Chapter 2

QUEUING MECHANISM

2.1 Introduction

As part of the resource allocation mechanisms, each router must implement some queuing discipline that governs how packets are buffered while waiting to be transmitted. Various queuing disciplines can be used to control which packets get transmitted (bandwidth allocation) and which packets get dropped (buffer space). The queuing discipline also affects the latency experienced by a packet, by determining how long a packet waits to be transmitted. Examples of the common queuing disciplines are first-in first-out (FIFO) queuing, priority queuing (PQ), and weighted-fair queuing (WFQ)[6].

Queuing can send higher-priority traffic ahead of lower-priority traffic and makes specific amounts of bandwidth available for those traffic types.Bandwidth allocation is a fundamental problem in the design of networks where bandwidthhas to be reserved for connections in advance. Examples of queuing strategies considered later in this thesis include the following:

- First In First Out (FIFO).
- Priority Queuing (PQ).
- Fair Queuing (FQ).
- Weighted Fair Queuing (WFQ).
- Custom Queuing (CQ).
- Deficit Weighted Round Robin (DWRR).

2.2 First-In First-Out (FIFO)

FIFO queuing is the most basic queue scheduling discipline. In FIFO queuing, all packets are treated equally by placing them into a single queue, and then servicing them in the same order that they were placed into the queue. FIFO queuing is also referred to as First-come, first-serverd (FCFS) queuing[7].

The simplest way to schedule a packet in any network is FIFO. Here the first packet in the queue is served first in a particular time slot, regardless of any prioritization, protection or even fairness. Hence it is very simple to implement. However, it fails to achieve all other scheduling properties except complexity. FIFO suffers from head of line (HOL) issue, which means that if the first packet in the queue is blocked for any reason, the rest is blocked even though the link is idle.If a packet arrives and the queue (buffer space) is full, then the router discards that packet. This is done without regard to which flow the packet belongs to or how important the packet is. This is sometimes called tail drop, since packets that arrive at the tail end of the FIFO are dropped. FIFO accumulates packets when the link is congested and it will forward packets which are stored in queue, as far as network is not congested. FIFO handles all packets equally and it has only one queue [8].Figure 2.1.

FIFO is the default queuing method on interfaces that run at speeds greater than 2.048 Mbps. Although FIFO is supported widely on all Internetwork Operating System (IOS) platforms, it can starve out traffic by allowing bandwidth-hungry flows to take an unfair share of the bandwidth[13].

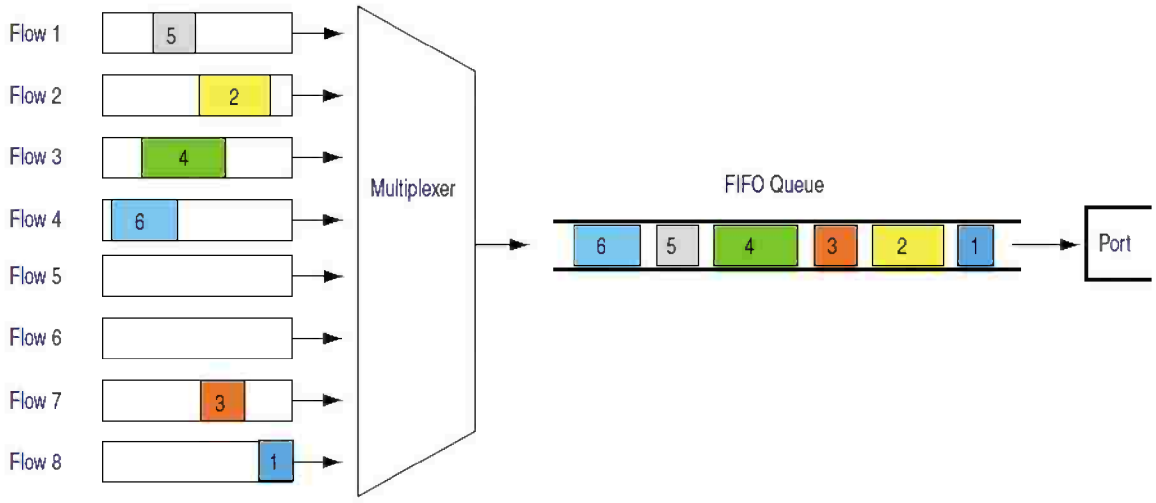

Figure 2.1 FIFO System[7].

2.2.1 FIFO benefits and limitations

FIFO queuing offers the following benefits:

- For software-based routers, FIFO queuing places an extremely low computational load on the system when compared with more elaborate queue scheduling disciplines.
- The behavior of a FIFO queue is that very predictable-packets are not reordered and the maximum delay is determined by the maximum depth of the queue.
- As long as the queue depth remains short, FIFO queuing provides simple contention resolution for network resources without adding significantly to the queuing delay experienced at each hop.

FIFO queuing also poses the following limitations:

- A single FIFO queue does not allow routers to organize buffered packets, and then service one class of traffic differently from other classes of traffic.
- A single FIFO queue impacts all flows equally, because the mean queuing delay for all flows increases as congestion increases. As a result, FIFO queuing can result in increased delay, jitter, and loss for real-time applications traversing a FIFO queue.
- During periods of congestion, FIFO queuing benefits User Datagram Protocol (UDP) flows overTransmission Control Protocol(TCP) flows. When experiencing packet loss due to congestion, TCP-based applications reduce their transmission rate, but UDP-based applications remain oblivious to packet loss and continue transmitting packets at their usual rate. Because TCP-based applications slow their transmission rate to adapt to changing network conditions, FIFO queuing can result in increased delay, jitter, and

a reduction in the amount of output bandwidth consumed by TCP applications traversing the queue.

- A bursty flow can consume the entire buffer space of a FIFO queue, and that causes all other flows to be denied service until after the burst is serviced. This can result in increased delay, jitter, and loss for the other well-behaved TCP and UDP flows traversing the queue[7].

2.2.2 FIFO implementations and applications

Generally, FIFO queuing is supported on an output port when no other queue scheduling discipline is configured. In some cases, router vendors implement two queues on an output port when no other queue scheduling discipline is configured: a high-priority queue that is dedicated to scheduling network control traffic and a FIFO queue that schedules all other types of traffic [7].

2.2.3 FIFO router configuration

Traffic within a single software queue (sometimes referred to as sub-queuing) is always processed using FIFO. Remember, software-based queuing is only used when the hardware queue is congested. Software queues serve as an intermediary, deciding which traffic types should be placed in the hardware queue first and how often, during periods of congestion.

2.3 Priority Queuing (PQ)

PQ is a simple variation of the basic FIFO queuing. The idea is to mark each packet with a priority.The mark could be carried, for example, in the IP Type of Service (ToS) field. The routers then implement multiple FIFO queues, one for each priority class. Within each priority, packets are still managed in a FIFO manner. The Priority Queuing algorithm dequeues the packet with highest priority, then the packet with the next higher priority, till finally the packet with the lowest priority. The problem with priority queuing is that the high-priority queue can starve out all the other queues. That is, as long as there is at least one high-priority packet in the high-priority queue, lower-priority queues do not get served [10].Figure 2.2.

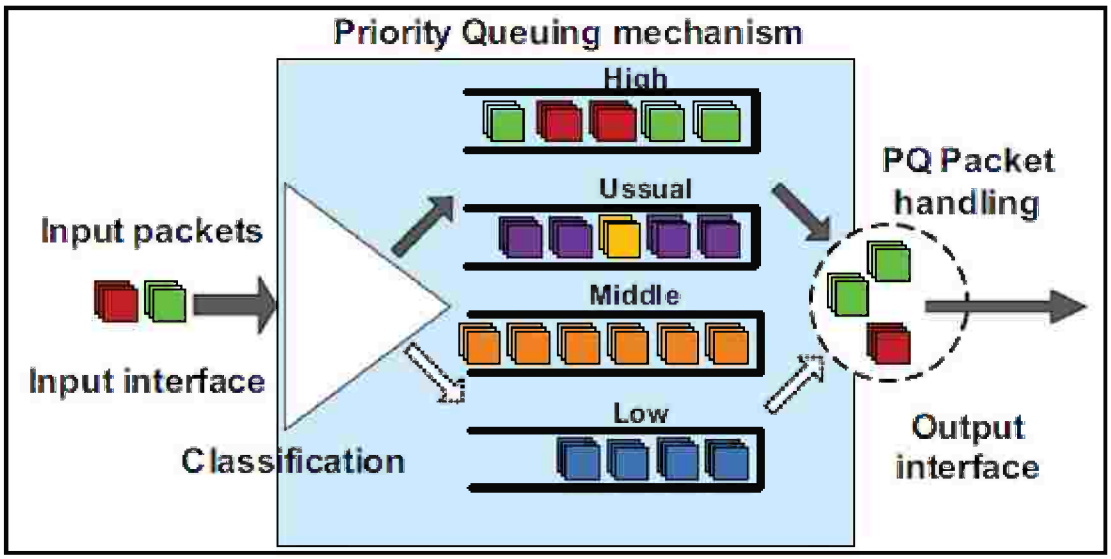

Figure 2.2 Priority Queuing System[9].

2.3.1 PQ benefits and limitations

PQ queuing offers the following benefits:

- The strength of PQ is that it is able to ensure highest priority to multimedia applications (especially for voice transfer), but this can also lead to infinite delay for the packets belonging to the lower priority queues[14].
- For software-based routers, PQ places a relatively low computational load on the system when compared with more elaborate queuing disciplines.
- PQ allows routers to organize buffered packets, and then service one class of traffic differently from other classes of traffic. For example, you can set priorities so that real-time applications, such as interactive voice and video, get priority over applications that do not operate in real time.

PQ queuing also poses the following limitations:

- If the amount of high-priority traffic is not policed or conditioned at the edges of the network, lower-priority traffic may experience excessive delay as it waits for unbounded higher-priority traffic to be serviced.
- If the volume of higher-priority traffic becomes excessive, lower-priority traffic can be dropped as the buffer space allocated to low-priority queues starts to overflow. If this occurs, it is possible that the combination of packet dropping, increased latency, and packet retransmission by host systems can ultimately lead to complete resource starvation for lower-priority traffic. Strict PQ can create a network environment where a reduction in the quality of service delivered to the highest-priority service is delayed until the entire network is devoted to processing only the highest-priority service class.
- PQ is not a solution to overcome the limitation of FIFO queuing where UDP flows are favored over TCP flows during periods of congestion. If you attempt to use PQ to place.TCP flows into a higher-priority queue than UDP flows, TCP window

management and flow control mechanisms will attempt to consume all of the available bandwidth on the output port, thus starving your lower-priority UDP flows.

2.3.2 PQ implementations and applications

Typically, router vendors allow PQ to be configured to operate in one of two modes:

- Strict priority queuing.
- Rate-controlled priority queuing.

Strict PQ ensures that packets in a high-priority queue are always scheduled before packets in lower-priority queues. Of course, the challenge with this approach is that an excessive amount of high-priority traffic can cause bandwidth starvation for lower priority service classes. However, some carriers may actually want their networks to support this type of behavior. For example, assume a regulatory agency requires that; in order to carryVoice over Internet Protocol(VoIP) traffic a service provider must agree (under penalty of a heavy fine) not to drop VoIP traffic in order to guarantee a uniform quality of service, no matter how much congestion the network might experience. The congestion could result from imprecise admission control leading to an excessive amount of VoIP traffic or, possibly, a network failure. This behavior can be supported by using strict PQ without a bandwidth limitation, placing VoIP traffic in the highest-priority queue, and allowing the VoIP queue to consume bandwidth that would normally be allocated to the lower-priority queues, if necessary. A provider might be willing to support this type of behavior if the penalties imposed by the regulatory agency exceed the rebates it is required to provide other subscribers for diminished service.

There are two primary applications for PQ at the edges and in the core of your network:

- PQ can enhance network stability during periods of congestion by assigning routing-protocol and other types of network-control traffic to the highest-priority queue.
- PQ supports the delivery of a high-throughput, low-delay, low-jitter, and low-loss service class. This capability allows you to deliver real-time applications, such as interactive voice or video[7].

2.4 Fair Queuing (FQ)

FQ was proposed by John Nagle in 1987. FQ is the foundation for a class of queue scheduling disciplinesthat are designed to ensure that each flow has fair access to network resources and to prevent a bursty flow from consuming more than its fair share of output port bandwidth. In FQ, packets are first classified into flows by the system and then assigned to a queue that is specifically dedicated to that flow. Queues are then serviced one packet at a time in round-robin order. Empty queues are skipped. FQ is also referred to as per-flow or flow-based queuing [11].Figure 2.3.

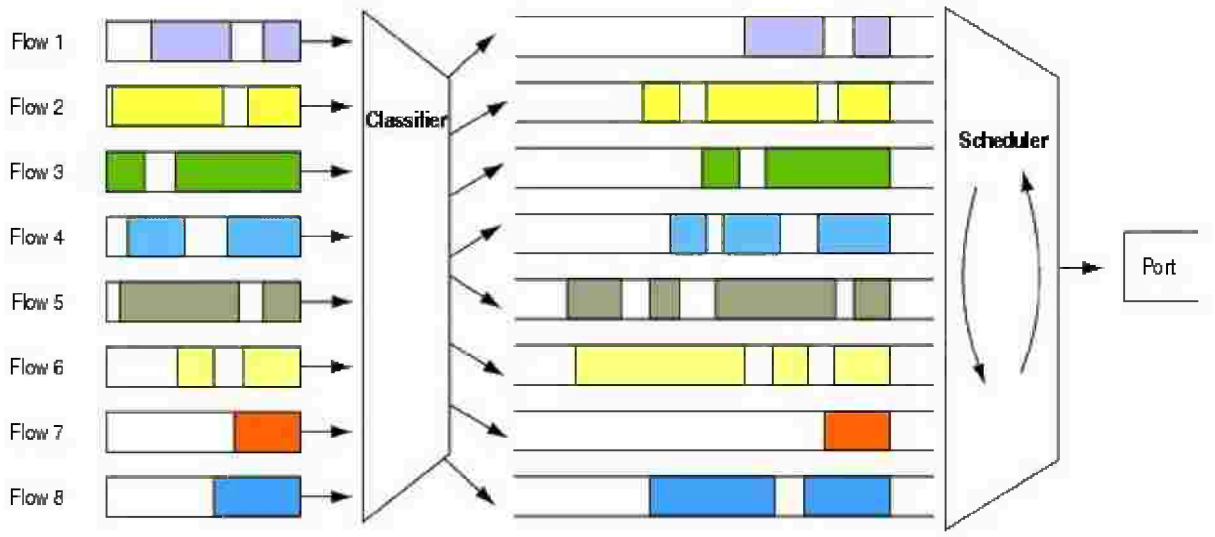

Figure 2.3FQ System[7].

2.4.1 FQ benefits and limitations

The primary benefit of FQ is that an extremely bursty or misbehaving flow does not degrade the quality of service delivered to other flows, because each flow is isolated into its own queue. If a flow attempts to consume more than its fair share of bandwidth, then only its queue is affected, so there is no impact on the performance of the other queues on the shared output port.

FQ also involves several limitations:

- Vendor implementations of FQ are implemented in software, not hardware. This limits the application of FQ to low-speed interfaces at the edges of the network.
- The objective of FQ is to allocate the same amount of bandwidth to each flow over time. FQ is not designed to support a number of flows with different bandwidth requirements.
- FQ provides equal amounts of bandwidth to each flow only if all of the packets in all of the queues are the same size. Flows containing mostly large packets get a larger share of output port bandwidth than flows containing predominantly small packets.
- FQ is sensitive to the order of packet arrivals. If a packet arrives in an empty queue immediately after the queue is visited by the round-robin scheduler, the packet has to wait in the queue until all of the other queues have been serviced before it can be transmitted.
- FQ does not provide a mechanism that allows you to easily support real-time services, such as VoIP.
- FQ assumes that you can easily classify network traffic into well-defined flows. In an IP network, this is not as easy as it might first appear. You can classify flows based on a packet's source address, but then each workstation is given the same amount of

network resources as a server or mainframe. If you attempt to classify flows based on the TCP connection, then you have to look deeper into the packet header, and you still have to deal with other issues resulting from encryption, fragmentation, and UDP flows. Finally, you might consider classifying flows based on source/destination address pairs. This gives an advantage to servers that have many different sessions, but still provides more than a fair share of network resources to multitasking workstations.

- Depending on the specific mechanism you use to classify packets into flows, FQ generally cannot be configured on core routers, because a core router would be required to support thousands or tens of thousands of discrete queues on each port. This increases complexity and management overhead, which adversely impacts the scalability of FQ in large IP networks.

2.4.2 FQ implementations and applications

- FQ is typically applied at the edges of the network, where subscribers connect to their service provider. Vendor implementations of FQ typically classify packets into 256, 512, or 1024 queues using a hash function that is calculated across the source/destination address pair, the source/destination UDP/TCP port numbers, and the IP Type of Service (ToS) byte.
- FQ provides excellent isolation for individual traffic flows because, at the edges of the network, a typical subscriber has a limited number of flows, so each flow can be assigned to a dedicated queue, or else a very small number of flows, at most, are assigned to each queue. This reduces the impact that a single misbehaving flow can have on all of the other flows traversing the same output port[7].

2.5 Weighted Fair Queuing (WFQ)

Weighted fair queuing (WFQ) was developed independently in 1989 by Lixia Zhang and by Alan Demers, SrinivasanKeshav, and Scott Shenke. WFQ is the basis for a class of queue scheduling disciplines that are designed to address limitations of the FQ model.The idea of the fair queuing (FQ) discipline is to maintain a separate queue for each flow currently being handled by the router. The router then services these queues in a round-robin manner. WFQ allows a weight to be assigned to each flow (queue). This weight effectively controls the percentage of the link's bandwidth each flow will get. We could use ToS bits in the IP header to identify that weight [11].

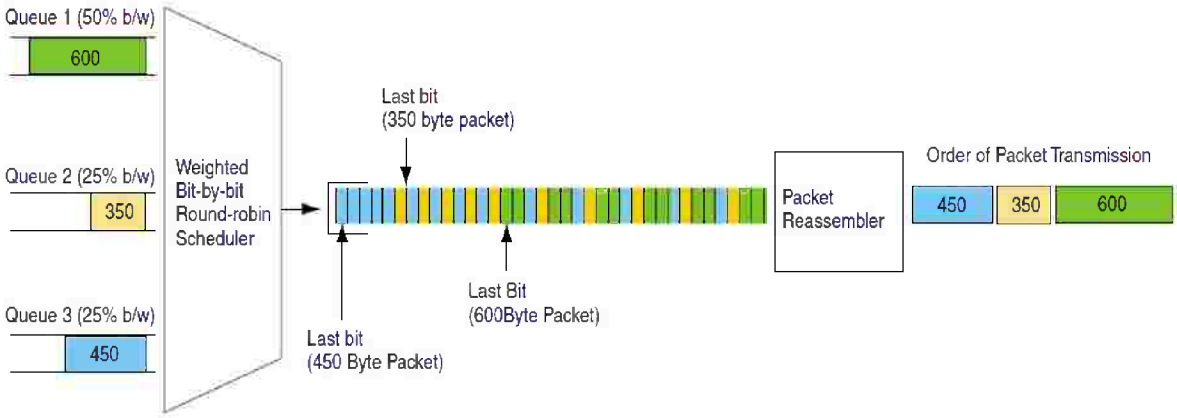

Figure 2.4 A Weighted Bit-by-bit Round-robin[7].

Figure 2.4 shows a weighted bit-by-bit round-robin scheduler servicing three queues. Assume that queue 1 is assigned 50 percent of the output port bandwidth and that queue 2 and queue 3 is each assigned 25 percent of the bandwidth. The scheduler transmits two bits from queue 1, one bit from queue 2, one bit from queue 3, and then returns to queue 1. As a result of the weighted scheduling discipline, the last bit of the 600-byte packet is transmitted before the last bit of the 350-byte packet, and the last bit of the 350-byte packet is transmitted before the last bit of the 450-byte packet. This causes the 600-byte packet finishes (complete reassembly) before the 350-byte packet, and the 350- byte packet finishes before the 450-byte packet[6].

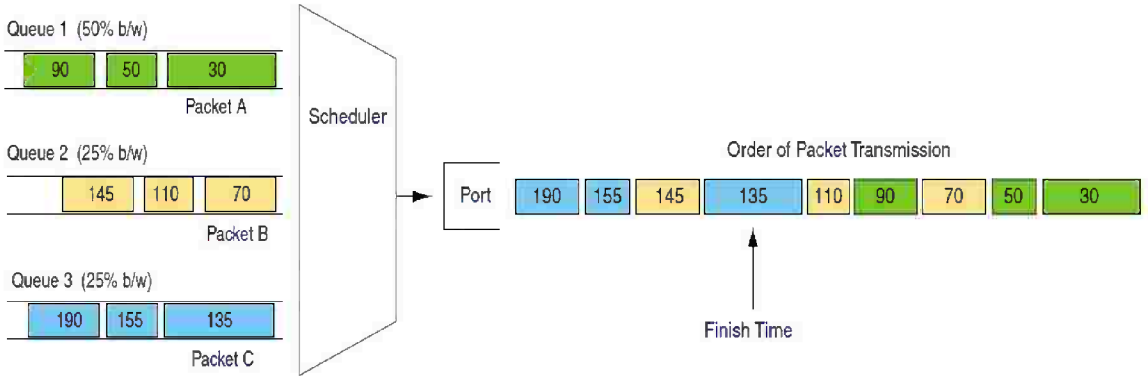

Figure 2.5 Weighted Fair Queuing (WFQ)[7].

When each packet is classified and placed into its queue, the scheduler calculates and assigns a finish time for the packet. As the WFQ scheduler services its queues, it selects the packet with the earliest (smallest) finish time as the next packet for transmission on the output port. For example, if WFQ determines that packet A has a finish time of 30, packet B has a finish time of 70, and packet C has a finish time of 135, then packet A is transmitted before packet B or packet C. In Figure 2.5 observe that the appropriate weighting of queues allows a WFQ scheduler to transmit two or more consecutive packets from the same queue[6].

2.5.1 WFQ benefits and limitations

Weighted fair queuing has two primary benefits:

- WFQ provides protection to each service class by ensuring a minimum level of output port bandwidth independent of the behavior of other service classes.
- When combined with traffic conditioning at the edges of a network, WFQ guarantees a weighted fair share of output port bandwidth to each service class with a bounded delay.

However, weighted fair queuing comes with several limitations:

- Vendor implementations of WFQ are implemented in software, not hardware. This limits the application of WFQ to low-speed interfaces at the edges of the network.
- WFQ implements a complex algorithm that requires the maintenance of a significant amount of per-service class state and iterative scans of state on each packet arrival and departure.Computational complexity impacts the scalability of WFQ when attempting to support a large number of service classes on high-speed interfaces.
- Finally, even though the guaranteed delay bounds supported by WFQ may be better than for other queue scheduling disciplines, the delay bounds can still be quite large.

2.5.2 WFQ implementations and applications

WFQ is deployed at the edges of the network to provide a fair distribution of bandwidth among a number of different service classes. WFQ can generally be configured to support a range of behaviors:

- WFQ can be configured to classify packets into a relatively large number of queues using a hash function that is calculated using the source/destination address pair, the source/destination UDP/TCP port numbers, and the IP ToS byte.
- WFQ can be configured to allow the system to schedule a limited number of queues that carry aggregated traffic flows. For this configuration option, the system uses QoS policy or the three low-order IP precedence bits in the IP ToS byte to assign packets to queues. Each of the queues is allocated a different percentage of output port bandwidth based on the weight that the system calculates for each of the service classes. This approach allows the system to allocate different amounts of bandwidth to each queue based on the QoS policy group or to allocate increasing amounts of bandwidth to each queue as the IP precedence increases[7].

2.5.3 WFQ router configuration

Dynamically creates queues based on traffic flows. Traffic flows are identified with a hash value generated using the following header fields:

- Source and Destination IP address.
- Source and Destination TCP (or UDP) port numbers.
- IP Protocol number.
- Type of Service value (IP Precedence orDifferentiated Service Code Point DSCP).

Traffics of the same flow are placed in the same flow queue. By default, a maximum of 256 queues can exist, though this can be increased to 4096. If the priority (based on the ToS field) of all packets are the same, bandwidth is divided equally among all queues. This results in low-traffic flows incurring a minimal amount of delay, while high-traffic flows may experience latency. Packets with a higher priority are scheduled before lower-priority packets arriving at the same time. This is accomplished by assigning a sequence number to each arriving packet, which is calculated from the last sequence number multiplied by an inverse weight (based on the ToS field). In other words a higher ToS value results in a lower sequence number, and the higher-priority packet will be serviced first.

2.6 Custom Queuing (CQ)

The primary purpose of custom queuing (CQ) is proportional sharing of the available network bandwidth among applications or organizations to avoid congestions in the network. CQ reserves the guaranteed bandwidth amount at a possible congestion point in the form of a constant ratio of bandwidth assurance, while the rest of the available bandwidth is left for other network traffic. Traffic management is performed by the allocating procedure according to the free space in the queue for each class of packets. It then starts the serving queue process in a circular manner. Furthermore, in each of the internal queue classes (up to 17), the amount of bandwidth, necessary for individual packets' transmission at the output connection is always calculated[9].

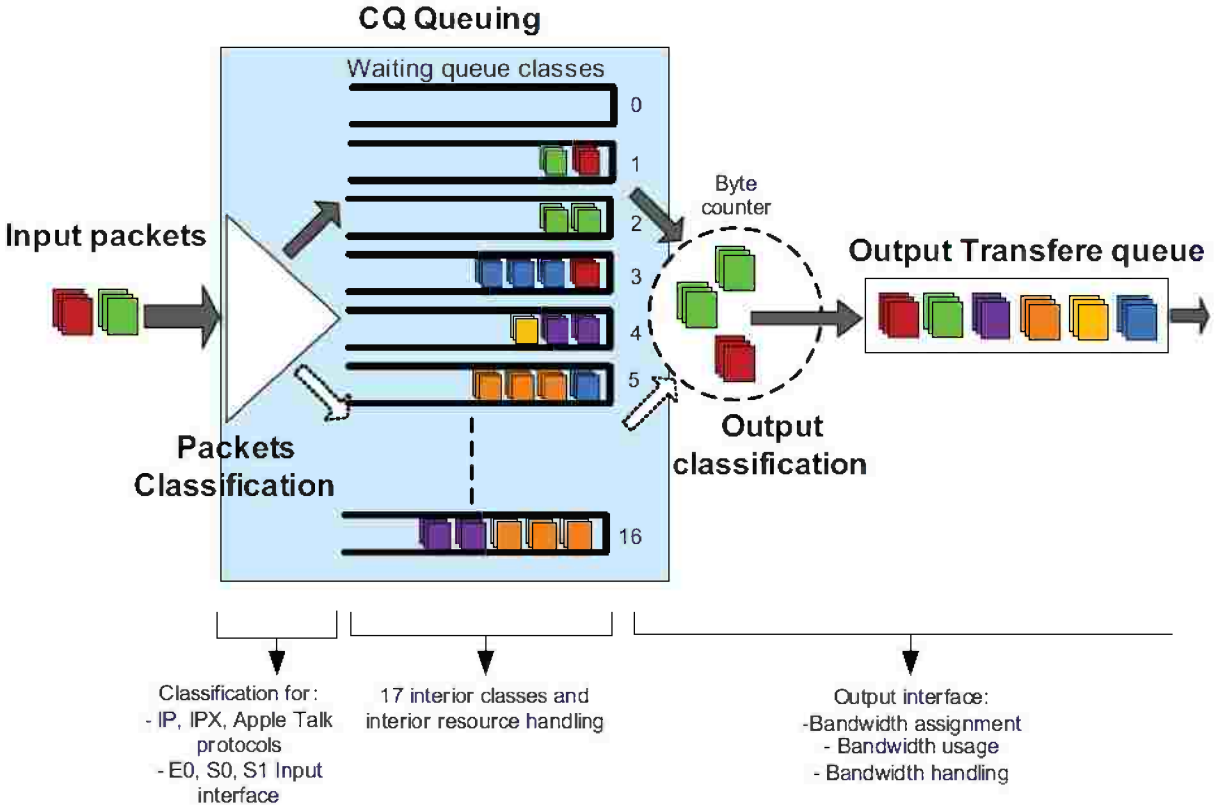

Figure 2.6 Custom queuing (CQ)[9].

Custom queuing algorithm places the packets in one of the seventeen internal waiting queues, where the queue with the index 0 is reserved for system messages, such as the so-called "keep-alive" messages and various warning messages. Queues discharging procedure is executed according to packets' weights. This means that the message with a higher priority has a smaller "weight" compared to the message with a lower priority (larger weight). The routers this way manage queues from 1 to 16 in a circular mode. Such functionality ensures an order, where no application (or group of applications) can take up more than a predetermined level of the overall bandwidth capacity, even in situations, where the link is over 90% full. CQ mechanism is statically configured[9].

In CQ, packets are first classified into various service classes (for example, real-time, interactive, and file transfer) and then assigned to a queue that is specifically dedicated to that service class. Each of the queues is serviced in a round-robin order. Similar to strict PQ and FQ, empty queues are skipped. Custom queuing is also referred to as class-based queuing (CBQ) or Weighted Round Robin (WRR) Queuing.

CQ queuing supports the allocation of different amounts of bandwidth to different service class by either:

- Allowing higher-bandwidth queues to send more than a single packet each time that it visited during a service round.
- Allowing each queue to send only a single packet each time that it is visited, but to visit higher-bandwidth queues multiple times in a single service round.

In Figure 2.7, the real-time traffic queue is allocated 25 percent of the output port bandwidth, the interactive traffic queue is allocated 25 percent of the output port bandwidth, and the file transfer traffic queue is allocated 50 percent of the output port bandwidth. CQ supports this weighted bandwidth allocation by visiting the file transfer queue two times during each service round.

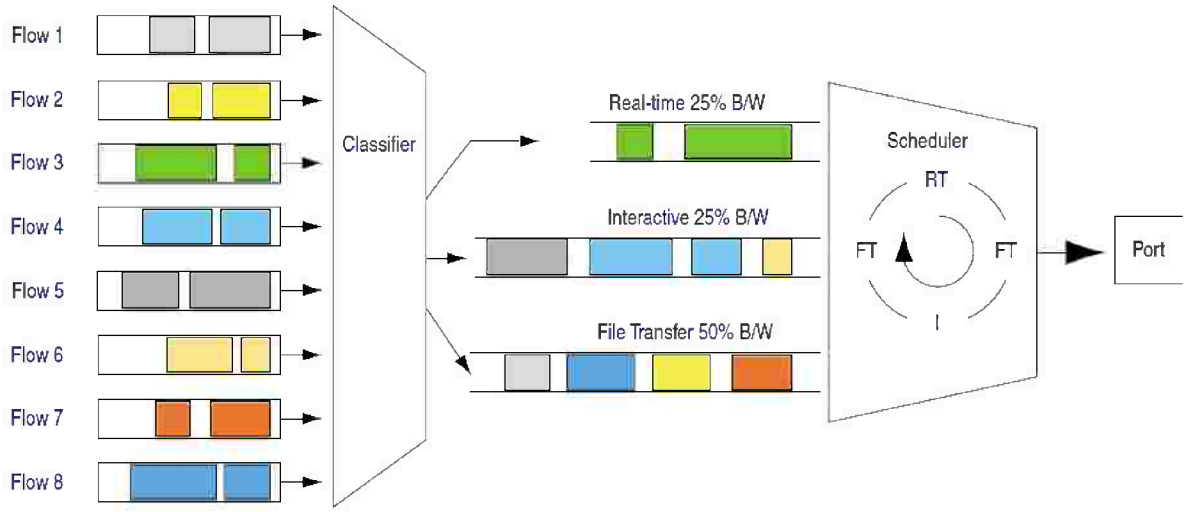

Figure 2.7 Custom Queuing (CQ)[7].

To regulate the amount of network resources allocated to each service class, a number of parameters can be tuned to control the desired behavior of each queue:

- The amount of delay experienced by packets in a given queue is determined by a combination of the rate that packets are placed into the queue, the depth of the queue, the amount of traffic removed from the queue at each service round, and the number of other service classes (queues) configured on the output port.
- The amount of jitter experienced by packets in a given queue is determined by the variability of the delay in the queue, the variability of delay in all of the other queues, and the variability of the interval between service rounds.
- The amount of packet loss experienced by each queue is determined by a combination of the rate that packets are placed into the queue, the depth of the queue, the aggressiveness of the RED profiles configured for the queue, and the amount of traffic removed from the queue at each service round. The fill rate can be controlled by performing traffic conditioning at some upstream point in the network.

Custom Queuing (CQ) or Weighted round robin (WRR) is the foundation for a class of queue scheduling disciplines that are designed to address the limitations of the FQ and PQ models.

- CQ addresses the limitations of the FQ model by supporting flows with significantly different bandwidth requirements. With CQ, each queue can be assigned a different percentage of the output port's bandwidth.
- CQ addresses the limitations of the strict PQ model by ensuring that lower-priority queues are not denied access to buffer space and output port bandwidth. With CQ, at least one packet is removed from each queue during each service round[7].

2.6.1 CQ benefits and limitations

Custom queuing includes the following benefits:
CQ queuing can be implemented in hardware, so it can be applied to high-speed interfaces in both the core and at the edges of the network.

- CQ queuing provides coarse control over the percentage of output port bandwidth allocated to each service class.
- CQ queuing ensures that all service classes have access to at least some configured amount of network bandwidth to avoid bandwidth starvation.
- CQ queuing provides an efficient mechanism to support the delivery of differentiated service classes to a reasonable number of highly aggregated traffic flows.
- Classification of traffic by service class provides more equitable management and more stability for network applications than the use of priorities or preferences. For example, if you assign real-time traffic strict priority over file-transfer traffic, then an excessive amount of real-time traffic can eliminate all file-transfer traffic from your network. CQ is based on the belief that resource reduction is a better mechanism to control congestion than resource denial. Resource denial not only blocks all traffic from lower-priority service classes but also obstructs all signaling regarding the denial of resources. As a result, TCP applications and externally clocked UDP applications are unable to correctly adapt their transmission rates to respond to the denial of network resources.

The primary limitation of weighted round-robin queuing is that it provides the correct percentage of bandwidth to each service class only if all of the packets in all of the queues are the same size or when the mean packet size is known in advance. For example, assume that you are using CQ to service four queues that are assigned the following percentages of output port bandwidth: queue 1 is allocated 40 percent, queue 2 is allocated 30 percent, queue 3 is allocated 20 percent, and queue 4 is allocated 10 percent. Assume also that all of the packets in all of the queues are 100 bytes.

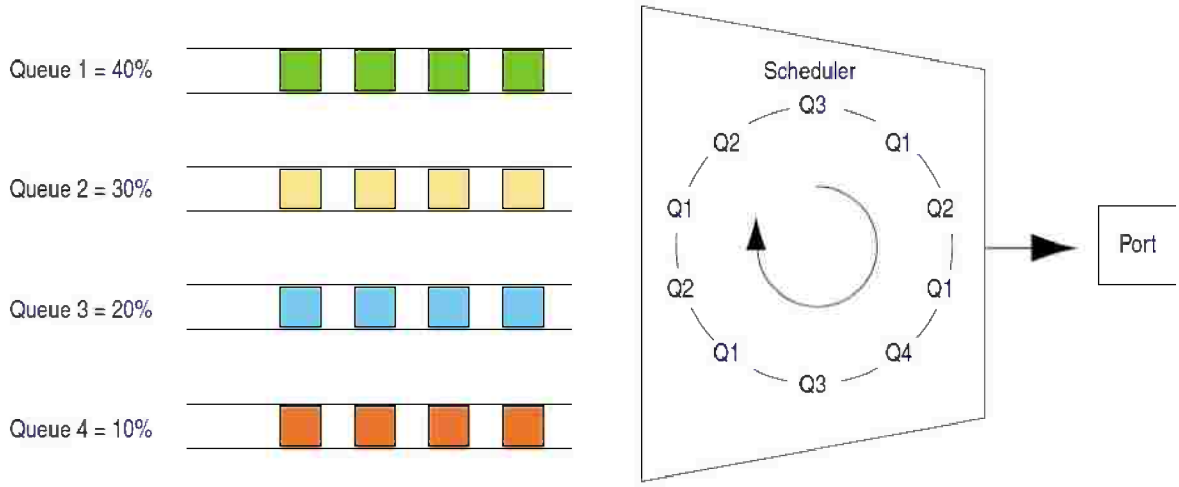

Figure 2.8CQ is Fair with Fixed-length Packets[7].

At the end of a single service round, queue 1 transmits 4 packets (400 bytes), queue 2 transmits 3 packets (300 bytes), queue 3 transmits 2 packets (200 bytes), and queue 4 transmits one packet (100 bytes). Since a total of 1000 bytes are transmitted during the service round, queue 1 receives 40 percent of the bandwidth, queue 2 receives 30 percent of the bandwidth, queue 3 receives 20 percent of the bandwidth, and queue 4 receives 10 percent of the bandwidth. In this example, CQ provides a perfect distribution of output port bandwidth. This behavior is similar to what you would expect to find in a fixed-length, cell-based ATM network, because all packets are the same size.Figure 2.8.

However, if one service class contains a larger average packet size than another service class, the service class with the larger average packet size obtains more than its configured share of output port bandwidth. Assume that the CQ scheduler is configured exactly as in the previous example. However, all of the packets in queue 1 have an average size of 100 bytes, all of the packets in queue 2 have an average size of 200 bytes, all of the packets in queue 3 have an average packet size of 300 bytes, and all of the packets in queue 4 have an average packet size of 400 bytes[7].

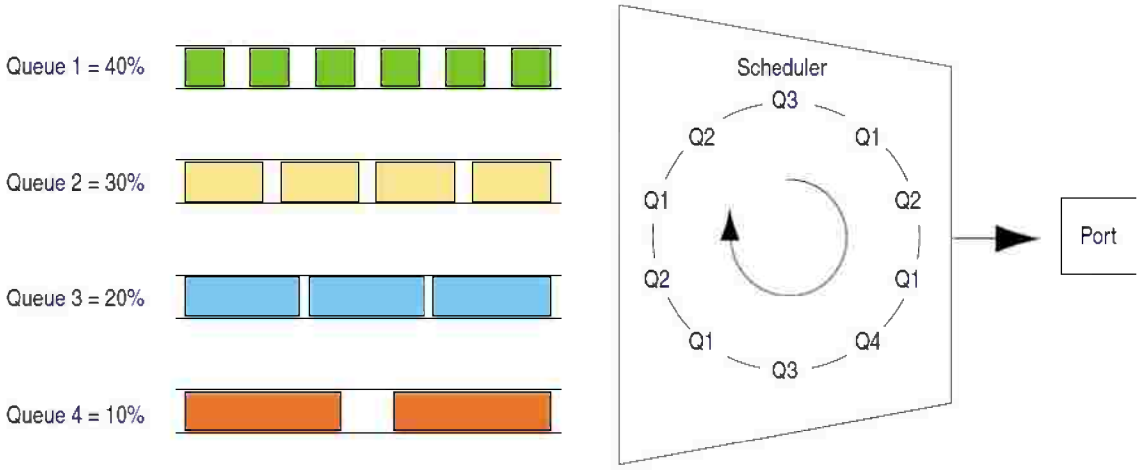

Figure 2.9 CQ is Unfair with Variable-length Packets[7].

Assuming these average packet sizes, at the end of a single service round, queue 1 transmits 4 packets (400 bytes), queue 2 transmits 3 packets (600 bytes), queue 3 transmits 2 packets (600 bytes), and queue 4 transmits 1 packet (400 bytes). Since a total of 2000 bytes are transmitted during the service round, queue 1 receives 20 percent of the bandwidth, queue 2 receives 30 percent of the bandwidth, queue 3 receives 30 percent of the bandwidth, and queue 4 receives 20 percent of the bandwidth. When faced with variable length packets, CQ does not support the configured distribution of output port bandwidth.Figure 2.9.

2.6.2 CQ implementations and applications

Because the CQ scheduling discipline can be implemented in hardware, it can be deployed in both the core and at the edges of the network to arbitrate the weighted distribution of output port bandwidth among a fixed number of service classes. CQ effectively overcomes the limitations of FQ by scheduling service classes that have different bandwidth requirements. CQ also overcomes the limitations of strict PQ by ensuring that lower-priority queues are not bandwidth-starved. However, CQ's inability to support the precise allocation of bandwidth when scheduling variable-length packets is a critical limitation that needs to be addressed[7].

2.7 Deficit Weighted Round Robin (DWRR)

DWRR queuing was proposed by M. Shreedhar and G. Varghese in 1995. DWRR is the basis for a class of queue scheduling disciplines that are designed to address the limitations of the CQ and WFQ models.

- DWRR addresses the limitations of the CQ model by accurately supporting the weighted fair distribution of bandwidth when servicing queues that contain variable-length packets.
- DWRR addresses the limitations of the WFQ model by defining a scheduling discipline that has lower computational complexity and that can be implemented in hardware. This allows DWRR to support the arbitration of output port bandwidth on high-speed interfaces in both the core and at the edges of the network.

2.7.1 DWRR algorithm

In the classic DWRR algorithm, the scheduler visits each non-empty queue and determines the number of bytes in the packet at the head of the queue. The variable DeficitCounter is incremented by the value quantum. If the size of the packet at the head of the queue is greater than the variable DeficitCounter, then the scheduler moves on to service the next queue. If the size of the packet at the head of the queue is less than or equal to the variable DeficitCouner, then the variable DeficitCounter is reduced by the number of bytes in the packet and the packet is transmitted on the output port. The scheduler continues to dequeue packets and decrement the variable DeficitCounter by the size of the transmitted packet until either the size of the packet at the head of the queue is greater than the variable DeficitCounter or the queue is empty. If the queue is empty, the value of DeficitCounter is set to zero. When this occurs, the scheduler moves on to service the next non-empty queue. Figure 2.10

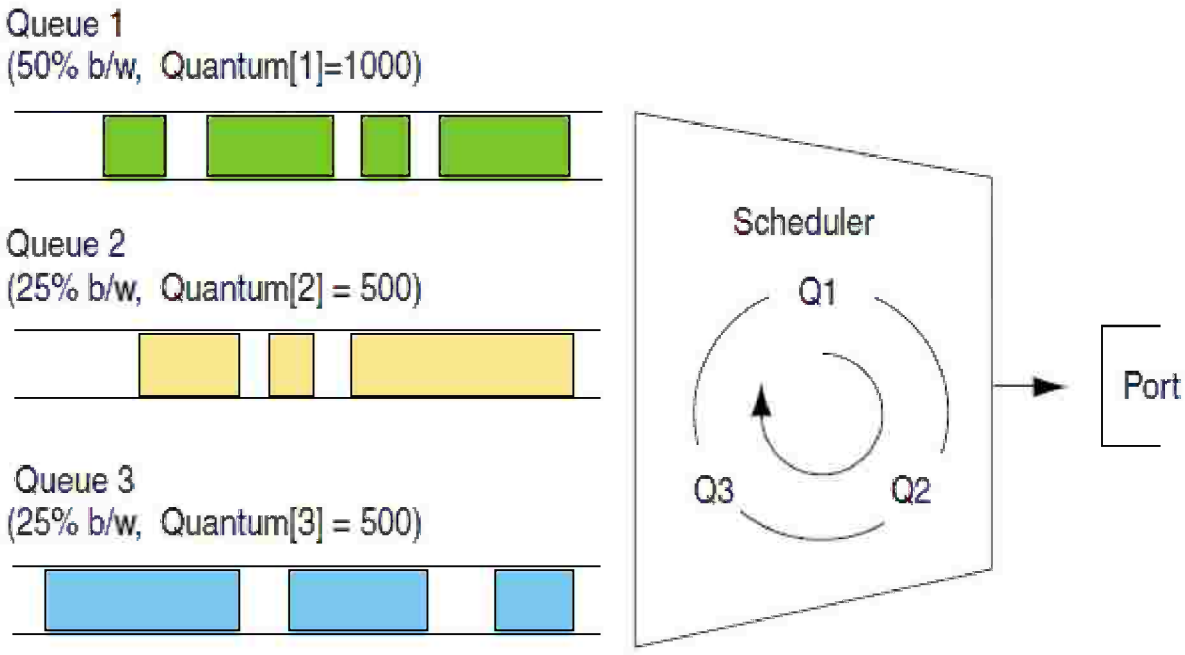

Figure 2.10 Deficit Weighted Round Robin Queuing[7].

2.7.2 DWRR example

For this example, assume that DWRR scheduling is enabled on an output port that is configured to support 3 queues. Queue 1 is allocated 50 percent of the bandwidth, while queue 2 and queue 3 are each allocated 25 percent of output port bandwidth. Initially, the array variable DeficitCounter is initialized to 0. Assume that the round robin pointer points to queue 1, which is at the top of the ActiveList. Before the DWRR scheduling discipline begins to service queue 1, Quantum[1] = 1000 is added to DeficitCounter[1], giving it a value of 1000[7].Figure 2.11.

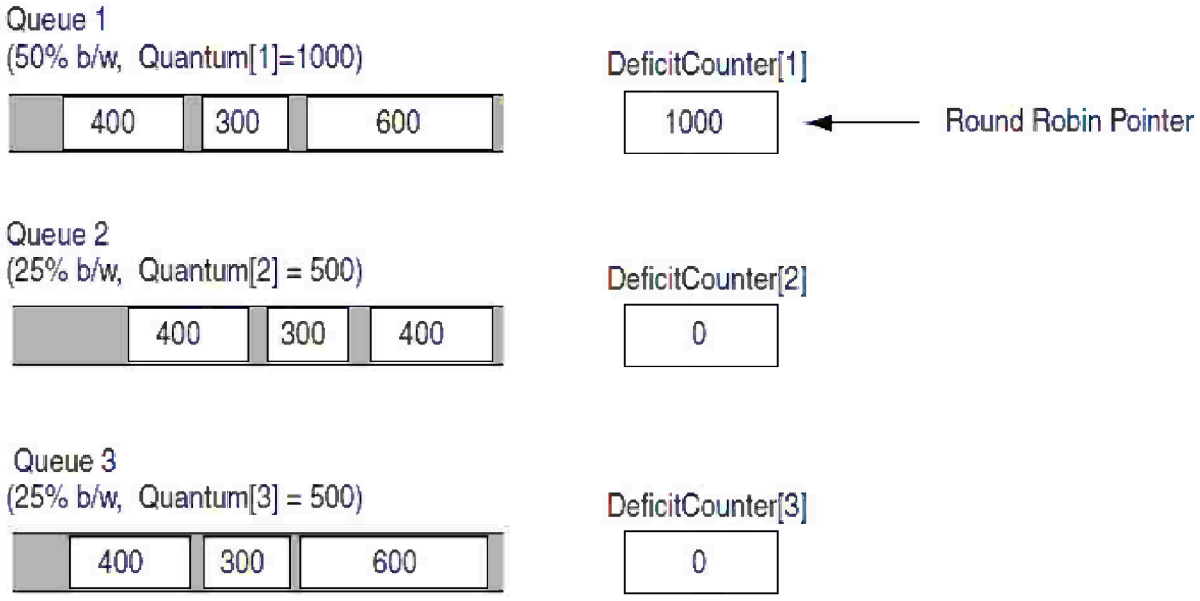

Figure 2.11 DWRR Example Round 1 with Round Robin Pointer = 1[7].

Because the 600-byte packet at the head of queue 1 is smaller than the value of DeficitCounter[1] = 1000, the 600-byte packet is transmitted. This causes the DeficitCounter[1] to be decremented by 600 bytes, resulting in a new value of 400. Now, since the 300-byte packet at the head of queue 1 is smaller than DeficitCounter[1] = 400, the 300-byte packet is also transmitted, and this causes DeficitCounter[1] to be decremented by 300 bytes, creating a new value of 100. Because the 400-byte packet at the head of queue 1 is larger than the value of DeficitCounter[1] = 100, the 400-byte packet cannot be transmitted. This causes the round robin pointer to point to queue 2, which is now at the top of the ActiveList. Figure 2.12.

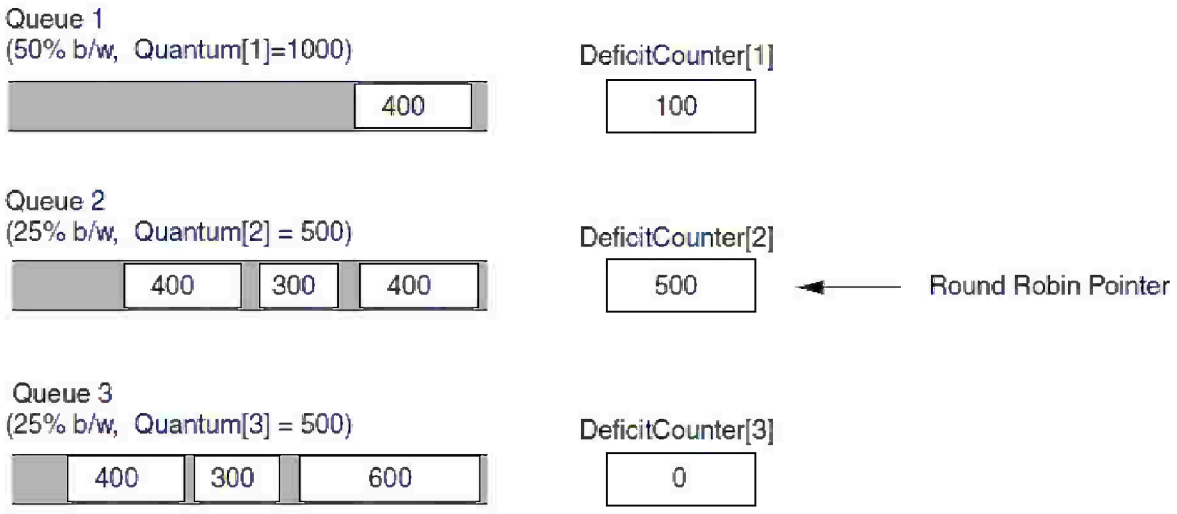

Figure 2.12 DWRR Example Round 1 with Round Robin Pointer = 2[7].

Before the DWRR scheduling discipline starts to service queue 2, Quantum[2] = 500 is added to DeficitCounter[2] giving it a value of 500. Since the 400-byte packet at the head of queue 2 is smaller than the value of DeficitCounter[2] = 500, the 400-byte packet is transmitted. This causes DeficitCounter[2] be decremented by 400 bytes and thus have a new value of 100. Because the 300-byte packet at the head of queue 2 is larger than the value of DeficitCounter[2] = 100, the 300-byte packet cannot be transmitted. This causes the round robin pointer to point to queue 3, which is now at the top of the ActiveList.Figure 2.13.

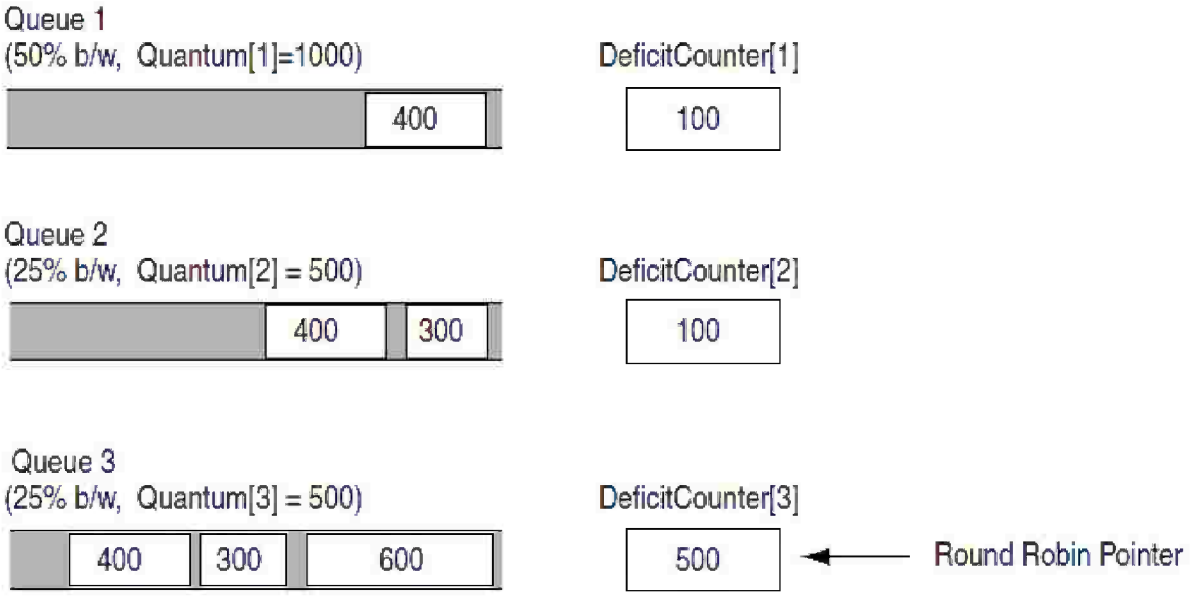

Figure 2.13 DWRR Example — Round 1 with Round Robin Pointer = 3[7].

Before the DWRR scheduling discipline starts to service queue 3, Quantum[3] = 500 is added to DeficitCounter[3], giving it a value of 500. Since the 600-byte packet at the head of queue 2 is larger than the value of DeficitCounter[3] = 500, the 600-byte packet cannot be transmitted. This causes the round robin pointer to point to queue 1, which is now at the top of the ActiveList.Figure 2.14.

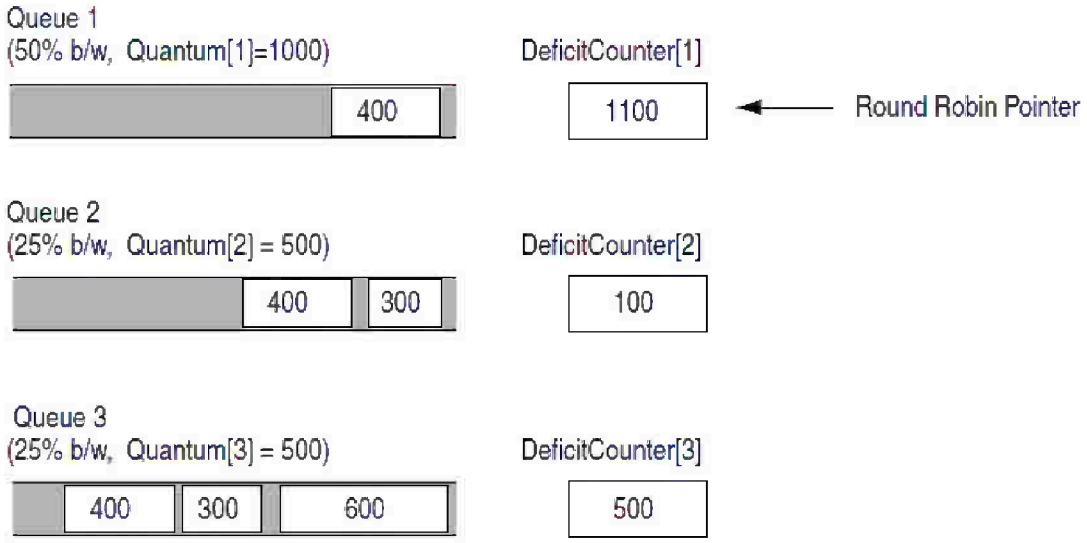

Figure 2.14 DWRR Example — Round 2 with Round Robin Pointer = 1[7].

Before the DWRR scheduling discipline starts to service queue 1, Quantum[1] = 1000 is added to DeficitCounter[1] giving it a value of 1100. Since the 400-byte packet at the head of queue 1 is smaller than the value of DeficitCounter[1] = 1100, the 400-byte packet is transmitted. This causes DeficitCounter[1] be decremented by 400 bytes and have a new value of 700. Now queue 1 is empty. This causes DeficitCounter[1] to be set to zero, queue 1 to be removed from the ActiveList, and the round robin pointer to point to queue 2, which is now at the top of theActiveList.Figure 2.15.

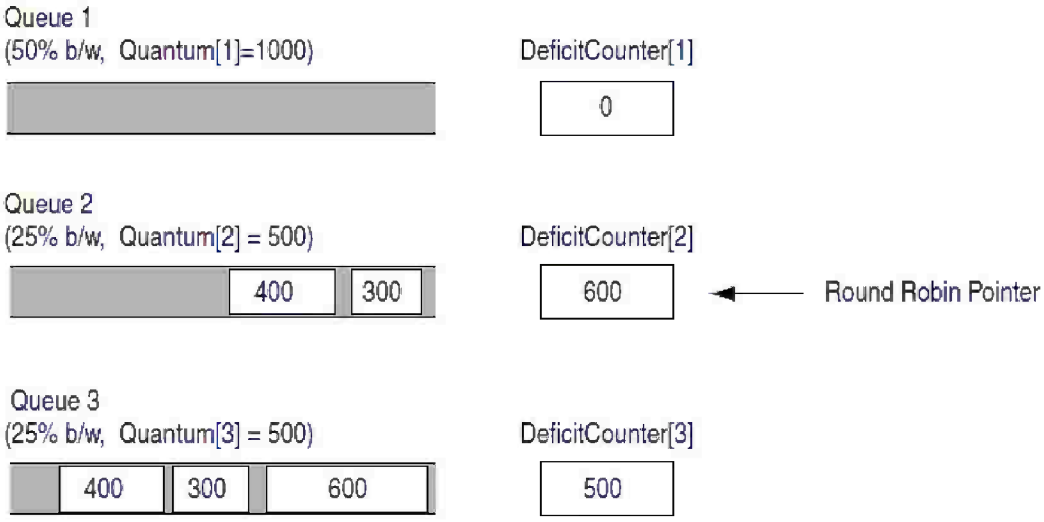

Figure 2.15 DWRR Example — Round 2 with Round Robin Pointer = 2[7].

Before the DWRR scheduling discipline starts to service queue 2, Quantum[2] = 500 is added to DeficitCounter[2], giving it a value of 600. Since the 300-byte packet at the head of queue 2 is smaller than the value of DeficitCounter[2] = 600, the 300-byte packet is transmitted. This causes DeficitCounter[2] to be decremented by 300 bytes, to a new value of 300. Since the 400-byte packet at the head of queue 2 is larger than the value of DeficitCounter[2] = 300, the 400-byte packet cannot be transmitted. This causes the round robin pointer to point to queue 3, which is now at the top of the ActiveList.Figure 2.16.

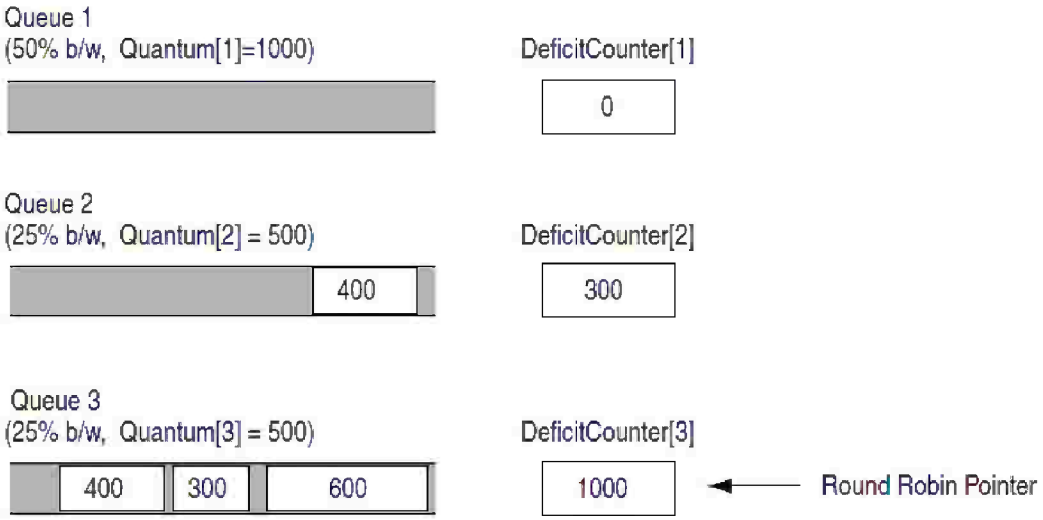

Figure 2.16 DWRR Example — Round 2 with Round Robin Pointer = 3

Before the DWRR scheduling discipline starts to service queue 3, Quantum[3] = 500 is added to DeficitCounter[3] giving it a value of 1000. Since the 600-byte packet at the head of queue 3 is smaller than the value of DeficitCounter[3] = 1000, the 600-byte packet is transmitted. This causes DeficitCounter[3] be decremented by 600 bytes and have a new value of 400. Since the 300-byte packet at the head of queue 3 is smaller than the value of DeficitCounter[3] = 400, the 300-byte packet is transmitted, and this causes DeficitCounter[3] to be decremented by 300 bytes, to a new value of 100. Since the 400-byte packet at the head of queue 3 is larger than the value of DeficitCounter[3] = 100, the 400-byte packet cannot be transmitted. This causes the round robin pointer to point to queue 2, which is now at the top of the ActiveList.Figure 2.17.

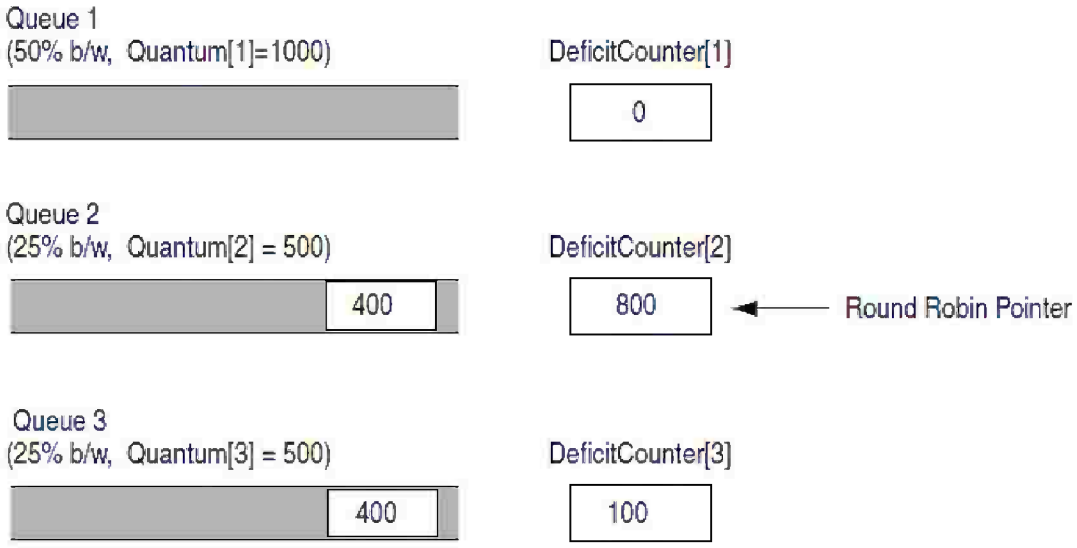

Figure 2.17 DWRR Example — Round 3 with Round Robin Pointer = 2[7].

Before the DWRR scheduling discipline starts to service queue 2, Quantum[2] = 500 is added to DeficitCounter[2], giving it a value of 800. At this point, the DWRR scheduling discipline continues to service queues by providing the user-configured percentage of output port bandwidth to each service class[7].

2.7.3 DWRR benefits and limitations

The benefits of DWRR queuing are that it:

- Provides protection among different flows, so that a poorly behaved service class in one queue cannot impact the performance provided to other service classes assigned to other queues on the same output port.
- Overcomes the limitations of WRR by providing precise controls over the percentage of output port bandwidth allocated to each service class when forwarding variable-length packets;
- Overcomes the limitations of strict PQ by ensuring that all service classes have access to at least some configured amount of output port bandwidth to avoid bandwidth starvation; and
- Implements a relatively simple and inexpensive algorithm, from a computational perspective, that does not require the maintenance of a significant amount of per-service class state.

As with other models, DWRR queuing has limitations:

- Highly aggregated service classes mean that a misbehaving flow within a service class can impact the performance of other flows within the same service class. However, in the core of a large IP network, routers are required to schedule aggregate flows, because the large number of individual flows makes it impractical to support per-flow queue scheduling disciplines.

- DWRR does not provide end-to-end delay guarantees as precise as other queue scheduling disciplines do.
- DWRR may not be as accurate as other queue scheduling disciplines. However, over high-speed links, the accuracy of bandwidth allocation is not as critical as over low-speed links.

2.7.4 DWRR implementations and applications

Because the DWRR queue scheduling discipline can be implemented in hardware, it can be deployed in both the core and at the edges of the network to arbitrate the weighted distribution of output port bandwidth among a fixed number of service classes. DWRR provides all of the benefits of WRR, while also addressing the limitations WRR by supporting the accurate allocation of bandwidth when scheduling variable-length packets[7].