue gets a traffic burst, it can temporarily starve the low priority queues. Thus, in a priority-scheduling scheme, it is difficult to provide service guarantees for lower priority traffic (e.g. ABR). When multiple sub-classes share the same queue, it is possible to order a queue on a non-FIFO basis, as long as the cells within a sub-class maintain a logical FIFO structure. That is, the cells from within a sub-class should be ordered as a FIFO. Though complex to implement, this kind of arrangement provides priorities within sub-classes when they share the same physical queue.

Multi-priority scheduling is very simple and efficient to arbitrate between a small number of queues. It can provide for example, a class of real-time traffic and a class of non-real-time traffic with guaranteed loss rate and a “best effort” class. Increasing the number of classes beyond this may be insufficient to provide fine-grained QoS commitments unless some level of fair access to the bandwidth is provided as described in the next section.

Fair-Share Scheduling

A robust queuing structure should protect the traffic flows from malicious (or misbehaving) sources (or users) without solely relying on the policing function. It should also be efficient under any network load variations. Misbehaving sources transmit data at a rate higher than their advertised rate or allocated rate. The variations in network load can cause reduced service rates at contention points, which has the equivalent effect of higher arrival rate which necessitates the use of rate-based service schemes.

An alternative approach to priority scheduling is fair-share scheduling, in which, each queue is guaranteed to get its share of link bandwidth according to its need. Different queues may have different bandwidth requirements and thus a weight can also be associated with a queue. The scheduler divides the bandwidth amongst each queue based on the weighted fair-share. Fair-share scheduling is a concept that introduces firewalls (or flow isolation) between various traffic flows in a queuing system and thus the interaction between the flows is minimized. Fair-share scheduling basically deals with how the leftover (or idle) bandwidth is distributed among the connections. Fair-share scheduling attempts to distribute the bandwidth among competing connections in a fair manner. It is very useful for ABR, GFR and UBR services where the bandwidth is assigned on the basis of availability. One can assign some weight for UBR traffic so that UBR connections get a share of link capacity. This flexibility would not be possible in multi-priority scheduling scheme.

Note that the fair-bandwidth allocation is possible only between the flows and not between the sub-flows, if the traffic is divided as such. For example, if each service category is considered as a flow then a fair-allocation is possible between service categories, but not between connections within a service category. Therefore, the definition of a flow depends on the queuing and scheduling architecture being considered.

The fair-share scheduling algorithms guarantee certain minimum rate between the flows. These “rate-based” mechanisms can be classified into two categories [ZK91]: *rate-allocation service* discipline and *rate-controlled* service discipline (see Figure 5.9).

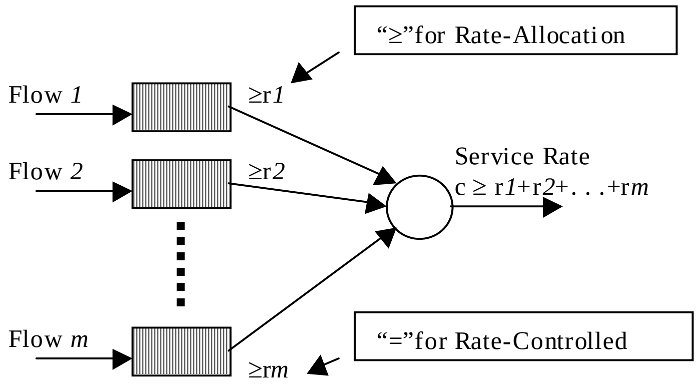

Figure 5-9 Rate-Controlled versus Rate-Allocation

In the rate-allocation service discipline, a queue may be served at a higher rate than the minimum service rate as long as the guarantees made to other services do not suffer. In the rate-controlled service discipline, the mechanism does not serve any queue at a rate higher than its assigned service rate under any circumstance. Figure 5.9 depicts the difference between the two. Each flow needs a *minimum bandwidth guarantee* of r_i in this example. The scheduler should try to achieve this minimum bandwidth guarantee for all traffic flows.

These rate-based schemes are also classified as *work conserving* or *non-work conserving.* A work conserving scheduler is never idle when there are cells to send in the queues. Since rate-controlled schedulers do not serve a queue with more than its allocated rate, these are classified as non-work conserving while the rate-allocation schedulers are work conserving. It can be expected that the control of delay jitter and buffer requirements for each of the traffic flows depends upon the type of scheduler, i.e., whether it is work conserving or not. Some of the rate-based schemes proposed in the literature are discussed below. Note that some of the schemes are developed in the context of data networks where the data unit is a packet (of variable length). Therefore, the names packet and cell are interchangeably used in the following discussion. Also, the terms: sessions, connections and VCs are used interchangeably.

An ideal fair-share scheduler is the one, which employs Processor Sharing (PS) among the traffic flows. The processor divides the link capacity equally among the contending connections. When connections are given different weights and the link capacity is divided not as equal share, but in proportion to the weight assigned, then the processor sharing is referred to as Generalized Processor Sharing (GPS). It is very difficult to implement the PS or GPS, as it assumes that the traffic is infinitely divisible and all connections with non-empty queues can be served simultaneously. In reality, a cell can only be served in its entirety (i.e., cannot be split). Therefore, fair queuing or the weighted fair queuing can be thought of as a practical implementation of the PS or GPS. There are many schemes proposed in the literature, which try to achieve this fair-share scheduling by various packet approximation algorithms. The fair-share schedulers can be implemented in two ways:

Method 1: Assign a *service-deadline* (or *time-stamp*, or *virtual finishing times,* or *finish number*) for each cell and serve them in the increased order of the deadline. If there is a tie between cells, the cells are re-ordered randomly. Assuming a unit link capacity, the service-deadline F_i^j assigned to the cells of each flow *j* at instant *i* has the following form:

$$\begin{aligned} F_i^j &= \max\{F_{i-1}^j, v_i^j\} + \left(1/r^j\right) \\ F_0^j &= 0 \end{aligned} \tag{5.2}$$

Here, r^j is the minimum guaranteed bandwidth for connection *j* and v_i^j is the virtual time for connection *j* at instant *i*. The main difference between various algorithms is the way the virtual time is computed.

Method 2: Serve the queues in a round-robin fashion using frames or cycles. In each cycle, all the queues are given a transmission opportunity to transmit a cell or packet. Queues with larger weights can be given multiple transmission opportunities in each cycle. If all the queues have the same weight, then it is a round robin (RR) system otherwise it is weighted round-robin (WRR). This behavior is similar to a TDM system, except that if a queue is given a transmission opportunity and if the queue does not have any cell to transmit, the scheme does not waste the bandwidth. Instead, the scheduler examines rest of the queues for any possible cell transmission. With *N* backlogged flows, a round robin discipline assigns a bandwidth of *Link Capacity/N* to each flow, while for each flow with weights w_i, WRR scheme assigns a bandwidth of $w_i.LinkCapacity/\left(\sum_{1\le i\le N} w_i\right)$.

There are two major aspects to these algorithms:

- Minimum bandwidth guarantees for the traffic flows;
- Bandwidth granularity.

Each of the schemes has different properties with respect to these aspects, which is discussed next.

Work-Conserving Fair-Share Scheduling Algorithms

A. Fair Queuing [DKS89]

Fair queuing algorithm proposed by Demers et al. [DKS89] emulates bit-by-bit round robin service discipline. It is assumed that packets from each traffic flow are separately queued. Each packet is assigned a service deadline, which is the finishing time, if the packets are served using a bit-by-bit round robin (BR) scheme, i.e., each queue is served one bit at a time in a round-robin fashion. That is, the service deadline is the time when a packet would have left under the BR servicing scheme. When packets are served in the order of finish times, it is shown that the system emulates the BR scheme. A finish number is assigned to each packet as:

$$F_i^j = \max\left\{F_{i-1}^j, R_i^j\right\} + P^j \qquad (5.3)$$

Here, P^j is the packet length, R_i^j is the number of rounds made in the round-robin service discipline.

B. Packet-based Generalized processor Sharing [PG93]

The packet-based generalized processor sharing (PGPS) concept is a generalized version of fair queuing using virtual finishing times and weights to each traffic flow. The weight allows each flow to get a weighted share of link bandwidth rather than just an equal share. The packets are stamped with a virtual finishing time as:

$$F_i^j = \max\left\{F_{i-1}^j, v_i^j\right\} + \left(P^j / \phi^j\right) \qquad (5.4)$$

Here, P^j is the packet length, ϕ^j is a weighting factor, v_i^j is a virtual time. The virtual time v_i^j during a busy period [t_1, t_2] is defined as:

$$\frac{\partial V(\tau)}{\partial \tau} = \frac{1}{\sum_{i \in B_k} \phi_i} \quad \forall t_1 \le \tau \le t_2; \quad V(t_1) = 0;$$

where B_k is the set of back logged connections. It is shown by Parekh [PG93] that, regardless of traffic arrival pattern, PGPS approximates to GPS within one packet transmission time. Delay bounds for sessions traversing through a set of GPS or PGPS servers is derived in [PG94]. These bounds are tighter than those derived using summation of worst-case delays at each node.

C. Self-clocked Fair Queuing [Gol94]

Both PGPS and WFQ algorithms developed the virtual finish times based on a hypothetical fluid-flow fair queuing system. The computational complexity in computing these virtual finish times can be very high to implement the schemes in real networks. Therefore, a Self-Clocked Fair Queuing (SCFQ) is proposed by Golestani [Gol94]. As per this scheme, the packets are stamped with a virtual finishing time as:

$$F_i^j = \max\left\{F_{i-1}^j, \hat{v}_i^j\right\} + \left(P^j / \phi^j\right) \qquad (5.5)$$

where, $\hat{v}_i^j$ is regarded as the system's virtual time and is defined as the service tag of the packet receiving at that time. That is, $\hat{v}_i^j$ is the virtual finish time of the last packet to leave (hence the name self-clocked) the queuing system during a busy period. Once the busy period is over and there are no more packets to serve in the queue $\hat{v}_i^j$ is reset to zero.

D. Virtual Spacing [Rob94]

This scheme is the same as SCFQ, independently developed and applied to the context of ATM with constant length packets. In this scheme, cells are stamped with a virtual finishing time as:

$$F_i^j = \max\left\{F_{i-1}^j, \hat{v}_i^j\right\} + \left(1/r_i\right) \qquad (5.6)$$

where, $\hat{v}_i^j$ cannot be greater than the time stamp of any cell already waiting when its value is updated. This result implies that back logged cells are spaced at an interval of ($1/r_i$).

E. Virtual Clock [Zha90]

Virtual Clock scheduling algorithm is an emulation of TDM (Time Division Multiplexing) service discipline. TDM is a slot-based system, which does not allow any statistical multiplexing and guarantees a fixed bandwidth for the traffic flow by assigning fixed slots to the traffic flows. If a flow does not have any data to send in the assigned slot, the slot is not re-used and the bandwidth wasted. Thus TDM is a non-work conserving service discipline. With Virtual clock statistical multiplexing is feasible and slots are utilized if there are any non-empty queues (i.e., work conserving). While a real-time clock is used in TDM system to define the service slots, a "virtual clock" is used to stamp the cells in this Virtual Clock scheduling. Each cell is assigned a service deadline (or virtual time), which is the time a cell would leave if it were using a TDM system. Then, cells are sent in the order of virtual times. The virtual time v_i^j assigned to a traffic flow *j* is the real-time itself, i.e., the time *t* at which the cell of the flow arrived. That is, the cells are marked with a time-stamp as:

$$F_i^j = \max\left\{F_{i-1}^j, t\right\} + \left(1/r^j\right) \qquad (5.7)$$

The delay bounds for both single server as well as network of servers using VC discipline is derived in [FP95].

E. Deficit Round Robin [SV96]

Deficit round robin is an extension to round-robin scheduler, which serves variable length packets. This scheme maintains a "deficit counter" (DC_i) for each traffic flow. The deficit counter is reset to zero whenever the queue falls empty. The deficit counter is initialized to the weight of connection (Q_i) when the queue becomes backlogged. The weight in this scheme is the number of bits a flow is allowed to send in a given round. Since each packet is of different length, a deficit can be built up for each traffic flow. The current deficit is set equal to the previous deficit if any less the number of bits sent in the current packet, i.e., $DC_i = DC_i + (Q_i\text{-}bytes\_sent_i)$. The flow can take advantage of the available deficit in its next round. That is, the amount of bandwidth usable by the flow is the sum of DC_i added to the weight of the connection Q_i.

F. Pulse Scheduling [MLF92]

Pulse scheduling generalizes the virtual clock method of Zhang [Zha90] to bursty sources. Recall that virtual clock uses only the average rate (r^j) of the traffic source in the computation of finish times. The pulse scheduling extends this concept by taking into account the burst properties of the connection. Let a connection *i* transmit at a peak rate λ_i for a duration $t1_i$ and off (nothing transmitted) by duration $t2_i$. Then, the pulse scheduling assigns finish times using the peak rate for $\lambda_i.t1_i$ packets. The next packet is assigned a virtual time of $t2_i$ and the process is repeated. Thus, the virtual time follows the burst properties of the connection and the pulse scheduling thus provides preferential treatment to the connections based on the advertised peak rate. This scheme is identical to the virtual clock when $t2_i$=0.

G. Delay Earliest Due Date [FV90]

The methods described above are only concerned with guaranteeing bandwidth to traffic flows. Delay guarantees are very essential for real-time services. A method for guaranteeing delays in packet-switched wide area networks is proposed by Ferrari et al. in [FV90], called Delay Earliest Due Date (Delay-EDD). The scheme provides two services, one with deterministic delay bounds and the other with statistical delay bounds. Deadlines are assigned to packets based on the service. The deadlines for packets belonging to deterministic delay bound service are “aligned” in such a way that, if the deadlines of the packets overlap, the deadlines are appropriately reduced so that the delay guarantees are met. When it is time to serve the queues, the service compares the deadline stamps of the head-of-line packets waiting at both deterministic delay and statistical delay service queues. The comparison is with respect to the end time (i.e., deadline of statistical queue packet) and beginning time (i.e., deadline – service time) of deterministic queue packet. If the latter is lower than the former, the determinist delay queue is served immediately. The deadlines assigned to each packet are of the form:

$$F_i^k = \max\left\{a_i^k + d_i^k, F_i^k + x_{\min}^k\right\} \qquad (5.8)$$

where, a_i^k is the arrival time of the packet, d_i^k is the delay bound assigned by the server to the connection, $x_{\min}^k$ is the minimum inter-arrival time for the connection.

H. Worst-case Fair Weighted Fair Queuing [BZ96]

All the fair queuing systems described so far are packet based and try to emulate the GPS system as closely as possible so that fair queuing is practical to implement. The main difference between the GPS and packet-based systems is that, in a fluid system multiple packets are served simultaneously while in packet-based systems only one packet can be served at a time. This effect causes the service provided by both systems to be different for a given connection. Bennet and Zhang [BZ96] show that it is exactly the case by considering the worst-case fairness properties. To minimize the difference between a packet system and the fluid system, [BZ96] proposed a packet policy called Worst-case Fair Weighted Fair Queuing (WF^{2}Q). In a WFQ system, the next packet chosen to serve is the one which would have completed service in the corresponding GPS system at time *t*. In WF^{2}Q system, the server considers the set of packets that would have started and possibly finished receiving service in the corresponding GPS system. By doing so, it is shown that WF^{2}Q provides almost identical service to GPS not differing by more than one maximum size packet. To reduce the complexity of implementation, [BZ97] extends this scheme to WF^{2}Q+ scheme using a lower complexity virtual time function.

I. Hierarchical Packet Fair Queuing [BZ97]

The fair queuing algorithms discussed above work at a single level and distribute bandwidth fairly among the single class of service. In a multi-service ATM network, there are many services supported and the excess unused bandwidth can be distributed as per the service hierarchy rather than the single level model. [BZ97] proposed a Hierarchical Packet Fair Queuing (H-PFQ) as an approximation to Hierarchical Generalized Processor Sharing (H-GPS), which is shown in Figure 5.10.

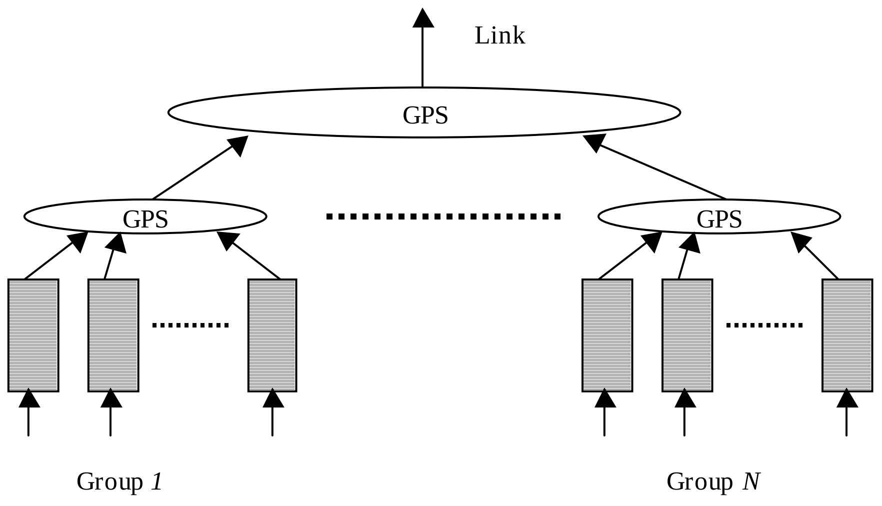

Figure 5.10 Hierarchical GPS

Here, individual physical groups are arranged into groups, where each group gets its own GPS server. There is another GPS server at the top level, which serves the individual logical groups. With an Hierarchical GPS server, each logical group gets its own fair share which is further re-distributed to the individual physical queues in that logical group. At the physical group level any of the fair queuing techniques discussed above can be used. At the logical group level, [BZ97] presented an implementation framework. The major difference between the logical group and physical group level is that, at the logical group the queue need not be a FIFO.

J. Weighted Round-Robin Scheduling

Unlike the schemes described above which are based on virtual finish times, the Weighted Round-Robin (WRR) schemes can be built with frames for fixed packet lengths. The simple case of WRR is the round robin. If there are *N* traffic flows, each separately queued, the RR server in each cycle visits each of the queues and serves a packet if any is waiting. Thus, the RR server shares the link bandwidth equally among the traffic flows. The frame (or cy-cle) length of the RR sever is *N* packet length slots. Instead of equal share, it is also possible to obtain a weighted share per traffic flow by assigning weights to each traffic flow and giving slots proportional to weight in each sever cycle. Then, frame length is much larger than *N* and depends upon the bandwidth granularity supported by the WRR server. For a frame length of *F* slots, the minimum bandwidth assigned for a connection is *1/F* of the link capacity. By assigning slots proportional to weight w_i in a frame, each flow gets a fair share of $w_i.LinkCapacity/\left(\sum_{1\le i\le N} w_i\right)$. To achieve finer bandwidth granularity, the frame length *F* can be made large which has implications on the delay. A better way to achieve finer granularity is to use Hierarchical Round-Robin (HRR), which is described latter in the chapter.

K. Rotating Priority Queues$^+$ (RPQ$^+$) Scheduling [WL97]

The RPQ$^+$ scheduler uses a set of prioritized FIFO queues. The order of the priorities is re-arranged periodically to increase the priority of waiting packets. By doing so, it is proven [WL97] that RPQ$^+$ can provide worst-case delay guarantees, superior to static priority queuing and approximates the optimal Delay-EDD scheduler. The efficiency of RPQ$^+$ increases with the priority re-labeling and approaches to that of Delay-EDD in the limit. The RPQ$^+$ employs *2P* ordered FIFO queues. These queues are labeled from highest to lowest priority with indexes as: $0^+,1,1^+,2,2^+,\ldots,(P-1),(P-1)^+,P$. The RPQ$^+$ always selects a packet from the highest priority non empty FIFO for transmission. Connections are divided into traffic sets as $C_1, C_2, \ldots, C_P$. Connections in the set C_i assumed to have identical delay bounds $d_i=i\Delta$, where Δ is the priority rotation (or re-labeling) interval. The queue rotation is a two step process, one a concatenation step and the second a promotion step. In the concatenation step, the current FIFO *i* and FIFO i^+ are merged to form a FIFO *i* for all $1\le i\le P$. Packets from FIFO i^+ are concatenated to the end of FIFO *i*. In the promotion step, FIFO *i* is relabeled as FIFO $(i-1)^+$ for all $1\le i\le P$. A new empty FIFO *i* is also created to hold arriving packets during the next rotation interval. All packets reside in FIFO i^+ queue after the promotion step.

L. Leap Forward Virtual Clock [SVC97]

Leap Forward Virtual Clock (LFVC) is a modification to the Virtual Clock (VC) algorithm to provide throughput fairness, using two mechanisms: a quarantine mechanism and a leap forward mechanism. The original virtual clock has in general throughput fairness problems when some sources send bursty data while others remain idle. When the rates of traffic flows vary widely, it also can give short-term unfairness. The LFVC tries to avoid this problem by providing two queues: a high priority queue H and low priority holding queue L. Traffic from well behaved sources is always placed in queue H. When a traffic flow becomes oversubscribed, the packets of the flow are placed in queue L. The server always serves packet from queue H. It is possible that packets in queue H may have larger virtual finishing times than the queue L, which is a departure from the original algorithm to serve in increasing order of finish times. For a given traffic flow, the cell at the head of the queue in low priority queue is transferred to the high priority queue, before its delay condition is violated. Since the LFVC is work conserving, to serve a low priority queue L when there are no packets waiting in the high priority queue, the server clock is advanced as far as possible (called Leap Forward) without violating the delay condition for any traffic flows. The resulting scheme provides throughput fairness and end-to-end delay bounds.

M. Frame-based Fair Queuing [Sva96-1]

A frame-based fair queuing algorithm that provides the same end-to-end delays as a PGPS server is proposed in [Sva96-1]. This algorithm uses a potential function $P_i(t)$ for connection i at time t, defined as: $P_i(t) - P_i(\tau) = \left(W_i(\tau,t)/\rho_i\right)$, where $W_i(\tau,t)$ is the amount of service received by the connection i during the interval $(\tau,t]$ and ρ_i is the rate allocated to the connection. The potential of a connection is non-decreasing function of time during the system busy period and when the connection is backlogged, it increases by the normalized service it received. A fair algorithm is the one, which attempts to increase the potentials of all backlogged connections at the same rate.

The fluid version of the Frame-based Fair Queuing (FFQ) is defined as the server, which services the set of backlog connections with the minimum potential and the connections in the set are served at a rate proportional to their requested rates. The time is split into frames of F bits. If LR is the link rate, then the frame period T is F/LR. Let $\varphi_i=\rho_i T$, which denotes the maximum amount of traffic session i can send during the frame. If L_i is the maximum size of a packet for the session, then it is also assumed that $L_i \leq \varphi_i$. Let $f(t)$ be the frame which is in progress at time t. When the system is empty, $f(t)$ is reset to zero and the potentials of all backlogged connections is also reset to zero. When the potentials of all backlogged connections equals to T, then $f(t)$ is increased by T. The system potential function is set as follows: Let the frame update occur at time τ. Then, $f(t)$ and $P(t)$ are set as: $f(\tau)=f(\tau^{-})+T$ and $P(\tau)=max(P(\tau^{-}),f(\tau))$. At other times, system potential is computed as $P(t)=P(\tau)+(t-\tau)$. The scheduler uses this potential function in order to compute the timestamp of the packet. This FFQ algorithm belongs to the general class of Rate-Proportional Servers (RPS) which provide low latency and bounded fairness.

Two algorithms are executed on the arrival and departure of a packet. On arrival, a packet is stamped with its finishing potential, computed from the knowledge of the packet length, the reserved rate and starting potential. The starting potential itself is estimated as the maximum of finishing potential of its previous packet and the system potential. When the packet finishes transmission, another algorithm is executed which updates the state of the system.

N. Variation Fluctuation Smoothing [MLG97]

To optimize CDV performance for circuit emulation services in ATM networks, a scheduling algorithm called Variation Fluctuation Smoothing (VFS) is proposed in [MLG97]. The VFS algorithm estimates the clock of each connection by using on-line traffic measurements. Then it computes the lateness of the head-of-line cell for each connection and assigns highest priority to the latest cell. The algorithm stores two variable called lateness L_{i-1} and arrival time Y_{i-1} of the last cell. When the next cell i arrives, its lateness is computed as $L_i=\alpha L_{i-1}+(Y_i-Y_{i-1}-T)$, where T is the nominal assembly time of the connection corresponding to a given rate. When this cell is at the head-of-line position, its current lateness is calculated by adding the waiting time to L_i. This final lateness for the head-of-line cells for all streams is compared and the cell with greatest lateness is sent. With a jittered CBR traffic, it is shown in [MLG97] that this VFS scheme outperforms both FIFO and EDD in providing reduced jitter for connections.

Non-Work Conserving Fair Share Scheduling Algorithms

Many of the real-time services require bounded delay guarantees for packets transmitted across a network. If the traffic streams are leaky-bucket controlled, it is possible to obtain delay performance bounds for some of the WFQ schemes discussed in the above section. So these schemes can be used at the edge of networks where traffic policers are in place. But with work conserving disciplines, traffic distortions take place within the network. Due to this problem, traffic characterization in a network becomes very difficult and the end-to-end delay bounds using WFQ schemes becomes harder to achieve. To obtain end-to-end delay bounds for the connections, one may control the traffic at every switch in the network by using non-work conserving disciplines to serve the traffic flow. That is, a framing-strategy is used, whereby, packets are not serviced until a predetermined time and held in their own queues. This behavior may increase the average delay each packet experiences over a corresponding WFQ system, but the end-to-end delay is bounded. This section discusses at some of the methods proposed for such service.

A. Stop-and-Go [Gol90]

This scheme is based on a framing strategy and assumes that traffic streams which need delay bounds are *(r,T)*-smooth. A packet stream is defined to be *(r,T)*-smooth, if during each frame of length *T*, the arrived packets have no more than *r.T* bits. If packets are of fixed length, this scheme is equivalent to receiving *(r.T)/packet size* packets during the interval *T*. In the Stop-and-Go service discipline, if a connection *k* is (r_k,T) smooth when it enters the network, the property continues to hold at any switching node. The service strategy is based on framing. That is, both incoming and outgoing links are assumed as carrying fixed size frames. These frames need not be synchronous across input and output, i.e., output frames can be time lagged with respect to the input frames. The transmission of packets arriving on an input frames is postponed until the beginning of the next frame on the output link. When a connection traverses *H* links, the total delay experienced is a fixed value between *HT* and *2HT*, where *T* is the frame size. Since delay is proportional to *T*, the frame size, for small delay connections *T* should be small. However, the bandwidth granularity is inversely proportional to *T*, that is, if all packets are fixed size, the minimum bandwidth allocated to a connection is *packet size/T* bits/sec. Thus, *T* should be large for smaller bandwidth granularity. These are two opposing requirements. Therefore, Golestani [Gol90] also proposed a multi-level framing strategy, a generalized version of stop-and-go.

B. Hierarchical Round Robin [KKK90]

The Hierarchical Round Robin (HRR) also uses framing strategy like Stop-and-Go. The frames are divided into various levels, each level can consist of different frame length. At each level, slots are assigned to connections directly on that level or reserved for usage by lower levels. Thus, a certain fraction of link bandwidth is assigned to lower level frames. This behavior is depicted in Figure 5.11, where the slot "C" is applied to any connection and the slot "L" is applied to next level.

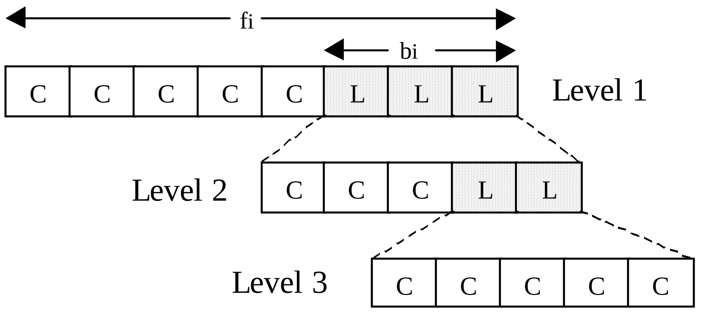

Figure 5.11 Levels in Hierarchical Round Robin

Finer bandwidth granularity can be achieved by allocating slots at various levels for the same connection. Let b_i represent the number of slots allotted to the next level and f_i represent the slots in a frame at level i. If a connection is assigned k_i slots at level i, then the bandwidth a connection gets in a L level HRR scheme is:

$$ConnectionBandwidth = \left(\sum_{i=1}^{L} k_i \frac{b_1 b_2 \ldots b_i}{f_1 f_2 \cdots f_i} \right) \times LinkCapacity \qquad (5.9)$$

Note that if there is only one level, HRR turns into a WRR. It should also be noted that the last level is a simple WRR. The original HRR scheme is non-work conserving. That is, if there are no packets waiting for a connection when serving a given slot, instead of sending packets assigned to other slots, the server is left idle. This way, the total duration of the cycle (i.e., cycling through all frames) is always be constant. Due to this bahavior, HRR maintains the traffic smoothness inside the network and can guarantee delay bounds. Keeping the frame size constant, the HRR can be made work conserving by assigning unused slots temporarily to the demand traffic. Queuing delays when HRR servers are used and in general with rate controlled schedulers is developed in [BK93].

C. Dynamic Time Slice [Sri92]

In the Dynamic Time Slice (DTS) method, each service class is assigned a time-slice and the queue is served until the expiration of the time-slice duration or empty queue which ever occurs first. If there are *n* service classes and each is assigned a time slice T_i, then each service class gets a link bandwidth of $\left(T_i / \sum_i T_i\right).LinkCapacity,\ 0 \le i \le n$ and $\sum_i T_i \le D_C$, where D_C is the DTS cycle time. Since the cycle time is fixed, delay guarantees can be met for the services, which require it.

D. Jitter Earliest Due Date [VZF91]

The Jitter Earliest Due Date (EDD) is an extension to Delay-EDD to provide delay jitter bounds. That is, the method provides both minimum as well as maximum delay bounds on the delay. To guarantee a delay jitter bound, each node in the network has to preserve the packet arrival traffic pattern on a given connection. In this scheme, each packet is stamped with a value (called *correction term*), which is the time difference between the instant the packet is served and the instant it was supposed to be served (its deadline). A traffic regulator is used at the next switch, which holds the packet for this duration before the packet is declared eligible. The *holding time* is defined as *(correction term + delay bound-jitter bound).* Then the packet's *eligibility time = holding time + arrival time* computed and the packet's deadline is determined as *max{(eligibility time + jitter bound), (deadline of last packet + x_{min})}*, where x_{min} is the minimum packet inter-arrival time. The scheme can be viewed as consisting of traffic regulators for each channel followed a bounded delay server, which is shared by all channels.

Traffic-Shaping

As discussed in Chapter 3, a source, an adaptation device or the network can apply a traffic shaping function. The objective of traffic shaping is to create a cell flow, conformant to the traffic descriptors. In this chapter, traffic shaping function of a switch within a network is discussed. Traffic shaping can be applied to both per-Group and per-VC queuing structures. In the former case, the aggregate traffic is shaped while in the latter per connection traffic shaping is performed. Actually the traffic shaping can be seen as the "rate-controlled" fair-share scheduling technique. Since it is a little different from the rate-controlled fair-share scheduling techniques, the traffic shaping is considered as a separate scheduling discipline.

In rate-controlled scheduling described above, the cells are held (or delayed) to control delay jitter (e.g., Stop-and-Go, Jitter-EDD, DTS) or rate jitter as in HRR. The rate control in HRR is based on average connection rate and does not take into account the burst properties of the ATM traffic flows. The traffic shaping schemes control the rate of connections as per the traffic conformance definitions (such as GCRA or leaky-bucket algorithm). This technique is becoming very popular and many modern switches implement it to shape the traffic flow. ATM networks support both real-time and non real-time services. Thus a combination of work conserving and rate-controlled service disciplines makes a good compromise to support a multi-service platform. This section looks at a few such architectures.

An integrated traffic shaping and link scheduling architecture is described by Rexford et al. in [RBGW97]. The architecture assumes that connections are leaky bucket policed (see Chapter 3) with (σ,ρ) being the parameters, where σ is the token bucket size and ρ the token generation rate. The "conforming" cells are then fairly scheduled using rate-based algorithms. Instead of dropping, traffic shaper delays the incoming cells until they conform to the traffic descriptors. Cell conformance times are estimated using ATM forum's virtual scheduling algorithm. Let t be the cell arrival time, c the conformance time, X the estimated arrival time, (σ,ρ) are the leaky bucket parameters. Then the cell conformance computation is performed as [RBGW97]:

```
$X = X + 1/\rho$                      // Estimated Cell arrival time
$if (X \leq t)$                       // Full token bucket : reset $X$
    $c = X = t;$
$else\ if (X \leq t + \sigma/\rho)$   // Partially full token bucket
    $c = t;$
$else$                                // Empty token bucket : delay cell
    $c = (X - \sigma/\rho);$
```

Figure 5.12 Cell Conformance Computing

The cells are stamped with this conformance time. The shaper can also enforce both peak and average rate policing by incorporating a dual-leaky bucket algorithm conforming to *$(\sigma,\rho,\rho_{peak})$* for each connection. The shaper then uses the conformance times *c* to schedule cells on the link. A calendar queue can be used to sort the cells in increase order of conformance times. A calendar queue can be thought of as an array of linked lists. Each position of the array corresponds to a given conformance time and the linked list at a given position corresponds to the cells that are to leave at that time. Normally there are many connections being multiplexed on to a given a link and therefore, there is a chance that quite a few cells can have the same conformance time. This event is called cell “collisions” and was discussed in Chapter 3. Since it is only possible to transmit a single cell at a time on a given link, a transmission FIFO is used to hold the cells, which have reached their conformance time.

Instead of including all the backlogged cells of a connection, only the head-of-line is scheduled on to the calendar queue to reduce the complexity of the shaper and prevent large number of collisions. This simplification is possible because the head-of-line cell has the conformance time at most *$1/\rho$* into the future; otherwise there would be a previous cell in the shaper for this connection. Therefore, a per-connection FIFO can be used to hold the backlogged cells of a connection, while the head-of-line cell is scheduled on to the calendar queue and waiting to go at its scheduled time. This system is shown in Figure 5.13. It is assumed that each traffic flow is separately queued. The conformance times for the waiting cells is shown as the label *c=x*. The calendar sorter lists either the flow identification numbers or the cells at the heads of the queues at their respective conformance times. In the Figure 5.13, a slot *E* indicates that no cell is scheduled to go at that time and is empty. The calendar sorter reads the cells at the current calendar time and appends them to the transmission FIFO from which the cells are eventually transmitted.

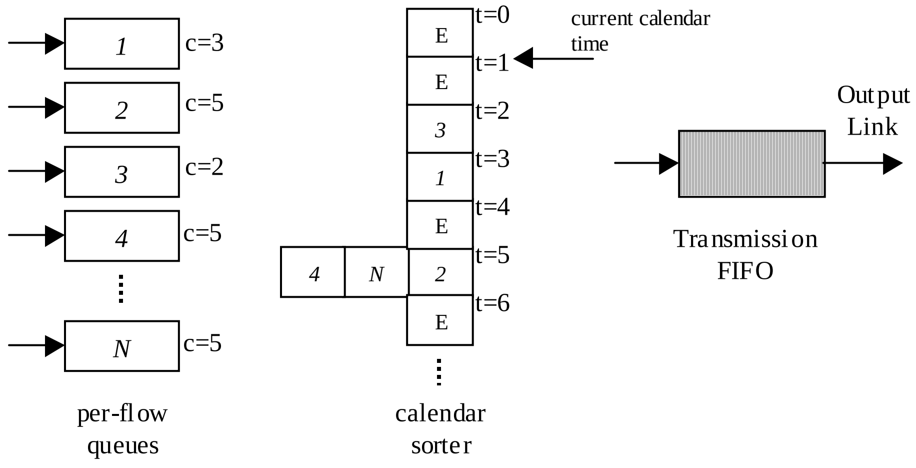

Figure 5.13 Calendar Queue Sorter

But this scheme can complicate the implementation of exact sorting, because the new cell entering the calendar might have already reached its conformance time and need to be inserted between other cells waiting for their service. Approximate sorting algorithms become necessary to reduce these overheads. For example, head-of-line cells, which have reached their conformance times, can be directly sent to the transmission FIFO instead of scheduling on the calendar. One of the problems with calendar queue implementation is that a low bandwidth connection may be scheduled very far into the future, thus needing a very large array of calendar sorter. This problem can be avoided by incorporating multiple level (granularity) calendar sorters. For example, in Figure 5.13, the calendar granularity is shown to be one cell transmission time (or one cell slot). As the granularity (*g*) increases, the calendar sorter maps the conformance times between time *t* and (*t+g)* on to the same slot on the sorter, thus needing smaller array of linked lists. As an example, if *g=4*, then conformance times 0 to 3 time slots are mapped on to the same slot in the calendar sorter. Therefore, the time "resolution" of the calendar is poor for large granularity, which causes an increase in the jitter of the transmitted cells. Generally, high-speed connections cannot tolerate large jitter and therefore should be mapped on to low granularity sorters. In a network with wide variety of traffic connections, it is possible to arrange the calendar sorter in-groups. Then, a group level arbitration becomes necessary.

Since the calendar sorter schedules head-of-line cells as per traffic shaping rules, it can also be used for other implementations, such as WFQ, where the shaping is as per weight of the connection. Instead of using a leaky bucket shaper, virtual finishing times can be used to schedule cells on to the calendar to achieve WFQ. Numerous possibilities occur to design a multi-service cell scheduler using a combination of leaky-bucket traffic shaping and WFQ schedulers in one integrated approach using multiple calendar sorters. One such example is shown in Figure 5.14, where, some traffic flows use the WFQ calendar and others use leaky bucket shaping calendars.

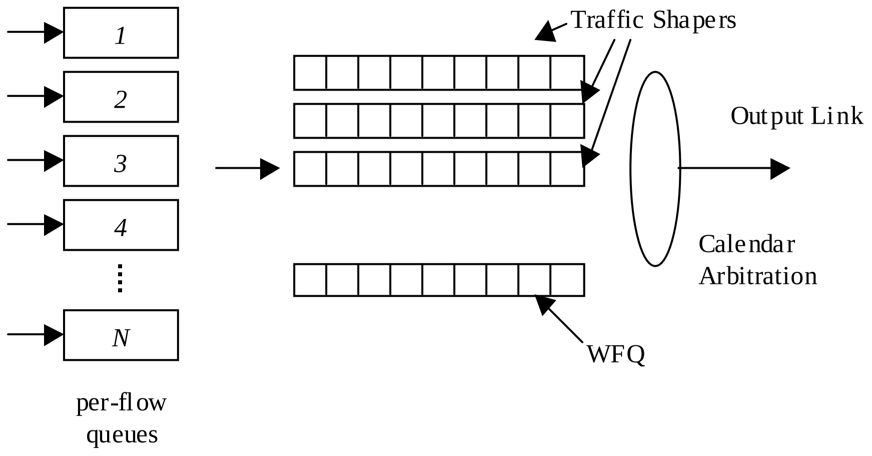

Figure 5.14 Integrated Traffic Shaping and WFQ

Each of the traffic shapers and WFQ shown in Figure 5.14 are the complete calendar sorters, each slot is a bin which holds multiple connection identifiers scheduled to go at that instant. Note that, it is also possible to avoid the use of a transmission FIFO by using two pointers for every traffic shaper. One is pointing to the real calendar time and the other to the virtual calendar time. New connections are scheduled on to the calendar using the real calendar time pointer, while cells are actually read from the virtual calendar time. The virtual calendar pointer always lags behind the real calendar time pointer and tries to catch up whenever cell collisions occur.

A Rate Controlled Static Priority (RCSP) queuing is proposed by Zhang [ZF93]. This scheme uses traffic regulators for each connection followed by a static priority queuing structure (see Figure 5.15). By constructing either partial or full traffic pattern, both rate and delay jitter control regulators can be used respectively. Calendar sorting is used for traffic shaping. The scheduler consists of one more stage of queuing, which is comprised of a set of priority queues. This scheme is similar to the approach shown in Figure 5.14, where the calendar arbitration is done by a set of priority queues.

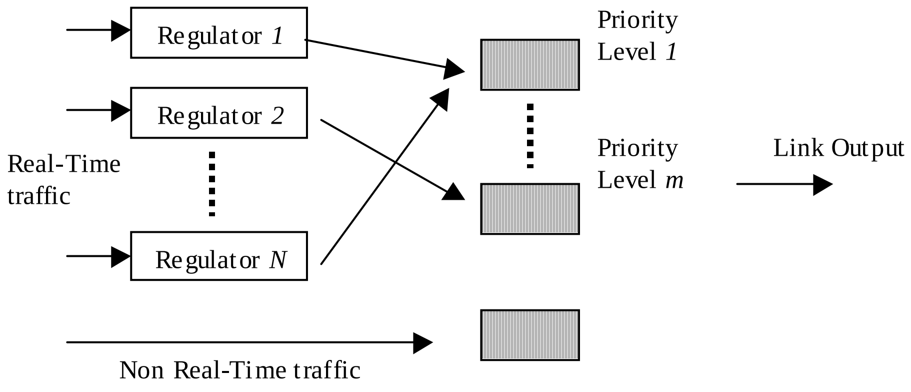

Figure 5.15 Rate Controlled Static Priority (RCSP) Queuing

A general methodology for designing integrated shaping and scheduling based on FFQ is presented by Stiliadis et al. [Sva97]. The FFQ server belongs to the class of Rate Proportional Server (RPS) whose aim is to equalize the potential for all backlogged connections at each instant. See the definition of potential function in FFQ description. A Shaped Rate Proportional Server (SRPS) is the one which uses both a shaper and scheduler. The shaper is used to admit eligible packets into the scheduler, the admission criterion being the current value of system potential in the RPS is equal to or greater than the finishing potential of the pervious packet admitted from that connection. The logical structure is shown in Figure 5.16.

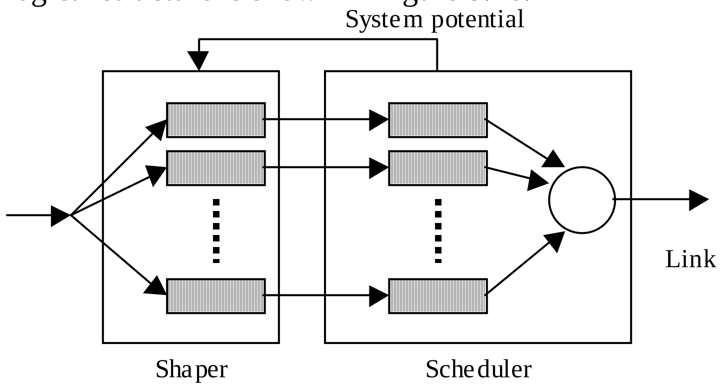

Figure 5.16 Structure of Integrated Shaper and Scheduler in SRPS Class

The scheduler has a separate queue to hold eligible packets for each connection. The reference [Sva97] also proposed another scheduling algorithm called Shaped Starting Potential-based Fair queuing (SPFQ). It is also shown that RPS algorithms like Virtual Clock, which are not fair can yield worst-case fairness identical to WFQ.

When Rate Controlled (or non-work conserving) Servers (RCS) are used in conjunction with shapers, end-to-end delay bounds can be provided for sessions traversing multiple hops. This scheme is shown in [GGPS96], where the traffic is re-shaped at every hop before entering the next switch. Main disadvantage of RCS servers is that the end-to-end delay bounds are generally obtained as sum of worst-case delays at each node. This problem is alleviated by proper re-shaping function at every node and performance similar to a GPS system is obtained. The system can be made work conserving by maintaining two queues, a high priority and low priority, like in Leap Forward Virtual Clock. High priority queue is used to hold packets that are eligible for scheduling (i.e., re-shaped) and low priority queue is used to hold packets, which are not yet eligible. A non-preemptive arbitration is used to schedule packets and packets from low priority queue are served only when the high priority queue is empty. Delay guarantees are not affected when operating in the work-conserving mode.

Comparison of Scheduling Schemes

Several scheduling algorithms are reviewed and discussed in this chapter. There are several measures proposed in the literature like fairness index, complexity, delay guarantees, latency, minimum bandwidth property etc., which can be used to compare these algorithms.

Golestani [Gol94] defined the notion of *fairness* as the difference between the normalized service received by different sessions. The normalized service w_k of a session k is defined as the ratio of aggregate traffic $W_k(t_1,t_2)$ served during an interval (t_1,t_2) of the session to the service rate r_k allocated to that session. Then the fairness index (FI) for two sessions i and j is given by:

$$FI = \left| w_i(t_1,t_2) - w_j(t_1,t_2) \right| = \left| \frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j} \right| \quad (5.10)$$

Bennet and Zhang [BZ96] defined *worst-case fairness* of a session as follows: a session i is called worst-case fair, if for any time t, the delay of a packet $d_i(t)$ arriving at t is bounded above by $C_i+Q_i(t)/r_i$, where r_i is the guaranteed throughput of the session i, $Q_i(t)$ is the queue size of the session at time t and C_i is a constant independent of queues of other sessions sharing the multiplexer. That is, worst-case fair index C_i is such that,

$$d_i(t) \le C_i + \frac{Q_i(t)}{r_i} \quad (5.11)$$

The *minimum bandwidth property* of the schedulers is defined in [HuK96]. This metric looks at departure times from a processor-sharing node versus the departure times from a FIFO buffer, if the server is given a devoted bandwidth. Consider a scheduler for N FIFO buffers. For connection j, let d_i^j be the departure time of i^{th} cell from the processor-sharing node using this scheduler. Then, the scheduler is said to have a *minimum bandwidth* *property* with parameter μ, if $d_i^j = \left| F_i^j \right| + \mu$ for all cell arrival processes, where $\lceil F_i^j \rceil$ is the departure time of i^{th} cell from the FIFO system having dedicated bandwidth. The parameter μ for some of the servers described in this chapter is as follows: $\mu=-1$for PGPS, $\mu=N-2$ for SCFQ, $\mu=-1$for Virtual clock; where N is the number of sessions.

Other comparison parameters are per connection buffer space requirements, order of computational complexity of the algorithms, end-to-end delay bounds and end-to-end delay-jitter bounds. Note that some of the delay bounds for work conserving disciplines are obtained under the assumption that traffic is constrained. The models used are either a leaky bucket model or other constrained traffic models such as $(X_{min}, X_{ave}, I, S_{max})$. Here, X_{min} is the minimum packet inter-arrival time, X_{ave} is the average packet inter-arrival time, I is the averaging interval over which X_{ave} is computed and $S_{max}$is the maximum packet size.

The classification introduced in this chapter as work conserving and non work-conserving scheduling is defined in a broad sense. Further classification is also possible based on what type of server mechanism used (i.e. whether server uses a single scheduler or uses a rate controller and a scheduler), type of queuing used (static priority versus sorted priority) and whether framing is used or not. Zhang [Zha95] introduced a taxonomy of service disciplines based upon which Stiliadis and Varma [Sva96] defined a general class of Latency Rate (LR) servers, describing the worst-case behavior of numerous scheduling algorithms. A scheduling algorithm belongs to the class of LR servers, if the average rate of service offered by the scheduler to a busy session i at least equals to its reserved rate, over every interval starting at time θ_i from the beginning of a busy period. The parameter θ_i is called the latency of the scheduler and is the worst case delay seen by a first packet of the busy period session, i.e., packet arriving to an empty queue. All the work-conserving schedulers belong to this model. If the traffic is leaky bucket constrained with parameters (σ_i,ρ_i), then the delay bound for session i of such servers is given by: $D_i \le \theta_i + (\sigma_i/\rho_i)$. The delay is also bounded when the traffic is passed through a series of n such servers and is given by: $D_i^n \le \sum_{j=1}^{n} \theta_i^j + (\sigma_i/\rho_i)$.

Stiliadis and Varma [Sva96-1] further classified the LR servers to Rate Proportional Servers (RPS) based on system potential function with zero latency. Example of such server is based on Frame-based Fair queuing strategy.

Table 5-2 shows some comparisons (also see [SVC97, Sva96-1, Zha95]) for various LR servers. The following notations are used: N the number of connections; L_i the maximum packet size of session i; L_{max} is the maximum packet size among all sessions; F is the frame size; ϕ_i is the amount of traffic allocated in a frame to session i for frame-based servers; L_c is the size of fixed packet; r_i is the rate requested by the session.

Table 5-2 Some comparisons of LR servers

Server	Bandwidth fair?	Worst case fairness	Latency	Complexity
GPS	Yes	Excellent	0	Not practical
PGPS	Yes	Poor	$(L_i/\rho_i)+(L_{max}/r_i)$	*O(N)*
Virtual Clock	No	Poor	$(L_i/\rho_i)+(L_{max}/r_i)$	*O(logN)*
Frame-based Fair Queuing	Yes	Poor	$(L_i/\rho_i)+(L_{max}/r_i)$	*O(logN)*
SCFQ	Yes	Poor	$(L_i/\rho_i)+(N-1)(L_{max}/r_i)$	*O(logN)*
WF^2Q	Yes	Good	$(L_i/\rho_i)+(L_{max}/r_i)$	*O(logN)*
Deficit Round Robin	Yes	Poor	$(3F-2\varphi_i)/r_i$	*O(1)*
Weighted Round Robin	Yes	Poor	$(F-\varphi_i+L_c)/r_i$	*O(1)*
Leap Forward VC	Yes	Good	*Same as VC + a small constant*	*O(log logN)*

Table 5-3 compares (also see[ZK91]) some of the service disciplines, which provide delay guarantees. Some of these algorithms are non-work conserving, which are also referred to as Rate-Controlled Service disciplines.

Table 5-3 Server Comparison which provide Delay Gurantees

Server	Bandwidth fair ?	Delay guarantee?	Jitter guarantee?	Work con-serving ?
Stop-and-Go	Yes	Yes	Yes	No
HRR	Yes	Yes	No	No
Delay -EDD	Yes	Yes	No	Yes
Jitter -EDD	Yes	Yes	Yes	No
DTS	Yes	Yes	No	No
RCSP	Yes	Yes	Yes	No

OTHER ISSUES

The implementation of the queuing and scheduling mechanisms should also consider the following issues:

1. CLP transparency
2. CLP flooding
3. Programming the weights

CLP transparency

The CLP transparency support (see Chapter 3) applies to the traffic conformance definition of CBR.1, VBR.1 (both real-time and non real-time) classes. These service classes conformance definition is based on CLP0+1 stream. That is, the SCR and PCR conformance applies to the CLP0+1 stream. The per-Group queuing structures of Figure 5.3 (grouping is as per service category) and Figure 5.4 (grouping is as per service class) share the same queue for some of the service classes. For example, the queuing structure shown in Figure 5.3 uses one VBR queue for all conformance definitions of VBR. The queuing structure shown in Figure 5.4 shares one queue for CBR and rt-VBR. Therefore, the queue mechanism uses CLP discard thresholds for the support of VBR.2 and VBR.3, when service classes are mixed together. This affects the QoS guarantees to CBR.1 (if CBR shares the queue with rt-VBR) and VBR.1, as their traffic conformance definition applies to CLP0+1 stream and as such should not have any discard thresholds on their queues. Therefore, the queuing and scheduling must transparently support CLP1 traffic within the conformance framework of CBR.1 and VBR.1 services. One approach is to save the CLP status and reset it (i.e., mark them as CLP0) when the cells enter the switch. The saved status is copied back when the cells exit the switch. For this purpose, the CLP status context has to be maintained, i.e., whether the cell is deliberately marked as CLP0 or not. Generally, most of the switches maintain internal cell format, which is longer than 53 bytes, and one bit can be used for CLP transparency purposes. Note that this CLP transparency issue is easier to deal in per-Group queuing, where grouping is done as per traffic conformance (figure 5.5).

CLP flooding

This CLP flooding problem applies to per-Group queuing with fair or priority scheduling. The main problem with some of the service sub-classes is that the network/user is allowed to tag excess traffic. For example VBR.2 and VBR.3 conformance definitions (see Chapter 3) allow the user to exceed the SCR, up to PCR, as long as the traffic is tagged. This behavior creates the problem of “CLP1 flooding”, since the CLP1 traffic although confirming, is not eligible for CLR guarantees, has not allocated any bandwidth but tries to compete for the available resources. It is possible for some of the sources to send their complete traffic as CLP1 tagged up to PCR.

Proper handling of the CLP1 traffic becomes necessary so that it does not steal bandwidth away from services offering guarantees and gets some fair share of available bandwidth. The other significant problem that arises is guaranteeing bandwidth to ABR, if a priority scheduling is used which gives higher priority to VBR service. Since VBR.2 and VBR.3 are served at a higher priority than ABR in this case, the ABR service can get starved due to CLP1 flooding. It is a philosophical discussion whether this CLP1 traffic should be given priority over UBR traffic (which in most cases is true since VBR may be served as higher priority over UBR). It is also debatable how the excess bandwidth (i.e., the bandwidth available after guaranteed bandwidth is discounted from the link capacity) should be distributed among the CLP1 traffic, UBR and ABR.

Programming of the weights

The third issue relates to programming the weights of the fair-share schedulers in the case of group-based queuing. As seen in this chapter, all the fair-share schedulers use some sort of weights (either per-connection weights or per-connection guaranteed rates etc.) to guarantee bandwidth to each connection. When fair-share scheduling is used for per-Group queuing structure, it is the aggregate weight that need to be programmed. There are two possible paradigms:

> **Static weight setup**: guarantees certain bandwidth to the given group queue all the times, irrespective of how many connections are using it;
>
> **Dynamic weight setup**: is based on aggregation of weights of all connections using a given group queue.

When static weight set-up is used, a portion of the link bandwidth is kept available for a given group, which may cause fairness problems between groups. For example, consider the scenario in which, grouping is based on service category (see Figure 5.3). Assume 10 ABR connections and 1 UBR connection are sharing a link bandwidth. If static weight is used then the single UBR connection can take up the bandwidth available to the whole UBR service while 10 connections share the ABR bandwidth. This creates a fairness problem among groups. In the dynamic weight set-up, the efficiency of the scheduler depends upon, how often the weights are changed, i.e., on every connection setup or over a time interval, say every few seconds. However, the weights are re-programmed for each group depending upon the number of connections and bandwidth requirements. Therefore, the fairness problem between groups is minimized. This problem does not arise in per-VC/VP queuing as the connection weights are based on per-VC/VP bandwidth requirements.

REVIEW

The key to provide efficient multi-service platform in ATM networks is to provide appropriate queuing and scheduling mechanisms at various contention points in ATM networks. This chapter focused on some of these mechanisms. Though comprehensive, this chapter's intention is to introduce to the reader, the complex field of queuing and scheduling. Fair-share scheduling and per-VC queuing is necessary to provide guaranteed QoS. Generally, an ATM switch consists of a switching core and peripheral devices. The per-VC queuing can be limited to peripheral devices, where by traffic can be shaped (or controlled) into and out of switching core. Pushing congestion to the ingress peripheral devices by appropriate control can reduce the complexity of the core. Which scheduling algorithm to choose depends on implementation complexity and economics. The bandwidth scheduler in general must take care of:

1. CLP flooding
2. CLP transparency support
3. Fair allocation of bandwidth to connections
4. Fair allocation of bandwidth to service classes
5. Provide delay guarantees to real-time services.

Since a single type of scheduling mechanism cannot guarantee all QoS parameters, it may be necessary to use a combination of mechanisms to provide efficient services. For example, at a low-level traffic shapers are used for real time services, while a WFQ scheme is used between non real-time services. A priority queue can be used between the traffic shapers and WFQ at a high level.

It should be noted that main trade-off lies between simplicity of implementation and achieving fairness. Per-Group queuing and priority scheduling may be simple to implement while per-VC queuing and WFQ scheduler combination may prove to be expensive. But the former does not yield fairness while the latter provides the best possible fairness and allocation of bandwidth to connections. It should also be noted that fairness cannot be measured instantaneously. Thus, the schemes should at least try to achieve long term fairness. The bottom line is that the queuing and scheduling mechanism at a contention point should provide the contracted QoS objectives of the connections.

References

[AD89] H.Ahmadi, W.Denzel, “ A Survey of Modern High-Performance Switching Techniques”, *IEEE Journal on Selected Areas in Communications,* Vol.7, No.7, September 1989.

[BK93] A.Banerjea, S.Keshav, “Queuing Delays in Rate Controlled ATM Networks”, *Proceedings of IEEE INFOCOM’93,* SanFrancisco, 1993.

[BZ96] J.C.R.Bennet, H.Zhang, “WF^{2}Q: Worst-case Fair Weighted Queuing”, *Proceedings of IEEE INFOCOM 96, March 1996.*

[BZ97] J.C.R.Bennet, H.Zhang, “Hierarchical Packet Fair Queuing Algorithms”, *IEEE/ACM Transactions on Networking,* Vol. 5, No.5, October 1997.

[DKS89] A.Demers, S.Keshav, S.Shenker, “Analysis and Simulation of Fair queuing Algorithm”, *SIGCOMM89,* pp.1-12, September 1989

[Eng88] K.Eng, “A Photonic Knock-ot Swicth for High-Speed Packet Networks”, *IEEE Journal on Selected Areas in Communications,* August 1988.

[FV90] D.Ferrari, D.C.Verma, “A Scheme for Real-time Channel Establishment in Wide-Area Networks”, *IEEE Journal on Selected Areas in Communications,* Vol.8, No.3, April 1990.

[FP95] N.R.Figueira, J.Pasquale, “An Upper Bound on Delay for the Virtual Clock Service Discipline”, *IEEE/ACM Transcation on Networking,* Vol. 3, No.4, August 1995

[Gar96] M.W.Garrett, “A Service architecture for ATM: From Applications to Scheduling”, *IEEE Network Magazine*, May/June 1996.

[Gol90] S.J.Golestani, “A Stop-and-Go Queuing Framework for Congestion Management”, *Proceedings of SIGGCOMM90,* pp.8-29

[Gol94] S.J.Golestani, “A Self Clocked Fair Queuing Scheme for Broadband Applications”, *IEEE INFOCOM’94*, Toronto

[GGPS96] L.Georgiadis, R.Guerin, V.Peris, K.N.Sivarajan, “Efficient Network QoS Provisioning based on per Node traffic Shaping”, *IEEE/ACM Transactions on Networking,* Vol. 4, No.4, August 1996.

[HK88] M.G.Hluchyj, M.J.Karol, “Queuing in high-performance Packet Switching”, *IEEE Journal on Selected Areas in Communications,* Vol. 6, No.9, December 1988.

[HuK96] A.Hung, G.Kesidis, “Bandwidth Scheduling for Wide-Area ATM Networks Using Virtual Finish Times”, *IEEE/ACM Transactions on Networking*, February 1996.

[JH87] J.Hui, “A Broadband Packet Switch for Integrated Transport”, *IEEE Journal on Selected Areas in Communications*, October 1987.

[KKK90] C.R.Kalamanek, H.Kanakia, S.Keshav, “Rate Controlled Servers for Very High-Speed Networks”, *IEEE Globecom’90*, SanDiego, December 1990

[KLE75] L. Kleinrock, “Queuing Systems, Volume 1- Theory”, *John Wiley and sons*, 1975.

[MLG97] D.McDonald, R.Liao, N.giroux, “Variation Fluctuation Smoothing for ATM Circuit Emulation Service”, *Proceedings of ITC*

[MLF92] A.Mukherjee, L.H.Landweber, T.Faber, “Dynamic Time Windows and Generalized Virtual Clock: Combined Closed-Loop/Open-Loop Congestion Control”, *Proceedings of IEEE INFOCOM’92 Florence, Italy*

[PG93] A.K.Parekh, R.G.Gallager, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case”, *IEEE/ACM Transactions on Networking,* Vol. 1, No.3, June 1993

[PG94] A.K.Parekh, R.G.Gallager, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case”, *IEEE/ACM Transactions on Networking,* Vol. 2, No.2, April 1994

[RBGW97] J.Rexford, F.Bonomi, A.Greenberg, A.Wong, “Scalable Architectures for Integrated Traffic Shaoing and Link Scheduling in High-Speed ATM Switches”, *IEEE Journal on Selected Areas in Communications,* Vol.15, No.5, June 1997

[Rob94] J.W.Roberts, “Virtual Spacing for Flexible traffic Control”, *International Journal of Communication Systems,* Vol.7, 1994.

[Sri92] K.Sriram, “Methodologies for bandwidth Allocation, Transmission, Scheduling and Congestion Avoidance in Broadband ATM Networks”, *Proceedings of IEEE Globecom’92,* Orlando, Florida, pp.1545-1551.

[SV96] M.Shreedhar, G.Varghese, “Efficient Fair Queuing Using Deficit Round-Robin”, *IEEE/ACM Transactions on Networking,* Vol. 4, No.3, June 1996.

[Sva96] D.Stiliadis, A.Varma, “Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms”, *Proceedings of IEEE INFOCOM96*

[Sva96-1] D.Stiliadis, A.Varma, “Design and Analysis of Frame-based Fair Queuing: A New Traffic Scheduling Algorithm for Packet Switched Networks”, *Proceedings of IEEE Sigmetrics’96*

[SVa97] D.Stiliadis, A.Varma, “A General Methodology for Designing Efficient Traffic Scheduling and Shaping Algorithms”, *Proceedings of IEEE INFOCOM97.*

[SVC97] S.Suri, G.Varghese, G.Chandranmenon, “Leap Forward Virtual Clock: A New Fair Queueing Scheme with Guaranteed Delays and Throughput Fairness”, *Proceedings of IEEE Infocom’97*

[Tob90] F.A.Tobagi, “Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks”, *Proceedings of the IEEE,* Vol. 78, No.1, January 1990.

[VZF91] D.C.Verma, H.Zhang, D.Ferrari, “Delay Jitter Control for Real-Time Communication in a Packet Swtching Network”, *Proceedings of Tricomm’91*, pp.35-46, Chapel Hill, North Carolina, April 1991.

[WL97] D.E.Wrege, J.Liebeherr, “A Near-Optimal Packet Scheduler for QoS Networks”, *Proceedings of IEEE Infocom’97*

[YO89] Y.Oie, “Survey of Switching Techniques in High-Speed Networks and their Performance”, *ICC89.*

[ZF93] H.Zhang, D.Ferrari, “Rate Controlled Static-Priority Queuing”, *Proceedings of IEEE INFOCOM’93.*

[Zha90] L.Zhang, “Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks”, *SIGCOMM90*, pp.19-29

[Zha95] H.Zhang, “Service Disciplines for Guaranteed Performance Service in Packet Switching Networks”, *Proceedings of the IEEE*, Vol. 83, No.10, October 1995.

[ZK91] H.Zhang, S.Keshav, “Comparison of Rate-based Service Disciplines”, *Proceedings of ACM SIGCOMM'91*, pp.113-121, Zurich, Switzerland, September 1991.