Packet Buffer Management

In packet routing, as in any store-and-forward operation, there must be some place within the system to store the data while it is being routed on its way. The typical strategy is to create memory buffers to hold incoming packets while the switching operation is taking place. Because the capability to route packets is central to the IOS architecture, IOS contains a special component dedicated to managing just such buffers. This component is called the *buffer pool manager* (not to be confused with the memory pool manager discussed earlier). IOS uses this component to create and manage a consistent series of packet buffer pools for switching on every platform. The buffers in these pools are collectively known as the *system buffers*.

The buffer pool manager provides a convenient way to manage a set (or *pool*) of buffers of a particular size. Although it can be used to manage any type of buffer pool, the buffer pool manager is used primarily to manage pools of packet buffers, so that's what we'll focus on here.

Packet buffer pools are created using memory allocated from one of the available memory pools: for example, Processor, I/O, and so on. To create a pool, the packet buffer manager requests a block of memory from the kernel's memory pool manager and divides it into buffers. The packet buffer manager then builds a list of all the free buffers and tracks the allocation and freeing of those buffers.

Packet buffer pools can be either static or dynamic. Static pools are created with a fixed number of buffers—no additional buffers can be added to the pool. Dynamic pools are created with a particular base minimum number of buffers, called *permanent* buffers, but additional buffers can be added or removed on demand. With dynamic buffer pools, if the buffer pool manager receives a request while the pool is empty, it attempts to expand the pool and satisfy the request immediately. If it's not possible to expand the pool within the context of the request, it fails the request and expands the pool later in the **pool manager** background process.

Packet buffer pools are classified as either public or private. Public pools, as the name implies, are available for use by any system process, and private pools are created for (and known only to) a specific subset of processes that use them.

System Buffers

Every IOS system has a predefined set of public buffer pools known as the *system buffers*. These buffer pools are used for process switching packets through the platform and for creating packets that originate in the platform (such as interface keepalive packets and routing updates). IOS provides statistics about these and other buffer pools through the **show buffer** command. For an example, let's examine the **show buffer** output in Example 1-10.

Example 1-10. *show buffer* Command Output

router#>**show buffer** Buffer elements: 500 in free list (500 max allowed) 747314 hits, 0 misses, 0 created Public buffer pools: Small buffers, 104 bytes (total 50, permanent 50): 46 in free list (20 min, 150 max allowed) 530303 hits, 6 misses, 18 trims, 18 created 0 failures (0 no memory) Middle buffers, 600 bytes (total 25, permanent 25): 25 in free list (10 min, 150 max allowed) 132918 hits, 3 misses, 9 trims, 9 created 0 failures (0 no memory) Big buffers, 1524 bytes (total 50, permanent 50): 50 in free list (5 min, 150 max allowed) 47 hits, 0 misses, 0 trims, 0 created 0 failures (0 no memory) VeryBig buffers, 4520 bytes (total 10, permanent 10): 10 in free list (0 min, 100 max allowed) 26499 hits, 0 misses, 0 trims, 0 created 0 failures (0 no memory) Large buffers, 5024 bytes (total 0, permanent 0): 0 in free list (0 min, 10 max allowed) 0 hits, 0 misses, 0 trims, 0 created 0 failures (0 no memory) Huge buffers, 18024 bytes (total 0, permanent 0): 0 in free list (0 min, 4 max allowed) 0 hits, 0 misses, 0 trims, 0 created 0 failures (0 no memory) …

Notice the list of **Public buffer pools**. These are the standard system buffers for IOS. Each one has a name, such as **Small buffers**, **Middle buffers**, and so on, that identifies the particular pool. Following the pool name is the byte size of the buffers contained in that pool. Each buffer pool contains buffers of one and

only one size. The sizes vary from 104 bytes up to 18,024 bytes in order to accommodate the various maximum transmission unit (MTU) sizes encountered on different interface media.

The remainder of the fields in Example 1-10 are as follows:

- **total—**

  The total number of buffers in the pool (both allocated and free).

- **permanent—**

  The base number of buffers in the pool. For dynamic pools, the total number of buffers can fluctuate as the pool grows and shrinks. However, there will never be less than the **permanent** number of buffers in the pool.

- **in free list—**

  Indicates the number of free buffers available.

- **min—**

  The minimum number of buffers *desired* in the free list. If the number of free buffers drops below **min**, the buffer pool manager attempts to grow the pool and add more buffers until the number of free buffers is greater than **min**.

- **max allowed—**

  The maximum number of buffers *desired* in the free list. If the number of free buffers increases beyond **max allowed**, the buffer pool manager attempts to shrink the pool, deleting free buffers, until the number of free buffers drops below **max allowed**. Note that the buffer pool manager never shrinks the size of the pool below the **permanent** metric regardless of the value of **max allowed**.

- **hits—**

  The number of buffers that have been requested from the pool.

- **misses—**

  The number of times a buffer was requested from the pool when there were fewer than **min** buffers in the free list.

- **trims—**

  The number of buffers that have been deleted from the pool during pool shrinkage.

- **created—**

  The number of buffers that have been created and added to the pool during pool growth.

- **failures—**

  The number of times a buffer was requested from the pool but the request could not be satisfied. Failures can occur for two reasons:

  - A buffer request occurs during an interrupt and there are no free buffers. Pools can't grow during an interrupt.

- A buffer request occurs, there are no free buffers, and there is no more memory left in the memory pool to grow the buffer pool.

- **no memory—**

  The number of times a failure occurred because there was no available memory in the memory pool. Note that this counter is no longer used in recent versions of IOS.

To get an understanding of just how the buffer pool manager operates with these system buffers, let's step through a packet-switching scenario and monitor how the **show buffers** counters change for a particular pool. Consider the following example.

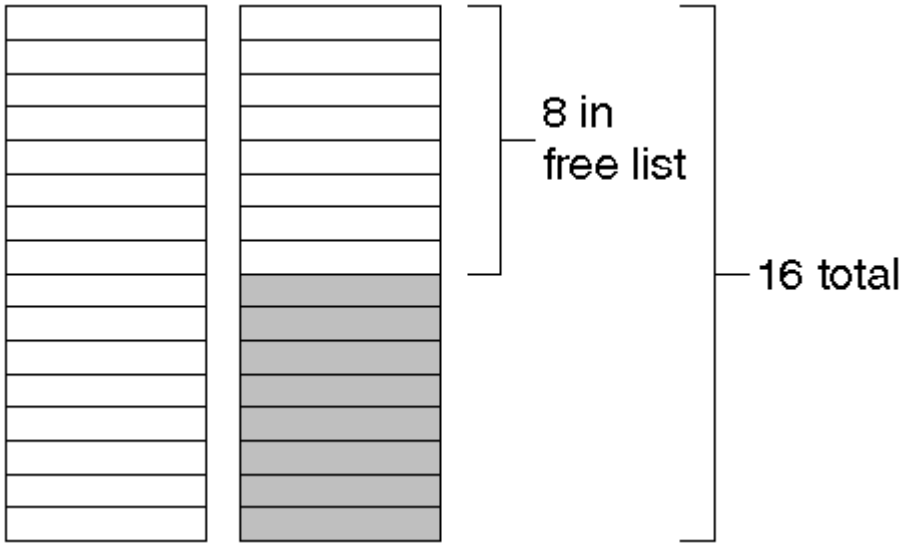
Let's start with 16 buffers in the free list, as shown in the following output and illustrated in Figure 1-7.

Figure 1-7. Beginning with an Empty Buffer


```
Small buffers, 104 bytes (total 16, permanent 16):
     16 in free list (8 min, 16 max allowed)
     0 hits, 0 misses, 0 trims, 0 created
     0 failures (0 no memory)
```

Now say IOS receives eight packets that fit nicely into the 104-byte small buffers. The packet receive software requests and receives eight buffers from the small buffer pool and copies in the packets. The free list now drops from 16 to 8, resulting in a count of eight free buffers and eight hits, as illustrated in the following output and in Figure 1-8.

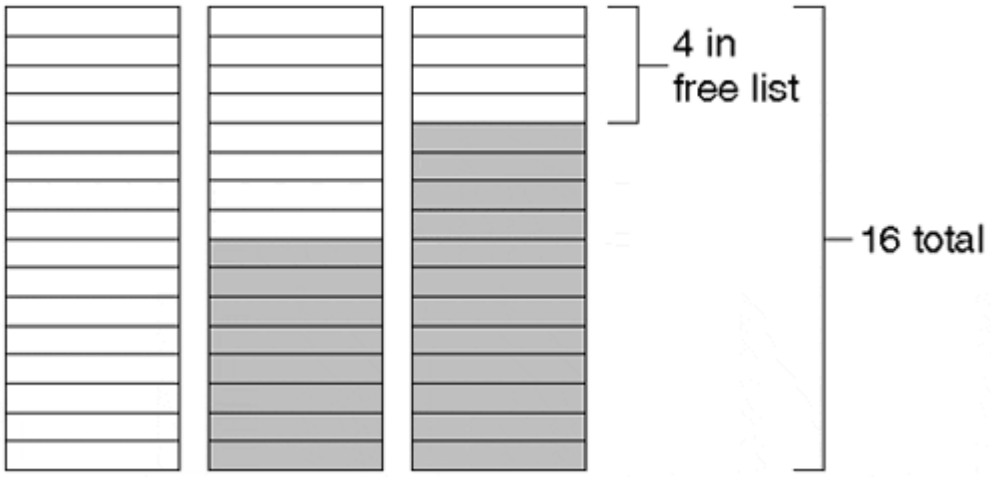
Figure 1-8. Eight Packets Received




```
Small buffers, 104 bytes (total 16, permanent 16):
     8 in free list (8 min, 16 max allowed)
     8 hits, 0 misses, 0 trims, 0 created
     0 failures (0 no memory)
```

Now IOS receives four more packets before processing any of the eight previously received. The packet receive software requests four more buffers from the **small** pool and copies in the data packets. Figure 1-9 illustrates the pool at this point.

Figure 1-9. Four More Packets Received



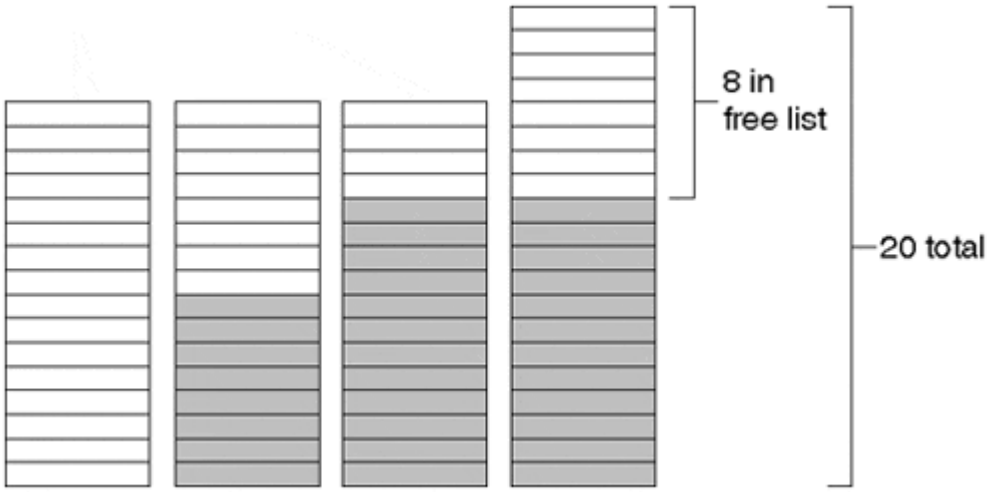
Looking at the output of **show buffers**, we see several counters have incremented. Let's inspect them to see what happened to the pool:

```
Small buffers, 104 bytes (total 16, permanent 16):
     4 in free list (8 min, 16 max allowed)
     12 hits, 4 misses, 0 trims, 0 created
     0 failures (0 no memory)
```

The number in the free list has now dropped to 4 and there are now 12 hits, reflecting the 4 new buffers requested. Why have the misses incremented to four, though? Because each time a buffer is requested from a pool where it causes the free list to drop below the minimum free buffers allowed, a miss is counted. We can see from the output that the minimum free is eight and we now have only four buffers in the free list; so, IOS counted four misses.

Because there are now fewer than **min** buffers in the free list, the buffer pool manager runs the pool manager process to grow the pool and to create more free buffers, as demonstrated in the following output and illustrated in Figure 1-10.

Figure 1-10. After Pool Manager Creates New Buffers

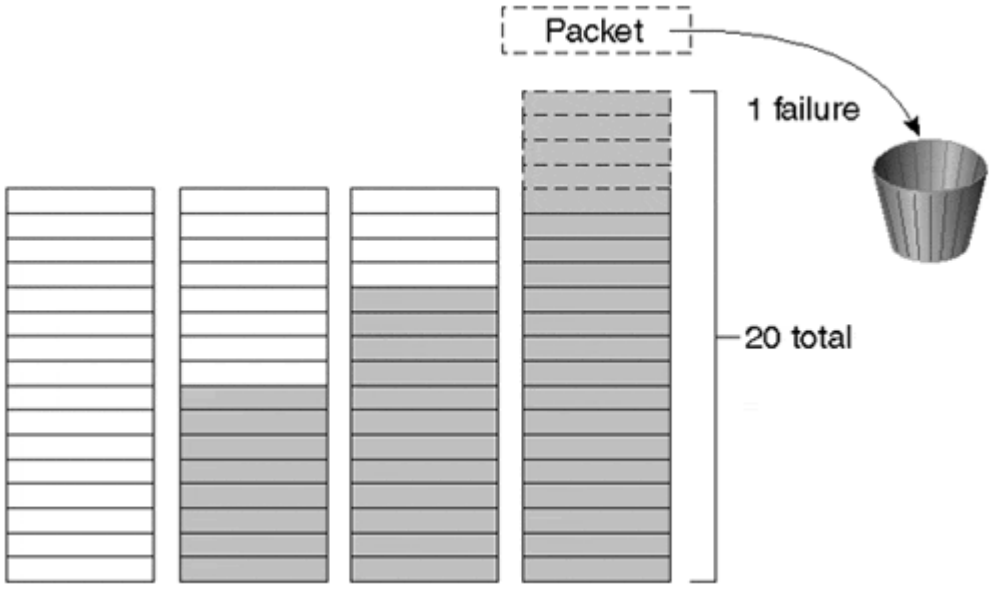



```
Small buffers, 104 bytes (total 20, permanent 16):
     8 in free list (8 min, 16 max allowed)
     12 hits, 4 misses, 0 trims, 4 created
     0 failures (0 no memory)
```

We now see that the pool manager process has created four new buffers to bring the number in the free list back up to a total of eight, the minimum number of buffers desired to be free at any time in this pool.

Now, Figure 1-11 illustrates what happens when IOS tries to request nine more packet buffers from this pool before returning any free buffers.

Figure 1-11. Storing Nine More Packets

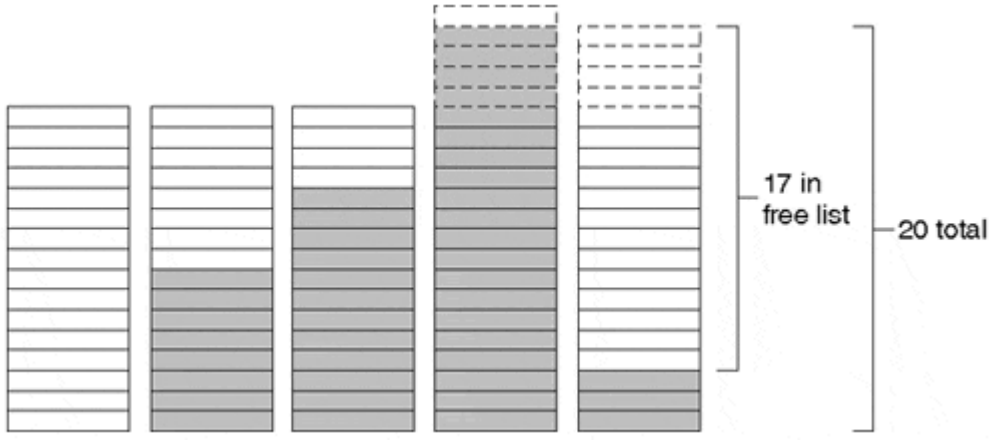


From the following output, we see that the free list has been completely exhausted—0 buffers remaining—and there have been 20 hits, 13 misses, and 1 failure. The failure accounts for the one request that overran the buffer pool.

```
Small buffers, 104 bytes (total 20, permanent 16):
     0 in free list (8 min, 16 max allowed)
     20 hits, 13 misses, 0 trims, 4 created
     1 failures (0 no memory)
```

Finally, IOS begins processing packets and returning buffers to the pool. IOS processes 17 of the 20 packets and returns the 17 free buffers, as illustrated in Figure 1-12 and the output that follows.

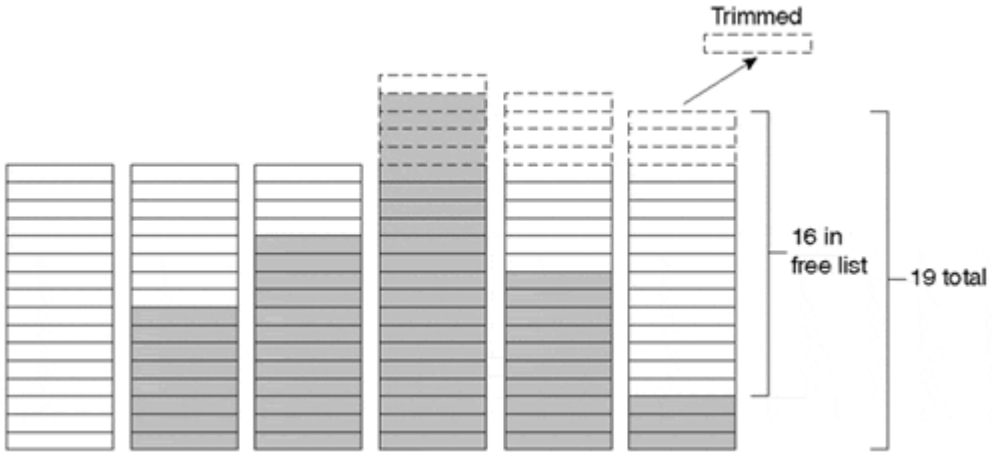
Figure 1-12. Returning 17 Buffers to the Pool




```
Small buffers, 104 bytes (total 20, permanent 16):
     17 in free list (8 min, 16 max allowed)
     20 hits, 13 misses, 0 trims, 4 created
     1 failures (0 no memory)
```

The **small** pool now has 17 in the free list. However, the desired maximum free buffers is only 16. Because the number in the free list exceeds the **max allowed**, the next time the **pool manager** process runs it shrinks the buffer pool to adjust the number of free buffers, causing one to be trimmed, as shown in Figure 1-13 and the output that follows.

Figure 1-13. Trimming the Extra Buffer




```
Small buffers, 104 bytes (total 20, permanent 16):
     17 in free list (8 min, 16 max allowed)
     20 hits, 13 misses, 1 trims, 4 created
     1 failures (0 no memory)
```

Finally, IOS processes all the remaining packets and returns the buffers to the pool. Eventually, the pool manager process trims all the remaining buffers above the permanent pool size, resulting in the following counts:

```
Small buffers, 104 bytes (total 16, permanent 16):
     16 in free list (8 min, 16 max allowed)
     20 hits, 13 misses, 4 trims, 4 created
     1 failures (0 no memory)
```


< BACK Make Note | Bookmark CONTINUE >

Index terms contained in this section