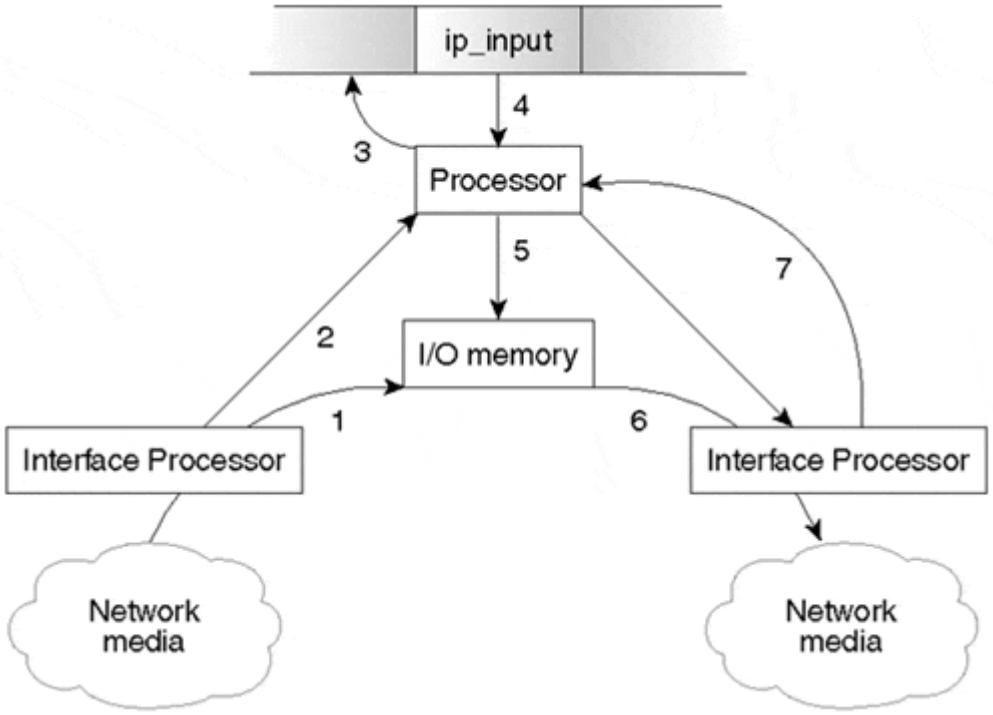
Routing 101: Process Switching

Process switching was the first switching method implemented in IOS. Basically, it uses the brute-force method to switch packets. Although process switching contains the least amount of performance optimizations and can consume large amounts of CPU time, it does have the advantage of being platform-independent, making it universally available across all Cisco IOS-based products. Process switching also provides some traffic load sharing capabilities not found in most of the other switching methods, which are discussed in detail in the section, "Traffic Load Sharing with Process Switching."

To understand just how process switching works, take a look at the steps required to process switch a packet. Figure 2-1 illustrates the process-switched path for an IP packet.

Figure 2-1. The Process Switched Path



This example begins with the network interface on the router sensing there is a packet on the wire that needs to be processed. The interface hardware receives the packet and transfers it into input/output (I/O) memory—Step 1 in Figure 2-1.

The network interface interrupts the main processor, telling it there is a received packet waiting in I/O memory that needs to be processed; this is called the *receive interrupt*. The IOS interrupt software inspects the packet's header information (encapsulation type, network layer header, and so on), determines that it is an IP packet, and places the packet on the input queue for the appropriate switching process—Step 2 in Figure 2-1. For IP packets, the switching process is named **ip_input**.

After at least one packet is in the input queue of the **ip_input** process, **ip_input** becomes eligible to run—Step 3 in Figure 2-1.

After the **ip_input** process is running (Step 4 in Figure 2-1), the actual packet-forwarding operation can begin. It is here that all the decisions are made about where to direct the received packet. In this example, **ip_input** looks in the routing table to see whether a route exists to the destination IP address. If one is found, it retrieves the address of the next hop (the next router in the path or the final destination) from the

routing table entry. It then looks into the ARP cache to retrieve the information needed to build a new Media Access Control (MAC) header for the next hop. The **ip_input** process builds a new MAC header, writing over the old one in the input packet. Finally, the packet is queued for transmission out the outbound network interface—Step 5 in Figure 2-1.

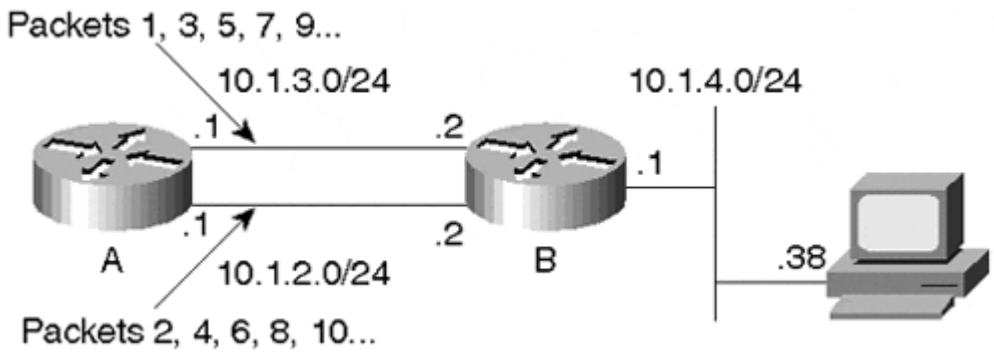
When the outbound interface hardware senses a packet waiting for output, it dequeues the packet from I/O memory and transmits it on to the network—Step 6 in Figure 2-1. After the outbound interface hardware finishes transmitting the packet, it interrupts the main processor to indicate that the packet has been transmitted. IOS then updates its outbound packet counters and frees the space in I/O memory formerly occupied by the packet. This is the final step in Figure 2-1, Step 7.

Traffic Load Sharing with Process Switching

One of the advantages of process switching is its per-packet load sharing capability. Per-packet load sharing provides a relatively simple way to route traffic over multiple links when multiple routes (paths) exist to a destination. When multiple paths exist, process-switched packets are automatically distributed among the available paths based on the routing metric (referred to as the *path cost*) assigned to each path.

The metric or cost of each path in the routing table is used to calculate a *load share* counter, which is actually used to determine which path to take. To better understand how this works, take a look at Figure 2-2.

Figure 2-2. Traffic Sharing Across Equal Cost Paths



Router A has two paths to the 10.1.4.0/24 network (as illustrated in Figure 2-2). From the output in Example 2-1, you can see these are equal-cost paths.

Example 2-1. Routing Table for Router A in Figure 2-2

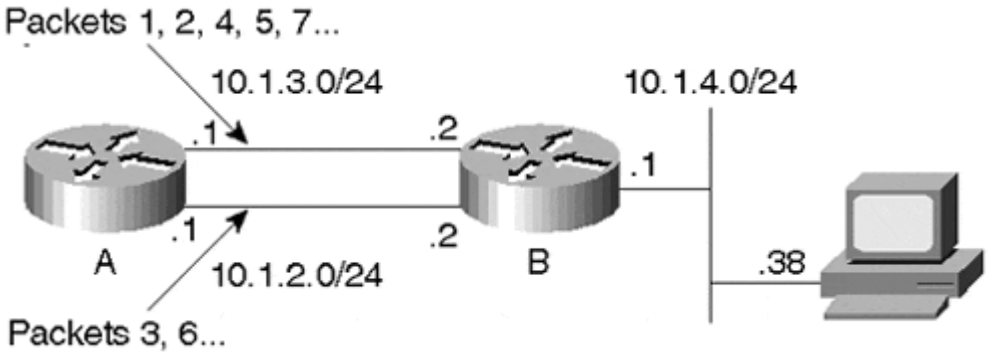
RouterA#**show ip route 10.1.4.0 255.255.255.0** Routing entry for 10.1.4.0/24 Known via "static", distance 1, metric 0 Routing Descriptor Blocks: 10.1.2.1 Route metric is 0, traffic share count is 1 * 10.1.3.1 Route metric is 0, traffic share count is 1

Note the asterisk (*) next to one of the two network paths in Example 2-1. This indicates the path that will be used for the next packet switched toward 10.1.4.0/24. The traffic-share count on both paths is 1, meaning packets are switched out each path in a round-robin fashion.

In this example, the very next packet arriving for this network will be routed to next hop 10.1.3.1. The second packet arriving will be routed to next hop 10.1.2.1, the third to 10.1.3.1, and so on (as shown by the packet numbers in Figure 2-2).

Some IP routing protocols, in particular Interior Gateway Routing Protocol (IGRP) and Enhanced IGRP (EIGRP), can install unequal cost paths in the routing table; in cases where path costs differ, the traffic-sharing algorithm changes slightly. If a link is changed in the setup in Figure 2-2, for example, so one path has about twice the bandwidth of the other, the resulting network is shown in Figure 2-3.

Figure 2-3. Traffic Sharing Across Unequal Cost Paths



The routing table would look something like Example 2-2.

Example 2-2. Routing Table for Router A in Figure 2-3

RouterA#**show ip route 10.1.4.0 255.255.255.0** Routing entry for 10.1.4.0/24 Known via "EIGRP", distance 90, metric 284600 Routing Descriptor Blocks: 10.1.2.1 Route metric is 569200, traffic share count is 1 * 10.1.3.1 Route metric is 284600, traffic share count is 2

Note the traffic share counts in this **show ip route** output. The lower-cost path through 10.1.3.1 has a traffic share count of 2; the higher-cost path through 10.1.2.1 has a traffic share count of 1. Now for every two packets switched out the higher-cost path, one packet is switched out the lower-cost path, as indicated by the packet numbers in Figure 2-3.

NOTE

Although per-packet traffic sharing is very good at balancing traffic load across multiple links, it does have one significant drawback: It can result in packets arriving out of order at the destination. This is especially true if there is a wide variation in latency among the available routes. Out-of-order packet processing can significantly degrade end-station performance.

Disadvantages of Process Switching

As noted before, a key disadvantage of process switching is its speed—or lack thereof. Process switching requires a routing table lookup for every packet; as the size of the routing table grows, so does the time required to perform a lookup (and hence the total switching time). Recursive routes require additional lookups in the routing table (to resolve the recursion), increasing the length of the lookup time.

Longer lookup times also increase the main processor utilization, an effect multiplied by the incoming packet rate. Although this effect might not be noticeable on very small networks with few routes, large networks can have hundreds or even thousands of routes. For these networks, routing table size can significantly impact main processor utilization and routing latency (the delay between the time the packet enters and exits the router).

Another major factor affecting process-switching speed is in-memory data transfer time. On some platforms, process switching requires received packets to be copied from I/O memory to another memory area before they can be switched. After the routing process finishes, the packets must be copied back to I/O memory before being transmitted. Memory data copy operations are very CPU intensive, so on these platforms, process switching can be a very poor performer.

It became quite clear to the early IOS developers that a better switching method would be required if IOS was to be viable in the world of ever-growing routed networks. To understand the solution they devised, take a look at some of the more obvious areas for improvement in process switching.

Looking back at the IP process switching example, the **ip_input** process needs three key pieces of data to switch the packet:

- **Reachability—**

  Is this destination reachable? If so, what is the IP network address of the next hop toward this destination? This data is in the routing table (also called the forwarding table).

- **Interface—**

  Which interface should this packet be transmitted on to reach its destination? This data is also stored in the routing table.

- **MAC layer header—**

  What MAC header needs to be placed on the packet to correctly address the next hop? MAC header data comes from the ARP table for IP or other mapping tables, such as the Frame Relay map table.

Because every incoming packet can be different, **ip_input** must look up these key pieces of data anew each time it switches a packet. It has to search through a potentially huge routing table for reachability and interface data, and then search another potentially large table for a MAC layer header. Wouldn't it be useful if the **ip_input** process could "remember" the results of these lookups for destinations it has already seen before? If it could keep a smaller table of the reachability/interface/MAC combinations for the most popular destinations, it could significantly reduce the lookup time for most of the incoming packets. Furthermore, because looking up the forwarding information is the most intensive part of the switching process, using a smaller table for these lookups could speed up switching enough so the entire operation could be performed during the packet receive interrupt. If the packet can be switched during the receive interrupt, it would also remove the need for in-memory data copying, saving even more time. So how can IOS keep a smaller lookup table and take advantage of these performance savings? The answer is the *Fast Cache*.

Last updated on 12/5/2001
Inside Cisco IOS Software Architecture, © 2002 Cisco Press

< BACK Make Note | Bookmark CONTINUE >

Index terms contained in this section

About Us | Advertise On InformIT | Contact Us | Legal Notice | Privacy Policy

reviewed by TRUSTe site privacy statement

© 2001 Pearson Education, Inc. InformIT Division. All rights reserved. 201 West 103rd Street, Indianapolis, IN 46290