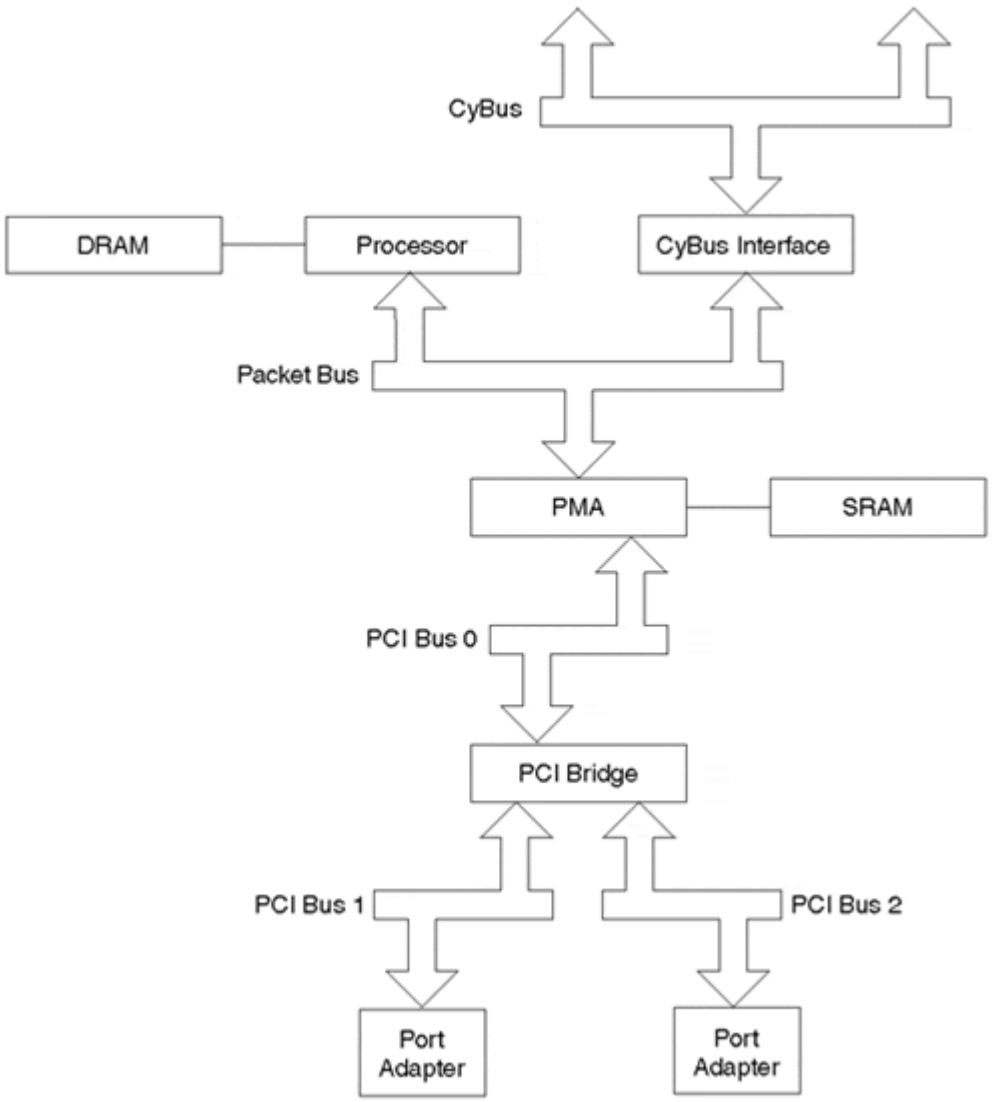
VIP Architecture

Cisco introduced the *Versatile Interface Processor* (VIP) to provide a scalable distributed architecture on top of an already proven platform. The VIP is based on a single motherboard and multiple port adapters that can have mixed media types (hence the name *versatile*).

VIP Components

VIPs are essentially routers mounted on Cisco 7500 interface processor cards. Each VIP has its own processor, packet memory, and main memory, and each runs its own copy of IOS. Figure 6-7 shows a high level architecture of the VIP.

Figure 6-7. VIP Architecture



CyBus Interface

The VIP is a fully CyBus-capable interface processor, meaning it can perform two 4-byte reads in each bus clock cycle and exploit the full 1.066 Gbps bandwidth of the CyBus.

CPU

The local processor (a MIPS R4700, R5000, or R7000 CPU) on the VIP runs a special version of the IOS operating system. It's capable of performing packet switching, offloading this task from the main CPU on the RSP. The VIP IOS also supports local, advanced Layer 3 services, such as packet filtering, weighted fair queuing (WFQ), and weighted random early drop (WRED). This local packet switching and filtering capability forms the basis of the 7500's *distributed switching* architecture.

Main Memory

VIPs use upgradable dynamic RAM (DRAM) for their main memory regions. Like the main memory on the RSP, VIP main memory contains the IOS instructions, program data, switching control structures, and heap.

Packet Memory

VIPs have an upgradable bank of SRAM used for storing packet buffers and certain data structures used by the interfaces' media controllers. This bank of memory, called *packet memory*, is designed to provide high speed access to packet data, particularly for high speed interfaces.

VIP IOS divides the packet memory into pools of packet buffers used by the interfaces. Like IOS on the Cisco 7200, VIP IOS uses the particle buffer approach to provide packet buffers for interfaces.

VIPs have a fixed particle size, typically 256 bytes or 512 bytes, which applies to all interfaces on the board. The particular particle size used depends on the type of port adapters installed on the VIP. Executing the IOS command **show controller vip slot** *slot* **tech-support** displays the particle size.

Port Adapter

VIPs use the same port adapters as the Cisco 7200. Each VIP can accommodate up to two port adapters on its board, accepting either two single-wide port adapters or one double-wide port adapter. Note that VIP IOS does not support insertion or removal of the port adapters while the VIP is operating.

PCI Bus

VIPs use the CyBus to connect to the RSP, but within the VIP card PCI buses are used to connect the CPU to the port adapters, just like the 7200. A VIP has two PCI busses, bridged together, each 32 bits wide and operating at 25 Mhz, providing a theoretical bandwidth of 800 Mbps (25 MHz * 32 bits/cycle).

VIP Models

There are three VIP models available as of this writing: the VIP2-15, the VIP2-40, and the VIP2-50. The particular VIP model can be determined from the output of the **show diag** command. The VIP2-40 has an R4700 MIPS CPU, 2 MB of SRAM, and 32 MB of DRAM. The memory options on a VIP2-40 are static—they cannot be upgraded. A VIP2-50 has a R5000 MIPS CPU and can have up to 8 MB SRAM and 64 MB of DRAM.

Entering the **show diag** command on a VIP2-40 displays the output that appears in Example 6-5

Example 6-5. *show diag* Command Output for the VIP2-40 Model

Router>**show diag** Slot 2: Physical slot 2, ~physical slot 0xD, logical slot 2, CBus 0 Microcode Status 0x4 Master Enable, LED, WCS Loaded Board is analyzed Pending I/O Status: None EEPROM format version 1 VIP2 controller, HW rev 2.04, board revision D0 Serial number: 06745223 Part number: 73-1684-03 Test history: 0x0E RMA number: 16-96-31 Flags: cisco 7000 board; 7500 compatible EEPROM contents (hex): 0x20: 01 15 02 04 00 66 EC 87 49 06 94 03 0E 10 60 1F 0x30: 68 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 Slot database information: Flags: 0x4 Insertion time: 0x32B7F4 (3d04h ago) Controller Memory Size: 32 MBytes DRAM, 2048 KBytes SRAM

Entering the **show diag** command on a VIP2-50 displays output similar to Example 6-6

Example 6-6. *show diag* Command Output for the VIP2-50 Model

Router>**show diag** Slot 1: Physical slot 1, ~physical slot 0xE, logical slot 1, CBus 0 Microcode Status 0x4 Master Enable, LED, WCS Loaded Board is analyzed Pending I/O Status: None EEPROM format version 1 VIP2 R5K controller, HW rev 2.02, board revision A0 Serial number: 12345678 Part number: 73-2167-04 Test history: 0x00 RMA number: 00-00-00 Flags: cisco 7000 board; 7500 compatible EEPROM contents (hex): 0x20: 01 1E 02 02 00 93 63 AC 49 08 77 04 00 00 00 00 0x30: 50 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 Slot database information: Flags: 0x4 Insertion time: 0x14D4 (9w3d ago) Controller Memory Size: 32 MBytes DRAM, 4096 KBytes SRAM

VIP Packet Operations: Distributed Switching

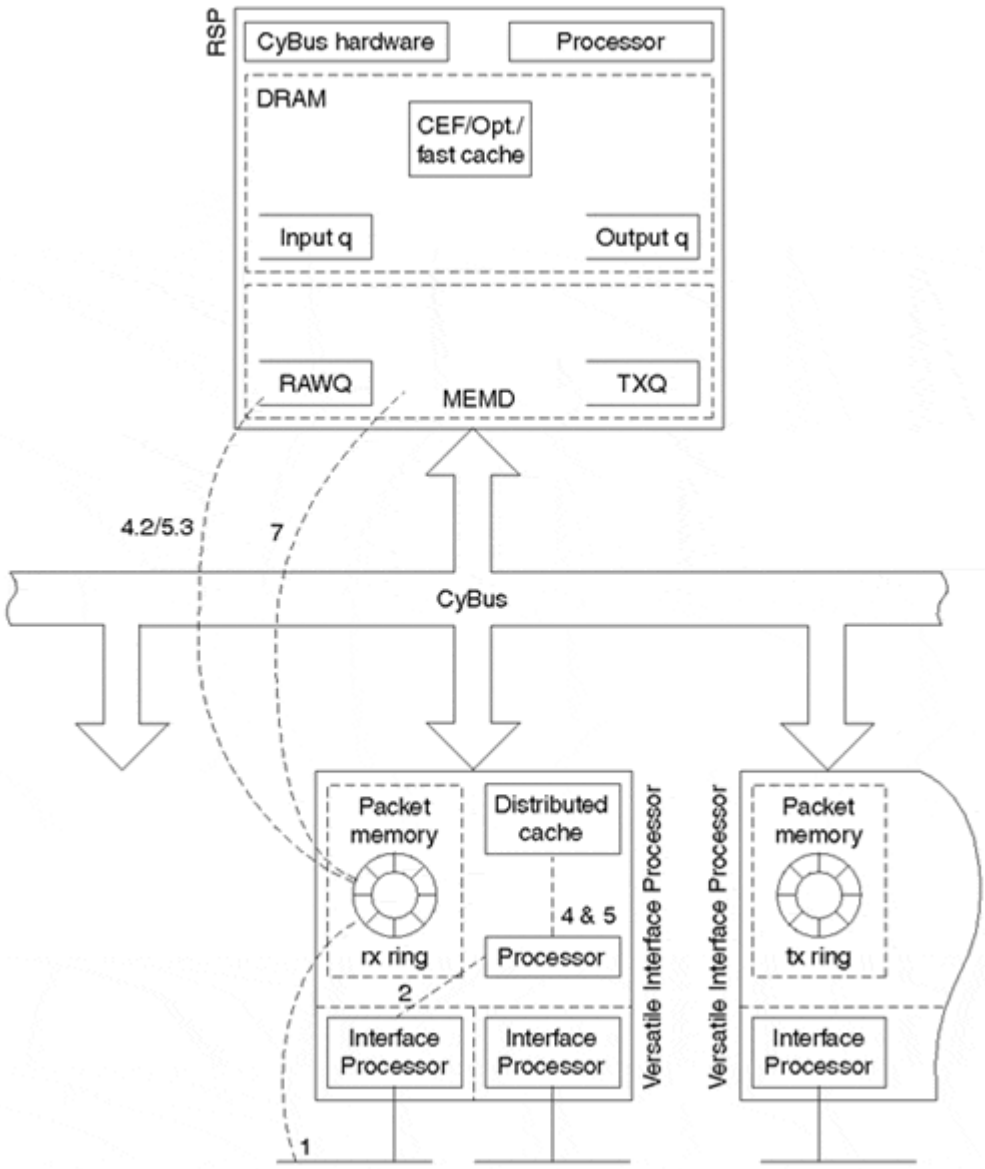
At their most basic level, VIPs perform the same functions other, less sophisticated, microcoded interface processors do; they assemble received packets from interface media for the RSP and transmit packets from the RSP out media interfaces. However, as mentioned previously, VIPs can also do something their microcoded predecessors cannot do. Because VIPs run IOS, they have built-in logic that allows them to make packet switching decisions—a very powerful feature. VIPs also contain hardware allowing them to use the CyBus to send data to other interface processors without involving the RSP CPU. These two capabilities make it possible for VIPs to operate like independent packet switching engines in the 7500 chassis and provide a means to distribute the packet switching work to multiple CPU's in the router. This concept is the basis of the IOS *distributed switching* feature.

To understand how distributed switching works, the following sections examine the three packet operation stages within the context of the VIP IOS—receiving the packet, switching the packet, and transmitting the packet.

Distributed Switching: Receiving the Packet

Figure 6-8 illustrates the steps of the packet receive and packet switching stages for distributed switching. Each step is described in the numbered list that follows.

Figure 6-8. Distributed Switching: Receiving the Packet



Step 1. An interface's media controller (on a port adapter) detects an incoming packet on the media. The media controller receives the packet into one or more free particle buffers held within its local receive area in packet memory (called a *receive ring*).

Step 2. The media controller raises a network interrupt to the VIP CPU. The VIP CPU acknowledges the interrupt and VIP IOS begins interrupt processing by calling on packet receive processing for the interrupting interface.

Step 3. Packet receive processing logic attempts to take the packet's particle buffers from the receive ring and replace them with empty particle buffers acquired from the global particle buffer pool (this step is not shown in Figure 6-8).

Step 3.1. If there aren't sufficient empty particle buffers in the global pool to replace the ones in the receive ring, the packet is dropped and the interface's **ignore** counter is incremented. The packet's particle buffers are left in the receive ring and are marked as free to be reused by a later packet.

Step 3.2. If the receive processing logic successfully replenishes the receive ring, it continues by inspecting the particle buffer's contents to determine the packet type.

Step 3.2.1. *If the received packet is an IP packet, then go to distributed switching (distributed optimum or distributed CEF).*

Step 3.2.2. *If the packet is not an IP packet, then distributed switching is not supported. The packet is passed to the RSP in MEMD via the CyBus using the same procedure as the microcoded interface processors. VIP IOS then finishes the interrupt processing and the VIP CPU dismisses the network interrupt.*

Distributed Switching: Switching the Packet

Figure 6-8 also shows the steps in the packet switching stage for distributed switching. All distributed switching decisions are made by the VIP CPU. The following steps describe the distributed packet switching stage.

Step 4. Distributed Optimum Switching. A full copy of the optimum switching cache from the RSP is kept in the VIP's main memory so switching decisions can be made by the VIP CPU. The RSP cache and the distributed cache are kept in sync by messages between the two CPUs. While still processing the network interrupt, VIP IOS looks for forwarding information in the local optimum cache.

Step 4.1. If the information needed to switch the packet is found in the local optimum cache, the packet is encapsulated with the new MAC information and the packet is passed to the packet transmit stage.

Step 4.2. If the information needed to switch the packet is not found in the local optimum cache, the packet is passed to the RSP in MEMD via the CyBus using the same procedure as the microcoded interface processors. VIP IOS then finishes the interrupt processing and the VIP CPU dismisses the network interrupt.

Step 5. Distributed CEF switching. A copy of the CEF table and the adjacency table from the RSP are kept in the VIP's local main memory so the VIP CPU can CEF switch incoming packets. Messages passed between the RSP CPU and the VIP CPU keep the master tables on the RSP and the slave tables on the VIP in sync. While still processing the network interrupt, the VIP IOS searches the CEF table for the information needed to switch the packet.

Step 5.1. If the information needed to switch the packet is found in the local CEF table, the packet is re-encapsulated and is passed to the packet transmit stage.

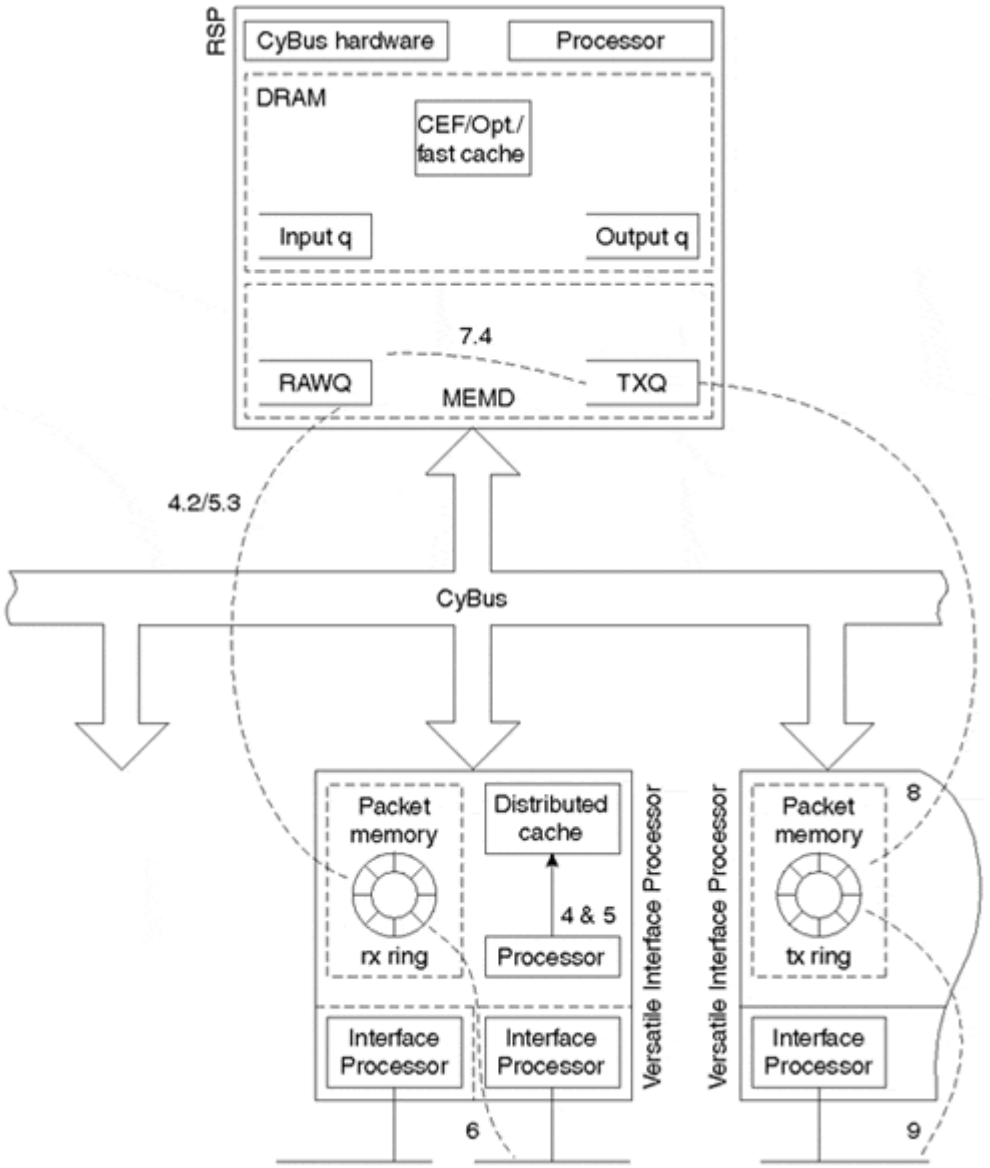
Step 5.2. If searching the local CEF table fails to produce the information needed to switch the packet, the packet is dropped. VIP IOS finishes interrupt processing and the VIP CPU dismisses the network interrupt.

Step 5.3. If the results of the CEF search result in a punt or receive adjacency, the packet is passed to the RSP in MEMD via the CyBus using the same procedure as the microcoded interface processors. VIP IOS then finishes the interrupt processing and the VIP CPU dismisses the network interrupt.

Distributed Switching: Transmitting the Packet

Figure 6-9 illustrates the steps of the packet transmit stage for distributed switching. Each step is described in the numbered list that follows.

Figure 6-9. Distributed Switching: Packet Transmit Stage



Step 6. If the packet is destined for an interface on the same VIP, it can be transmitted locally and never needs to cross the CyBus.

Step 6.1. The software running on the VIP places the packet's particle buffers in the local transmit area (called the *transmit ring*) for the outbound interface. VIP IOS finishes interrupt processing and the VIP CPU dismisses the network interrupt.

Step 6.2. The outbound interface's media controller (on the port adapter) transmits the packet data from its transmit ring onto the media.

Step 6.3. When the media controller completes transmitting the packet, it marks the packet's particle buffers as empty and notifies the VIP IOS (via another CPU interrupt) that the particles have been freed and might be returned to the global particle pool.

Step 7. If the packet is destined for an interface on some other interface processor, the VIP IOS transfers the packet to the other interface processor via MEMD across the CyBus.

Step 7.1. VIP IOS attempts to obtain a free MEMD buffer, first from its local free queue and then from the global free queue.

Step 7.2. If there are no MEMD buffers available, VIP IOS tries to hold the packet temporarily in packet

memory rather than drop it (see the section, "VIP Receive Side Buffering"). Similarly, if the outbound interface's MEMD transmit queue is full, VIP IOS also tries to hold the packet temporarily in packet memory.

Step 7.3. If VIP IOS succeeds in obtaining a MEMD buffer, it copies the packet data from the particle buffers into the MEMD buffer across the CyBus.

Step 7.4. The MEMD buffer header for the packet is then placed directly on the outbound interface's transmit queue. The RSP's CPU is never interrupted during this process; it continues doing background processes, handling process switched traffic, maintaining the routing table, and so on.

Step 7.5. VIP IOS finishes processing the interrupt and the VIP CPU dismisses the network interrupt.

Step 8. The outbound interface can be on a legacy interface processor or on another VIP. In either case, the outbound interface processor code detects a packet in the outbound transmit queue and copies the contents of the MEMD packet buffer across the CyBus into its internal memory.

Step 9. The outbound interface media controller transmits the packet from the interface processor's memory out to the media.

VIP Receive Side Buffering

Cisco 7500 interface processors, including the VIP, do not have the capability to directly access each other's internal memory. They only have access to their own internal memory and the RSP's MEMD memory. Because MEMD is the only common memory resource, every packet that passes from one interface processor to another over the CyBus must pass through an RSP MEMD packet buffer. MEMD, however, is a limited resource. So there is no guarantee a MEMD packet buffer will always be available when a VIP wants to transfer a packet to another interface processor.

In cases where the VIP is distributed switching a packet and no MEMD buffer is available, VIP IOS has the capability to buffer a received packet locally through a process called *receive side buffering*. Receive side buffering helps to reduce packet drops on the VIP by utilizing the packet memory on the VIP to temporarily hold packets that would otherwise be dropped due to lack of MEMD resources or a full transmit queue. It is particularly useful in situations where bursts of packets are received on a fast VIP interface and are destined for a slower interface on another interface processor.

With receive side buffering, VIP IOS creates an internal queue for every outbound interface requiring packet buffering. The size of each queue is derived from the configured bandwidth of the associated interface. At most, one second's worth of packets can be held on each queue.

You can display receive side buffering statistics by the IOS **show controller vip accumulator** command, as demonstrated in Example 6-7.

Example 6-7. *show controller vip accumulator* Command Output

Router#**show controller vip 2 accumulator** Buffered RX packets by accumulator: Forward queue 0 : 573 in, 0 drops (0 paks, 0 bufs) 1 FastEthernet1/0: 2 MEMD txacc 0x0082: 4500 in, 300 drops (0 paks, 0/24414/24414 bufs) 100000kbps 3 No MEMD acc: 4500 in, 300 limit drops, 0 no buffer 4 No MEMD buf: 0 in, 0 limit drops, 0 no buffer ATM2/0/0: local txacc 0x1A02: 0 in, 0 drops (0 paks, 0/37968/37968 bufs) 155520kbps

The set of lines numbered 1 through 4 constitutes the set of receive side buffering statistics for a particular outbound interface. These lines are repeated for each outbound interface.

In Example 6-7, line 2 indicates that a total of 4500 packets destined for FastEthernet1/0 were buffered in packet memory and 300 packets were dropped. There are currently 0 packets in the receive side buffer queue and a maximum of 24,414 packets will be buffered. This maximum number of buffers is the number of buffers needed to buffer one second's worth of traffic destined out the transmit interface. This can be expressed by the formula:

Max buffers = (Transmit interface bandwidth in bps/8) / particle size

In this example, bandwidth is 100 MB and the particle size is 512 bytes.

Line 3 indicates that all the packets counted as buffered in line 2 (4500 total) were buffered due to a transmit queue full condition (**No MEMD acc**). The 300 dropped packets occurred for the same reason. Note that if you see a non-zero **no buffer** drops counter on this line, it means there was insufficient packet memory resources to perform receive side buffering. This could indicate a shortage of packet memory (SRAM) on the VIP. Line 4 indicates there were no packets buffered due to lack of available MEMD packet buffers (**No MEMD buf**).

Last updated on 12/5/2001

< BACK Make Note | Bookmark CONTINUE >

Index terms contained in this section