Packet Switching with the Cbus

Recall the descriptions of fast switching and process switching from Chapter 2, "Packet Switching Architecture." In both of these methods, the receiving interface hardware copies the received packet into I/O memory and interrupts the *main* processor to begin switching the packet. Whether during the interrupt or in a background process, the main processor has to do all the work of actually switching the packet; the interface hardware just receives the packet and transmits it to the media. Also, packet switching is just one of the many tasks the main processor has to do, so not all of its capacity is available for packet switching.

The Cbus and the Cbus Controller provided another processing alternative on the AGS+. Instead of IOS performing all the switching on the main processor, much of the switching could be performed by the Cbus Controller without involving the main processor at all. The Cbus Controller was capable of performing fast switching autonomously on the switching processor as long as both the receiving and the transmitting interfaces were on Cbus cards. This capability provided a new method of switching appropriately called *autonomous switching*.

Autonomous Switching

Autonomous switching worked essentially like fast switching—except on a different processor. It was triggered by a packet receive interrupt (this time to the switching processor) and used a local cache to look up forwarding information, just like fast switching. In fact, autonomous switching used the same hash table structure for its fast cache that IOS used prior to release 10.2. IOS maintained both the main fast cache and the local switching processor (Cbus) cache during process switching in the same way. Each time an entry was added or invalidated in the main cache it also was added or invalidated in the Cbus cache.

Autonomous switching did differ from fast switching in a couple ways, though. Autonomous switching didn't support as many protocols as fast switching—only IP, IPX, and bridging—and it handled cache misses differently.

Recall that in fast switching, if a packet is received and there is no fast cache entry for it, the packet is queued for process switching. In autonomous switching, however, if there was no Cbus cache entry for a packet, the packet was sent to the main processor for possible *fast* switching. If the packet was destined to another Cbus interface, the main processor could fast switch the packet while it was still in the Cbus Controller. However, if the packet was destined to a Multibus interface, the whole packet had to be copied across the slow Multibus to a system buffer, and then process switched. Although the Cbus Controller had a Multibus interface, it couldn't use it to autonomously switch packets.

Cbus Fast Packet Memory

The Cbus Controller design introduced a new strategy for IOS packet buffering that is still in use today on the Cisco 7500 series. On the AGS+, process switching and fast switching used the system buffers for packets as described in Chapter 1, "Fundamental IOS Software Architecture." However, these system buffers held a distinct disadvantage for autonomous switching: They were located in main memory on the main processor card, which made them inaccessible to Cbus interface cards.

To provide packet buffers for the Cbus interfaces, Cisco designed the Cbus Controller with its own local dedicated packet buffer memory. The memory, referred to as *MEMD*, was a fixed 512 KB region that could be addressed by both the switching processor and the Cbus interface cards. Of the total 512 KB available, the first 8 KB (called *page zero*) was set aside for control structures, leaving the rest available to be carved into packet buffers.

NOTE

You might be wondering where the name MEMD came from. Contrary to popular belief, MEMD is not the name of a type of memory but is instead the name Cisco assigned to a class of memory used on its Cbus-based routers. MEMD is really just SRAM, but it's always used for packet

buffering to and from Cbus interfaces.

The name MEMD was derived from the word *memory* (MEM) plus the alphabetic letter *D*, indicating it is the fourth in a series of Cbus memory classes. There were also other Cbus memory classes used on the AGS+. For example, *MEMA* referred to memory used to hold the autonomous cache and the global pointers used by the switching processor.

What's most interesting about MEMD is how it was logically organized. Like the system buffers, MEMD was divided into buffers of varying sizes to accommodate different sized packets. Unlike the system buffers, MEMD buffer sizes were determined by the actual MTU sizes of the Cbus interfaces present on the router—not by a predetermined set of sizes.

The buffers were divided into a maximum of four pools, and then they were allocated to the interfaces from those pools. Within each pool, all buffers were the same size. Due to the limited size of MEMD and its control structures in page zero, the maximum number of packet buffers that could be defined was limited to 470 regardless of the number of Cbus interfaces present.

The switching processor maintained two queues in MEMD's page zero for each Cbus interface: a *receive queue* and a *transmit queue*. These queues were similar in function to the input queues and the output queues maintained by IOS. When a Cbus interface detected an incoming packet, it allocated a MEMD buffer from the appropriate pool, copied the packet data into the buffer, and placed the buffer on its receive queue to await switching by the switching processor. If the packet was autonomously switched, the switching processor switched it and placed the modified packet on the transmit queue of the destination Cbus interface. If the packet was fast switched, the main processor switched it and directed the switching processor to place it on the appropriate Cbus transmit queue. For packets that had to be process switched, the switching processor copied the packet to the main processor memory over the Multibus and returned the MEMD buffer to its original pool.

The switching processor also maintained counters and limit values in MEMD for each of the receive queues and the transmit queues. Each receive queue had a *receive queue limit* (RQL) and each transmit queue had a *transmit queue limit* (TQL) to prevent a single interface from hoarding all the MEMD buffers.

The RQL value determined the maximum number of packet buffers that could be held on the receive queue at any time. When a receive queue contained the RQL number of buffers, it was considered full and any additional packets received were dropped until the number of buffers dropped below the RQL level. Similarly, the TQL value determined the maximum number of packet buffers that could be held on the transmit queue at any given time. Any additional packets switched to an interface while its transmit queue was full were dropped until its transmit queue depth dropped below the TQL value.

You can find a more detailed description of the MEMD counters and the buffer carving process in Chapter 6, "Cisco 7500 Routers."

Last updated on 12/5/2001

Index terms contained in this section