QoS Overview

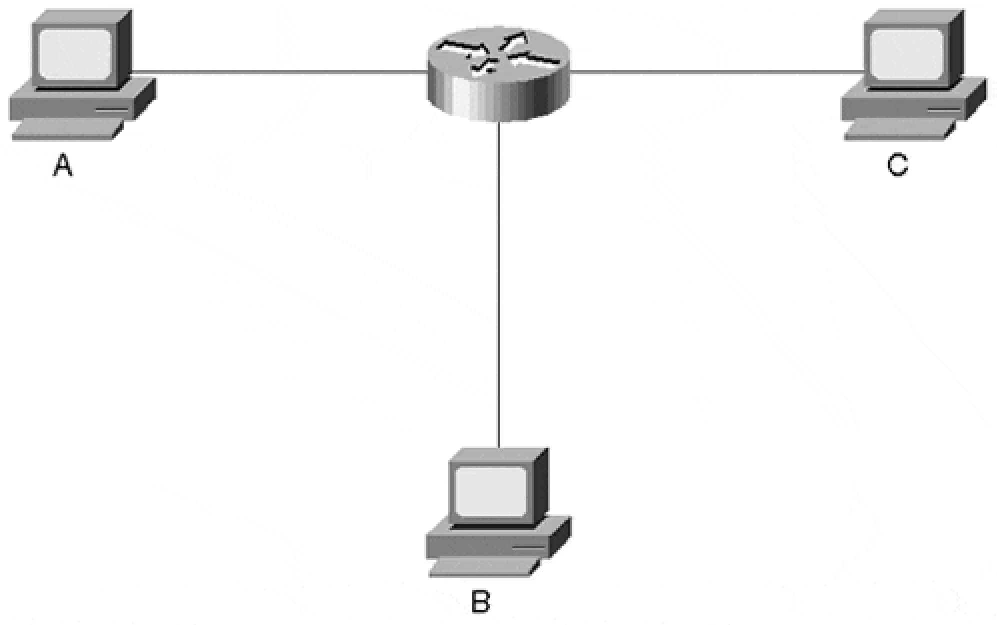
Consider a basic network of two computers connected by a single link. In a perfect world, these computers could send and receive as much data as they want without worrying about transit delay; the only thing bounding the number of file transfers, voice calls, and video sessions between them is the processing power of the systems themselves. They have a dedicated link with infinite bandwidth!

Real networks consist of shared links with limited bandwidth and some (measurable) amount of delay. Networks able to handle the full output rate of every connected node are rarely economical; instead, network designers attempt to balance costs with the average and peak rates of traffic they expect. Devices within the network, such as routers, employ various buffering and queuing mechanisms to handle temporary bursts above the average.

In previous chapters, we looked at the basic packet queuing mechanisms used in IOS, which is first-in, first-out (FIFO) in most cases. As long as there's sufficient link bandwidth and switching capacity, FIFO can meet all the service requirements for all traffic types.

To see why FIFO isn't always enough, though, take a look at the simple network shown in Figure 8-1.

Figure 8-1. QoS in a Simple Network



Let's assume all the links illustrated are 10 Mbps, and the router can switch at 50 Mbps. If the three hosts transmit an average of 700 Kb/s with an occasional burst of 5 Mbps, FIFO queuing on the router performs fine; the router's interfaces can always keep their transmit queues drained.

Increasing the maximum burst from 5 Mbps to 10 Mbps, however, quickly shows FIFO's weakness queuing. Switching capacity is not a problem—the router can handle up to five Ethernet interfaces going full speed. But if two of the hosts, A and B, try to send at 10 Mbps toward C at the same time, the total data arrival rate at the router (20 Mbps) exceeds the capacity of the output interface (10 Mbps).

Some number of the packets toward C need to buffered and queued on the transmit queue of the outbound interface—and anytime queues begin to grow, it's probable that packets waiting in those queues must wait longer than their acceptable delay.

The problem becomes even more pronounced if a significant difference in link speeds exists along a path.

For example, if the link to C in Figure 8-1 is replaced with a T1 (1.544 Mbps), it only takes a burst from one host to overrun the router's interface to C. In fact, the T1 barely keeps up with the average transmission rates of A and B.

Two common QoS strategies are used to meet the service requirements of the traffic: *congestion management* and *congestion avoidance*. Congestion management assumes packets wait in queues, and attempts to minimize the effects of queuing by dequeuing packets in an order that satisfies their service levels. Congestion avoidance attempts to limit queuing itself by signaling traffic sources to slow down their transmission rates.

Congestion Management

Congestion management methods are distinguished by how they classify packets and the order in which they dequeue packets. Two general classes of congestion management exist in IOS: *platform independent* and *platform dependant*. Platform-independent methods are available on most Cisco routers, and run on the main processor, such as the RSP in a Cisco 7500. Platform-dependant methods are implemented on intelligent interface processors, and are only available on selected platforms.

Currently, two platform-dependant congestion management methods exist: Distributed Weighted Fair Queuing (DWFQ) and Modified Deficit Round Robin (MDRR). DWFQ is implemented on Cisco 7500 VIP cards, and MDRR is implemented on Cisco 12000 line cards.

Platform-independent congestion management techniques implemented in IOS include:

- Priority Queuing (PQ)
- Custom Queuing (CQ)
- Weighted Fair Queuing (WFQ)
    - Flow-Based WFQ
    - Class-Based WFQ

All these congestion management techniques, with the exception of MDRR, are applied between an interface's output hold and transmit queues.

NOTE

IOS platform-independent congestion management techniques occur between an interface's output hold and transmit queues (or transmit rings). The interface transmit queue is a first-in, first-out (FIFO) queue; when a packet is placed in the interface transmit queue, it is scheduled for transmission on the physical wire. Thus, for any type of fancy queuing (congestion management techniques) to take place, packets must pass through the output queue rather than be placed directly on the interface's transmit queue.

How does IOS decide when a packet should be placed on an interface's output queue rather than directly on its transmit queue?

- If a packet has been process switched (or it's generated by IOS itself), it's always placed on the output hold queue rather than on the interface transmit queue.
- If the interface transmit queue is full, any packets queued to the interface are placed on the output hold queue.
- Beyond this, if any packets are on an interface's transmit queue, all packets destined out that interface are placed on the output hold queue as well.

So, if the interface becomes congested, its transmit ring fills up and packets are placed on its output hold queue instead, triggering congestion management.

How many packets does it take for an interface's transmit queue to fill up? It depends on the platform and the type of interface. When any type of congestion management is configured on an interface, the transmit queue size is decreased to improve the odds of the congestion management taking effect.

Throughout this chapter, we assume congested interfaces for all examples given.

Congestion Avoidance

The objective of congestion avoidance algorithms is to prevent congestion from happening in the first place. Two general classes of congestion avoidance exist in IOS: platform independent and platform dependent. Platform-independent methods are implemented on a platform's main processor, such as the RSP on the Cisco 7500, whereas platform-dependent methods are implemented on intelligent interface processors on selected platforms. We discuss the Weighted Random Early Detection (WRED) implementation in this chapter.

< BACK   Make Note | Bookmark   CONTINUE >

Index terms contained in this section