CHAPTER

17

Make the Switch!

Up to now, we have considered the various facets of switch operation only in isolation; that is, the issues related to flow control, Multicast Pruning, VLANs, Priority Operation, and so on, were treated independently of one another. This chapter shows how many of the concepts and mechanisms that have been discussed can be integrated into a complete switch architecture. Because the operation of these switch functions has already been discussed in depth in earlier chapters, no detail is provided here. This chapter focuses on how the various functional elements are combined in practical switches.

The approach taken is to walk through a hypothetical switch from input port to output port. At each stage, we look at the operations that must be performed, and in particular, the order in which actions must be taken to achieve the desired end results. To get the maximum benefit from this process, we assume that our switch is fully featured; that is, in addition to the basic functionality of a transparent switch, we consider the implications of:

- IEEE 802.3x Flow Control
- IEEE 802.3ad Link Aggregation
- IEEE 802.1p Multicast Pruning
- IEEE 802.1Q Virtual LAN support
- IEEE 802.1p/Q Priority Operation
- Network management

Of course, many switches implement only a subset of these capabilities, either for the switch as a whole or for particular ports (for example, not

every switch port will be part of an aggregation). Those functions that are not implemented just simplify the design.

It is impossible to discuss in one chapter all of the possible permutations and design considerations for LAN switches. By necessity, we have narrowed the scope to what we believe are the important issues that affect and have driven modern switch architectures. In particular:

- The emphasis of the discussion is on the operations performed by the switch to achieve proper data flow from input to output port(s). In any practical switch there will also be numerous control paths among the various modules to communicate state information, policies, table updates, and so on. As these tend to be quite implementation-specific, they are not discussed in great detail here.

WARNING FOR READERS WHO DO NOT REGULARLY ENJOY MYSTERY NOVELS!

In a good mystery story, the pleasure comes from following the tale as the author weaves it — from beginning to end in a linear fashion. The author provides circuitous routes, flashbacks, and digressions (if appropriate) — readers do not jump from chapter to chapter in nonsequential order.

Many people (myself included) read technical literature with the goal being to glean useful information rather than to be entertained.[1] Unlike a mystery novel, it often makes sense to start reading a technical book at the end, going back to earlier chapters only as needed to fill in the gaps that were not understood.

Please resist this temptation here. The discussions contained in this chapter presume that you have read (and, hopefully, understood) most of the previous chapters. Many of the concepts and much of the terminology from prior chapters are used here without introduction or explanation, as that would require restating most of the contents of the entire book within this one chapter.

- While the operations discussed are those of a Layer 2 switch (transparent bridge), the architecture and data flow could also be generalized to support Network layer routing (Layer 3 switching). The only significant changes will be an increase in the complexity and organization of the Classification and Lookup Engines (see sections 17.2.4 and 17.2.6) and an increase in the workload relegated to the housekeeping processor subsystem (see section 17.1.2).

[1]When Rich wants an uproariously fun time, he reads patent specifications and IEEE network standards. He really needs to get out more.

- While no particular LAN technology is presumed for the various port interfaces, there is no discussion of:
  - Source routing as used on Token Ring and FDDI LANs
  - Handling of embedded source routing information in VLAN tags
  - Big Endian/Little Endian conversions
- A commercial switch may, of course, use a different architecture or model than the exemplary one given here, although the functions performed must necessarily be similar. In some products, some functional modules may be combined (for example, VLAN input filtering may be performed by the Lookup Engine rather than as a separate block, as discussed in sections 17.2.5 and 17.2.6).

Figure 17-1 depicts the architecture of our hypothetical switch from a very high-level viewpoint.

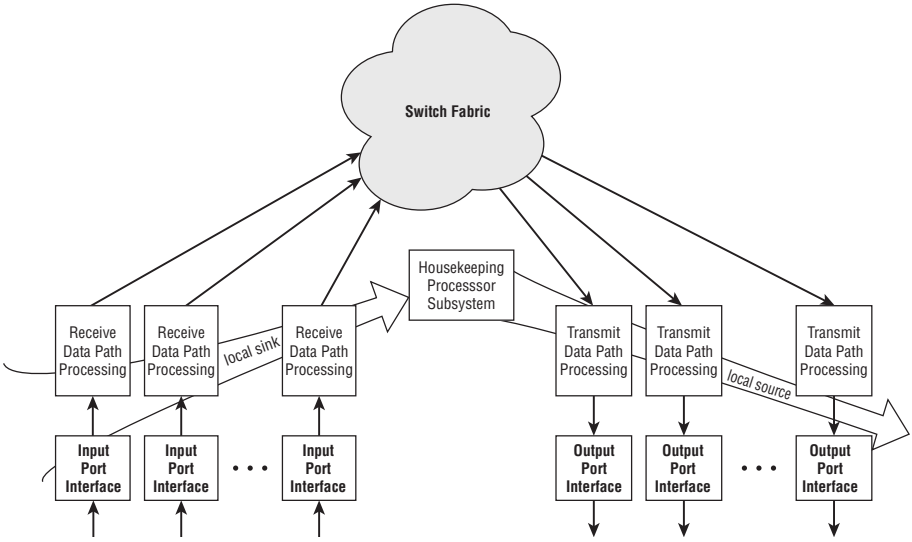


Figure 17-1 High-level block diagram

Following a discussion of embedded housekeeping functions, this chapter then delves into greater detail on:

- The path from where frames are received on the attached LANs to the point where the switch has decided to which output port(s) the frame should be transferred (input flow).
- Switch fabrics — the mechanisms by which frames are transferred among the multiple ports of a switch.
- The path between the switch fabric and the (possible) transmission of frames onto the LAN ports (output flow).

17.1 Keeping House

In the early days of internetworking, many bridge and router products implemented the majority of their functionality through software executed on an embedded microprocessor. While this may have been adequate for devices supporting small numbers of ports at relatively low data rates, it is rarely the approach taken today. The time-critical operations in the data path of a modern switch are invariably performed by dedicated hardware.

However, there is still a need for an embedded microprocessor in a hardware-based switch, except in the most basic, cost-driven, unmanaged desktop devices. While all of the fast path functions may be performed by specialized hardware, there are still myriad background processes, support functions, network management, and control protocols that are neither necessary nor practical to implement in dedicated hardware. In general, this *housekeeping processor* is used for all switch functions that are not in the typical fast path of the data flow.

Figure 17-2 depicts a common configuration for the housekeeping processor subsystem. The system includes a standard microprocessor (CPU) along with memory for storage of both the operational code and frame buffers. The code store is often implemented in nonvolatile memory; EEPROM or Flash ROM is typically used.[2] This allows the housekeeping capability to be available upon device initialization without the need for either code download or a local mass storage device (i.e., disk), while still allowing code modifications, bug fixes, and minor upgrades without a hardware change. Volatile storage for frame buffers and other data structures is provided in Dynamic RAM (DRAM). A serial port is normally provided for initial device configuration, secure out-of-band management, system debug, and code modification download (see Chapter 14, ''Switch Management'').

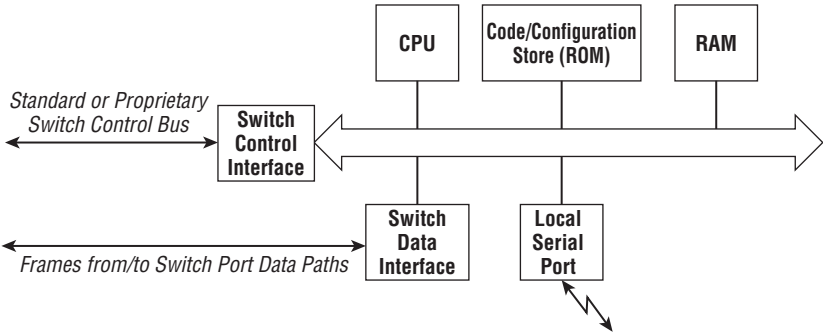


Figure 17-2 Housekeeping processor subsystem

[2]Typically, the operational code is transferred from ROM to RAM during device initialization for performance reasons.

There must also be some means to move data (typically frame buffers) between the switch's ports and the housekeeping subsystem, both for reception and transmission. This *switch data interface* is generally implementation-specific. It may vary from a Direct Memory Access (DMA) channel between the port memory and the housekeeping RAM to a proprietary internal bus structure. The *switch control interface* similarly allows the housekeeping processor to directly control the underlying hardware in an implementation-specific manner (switch fabric control registers, port configuration registers, and so on).

17.1.1 Housekeeping Functions

Most of the control and management functions discussed in earlier chapters will be implemented on the switch housekeeping processor, including:

- **Spanning Tree Protocol:** Spanning tree BPDUs, when received, are internally sunk to the housekeeping processor, where the received information is compared to the current spanning tree state. On those ports that are not in the *disabled* or *blocking* states, the housekeeping processor may regularly source spanning tree BPDUs for transmission either as the Root Bridge or in response to received BPDUs (see Chapter 5, ''Loop Resolution''). Similar to spanning tree operation, any proprietary loop resolution protocol (for example, on remote bridges using WAN links) may be implemented in the housekeeping processor as well.
- **Link Aggregation Control Protocol (LACP):** The protocol used to automatically configure and maintain aggregated links was specifically designed to make relatively low performance demands, so that it can be executed in software on an embedded microprocessor. LACP messages received on any switch port are passed to the protocol entity running on the housekeeping processor; messages are similarly sourced as needed on the appropriate output ports. Note that LACP must be able to source frames onto a particular physical link within an aggregation rather than to the aggregation as a whole; that is, it inserts frames into the physical port queues rather than the aggregated port queues, as discussed in section 17.4.2.
- **Marker Protocol:** When transferring conversations among physical links within an aggregation, the switch may accelerate the transfer through the use of the Marker Protocol discussed in Chapter 9, ''Link Aggregation.'' The housekeeping processor may be used to insert Marker messages into the frame stream as necessary. As discussed in sections 17.2.3 and 17.4.3, the Marker and Marker Response mechanisms are somewhat more time-sensitive than some other control protocols. For that reason, high-performance switches that anticipate a need to perform

frequent and/or rapid conversation shifts may implement the Marker functions either in dedicated hardware or in a separate microprocessor.

- **Enabled GARP Applications:** Depending on the features supported by the switch, one or more GARP applications may be in operation (for example, GMRP or GVRP; see Chapter 10, ''Multicast Pruning,'' and Chapter 12, ''Virtual LANs: The IEEE Standard''). Frames encapsulating messages for enabled GARP applications will be sunk to the housekeeping processor. Frames containing messages for GARP applications that are either not implemented or not currently enabled will be forwarded through the switch fabric without processor intervention. The differentiation between the two sets of messages is made through appropriate configuration of the Classification Engine, discussed in section 17.2.4.
- **Network Management:** All of the management software discussed in Chapter 14 will generally be implemented on the housekeeping processor, including:
  - The TCP/IP protocol stack as needed
  - The SNMP protocol implementation and MIB data structures
  - Telnet
  - Any internal Web server functions
  - Local serial port functions and user interface, and so on
- **Internal Diagnostics and Maintenance:** Generally, there will be a variety of implementation-specific functions that are performed on the housekeeping processor, including:
  - Device initialization
  - Power-on self test
  - Diagnostic and debug routines
  - ROM code update capability, and so on

A switch implementing Network layer routing capability may have another entire set of non–fast-path housekeeping functions to perform in addition to those just listed. Routing protocols (for example, RIP or OSPF), IP control protocols (for example, IGMP and ICMP), and even IP options processing could be delegated to the housekeeping processor, although these may significantly increase the processing burden and affect system performance.

As noted in section 17.2.2, the Ethernet flow control function (i.e., PAUSE mechanism as discussed in Chapter 8, ''LAN and Switch Flow Control'') must generally be implemented in hardware because of its severe timing constraints.

17.1.2 Implementation and Performance (or, It's Tough to Find a Good Housekeeper)

Historically, a number of popular embedded CPUs have been used as housekeeping processors. During the early-to-mid-1990s, the Intel i960 family was widely used. Recent implementations have favored the MIPS and MIPS-like families of RISC processors, ARM, and the Motorola PowerPC. All of these are available in a wide variety of configurations, processing powers, and costs, allowing the housekeeping subsystem to be closely tailored to fit the needs of a particular switch implementation.

Considered individually, none of the functions just discussed imposes a substantial processing burden. However, the performance demands of all of the functions combined can be significant. Depending on the number and complexity of the features implemented (in particular, the presence of Network layer routing capability), the number of switch ports, and the anticipated frame arrival rate, it may be necessary to use multiple housekeeping processors to distribute the workload.

17.2 Switch Data Receive Path Functions

Figure 17-3 depicts the path followed by data received from a given input port. The sections that follow examine each of the logical blocks within the receive path.

17.2.1 Port Interfaces (Receive)

The Media Access Control (MAC) and Physical Layer receiver (PHY) comprise the traditional network interface functions. On the receive side, the PHY takes the electrical or optical signals from the medium and decodes them into a bit, nibble, or byte stream as appropriate for the particular technology in use. The PHY will generally be different for different media (for example, twisted pair or optical fiber) and different technologies (for example, data rate and LAN type). The MAC performs the necessary framing and validity checking to reconstruct properly formed frames from data stream presented by the PHY.

Typically, the receiving MAC will compute the Frame Check Sequence (FCS) value in real-time while receiving data from the PHY. In the event of an invalid FCS, the MAC will discard the frame. Other than possibly incrementing an error counter for management purposes, there is no need for

a switch to continue to process a frame received in error.[3] The receiving MAC will normally maintain most of the real-time port-based statistics counters discussed in Chapter 14 (for example, RMON Ethernet statistics).

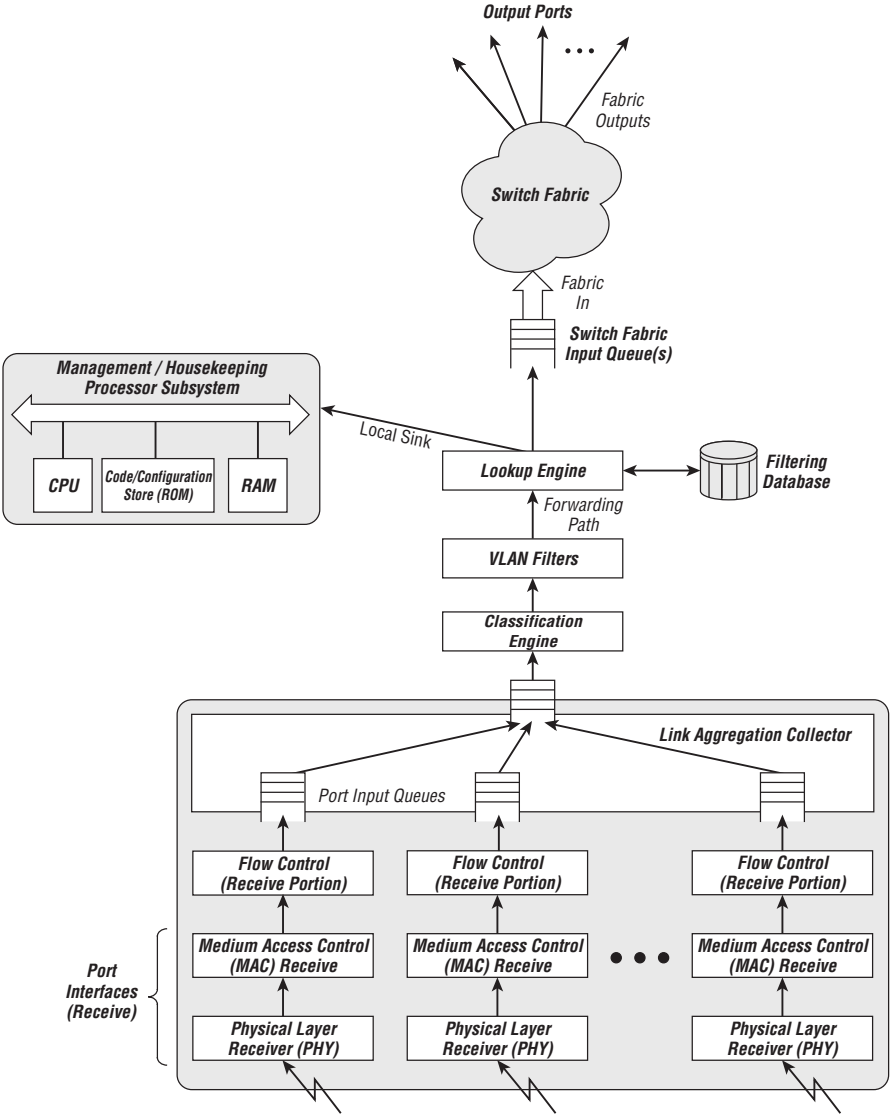


Figure 17-3 Switch input data flow (receive path)

[3]In the special case of a cut-through switch, a frame containing an FCS error may still be processed and forwarded. As discussed in Chapter 4, cut-through operation is rarely used today, as it requires that the input and output ports operate at the same data rate. Unless all of the switch's ports operate at the identical rate, it cannot be known whether cut-through forwarding is appropriate until the address table lookup operation is complete. In store-and-forward mode, a switch can transfer frames among ports operating at any data rate and can always discard frames containing FCS errors.

The receiving MAC must store incoming frames in a local buffer. If the switch uses a shared-memory fabric (see section 17.3.1), it may be possible to store the frame just once into a common buffer pool and avoid having to perform any buffer copy operations between input and output port(s). In this case, the switch simply maintains a list of outstanding buffer pointers and adjusts the various port queue data structures rather than actually moving frame data from port to port. With other fabrics (for example, cross-point matrices, see section 17.3.3), it is usually necessary to provide a frame buffer memory for each port (or group of ports) and transfer the frame data across the switch fabric.

Historically, PHY and MAC devices were built as individual packaged components with one instantiation per integrated circuit (IC). Modern silicon capability allows both horizontal and/or vertical integration, that is:

- PHY devices are available in quad, hex, octal, and denser configurations, both for Ethernet and other technologies.
- MAC devices are similarly available in multiple instantiations.

Both MAC and PHY devices are also available as semiconductor cores that can be integrated into a device that may be performing additional switch functions besides just the MAC and PHY interfaces. Depending on the nature of the target end product, it may be possible to integrate an entire switch system onto a single IC (system-on-a-chip), as discussed in Chapter 4, ''Principles of LAN Switches.''

17.2.2 Receive Flow Control

On the receive side of the interface, the flow control module inspects the incoming traffic stream in search of PAUSE frames, as discussed in Chapter 8. Upon receipt of a validly formed PAUSE command, this module updates the pause timer as indicated by the received pause_time value and provides control signals to the Transmit Flow Control module (see section 17.4.4) as appropriate. For example, upon receipt of a PAUSE frame with a non-zero value for the pause_time, the transmit side of the interface must be told to stop accepting frames from the MAC client (for example, the Link Aggregation Distributor) until the pause timer expires; in the event that a PAUSE frame is received with a zero value for the pause_time, the transmit side of the interface can be re-enabled to accept frames for transmission. The functions of parsing, interpreting, and acting upon received PAUSE frames are invariably implemented in dedicated hardware, because of the following:

- The operations are simple, well-defined, and bounded in nature.
- There are very tight restrictions on the length of time between reception of a PAUSE frame and the halting of the transmit frame flow (512 bit times for 10 Mb/s or 100 Mb/s interfaces, 1,024 bit times for 1000 Mb/s interfaces).

Of course, because link flow control is defined only for Ethernet, the Receive Flow Control module will not be present on a Token Ring or FDDI interface. In those cases, all frames would be delivered directly from the receiving MAC to the Link Aggregation Collector or Classification Engine.

17.2.3 Link Aggregation Collector

If multiple physical interfaces are being aggregated into a single logical interface, this module implements the frame collection function. The Collector gathers frames received from each of the underlying physical interfaces and presents a single, combined stream to the Classification Engine. As discussed in Chapter 9, ''Link Aggregation,'' the Collector does not need to take any special action to ensure proper frame order among frames received from each of the physical interfaces. The operation of the Distributor on the transmitting side of the link ensures that each conversation comprising a set of frames among which frame order must be maintained is mapped to a single physical link in the aggregation.

The Collector also includes the implementation of the responder portion of the Marker Protocol; this function may be implemented as a finite state machine or as software on a conventional microprocessor.[4] Depending on the design of the particular switch, the general-purpose housekeeping processor may be used to generate Marker Responses, although care must be taken to ensure very rapid response times. Typically, the Distributor at the other end of the aggregated link is waiting for a Marker Response before shifting one or more conversations from one physical link to another in the aggregation (see section 17.4.3). To the extent that Marker Responses are delayed, communications from those conversations will be stalled.

In general, the Link Aggregation Control Protocol (LACP) will be implemented in the housekeeping processor, as its operation is much less time-critical than that of the Marker Protocol. LACP frames always use a multicast Destination Address within the reserved range for link-constrained protocols; the Classification Engine will flag these for local sink to the housekeeping processor, as discussed later.

17.2.4 Classification Engine

Depending on the level of features and sophistication provided in a product, the Classification Engine can be one of the more complex and performance-intensive elements of a switch. This module takes the stream of received frames from an input port and parses them as needed to generate a

[4]The design of the Marker Protocol was intentionally kept simple to facilitate a pure hardware implementation at a reasonable cost.

vector of classifications based on various criteria and administrative policies. In particular, the Classification Engine may be used to implement:

- Local sinking of reserved multicast addresses
- The VLAN Ingress Rules
- Priority assessment

Each of these is discussed in the sections that follow.

17.2.4.1 Local Sinking of Reserved Multicast Addresses

Frames with Destination Addresses in the range reserved for link-constrained protocols (01-80-C2-00-00-00 through 01-80-C2-00-00-0F; see Chapter 2, ''Transparent Bridges'') must be identified and flagged to ensure that they are not forwarded to any other output ports through the switch fabric. Typically these frames will contain spanning tree BPDUs, Link Aggregation Control PDUs, and similar protocol messages, and must be locally sunk to the housekeeping processor. Note that while Ethernet flow control also uses one of these reserved multicast addresses, PAUSE frames will be sunk and acted upon by hardware within the Receive Flow Control module (see section 17.2.2) and will not normally be passed to the Classification Engine.

17.2.4.2 VLAN Ingress Rules

These are the rules used to associate a given frame with the VLAN to which it belongs. In a VLAN-aware switch, every frame must be classified as to VLAN membership.

As discussed in Chapter 11, ''Virtual LANs: Applications and Concepts,'' there are two ways to determine the VLAN to which a frame belongs. If the frame carries a VLAN tag, then the task of the Classification Engine is trivial; the VLAN identifier contained in the tag tells the switch which instant frame belongs to which VLAN.[5] Of course, the Classification Engine must parse the frame to determine whether it does indeed contain a tag. On an Ethernet, this implies a comparison of the Type field against the VLAN Protocol Identifier (0x8100). On a Token Ring or FDDI, the VLAN Protocol Identifier is encapsulated by LLC and SNAP, as discussed in Chapter 12.

If the frame is untagged, then its VLAN membership must be determined by applying the set of VLAN association rules to the contents of the frame. Depending on the capabilities of the switch and the nature of the administrative policies being applied, the implicit mapping of a frame to a VLAN could be quite complex. VLAN membership could be based on MAC addresses, protocol

[5]Note that if the VLAN identifier in a tag is 0x000, the frame is considered to be priority tagged rather than VLAN tagged, as discussed in Chapter 12. In this case, the VLAN membership must be determined implicitly from the frame contents just as if the frame carried no tag at all.

types, Network layer address information, or even TCP ports and higher layer application data, either individually or in various combinations. As a result, the Classification Engine may need to:

- **Parse multiple fields in the frame:** The VLAN membership may be based on a catenation of MAC address, protocol type, or other information taken in combination.
- **Perform serial, conditional parsing:** It may be necessary to first parse the frame to determine the higher-layer protocol type and then look at different fields deeper in the frame depending on the encapsulated protocol. For example, VLAN membership may be based on IP subnet information for those frames comprising IP packets and on IPX network identifiers for those frames containing IPX packets. The parsing must be done serially and conditionally, as the second qualifier will have a different syntax and be in a different position in the frame for each encapsulated protocol.
- **Deal with variable length fields:** Depending on the protocol and the fields of interest, the Classification Engine may need to calculate field offsets into the frame in real-time as a function of parsed information. For example, if an IP datagram contains options, the IP protocol header will be variable in length. An inspection of any fields positioned after the IP header will require a variable offset.
- **Perform extensive comparisons and/or table lookups to determine the VLAN mapping:** Depending on the VLAN association rules, the Classification Engine may need to compare MAC addresses, protocol types, or other parsed information against a stored database of mappings. This implies some form of lookup mechanism and local memory store accessible by the Classification Engine.

Regardless of the complexity or the set of policies in place, the end result is the assignment of a single VLAN Identifier to each frame in the range of 1 through 4,094.[6] If multiple policies are in place (for example, application-, MAC address-, and port-based VLAN association rules), a priority order must be established by policy. For example, an application-based VLAN mapping (if present) may take priority over a MAC address-based mapping that also renders a match. In the absence of a match of the frame's contents against any stated administrative policy, the default VLAN for the input port is used;

[6]Note that if a frame arrives in priority-tagged format (i.e., with a VLAN Identifier of 0x000), it will be assigned a non-zero VLAN Identifier by the Classification Engine. Ultimately, the frame will be forwarded on the appropriate output port(s) either untagged or tagged with the non-zero value. That is, priority-tagged frames never propagate through a switch without a change to the VLAN identifier; an IEEE 802.1Q–compliant switch will never emit a priority-tagged frame.

that is, port-based VLAN mapping is used only if no other rules apply to a given frame.

17.2.4.3 Priority Assessment

In addition to categorizing each frame according to VLAN membership, the Classification Engine also determines a user priority value for each frame, typically in the range of 0 to 7. As discussed in Chapter 13, ''Priority Operation,'' the priority value may be:

- Extracted from a VLAN or priority tag (if present)
- Indicated by LAN-specific priority signaling (for example, the native user priority field in a Token Ring or FDDI frame)[7]
- Determined implicitly from the frame's contents

From an implementation perspective, the last case poses the most difficult task, as it implies parsing the frame and comparing the results against a set of policies in a manner similar to implicit VLAN classification.

17.2.4.4 Do It Once and Save the Results

The Classification Engine performs the entire frame parsing and field comparisons needed to determine the salient characteristics of the frame from the switch's perspective. As a result of this processing, the switch can know:

- If the frame must be link-constrained (see section 17.2.4.1)
- The VLAN identifier associated with the frame (see section 17.2.4.2)
- The priority assigned to the frame (see section 17.2.4.3)
- The port on which the frame arrived (by virtue of the physical port or Aggregator that passed the frame to the Classification Engine)
- The time at which the frame arrived (by applying a timestamp upon receipt of the frame)
- Whether the frame was tagged or untagged upon receipt (from the VLAN and priority parsing processes)
- Whether the frame data payload is in canonical or non-canonical bit order (from the tag contents or the nature of the underlying physical port)
- Whether the frame carries a unicast or multicast destination, and so on

Once all of this information is collected, it makes sense to record the classification results along with the frame, so that other switch modules

[7]In the case of such a natively signaled priority, priority regeneration may also be used to map locally assigned priorities to globally significant values.

can simply inspect the classification vector rather than having to re-inspect the frame to determine its characteristics before taking appropriate action(s). Thus, most switches create an *internal switch header* for each received frame that is carried along with the frame during subsequent processing. The header is stripped prior to actual transmission through any output port. Thus, the format of the switch header does not have to conform to any particular standard. It can be tailored to the specific needs and implementation of a given switch. Figure 17-4 depicts a hypothetical internal switch header.

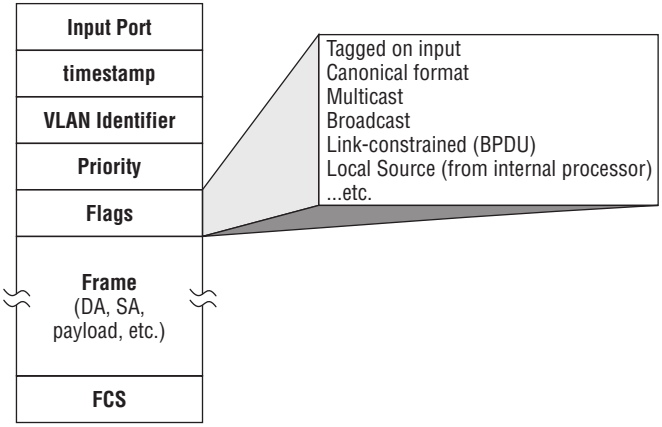


Figure 17-4 Internal switch header

Depending on the implementation, the internal switch header may actually be stored along with the frame contents (as depicted), or it may instead be associated with a queue entry for the frame; that is, the frame contents themselves may be stored separately from the header and linked with a buffer pointer.[8]

The timestamp is used so that the switch can determine the transit delay of a frame between reception and possible transmission through an output port. In this manner, the switch can enforce the Maximum Transit Delay constraint of IEEE 802.1D/Q (see Chapter 2). In general, the timestamp

[8]In Chapter 12, we discussed the theoretical problem of increasing the maximum length of an Ethernet frame to accommodate the 4-byte VLAN tag, and noted that no existing products were found that were unable to handle longer frames, as long as they weren't made too much longer. In practice, the limitation on the number of bytes added to the frame before implementation failed was a function of the length of practical internal switch headers. Many switch designs used (and still use) fixed-length buffers of 1,536 bytes (0x0600, a nice, round number), which includes both the frame itself and the internal switch header. The ''threshold of pain'' for an increased frame length was somewhere around 1,526 to 1,530 bytes maximum, which disappears quickly given the need to store a 1- or 2- byte port number, a 1- or 2-byte timestamp, 2 bytes of VLAN identifier and priority, a byte or so of flags, and perhaps a few bytes of buffer memory pointer(s) along with the Ethernet frame. Thus, we were able to increase the maximum Ethernet frame length from 1,518 (untagged), but could not have accommodated a much longer VLAN tag.

does not require either high resolution or absolute significance (i.e., we care only about the time difference between reception and transmission, not the time-of-day). Thus, a simple counter with a resolution on the order of tens or even hundreds-of-milliseconds is generally adequate. Ideally, the timestamp should be applied when the frame actually arrives at the input port's MAC entity; in practice, applying the timestamp within the Classification Engine is usually good enough. If desired, the timestamp can be normalized with respect to any known delay between reception by the MAC entity and subsequent processing by the Classification Engine.

17.2.4.5 Implementation of the Classification Engine

The performance demands placed on the Classification Engine may vary greatly, depending on:

- The nature of the classification rules supported (in particular, the allowable VLAN Ingress and Priority Rules)
- The data rate of the attached port
- The number of ports that may be aggregated prior to classification

In the simplest case (for example, a core switch), the Classification Engine may only see VLAN-tagged frames from a single port. With no implicit rules to be applied, the task of the engine is fairly easy; simply parse the tag and extract the relevant fields (for example, VLAN Identifier and Priority). These functions could be implemented in hardwired logic, as little flexibility is needed.[9]

On the other hand, an edge switch supporting multiple aggregated 100 Mb/s and/or 1000 Mb/s ports may have to deal with a large and complex set of VLAN and/or priority assignment rules. A simple hard-coded logic function will generally not provide either the capabilities or the flexibility required. In addition, wire-speed operation at 1000 Mb/s implies the ability to handle nearly 1.5 million frames per second.

As a result, the Classification Engine becomes a large and critical part of an edge switch designer's job. Some of the approaches taken to engine design include:

- **Hard-coded logic:** If the VLAN and priority rules are restricted to simple algorithms (for example, port-based, MAC address-based, and/or protocol-type-based), the Classification Engine can still be implemented in fixed-function logic; that is, circuitry can be designed

[9]This is one of the key justifications for the use of tags and the concept of core switches. High performance can be achieved with very little complexity at low cost.

specifically to parse and extract information from the relevant predetermined frame fields. Wire-speed operation at relatively low cost is achieved at the expense of low rules complexity. Many switches take this approach.

- **Classification Processors:** If the rules require greater flexibility, a microprocessor can be used to parse and analyze frames under program control. The exact fields inspected and the actions taken become a function of software running on this Classification Processor.

  As often happens, we are presented with a three-way tradeoff. A microprocessor-based approach can provide lots of flexibility, but the performance will not equal that of a pure hardware-based solution for the same cost. We must either use a lot of processing power (through faster or multiple processors) at a greatly increased cost, or live with a lower level of performance (either slower port data rates or non-wire-speed operation).

  A number of specialized *network processors* are now available that are tailored specifically for Classification Engine and similar purposes. They often include multiple RISC processors, embedded control memory, and optimized high-speed data paths for switch applications.

- **Programmable state machines:** While a microprocessor approach is extremely flexible, the performance of a software-based engine can sometimes be inadequate, especially for very high port data rates (for example, 1000 Mb/s or greater). However, a standard embedded microprocessor may provide more flexibility than is actually needed. The Classification Engine must parse and compare frame fields at high speeds against a set of pre-programmed rules, but it does not need many of the arithmetic or data manipulation features provided by a general-purpose processor. As a result, some high-performance switch designs have implemented parse-and-compare engines as programmable, finite-state machines, rather than using a traditional microprocessor.

  Ultimately, the set of VLAN and/or priority rules can be expressed as a sequence of bit strings and offsets that are compared against a received frame's contents. Rather than using software, you can design configurable hardware that is optimized for such arbitrary pattern matching in fixed and/or variable-position fields. The amount of logic required is much less than for a general-purpose microprocessor. As a result, optimized performance can be achieved at a lower cost than a pure software-based solution. The programmable-state machine may be implemented in traditional logic, as a programmable

micro-engine (essentially a specialized microprocessor with a highly restricted instruction set), or even as embedded logic blocks within a Field Programmable Gate Array (FPGA).[10] In most cases, embedded memory (typically SRAM) is used to store the rules set.

On a 1000 Mb/s Ethernet link, complete frames can arrive every 672 ns in the worst case scenario. Even dedicated logic would have difficulty applying a multitude of rules within this time limit. Thus, regardless of the approach taken, most designs resort to some form of pipelined processing for the application of multiple rules in sequence. Pipelining allows much more time for classification processing. However, it increases the absolute delay (latency) for all processed frames. That is, we can classify many more frames-per-second by setting up an assembly line and performing a variety of rules tests on multiple frames in parallel, but each frame experiences greater delay as it proceeds through the pipelined process before being forwarded by the switch.

It is especially important that the pipeline delay be bounded and within reasonable limits; that is, we cannot use an arbitrarily long pipeline to solve the problem of classification processing performance. Consider the plight of a high-priority frame belonging to a time-sensitive application flow (for example, interactive voice traffic). We must make sure that the frame does not get excessively delayed while we attempt to determine its priority. Conceivably, we could exceed the delay allowance before we even get to switch the frame! Such delay considerations inevitably place an upper bound on the number of rules that may be applied to a given frame.

17.2.5 VLAN Filters

The VLAN Filter module implements the two VLAN input filters discussed in Chapter 12.

Acceptable Frame Filter: A core switch may choose to admit for forwarding only those frames carrying an explicit VLAN tag and to discard untagged frames.[11] As shown in Figure 17-4, the switch header usually has a flag bit indicating whether a frame was tagged when it was received; the Acceptable Frame Filter need only inspect this flag and take the appropriate action.

[10] One example of an early specialized Classification Engine is 3Com's Flexible Intelligent Routing Engine (FIRE) [CIAM98]. Many other approaches are expected in emerging high-performance switch designs.

[11] Discarded untagged frames may additionally be counted for network management purposes.

Note that, regardless of the switch's policy with respect to forwarding untagged frames, frames whose Destination Address is in the reserved multicast range must still be locally sunk regardless of whether they are tagged or not. For example, a core switch must still implement the Spanning Tree Protocol, whose frames may be untagged.[12] Thus, the BPDU flag bit in the switch header (indicating that the destination was in the reserved multicast range), if set, overrides the VLAN Acceptable Frame Filter; BPDUs must always be passed to the local housekeeping processor.

VLAN Ingress Filter: Under administrative control, a switch may be willing to accept (or reject) frames when the port on which a frame was received is not in the member set for the VLAN to which the frame belongs. When set to reject, a VLAN becomes symmetric; frames will be accepted only from ports on which the switch is also permitted to emit frames for this VLAN. Asymmetric VLANs can be created by accepting frames from ports where the switch is prohibited from sending frames for the given VLAN.

Figure 17-5 depicts a flowchart for the combined set of VLAN filters. In the preceding discussion we described the filters as comprising a separate, discrete module. Practical implementations may integrate these functions into the Lookup Engine, discussed in section 17.2.6.

17.2.6 Lookup Engine

The Lookup Engine is the heart of the switch forwarding process. This module must decide what to do with frames that have successfully passed through all of the prior collection, classification, and VLAN Filter operations. The result of the lookup will be a set of output ports to which a given frame should be passed for further processing and possible transmission.

Table lookup is performed against the filtering database. This dynamically maintained database contains the current mapping of Destination Addresses and VLANs to the port(s) through which the target recipient(s) are reachable. Note that for a switch supporting IEEE 802.1p multicast pruning, the filtering database will include both unicast and multicast destination mappings. Frames being sent to unicast destinations should map to a single output port in the absence of a lookup failure; frames being sent to multicast destinations will map to one or more output ports.

[12]While an implementation of multiple spanning trees (spanning forest) may use tagged BPDUs, a switch may still need to deal with untagged BPDUs to support legacy devices capable only of using a single spanning tree.

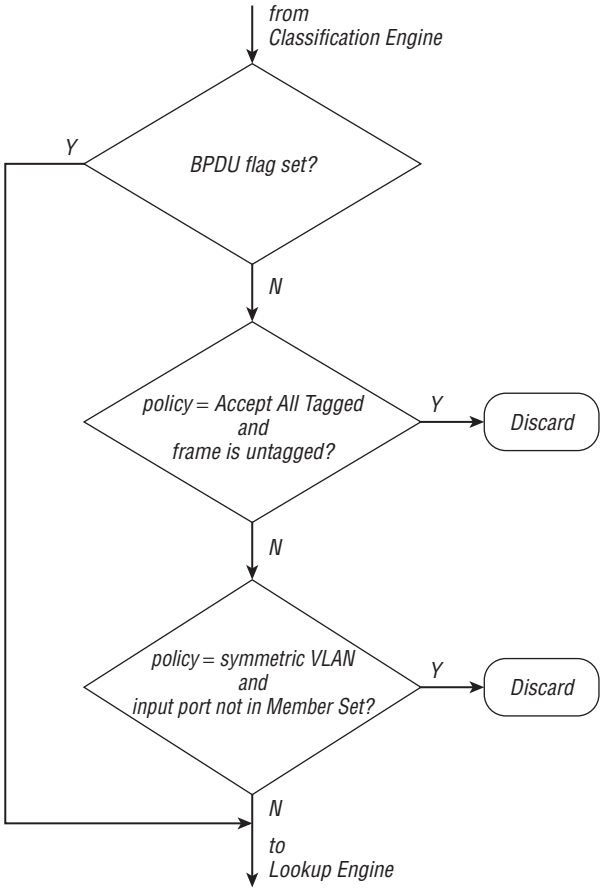


Figure 17-5 VLAN filter flow

17.2.6.1 Generating the Output Vector

As depicted in Figure 17-6, the Lookup Engine takes as its input the Destination Address in the received frame, the VLAN to which that frame belongs, and the flag bits in the internal switch header, and provides as an output a vector of ports to which this frame must be forwarded.[13] The Destination Address is in a known location within the frame buffer (typically the first field in the frame); the VLAN identifier and flag bits are stored in the internal switch header, as determined by the Classification Engine.

[13]This describes the lookup process for a Layer 2 switch. In a Layer 3 switch (i.e., a router), the lookup is generally more complex, being a function of protocol type, network and/or subnet identifiers, and so on. In addition, the result may need to include not only the output port vector but information about whether the target destination is locally or remotely connected and next-hop routing information.

In Chapter 2, we discussed the operation of the table lookup process for a VLAN-unaware bridge. In that case, only the Destination Address was needed to determine the appropriate output port(s); the lookup decision was not tempered by VLAN membership information. In a VLAN-aware switch, you must consider the VLAN membership when determining the port(s) on which to forward each received frame. You cannot send a frame onto ports that are not in the member set for the VLAN associated with that frame.

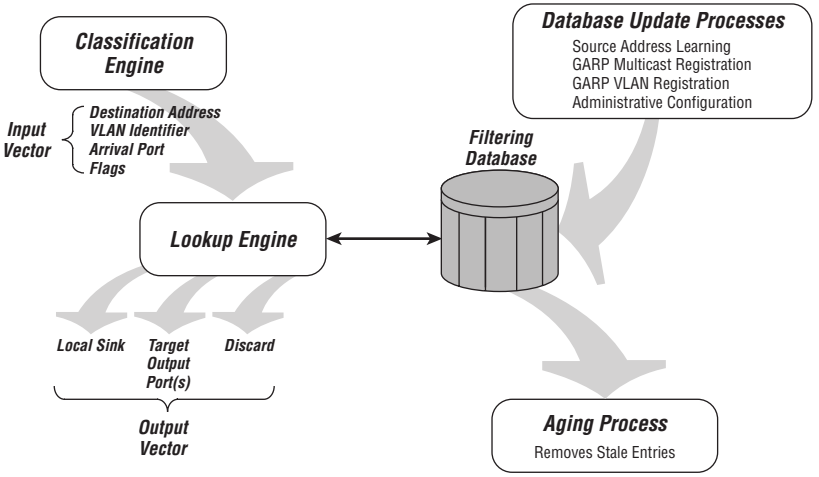


Figure 17-6 Filtering database processes

Depending on the contents of the filtering database, the output port vector may be:

- **A single output port:** This will be the typical case for a frame when:
  - The Destination Address is a known unicast and the port from which the database entry was learned is in the member set for the VLAN to which the received frame belongs.
  - The Destination Address is a known multicast, it is known that only the one port is needed to reach all devices listening to that multicast, and that port is in the member set for the VLAN to which the received frame belongs.
- **Multiple output ports:** A frame may be forwarded to multiple output ports if the Destination Address is a multicast or an unknown unicast. The frame will be directed to all ports in the member set for the VLAN to which the received frame belongs (except the port on which the frame arrived).

- **The local housekeeping processor port:** To simplify the design of the Lookup Engine, the local housekeeping processor generally appears as just another port on the switch; that is, one of the possible outputs of the engine is to locally sink a frame. Frames may be directed to the local processor port if:
  - The Classification Engine has determined that the frame is in the range of addresses reserved for link-constrained protocols (as indicated by a flag bit in the internal switch header).
  - The Destination Address is associated with a GARP application that is enabled on the switch (for example, GMRP or GVRP).
  - The Destination Address is equal to the MAC address of one of the switch's ports; that is, the frame is being sent to the switch itself as an end station. These frames may encapsulate network management traffic (SNMP, Telnet, and so on) or any other higher-layer application that may be running on the switch.[14]
  - A lookup failure occurs. The housekeeping processor can be used to handle lookup exception conditions, for example receiving a frame that belongs to a VLAN unknown to the switch.

  In some cases it may be appropriate to forward a frame to the local housekeeping processor in addition to one or more output ports of the switch. This might occur when a frame carries a multicast Destination Address and the local housekeeping processor is participating in the higher-layer application or protocol to which that multicast address maps.
- **No output port:** In some cases it is appropriate for the Lookup Engine to discard a frame without even sending it to the housekeeping processor. For example, in a switch using pure Independent VLAN Learning (IVL), if a frame is received with a known unicast Destination Address but with a VLAN association different from the one to which that destination is known to belong, the frame should be discarded. In lay terms, someone is trying to send a ''red frame'' to a ''blue station.'' Regardless of the fact that the destination is known, the VLAN is incorrect, and the frame should not be forwarded.

Note that a frame will never be forwarded out the same port from which it arrived. This would violate the non-duplication invariant of the Data Link.

[14]Note that this case allows the housekeeping processor to be used for Network layer routing in what would otherwise be a pure Layer 2 switch. Many switches provide such a software-based routing capability. While the performance may not equal that of a device optimized for routing, the feature can provide Network layer connectivity in cases where performance is not critical.

17.2.6.2 Maintaining the Filtering Database

As discussed in Chapter 2, the filtering database in a simple bridge can be maintained just by Source Address learning and the aging function. A full-featured switch needs to learn not only the port mappings for each known unicast source, but the mapping of registered multicast addresses as well as the port member set for each VLAN.

As such, the filtering database is maintained by a variety of processes as depicted in Figure 17-6:

- Static database entries are created and modified under human administrator control through network management.
- Dynamic unicast entries are learned by inspection of the Source Address in received frames.
- Multicast entries are learned through the GARP Multicast Registration Protocol (GMRP, see Chapter 10, ''Multicast Pruning'').
- Port VLAN mappings are maintained through a combination of administrator control and the GARP VLAN Registration Protocol (GVRP, see Chapter 12).

As discussed in Chapter 12, address learning may be performed independently for each VLAN (Independent VLAN Learning), or for multiple VLANs in the same data structure (Shared VLAN Learning).

17.2.6.3 Lookup Implementation

The implementation of the Lookup Engine is usually highly product- and vendor-specific. Depending on the complexity of the lookup operation and the number and data rates of the ports being supported, the Lookup Engine may incorporate:

- Content-addressable memory (CAM), providing almost instantaneous search and update capability through specialized associative storage (see Chapter 2).
- Pseudo-CAM, that is, using a standard memory (typically SRAM) together with a finite-state machine that emulates the operation of a CAM. Pseudo-CAMs generally provide both lower cost and lower performance when compared to true CAMs.
- Embedded microengines providing flexible, programmable lookup under software control.
- A combination of methods, for example, a small CAM for a most recently used entry cache, with a microprocessor-based fallback in the event of a cache miss.

Lookup Engines are generally modeled in one of two ways:

- **Centralized Lookup:** In this model, a single engine is used to perform the table lookup for all frames received from all ports, as depicted in Figure 17-7. While the burden placed on this engine is the sum of the lookup requirements from all ports combined, there need only be one engine for the entire switch, reducing overall cost. Because of the performance limitations of a single engine, this approach is used primarily with switches comprising moderate numbers of ports with relatively low data rates (for example, 10/100 Mb/s Ethernet desktop and workgroup switches). A centralized lookup model is most appropriate when a shared-memory or shared-bus switch fabric is present (see sections 17.3.1 and 17.3.2).
- **Distributed Lookup:** If a single Lookup Engine is incapable of keeping up with the frame arrival rate of all ports on the switch combined, it is usually necessary to provide a separate Lookup Engine for each port or group of ports, as depicted in Figure 17-8. While the cost will generally be greater than for a single Lookup Engine, this approach provides for greater and more scalable performance levels. The power of the combined Lookup Engines grows along with the number of ports deployed on the switch. In addition to the added cost, the switch designer must solve the problem of properly distributing and maintaining the contents of the filtering database so that the Lookup Engines operate in a coherent manner. Normally, this implies the presence of a control data path to distribute database updates among the port interfaces.[15]

The filtering database itself is most commonly implemented in high-speed RAM, external to the Lookup Engine itself. Depending on the memory bandwidth required by the Lookup Engine, the memory interface may be a conventional address/data bus or may use acceleration techniques such as Double Data-Rate (DDR) or Rambus technology.

As silicon density grows, emerging designs are more likely to use embedded memory; that is, the filtering database can be located within the Lookup Engine chip itself. Extremely high-speed memory exchange rates can be achieved because the width of the memory can be virtually unlimited. Rather than having to use a conventional 32- or 64-bit wide data path, internal memory widths can be 256, 512, or more bits, effectively multiplying the available memory bandwidth.[16]

[15] A PCI bus is often provided for this purpose.

[16] External memories are rarely practical in such wide organizations because the number of pins required on both the memory and Lookup Engine devices would increase their cost prohibitively. By embedding the memory, we eliminate the pin restriction. A 512 bit wide, 8 ns SRAM can provide 64 Gb/s of raw memory bandwidth; this is the equivalent of a 64-bit wide external memory with a 1000 MHz clock.

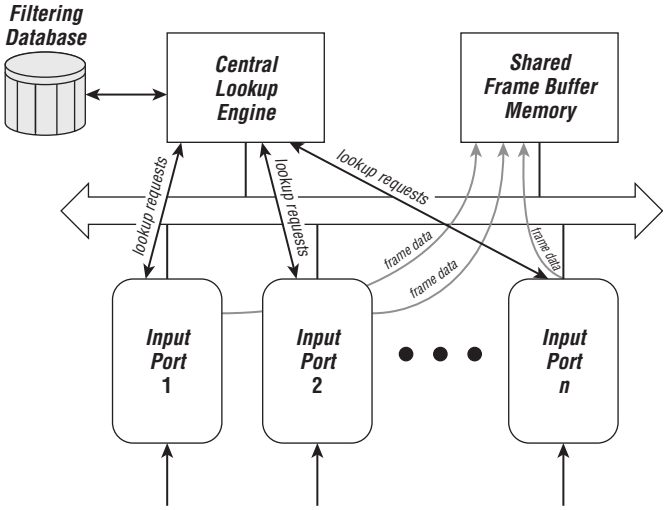


Figure 17-7 Centralized Lookup Engine

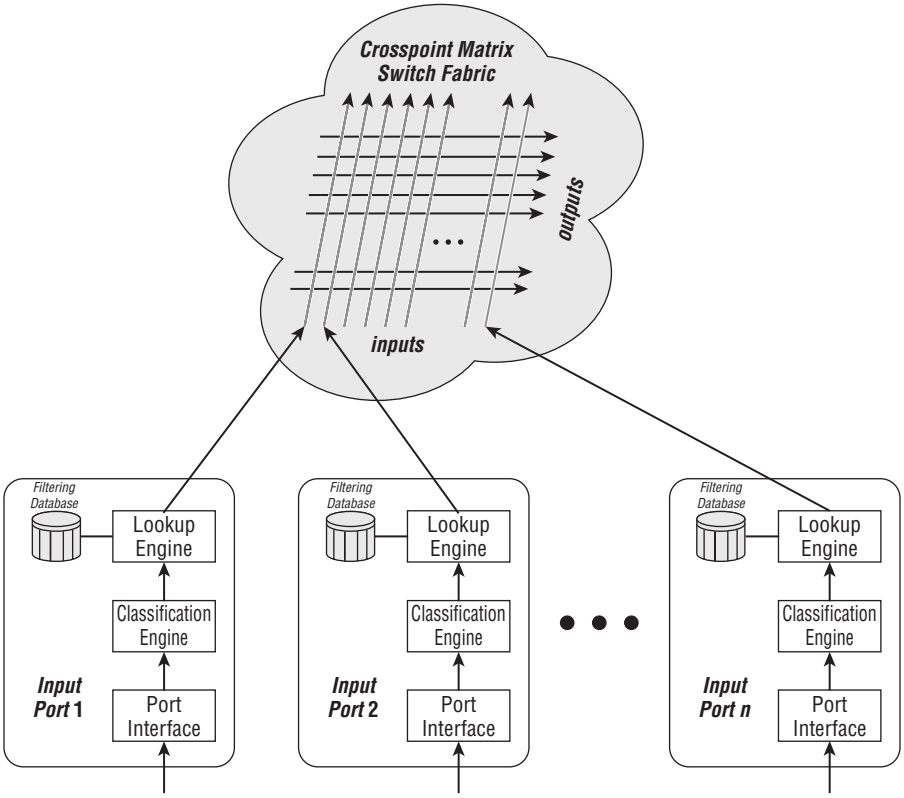


Figure 17-8 Distributed Lookup Engine

17.3 Switch Fabrics

The Receive data path performs all of the necessary qualification, classification, and table lookup functions needed to determine to which output port(s) a given frame should be forwarded. The Transmit data path takes these forwarded frames, applies additional qualifiers as needed, implements any prioritization and distribution policies, and sends the frames onto the appropriate output(s). A *switch fabric* sits between the Receive and Transmit data paths; its function is to transfer frames among all of the input and output ports of the switch. The design of the internal switch fabric is critical to the performance of a switch. While a comprehensive treatment of this subject could fill its own book, in this section we look at three switch fabric architectures that have been widely used in commercial LAN switch products:

- Shared memory
- Shared bus
- Crosspoint matrix

Each approach reflects a distinct method used to move frames from input to output ports, and has its own set of characteristics, limitations, and design issues.

17.3.1 Shared Memory

A shared memory architecture entails the lowest level of both design and implementation complexity, and usually provides the lowest-cost solution in those products where it can be used. As a result, it is the most popular approach to fabric design in LAN switches. The primary limitation of a shared memory switch fabric is the memory bandwidth; the speed at which data can be moved into and out of the shared memory will place an upper bound on both the number of ports in the switch and the data rates that can be supported on those ports.

17.3.1.1 Shared Memory Fabric Operation

A shared memory fabric uses a single common memory as the exchange mechanism for frames between ports, as depicted in Figure 17-9. Frames arriving through an input port's Receive data path are stored in the shared memory, where they can be assigned and channeled into the Transmit data path of the appropriate output port(s). Ideally, frames might be deposited directly from the receive port interface into the shared memory. In practice, there is usually some amount of per-port memory required in the Receive data path where frames are stored temporarily during reception and classification. However, the bulk of the system memory is in the common buffer pool.

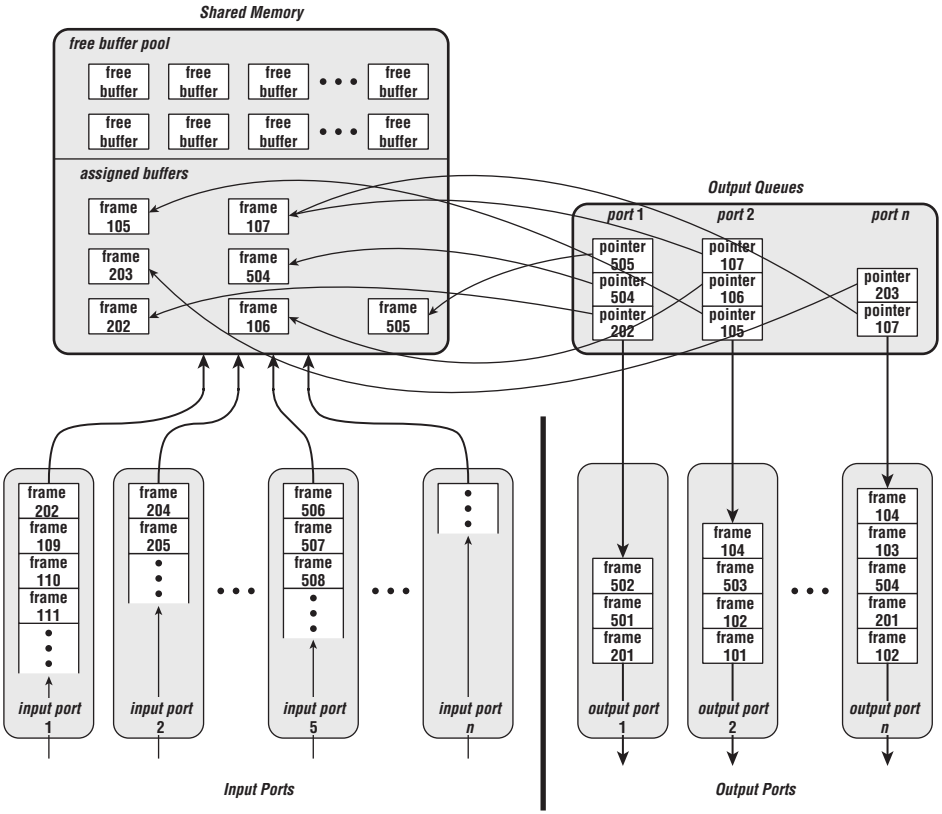


Figure 17-9 Shared memory architecture

Initially, all of the frame buffers in the shared memory are unused. They begin their life in the pool of free buffers. Free buffers are assigned to frames as they arrive from the various input ports. Depending on the results obtained from the Lookup Engine, frames in the shared memory are linked into the appropriate output queues for the ports onto which they will be forwarded. These output queues represent the stream of frames to be processed in the Transmit data path (see section 17.4). Frame discard is affected by returning a buffer to the free pool.

The following are advantages to a shared memory approach:

- Using a single large memory rather than individual buffer memories for each port lowers the total memory (and memory control logic) cost.
- The task of moving frames from one port to another becomes trivial. As shown in Figure 17-9, each port's output (transmit) queue comprises a linked list of buffer pointers representing frames stored in the shared memory. To move a frame from an

input port to an output port simply entails linking a pointer to the appropriate frame buffer into the target output port's queue.

- The Lookup Engine can be implemented either as a single, common engine for all ports or in a distributed manner, with each port performing lookups for its own received frames. This is a function of the performance demands being placed on the Lookup Engine (see section 17.2.6). Other fabric architectures (for example, crosspoint matrices, see section 17.3.3) make the use of a common Lookup Engine problematic.
- Lookup on a given frame may be completed before the frame is stored in the shared memory, or alternatively, frames may be stored while the lookup operation is performed in parallel. That is, a frame may be stored before the switch knows what to do with it; its buffer pointer can be linked to the appropriate output queue(s) once the Lookup Engine completes its work.

WHAT'S THE BIG DEAL ABOUT MEMORY ORGANIZATION?

In most computer contexts, we gladly accept greater complexity in the memory data structures in exchange for better efficiency. Why are we so concerned over the organization of the buffers in switch memory?

A computer program can easily manipulate complex memory data structures; other than requiring additional care and thought in the coding of the program, there is no cost impact associated with the implementation of discontiguous, variable-length buffers as linked-list queues in software. For performance reasons, high-speed switches cannot implement the shared memory data structures in software. All of the buffer manipulation and queue management must be performed by dedicated hardware — typically a set of finite-state machines — within the switch. Any complexity imposed on the organization of the memory buffers results in more complex and expensive logic for the memory interface. Thus, there is a real cost tradeoff between memory efficiency and data structure complexity that must be addressed by the switch fabric designer.

17.3.1.2 Multicasting in a Shared Memory Architecture

A shared memory fabric facilitates the ability to move frames from an input port to multiple output ports. Because the common memory is accessible from all ports, there is no need to perform multiple transfers or to provide special data paths for multicast traffic. Each frame buffer can be provided with a Transmit Port Map, as depicted in Figure 17-10.

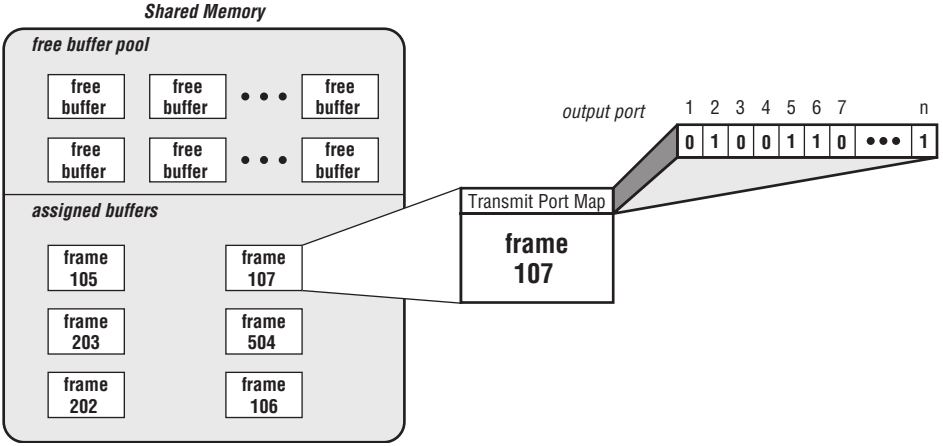


Figure 17-10 Transmit Port Map

The Lookup Engine determines the port(s) onto which the frame should be forwarded for transmission. If a frame is destined for more than one output port (for example, it is a multicast or unknown unicast frame), the appropriate bit for each target output port is set to 1 in the Transmit Port Map. When the output port Transmit data path no longer needs a copy of the frame in the shared memory (either because it has completed transmission or because it has copied the frame into a local port buffer), it clears the bit for its port in the Transmit Port Map and checks whether there are any bits still set. Once all of the bits in the Transmit Port Map are clear, the frame buffer can be returned to the free pool.

17.3.1.3 Buffer Organization

The shared memory is organized as a sequence of frame buffers (including any memory in the free buffer pool). However, there is no strict requirement that a given frame must be stored in one contiguous block. Frame buffers may be either contiguous or discontiguous. Each approach has its own set of advantages and disadvantages, as discussed in the following text.

17.3.1.3.1 Contiguous Buffers

The design of the memory interface and control logic is much simpler when buffers are contiguous. A contiguous buffer can be identified by a single pointer representing the location of the start of the frame. A Direct Memory Access (DMA) engine used to move frames into and out of the shared memory can work from this single buffer pointer. There is no need to scatter or gather frame fragments into disjoint portions of the shared memory.

The greatest simplicity occurs when the frame buffers are both contiguous and of fixed length. Using this scheme, the entire shared memory can be pre-allocated into distinct, numbered frame buffers. There is never any need to provide complex dynamic memory allocation when taking this approach. However, each fixed-length buffer must be large enough to store a maximum-length frame; to the extent that the received frames are smaller than the maximum possible length, the efficiency of memory utilization will drop. Thus, there is a clear tradeoff between the cost of the memory and the cost of the memory interface logic.

One common approach used in Ethernet switches is to allocate fixed-length contiguous buffers of 2,048 (2K) bytes. A 2-Kbyte buffer provides more-than-adequate storage for a maximum-length Ethernet frame (1,522 bytes, including VLAN tag), plus the internal switch header, Transmit Port Map, and any implementation-specific memory control information. In fact, the 526 bytes available for this internal overhead are much more than are generally necessary. However, if a round number (in binary) is used for the buffer length, the memory interface logic can use the frame number as the most-significant bits of each memory buffer pointer, as depicted in Figure 17-11. This represents a further simplification of the hardware logic at the expense of lower memory utilization.

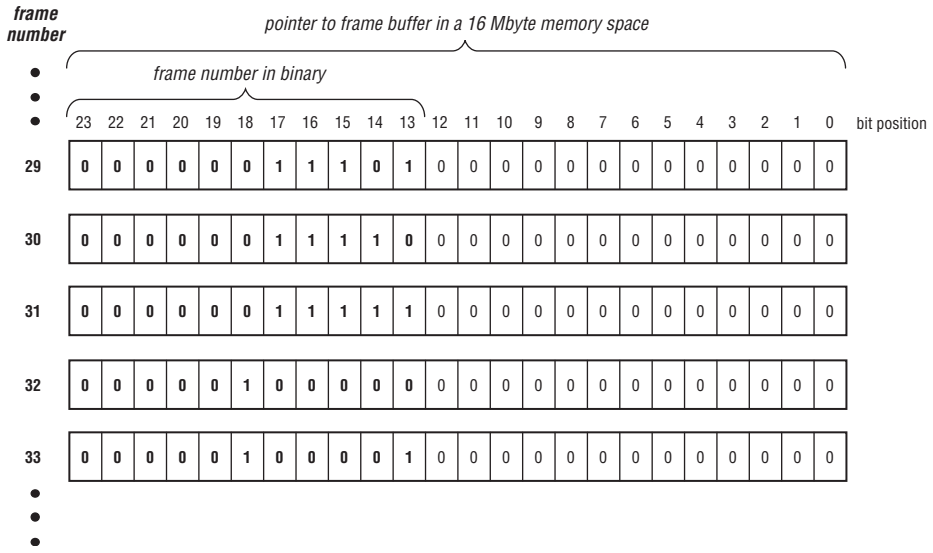


Figure 17-11 Using frame numbers as buffer pointers

While it is possible to design a shared memory switch fabric that uses variable-length contiguous buffers, this will generally result in a significant increase both in memory interface complexity and memory fragmentation.

As a result, if the system cannot tolerate the memory efficiency afforded by contiguous, fixed-length buffers, designers generally opt for a discontiguous buffer allocation scheme because this provides both higher memory utilization and variable-length buffer support without incurring significant memory fragmentation problems.

17.3.1.3.2 Discontiguous Buffers

Rather than storing a frame in a single block of memory, the use of discontiguous buffers allows a single frame to be stored as a series of frame segments scattered throughout memory. Thus, the start of a frame is identified by a single buffer pointer, but that pointer only indicates the first of the frame's possible segments. The entire frame is referenced by traversing a linked-list of buffer pointers, as depicted in Figure 17-12.

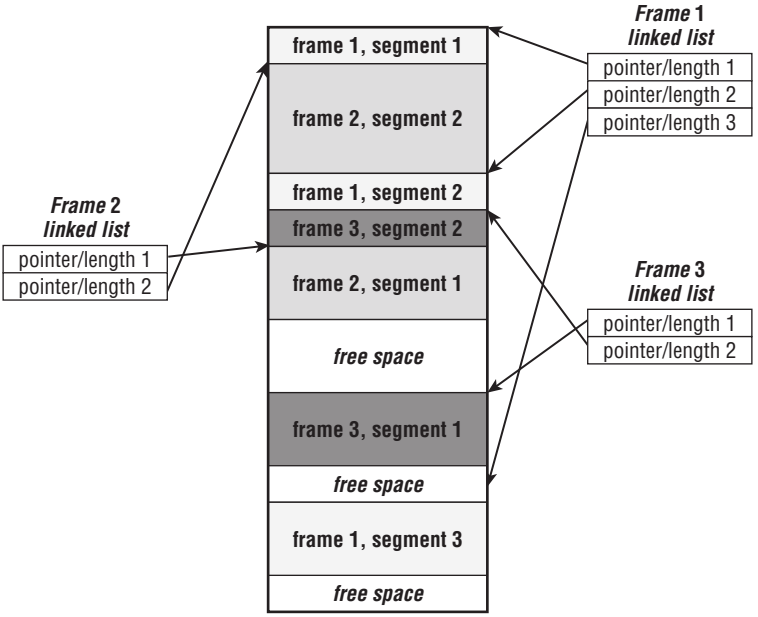


Figure 17-12 Discontiguous buffer pointers

The linked list of memory pointers may be stored either in the output queue data structure (as shown in Figure 17-12) or within the frame data buffer segments themselves by having the end of each data buffer segment provide either the next memory pointer or a flag indicating the end of the linked chain.

Each frame segment can be variable in length. Usually, there is some minimum value supported by the hardware, with the total length of each segment

being some multiple of this minimum. A discontiguous variable-length buffer approach uses memory much more efficiently than any fixed-length buffer scheme. In the worst case, the amount of memory allocated that goes unused is one minimum-length buffer less 1 byte, rather than a maximum frame.[17]

A linked-list buffer scheme can handle much larger frames than a fixed-length approach without sacrificing memory efficiency. Token Ring and FDDI LANs can support longer frames than can Ethernets; while the memory inefficiency of using fixed-length buffers is often tolerated for 1,522-byte Ethernet frames, an increase in memory interface complexity may be justified for the 18-Kbyte frames used in some Token Ring systems.

17.3.1.4 Memory Bandwidth Limitations

Quite simply, if a system can use a shared memory switch fabric, then it probably *should* use a shared memory fabric. A shared memory architecture is usually the simplest and lowest-cost solution available to the switch designer. Given this conclusion, why would any switch designer not choose a shared memory approach?

Because all frames must pass through the single, common shared memory, the speed of the memory interface will ultimately limit the capacity of the switch. For example, consider a memory system with a 32-bit wide data path, using 10 ns synchronous static RAM (SSRAM) and a 100 MHz clock. Each frame must pass across the memory interface at least twice — once when written from the input port and once when read into the output port.[18] This effectively halves the available memory bandwidth. Even in the best case of streaming data (i.e., 10 ns memory access times can only be achieved for multiple sequential reads or writes of data, not for individual random accesses) the aggregate switch capacity is limited to:

$$32 \text{ bits} \times 100 \text{ MHz} \div 2 = 1.6 \text{ Gb/s}$$

In practice, we cannot get the full benefit of data streaming through synchronous RAM; the actual performance improvement due to streaming will be a function of the design of the messaging system used to move frame data into and out of the memory. Typically, the usable memory bandwidth will be on the order of 50 to 70 percent of the maximum. Furthermore, to prevent unbounded queuing delays, the effective memory bandwidth should be undersubscribed by a factor of at least 30 percent. This results in a usable non-blocking switch

[17]It is common practice to use a 64-byte minimum segment length when using a discontiguous buffer approach in an Ethernet switch because this is the length of the minimum Ethernet frame. The larger the minimum length of a segment, the fewer links will be required for a given frame, at the expense of somewhat reduced memory efficiency.

[18]Multicast traffic may have to be read into multiple output ports, as discussed in section 17.3.1.2.

capacity on the order of 35 to 50 percent of the raw memory bandwidth, or about 550 to 800 Mb/s in the example just given.

Assuming full duplex operation on all ports, this capacity is adequate for a 10 Mb/s switch with a port count of 24 to 48 or more, or even a 48-port 10 Mb/s switch with two 100 Mb/s uplinks.[19] The switch cannot provide non-blocking operation for more than eight 100 Mb/s ports. Thus, the design example given is appropriate only for desktop or small workgroup switch applications.

17.3.1.5 Increasing the Memory Bandwidth

A number of approaches can be taken to increase the memory bandwidth so that a shared memory fabric can be used for higher-performance switches:

- **Use the fastest memories available:** Clearly, faster memory products will allow the system designer to use higher clock speeds without incurring wait states. However, there is always a physical limit to the speed of semiconductor RAM, based on currently available technology. In addition, the fastest memories are always the most expensive, negating some of the benefit of using a shared memory architecture.
- **Use a wider memory data path:** While the memories themselves may be clock rate–limited, the total memory bandwidth can be increased by widening the data path. As more bits can be transferred in a single clock cycle, the memory bandwidth will increase proportional to the width of the memory interface. In our example, changing to a 64-bit-wide data path would double the memory bandwidth. This would allow the fabric to support a non-blocking 100 Mb/s workgroup switch with 12 to 16 ports.

  Standard memory products are typically designed for use in 16-, 32-, or 64-bit-wide data paths. In addition, advances in memory technology tend to be directed toward achieving greater memory density — that is, stuffing more and more memory into each chip. The smallest SSRAM memory chip configurations today provide about 1 megabit of depth for each bit of width. Thus, a 64-bit-wide memory cannot generally be built with less than 8 Mbytes of total capacity (64 Mbits). If more memory bandwidth is needed, you can go to a 128- or 256-bit-wide data path; however, this will double or quadruple the total memory size (to 16 or 32 Mbytes) as a consequence of using standard memory chips. In many cases, the result is higher memory bandwidth, but with more total memory than is actually needed by the switch. Because conventional memory products are

[19] If the ports are operating in half duplex mode, the design can either support more ports or provide additional margin for non-blocking operation.

not optimized for such wide-yet-shallow memory arrays, you end up having to pay for more memory than you really wanted just to get a faster path.

- **Use non-traditional memory:** Rather than taking the brute force approach of simply widening the data path of a conventional memory array, there are alternative memory designs that provide higher interface bandwidth, including:
  - **Synchronous Graphic RAM (SGRAM):** As with LAN switches, graphic display subsystems often need memory configurations that are wider and shallower than traditional computer memories. Some semiconductor manufacturers provide specialty SGRAMs with this desirable characteristic. While they are occasionally used in LAN switches, they are more expensive than conventional RAMs because of their lower volume and limited market.
  - **Rambus Dynamic RAM (RDRAM):** Designed for use in high-performance PCs, workstations, and servers, RDRAM supports clock speeds of 600 MHz and more across the memory interface. Again, this higher-performance memory comes at a higher price, although RDRAM has the capability of extending the use of shared memory switch fabrics to much higher-end switch products.
- **Use embedded memory:** Even if you are willing to pay for a 32 Mbyte, 256-bit-wide SSRAM array to achieve a 4 to 6 Gb/s switch capacity, the sheer number of pins required to connect the memory array to the rest of the switch logic becomes problematic. Much of the cost of the custom-designed and manufactured ASIC devices used in high-performance switches is related to IC packaging requirements. If more than 256 pins are needed just to connect to the memory (there are a lot of arbitration, control, and power signals needed in addition to the memory data path lines), this will significantly increase the cost of the switch.

  An increasingly attractive alternative is to embed the memory within the switch ASIC itself. If the memory is internal rather than external, no pins (or their associated pads) are required. In addition, internal logic can be clocked much faster than external memories, because of lower lead inductance and reduced drive currents. Embedded memory data paths of 512 bits or more can be achieved at little or no cost penalty.

  For example, an 8 ns, 1,024-bit-wide embedded SRAM has a raw memory bandwidth of 128 Gb/s! Even if this bandwidth cannot be

used with perfect efficiency, such an array could easily support a non-blocking campus-class switch with 16 to 24 Gigabit Ethernet ports. In addition, the memory size can be tailored to the exact requirements of the switch. There is no need to stay within the strictures of merchant memory chips. Of course, embedded memory comes with its own set of problems:

- Large embedded memories can take up much of the available die space of an ASIC. SRAMs in particular are space-intensive; DRAM takes less space, but can generally run only at much slower clock speeds.
- Embedded memory is more expensive bit-for-bit than merchant memory chips.
- Not all semiconductor manufacturing processes can support embedded memory, which may limit the available suppliers.

Problems notwithstanding, as semiconductor densities increase and embedded memory becomes more of a commodity technology, this approach will become increasingly attractive for new switch designs and will extend the use of shared memory architectures to even higher levels of performance. Even when it is impractical to use for the switch fabric itself (because of the large amount of memory typically required), embedded memory may be used for the output port queue data structures or the internal storage required by the Lookup and/or Classification Engines (see sections 17.2.4 and 17.2.6).

17.3.2 Shared Bus

In the old days, memory was much slower than it is today, and most embedded processor systems were designed around 16-bit-wide internal data paths. In that environment, the available switching capacity of a shared memory fabric would be quite limited, typically on the order of about 150 Mb/s.[20] Thus, shared memories were practical only in 10 Mb/s switching systems with low port densities (15 or fewer ports for non-blocking operation). If a switch comprised a greater number of ports, or ports operating at higher data rates, it was usually necessary to change the fabric design.

A shared bus architecture (depicted in Figure 17-13) uses a common bus as the exchange mechanism for frames between ports, rather than a common memory.

[20]This is for a system using 25 ns SRAM and a 16-bit data path, after allowance for overhead, under subscription, and two memory operations (write/read) per frame.

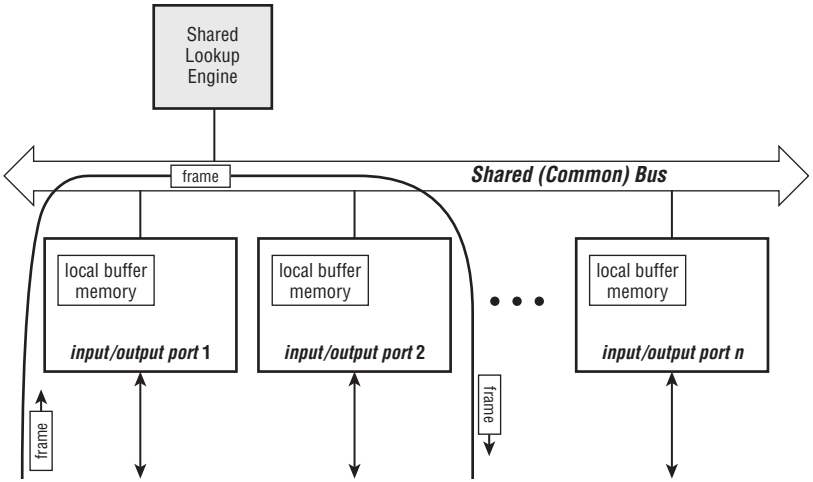


Figure 17-13 Shared-bus architecture

Each port (or small group of ports) has its own memory, both for input and output queues, depending on the design. The main advantages of this approach are:

- A shared bus can often provide higher bandwidth than an older-style shared memory. For example, a 32-bit, 33-MHz PCI has a theoretical maximum capacity of slightly over 1 Gb/s — more than 7 times the 150 Mb/s available from the preceding example. Proprietary buses can be used to achieve even higher capacities through the use of wider bus widths or faster clocks. A 1 Gb/s shared bus architecture can support a moderate number of 100 Mb/s ports (typically eight) and/or a very large number of 10 Mb/s ports.
- A shared bus is accessed only once per transaction (rather than once in, once out as in the shared memory case), so the effective capacity is doubled relative to the shared memory approach.
- Shared bus fabrics provide a natural mechanism for multicast transfers; it takes the same amount of time and bus capacity to transfer a frame to a single output port or to multiple output ports.
- A shared bus architecture can support either a distributed Lookup Engine (i.e., one engine on each interface) or a shared, common engine, depending on the performance demands of the system. A shared engine can be connected either to the same bus as is used for data transfers (as depicted in Figure 17-13) or to a separate, dedicated control bus. In the former case, some bus capacity is used to transfer header information to the Lookup Engine (and lookup results back to the requesting interface); this capacity cannot also be used for data transfers. The latter

case imposes the greater cost and complexity of providing two bus structures.

The disadvantages of a shared bus architecture include:

- **A separate memory is needed for each port (or group of ports):** As there is no common, shared memory among ports, each interface module must provide local memory for both input and output-queued frame storage. The memory on each interface must be sized for the anticipated worst-case load through the port. Depending on traffic patterns, some of the ports may become memory-starved while ample free buffers are available on other port interfaces. When all of the frames shared a common buffer pool, memory could be used more efficiently; that is, the statistical nature of load distribution allows a shared memory to support more traffic for a given memory capacity. Thus, for a given level of performance, a shared bus architecture will require more memory than a shared memory system, which increases cost.
- **Some means is required to arbitrate for the shared bus:** A shared bus appears very much like an internal, high-speed, shared-media LAN. As such, each port must contend for use of the bus. Once access is granted, the port uses the bus to transfer a frame from its local memory (input queue) to the local memory (output queue) of the target destination port.
- **Multiple input queues may be required:** Depending on the worst-case bus latency, there may be a need for a prioritized arbitration mechanism and/or multiple input queues at different priority levels. This effect is much more prevalent in a crosspoint matrix fabric, and is discussed in detail in section 17.3.3.5.

The product literature for many early modular internetworking devices prominently displayed and touted their backplane capacity. This was usually a clear sign that the system used a shared bus architecture; the capacity of the shared backplane bus guided the potential user as to the number and data rates of the interfaces that could be configured in the system while sustaining non-blocking operation.[21]

Shared bus architecture does not scale well; it is generally not possible to increase either the data rate or the bus width without redesigning all of the switch port interfaces (in particular, any ASIC devices designed to connect to, and arbitrate for, the shared bus). Most modern switches use either a shared memory or a crosspoint matrix fabric; rarely is there much call for

[21] For example, the first family of products from Wellfleet (later Bay Networks, then Nortel Network) used a standard VMEbus as the shared backplane. Operating with a 10 MHz clock and a 32-bit-wide bus, the system had an aggregate capacity of 320 Mb/s; this was considered quite a high-performance device at the time (late 1980s).

a shared bus anymore. As memories have gotten cheaper, faster, more sophisticated, and/or embedded, the small (if any) improvement possible from a shared bus is usually not enough of a change to provide sufficient benefit.

17.3.3 Crosspoint Matrix

As depicted in Figure 17-14, a crosspoint matrix switch fabric creates a transient connection between an input and an output port for the duration of a frame (or subset of a frame) exchange. There is effectively a matrix of electronic switches available to create a connection between every possible input and output port pair. The fabric control logic connects a given input to the appropriate output(s) on a frame-by-frame basis, in a matter of nanoseconds, and then disconnects the ports in preparation for the next frame exchange. A crosspoint matrix fabric is often called a *crossbar*; the terms are synonymous.

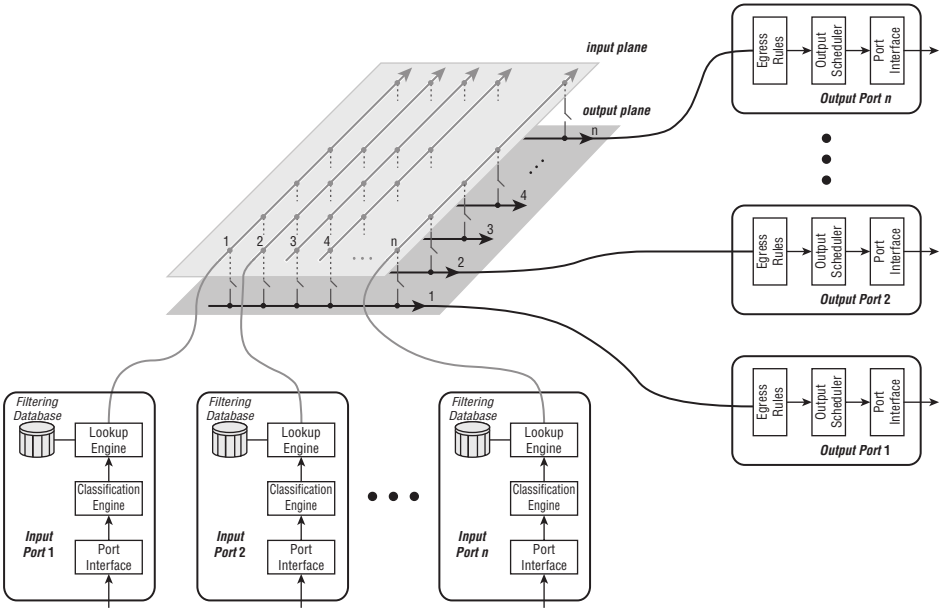


Figure 17-14 Crosspoint matrix fabric

Because every input port can be connected to some output port at the same time, the total data transfer capability of the crosspoint matrix can be much larger than that of a shared bus, and actually grows with the number of ports.

17.3.3.1 Multicasting in a Crosspoint Matrix Fabric

Unlike a shared memory or shared bus architecture, a crosspoint matrix provides no natural means to transfer frames from one input port simultaneously

to multiple output ports. There are two common approaches to dealing with multicast traffic in a crosspoint matrix:

- **Use multiple unicast transfers:** Simply put, any time a frame needs to be transferred to multiple output ports, the input port can execute a series of individual transfers to each target across the fabric. Just as in the shared memory approach, the input port maintains a Transmit Port Map indicating the set of output ports to which the instant frame must be transferred. As fabric arbitration and frame transfer proceeds, each target output port is checked off the list until the Transmit Port Map is all zeroes, after which the frame buffer can be released by the input port.

  The advantage of this approach is that it simplifies the design of the crosspoint matrix switch fabric. If multicast frames are propagated as multiple unicast transfers, there is no need to make any special accommodation within the fabric to handle the multicast traffic. The disadvantage is the increased input port complexity required to keep track of the progress of multicast transfers.

  While it may seem wasteful of fabric capacity to transfer the same frame multiple times, consider that the frame in question will be using bandwidth on all of the target output ports anyway.[22] If the fabric is designed to support the sum of the capacities of all ports combined, it will still be non-blocking when multicast frames are transferred separately to each output port. Any overprovisioning of capacity that was factored into the design will be reduced because of the frame replication caused by multicast traffic. The design can still handle the total capacity, but queuing delay (switch latency) will increase to the extent that multicast traffic comprises a significant portion of the total offered load.

- **Provide a dedicated multicast data path:** Alternatively, a switch can provide one (or more) logical output ports that are used specifically for multicast traffic. The multicast port is effectively a shared bus that is routed to each of the output ports in addition to the unicast line for that port, as depicted in Figure 17-15.

  This approach makes sense when multicast traffic is expected to be a small portion of the total offered load and the switch fabric is blocking in nature or has little excess capacity. By offloading the multicast traffic from the individual port outputs on the switch matrix, we avoid the frame replication discussed earlier and dedicate most of

[22]Some frames may not actually be transmitted at the output because of the operation of the Egress Filter (see section 17.4.1), but this is generally a very small fraction of the total traffic and can be ignored.

the capacity for unicast traffic, which is assumed to comprise the bulk of the frame load. The disadvantages of this approach are:

- Each output port must be equipped to receive frames from its dedicated unicast output line from the switch fabric as well as from the shared multicast port output.
- The capacity of the shared multicast bus may be a limiting factor if multicast traffic comprises a significant fraction of the total load.[23]

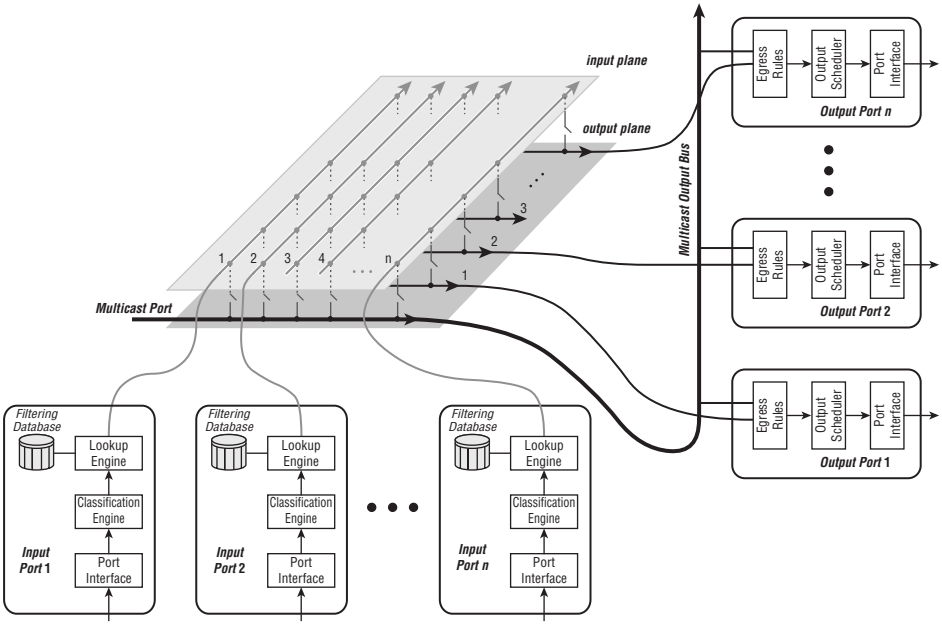


Figure 17-15 Shared multicast port

17.3.3.2 Crosspoint Matrix Implementation

The two popular approaches to implementation of the crosspoint matrix are as follows:

- **A physical matrix:** This matrix can be created by laying down strips of conductive material in parallel on one plane (input ports) and strips of conductive material at right angles to the input

[23] At least one switch fabric IC offers the capability to make a tradeoff between the amount of shared multicast capacity and the number of unicast ports that can be used. That is, the system designer can assign any of the fabric's output ports for multicast use; the shared multicast bus aggregates the selected number of multicast ports into a single logical channel. Those ports not assigned to multicast service can be used as unicast outputs.

ports on a second plane (output ports).[24] Each input port trace is connected to every output port trace by a transistor switching circuit; the switch control logic decides which of the transistor switches are closed or open based on the requests made by the input port controllers and the resolution of those requests by the fabric arbitrator (see section 17.3.3.5).That is, the depiction in Figure 17-14 represents the physical configuration of the crosspoint matrix.

Depending on the data rate supported by the fabric, it is often necessary to use multiple material traces for each port (both input and output). Multiple bits of data can then be transferred at each clock tick, which reduces the clock speed for a given rate of data transfer at the expense of increasing the pin count for the switch ports on the fabric chip.

- **A logical matrix:** This matrix can be created by defining the signals for each output port as a logic function derived from the signals at the input ports, under control of the fabric arbitrator. That is, a crosspoint matrix can be built from standard digital logic blocks. As with the physical matrix, clock speeds can be reduced by defining each input and output in terms of multiple parallel signals at the expense of increasing the pin count at the fabric interface.

In general, a physical matrix can be smaller and less complex, resulting in a lower-cost switch fabric. The disadvantage is that the integrated circuitry must typically be custom-designed and laid out. It is not generally possible to build a physical matrix in silicon using the conventional IC design tools used for synthesized logic. The advantage of a logical matrix is that because it uses synthesizable logic it becomes much more portable across semiconductor manufacturing processes and requires less specialized design skills.

Depending on the clock speed and the width of the port interface to the matrix, a huge switching capacity can be created. For example, a 16-port, 16-bit-wide (16 signals per port) matrix running on a 100-MHz clock has an effective data transfer capability of 25.6 Gb/s. Such a fabric would be suitable for huge numbers of 100-Mb/s ports and/or quite a few Gigabit Ethernet ports.[25]

17.3.3.3 The Head-of-Line Blocking Problem

Crosspoint matrices can provide an extremely port-scalable, high-capacity switch fabric. However, they suffer from a systemic problem known as *head-of-line blocking*. Consider what happens when multiple input ports must

[24]Normally, polysilicon is used for the conductive material in an integrated circuit.

[25]An ASIC implementation of such a crosspoint matrix would require (16 × 16 × 2 =) 512 pins just for the input and output port connections. Additional pins would be needed for switch control, power, and so on.

contend for the availability of a single output port. Figure 17-16 depicts a situation where a lot of traffic is destined for output port 9; ports 1, 2, and 3 each have frames queued and waiting for port 9 to become available. Assuming that the switch arbitration algorithm is fair, those frames will be transferred to port 9 in an orderly manner. The switch capacity will be evenly distributed among all of the input ports that have traffic destined for the single target output port.

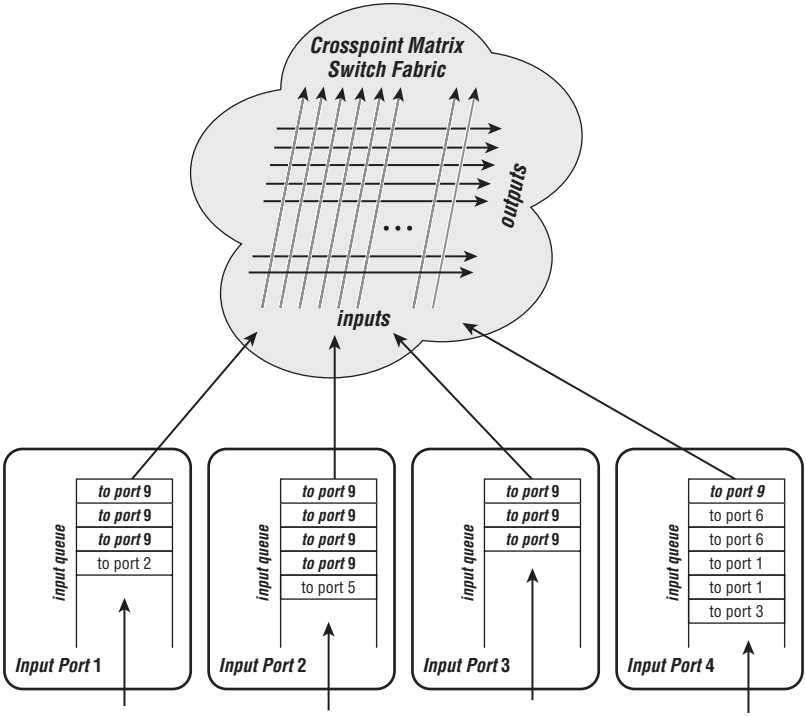


Figure 17-16 The head-of-line blocking problem

The problem is with port 4. Port 4 has frames queued and waiting to be transferred to a variety of output ports. However, the first frame in the queue is destined for output port 9, which we know to be experiencing short-term congestion. Because only one input port at a time is able to transfer frames to output port 9, each input port with a frame queued for port 9 must wait its turn. Assuming fair, round-robin scheduling, port 4 may have to wait for ports 1, 2, and 3 to transfer their respective frames to port 9 before it gets to unload the frame at the head of its queue. In the meantime, there are frames waiting in the input queue of port 4, behind the frame destined for port 9. These frames are targeted at uncongested ports; that is, the target output ports

(ports 6, 1, and 3 in Figure 17-16) are idle and awaiting traffic. However, port 4 cannot proceed to transfer those frames because the frame at the head of the line is blocking progress.

Note that this situation is not a result of any flaw in port 9. There is a transient overload condition where the traffic targeted at port 9 temporarily exceeds the capacity of the output link. The switch is (properly) resolving that short-term congestion by spreading the offered load over time. During that spreading time, frames must wait in queues within the switch. Ideally, we would like the frames to wait in the output queue for port 9, but we need to transfer frames across the switch fabric to achieve that state. To the extent that the switch fabric capacity is also experiencing transient congestion because of the many-to-one traffic pattern, some of the frames are still in the input queues waiting to cross the fabric. It is this condition that leads to the blocking problem.

Head-of-line blocking reduces the aggregate throughput of the switch as a function of traffic patterns. If some ports regularly experience short-term congestion (for example, ports connected to servers that are often the target for heavy offered load), throughput can be reduced on other ports that are completely uncongested. This is an undesirable situation. There is no good reason to reduce the throughput between ports 4 and 6 just because ports 1, 2, and 3 are sending a lot of traffic to port 9.

17.3.3.4 Solving the Head-of-Line Blocking Problem

A number of solutions to the head-of-line blocking problem have been implemented in practical switches, including:

- Queue lookahead
- Priority queues
- Per-output-port input queues

Each one differs in complexity of implementation and effectiveness in eliminating the blocking effect.

17.3.3.4.1 Queue Lookahead

One of the simplest approaches is to implement a lookahead mechanism on the input queue. Each input port arbitrates for the target output port as determined by the frame at the head of its queue. When multiple input ports contend for the same output port (for example, port 9 in Figure 17-16), one of the input ports will be granted access and the others will be temporarily blocked. When this occurs, the blocked input port can look ahead to the next frame in its input queue to see if the target output port needed by that second frame is available. If it is, the port can transfer that second frame rather than the one at the head

of the line. Once transferred, the input port can again arbitrate for the frame at the head of the queue and, if necessary, repeat the lookahead algorithm if the congested output port is still unavailable.

The hardware required for a simple one-stage queue lookahead is not especially complex and does not significantly increase the cost of the queue management logic. Of course, the mechanism is effective only if there is a single frame blocking the head of the line. If both the first and second frames in the queue are destined for congested output ports, then all of the other frames in the queue are still blocked (for example, input port 1 in Figure 17-16). However, the probability that two frames are both head-of-line blockers is lower than the probability that only one frame is causing trouble.[26]

Queue lookahead is most effective when the fabric arbitration mechanism provides a means to make a *standing request*, that is, to request an output port and have the switch provide notification when it is available without further action on the part of the input port. That way, the frame at the head-of-the-line will always be assured of delivery, regardless of the time distribution of the traffic destined for the congested output port. The input port can continue to process lookahead frames until the fabric arbitrator tells it that that (congested) output port has become available, ending the head-of-line block.

17.3.3.4.2 Priority Queues

An alternative approach to solving the head-of-line blocking problem is to equip each input port with multiple queues representing different priority levels, as depicted in Figure 17-17.

Frames are placed into one of the input queues based upon the priority as determined by the Classification Engine (and recorded in the internal switch header). The input port nominally schedules frames for fabric arbitration in priority order; that is, frames in the high-priority queue(s) receive preferential treatment relative to frames in the lower-priority queue(s).[27] Frames will be processed from lower-priority queues when all higher-priority queues are either empty or currently experiencing head-of-line blocking.

[26]The probability can be reduced further by incorporating a two-level lookahead, although this increases the hardware complexity. At some point (typically one or two levels), lookahead schemes provide diminishing returns, and it becomes more practical to implement one of the multiple-input-queue mechanisms discussed in sections 17.3.3.4.2 and 17.3.3.4.3.

[27]In theory, any of the mechanisms discussed could be used for output queue scheduling (for example, weighted fair queuing; see Chapter 13). In practice, strict priority is generally adequate and implies simpler queue management logic. A strict priority policy will starve lower-priority queues only when the sustained, steady-state, high-priority offered load exceeds the capacity of the switch fabric or its output ports. In a LAN switch environment, this is presumed not to be the case; sustained overload indicates a configuration problem (i.e., inadequate capacity) that cannot be resolved by complex priority mechanisms.

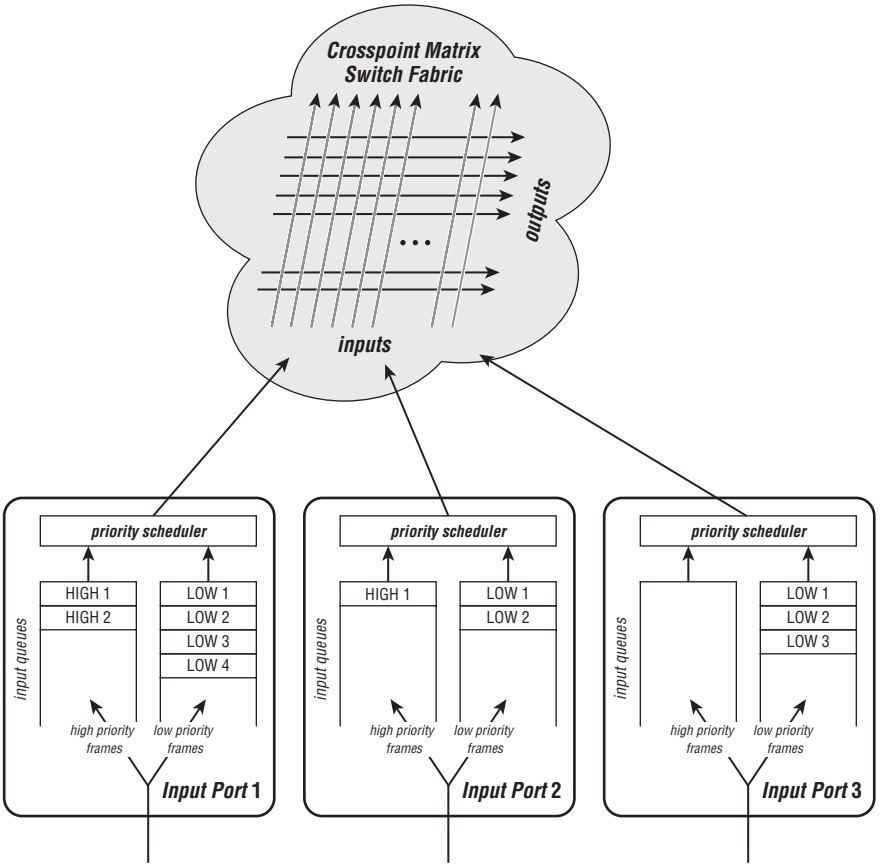


Figure 17-17 Priority input queues

Thus, head-of-line blocking may still occur, but at least a low-priority frame will never block one of higher priority. In addition, queue lookahead can be implemented in addition to priority queuing, further reducing the probability of blocking (especially for the higher-priority queues).

You should note the following regarding the implementation of priority input queues:

- Priority queues can be implemented on any input port without regard to whether this mechanism is being used on other input ports. That is, the fact that an input port has multiple queues and is selecting frames for fabric arbitration according to priority classification is invisible to the switch fabric or to other ports on the switch. Thus, to the extent that it is known in advance on which ports head-of-line blocking is anticipated, the additional cost can be incurred only when needed.
- Queue priority can be effectively combined with arbitration priority (see section 17.3.3.5). If the switch fabric itself supports arbitration at

multiple priority levels, the input queues can be mapped to the fabric arbitration priorities.

- The number of priority levels or queues provided does not have to be the same as either the number of priorities resolved by the Classification Engine or the number of Classes of Service provided at the output ports. The purpose of prioritization at the input queue is to avoid having high-priority traffic head-of-line blocked by lower-priority traffic. Regardless of input queue priority, all frames will still end up in an output queue and be scheduled according to their (end-to-end) user priority. Just as user priority is mapped to the available Classes of Service in the output queues, user priority is mapped to the available number of input queues and/or switch arbitration priority levels.

17.3.3.4.3 Per-Output-Port Input Queues

One complete solution to the head-of-line blocking problem is to provide an input queue for each possible output port, as depicted in Figure 17-18.

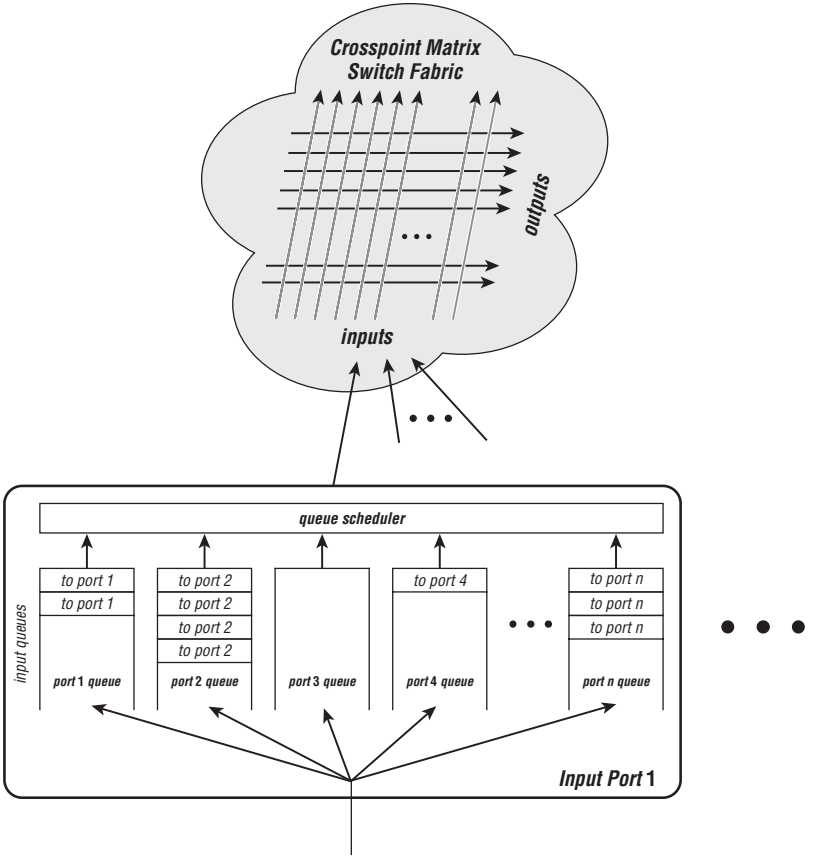


Figure 17-18 Per-output-port input queues

At the point where the Receive data path is ready to transfer a frame across the fabric, it must of course know the target output port. Rather than placing all of the frames waiting to be transferred into a single input queue (or a set of prioritized queues), a port can maintain a separate input queue for each possible output port. A scheduler selects frames from the non-empty queues in a fair (for example, round-robin) manner. If the target output port is available, the input port transfers the first frame in that queue across the fabric. If the target output port is busy, the input port moves on to the next non-empty queue. This mechanism completely eliminates head-of-line blocking because the unavailability of a given output port will never prevent traffic from moving to those ports that *are* available; the condition never occurs where a frame destined for an available port is waiting behind a frame that cannot currently be transferred.

As in the case of the queue lookahead scheme (see section 17.3.3.4.1), per-output-port queuing works best if the fabric arbitration mechanism provides a means to make a standing request, that is, to have the switch provide notification when a previously requested output port becomes available. The input port can continue to process frames for uncongested ports until the fabric arbitrator tells it that that (previously congested) output port is now available for use.

It is also possible to combine per-output-port queues with priority queues. Each output port can be served by multiple queues at different priority levels. On a per-port basis, high-priority frames will be given precedence over lower-priority frames, as discussed in section 17.3.3.4.2.

The price for this ultimate solution is not so much an increase in the total amount of input queue memory required — it is the same number of frames being queued either way — but in the complexity of the queue memory data structures and of the hardware used to manage and schedule those queues. The cost and complexity of a per-output-port queuing system can get quite high in a switch with a large number of ports. As always, we are riding the curve of semiconductor technology; as ICs become more dense and logic becomes less expensive, complexity of per-port queue mechanisms becomes ever more practical.

17.3.3.4.4 Arbitrating for the Fabric Connection

In a shared memory fabric, each port attempting to write into or read from the shared memory must by necessity arbitrate for the right to use the memory data path.[28] However, once access is obtained, any input port can store data destined for any output port, and an output port can read data written by any

[28]This arbitration is unrelated to the switching application; any shared memory requires a mechanism to allocate access among its multiple potential users.

input port. Similarly, the input ports on a switch must arbitrate for the right to post data onto a shared bus fabric, but the data transfer itself can be to any output port (or to multiple output ports simultaneously). Neither a shared memory nor a shared bus fabric embodies the concept of arbitrating for the right to send or receive data between a particular pair of switch ports.

A crosspoint matrix is different. Transient data paths are created between specific input and output ports as needed for data transfer. When an input port wishes to transfer a frame across a crosspoint matrix fabric, it must request access from itself to a specific target output port. In contrast, an input port in a shared memory or shared bus architecture only needs to request the use of the common fabric.

As depicted in Figure 17-19, crosspoint matrices must provide a mechanism for:

- Each input port to submit requests for a path to a selected output port
- Arbitrating among multiple simultaneous requests for the same output port
- Granting requests as a result of the arbitration
- Enabling the fabric control logic to create the desired path at the appropriate time

17.3.3.4.5 Arbitration Request Types

Under conditions of heavy offered load, it is important that the arbitration scheme operate both efficiently and in a fair and controlled manner. While the details of the arbitration logic tend to be highly product- and implementation-dependent, most crosspoint matrix arbitrators support one or more of the following types of fabric requests from the input ports:

- **Immediate request:** An input port can request access to a particular output port for immediate use. If the target port is available, the arbitrator responds by issuing a grant to the requesting input port. If not, the grant is denied, and no further action is taken by the arbitrator. The input port is free to make further requests for the same or different output ports at a later time.

  An immediate request might be appropriate for the first request made for a frame at the head of a queue equipped with lookahead capability (see section 17.3.3.4.1). If the request is granted, the input port can transfer the frame and proceed to the next frame in the queue. If the request is denied, the input port can make a standing request for the same output port followed by an immediate request for the next frame in the queue (lookahead).

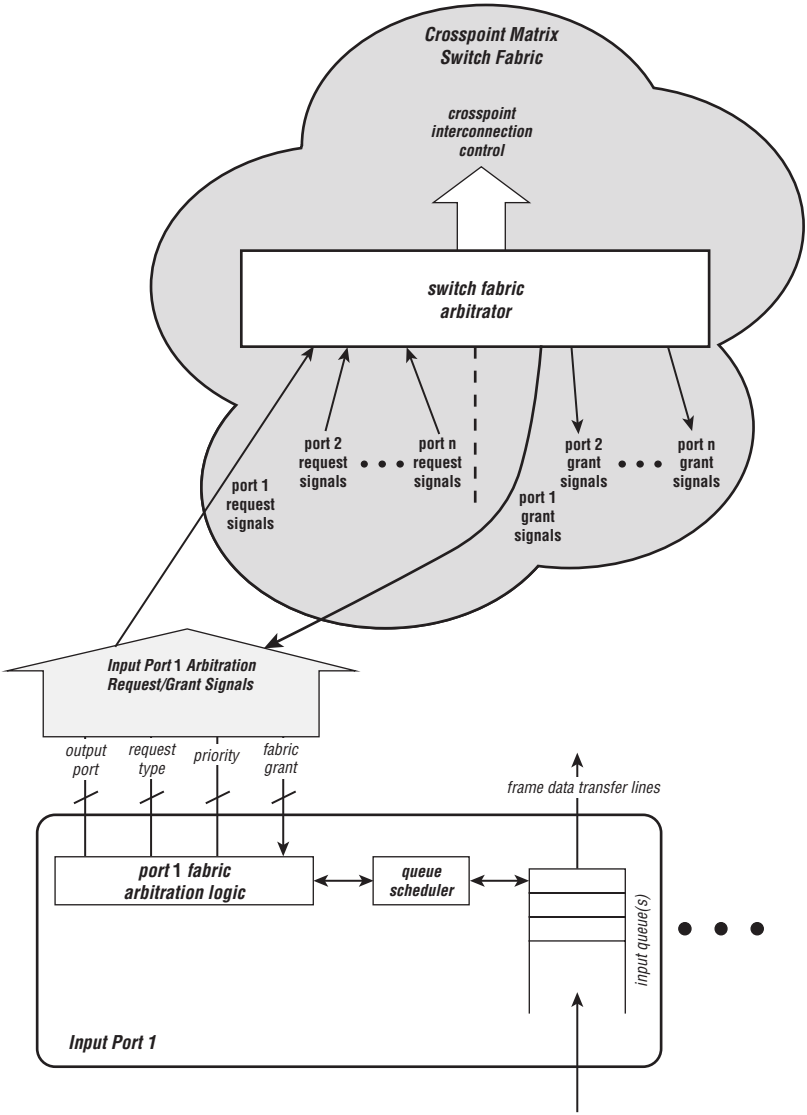


Figure 17-19 Crosspoint matrix arbitration

- **Synchronous request:** A second form of request asks for an output port and waits until the grant is issued. In this mode, the request-and-grant is a synchronous, atomic operation; no additional requests can be made by the input port until the outstanding request is granted (or times out because of some abnormal condition). Synchronous requests are appropriate for simple, single-input-queue systems that incorporate neither priority nor lookahead features. There is no need to provide additional port logic complexity if the design cannot take advantage of it.[29]

- **Standing request:** Some crosspoint arbitrators provide a means for an input port to request that it be notified when a desired output port becomes available. The arbitrator maintains an internal list of standing requests individually for each output port. When an output port becomes available, the fabric arbitrator decides which input port is to be granted access (based on the scheduling algorithm, priority level, and so on) and notifies the lucky winner that its request can be granted. The input port can then make an immediate or synchronous request, knowing that it will be successful.

  Standing requests are appropriate for more complex systems that support lookahead, priority, and/or per-output-port queuing. Any time an immediate request is denied, the input port can issue a standing request and move on to process a frame destined for a different output port. The input port will be notified when its standing request is fulfilled and can take appropriate action at that time.

  The capability to support standing requests imposes greater complexity on the logic both in the input ports and in the fabric arbitrator, but can provide significant improvements in switch fabric utilization and performance.

17.3.3.4.6 In-Band Versus Out-of-Band Arbitration

The implementation of a high-speed, high-port-density crosspoint matrix is often limited by the sheer number of pins required to move signals into and out of the fabric chip(s). To achieve data rates of 1000 Mb/s or higher, we often have to resort to 8- or 16-bit-wide interface paths. As the port density of the switching fabric increases, the number of pins on a switch chip can multiply into the hundreds or thousands.

In addition to the pins needed to move data through the fabric, we need a way to issue arbitration requests and grants between the input ports and the switch chip. Two approaches are commonly used to achieve this end:

- On switch fabrics that are not severely pin-limited (for example, those that have relatively low port density or that only support lower data rates, such that a 1- or 2-bit-wide data path can be used), we can simply add additional pins for the arbitration control. Each input port makes its requests on signal pins dedicated to the arbitration function; the switch can issue output port grants on similarly dedicated lines.[30]

[29]Time spent waiting for a grant against a synchronous request is exactly that time during which the input port is head-of-line blocked.

[30]In some switch designs, the grant signals are common to all ports. This reduces the number of pins required, but requires that the signaling on the output port grant lines be fast enough to accommodate the sum of the input port request rates with minimal delay.

- When pins are at a premium, we generally opt to make arbitration requests through the same data path used for frame transfers across the fabric. That is, output port requests are structured as ''frames'' being sent to the arbitrator within the fabric chip. Similarly, grant commands can be issued by the fabric arbitrator and sent to the respective ports through the fabric data output paths. In exchange for the reduction in pin count, we have:
  - **Increased the complexity of both the port and the fabric arbitration logic:** Information passing across the fabric interface must be parsed to separate arbitration requests and grants from data frame transfers.[31]
  - **Reduced the available bandwidth by using some of the data interface capacity for arbitration exchanges:** In practice, a switch designed for in-band arbitration signaling must include extra capacity to account for this effect.

17.3.3.5 Priority Levels in the Switch Fabric

Now, to make things *really* complicated, we can provide multiple priority levels for the switch arbitration. Under this scheme, an input port makes requests not only for a particular output port and request type but at a specified priority level. The arbitrator uses the request priority level as an input to the algorithm used to resolve contention for an output port. Higher-priority requests will be granted ahead of lower-priority requests for the same port. Note that the number of priority levels provided by an arbitrator is completely independent of any other priority mechanisms in the switch (for example, classification priorities or output queue Classes of Service). The purpose of prioritization here is to schedule the switch fabric in the manner that moves the most important traffic to the output ports in the most expeditious manner.[32]

17.3.4 Input Versus Output Queues

Regardless of the design of the fabric, any switch works best when the frames get to the output queue(s) as soon as possible. If there are frames in an output queue and the output port is available, those frames can be sent immediately. If the output port is available, but frames are still waiting on the input side of the switch fabric, the output port is unnecessarily idled. You can't send frames that are still sitting in an input queue. Thus, if frames have to wait, it is better

[31]Usually, some special codes or flag bits in the frame headers are used to uniquely identify arbitration requests. There is no need to parse or assign 48-bit addresses in the arbitrator.
[32]Typical fabrics provide two, four, or eight levels of priority.

to wait in an output queue than in an input queue. This maximizes channel utilization at the output.[33]

17.3.4.1 Input Queues and Shared Memory Switch Fabrics

Note that with a shared memory architecture there really isn't any input queue with respect to the switch fabric. Frames are stored in memory in one of two states:

- **The frame is awaiting processing by the Lookup and/or Classification Engine:** At this point in the process, we do not yet know the output port(s) to which we need to transfer the frame. The memory is being used for temporary storage within the Receive data path. There is no issue of head-of-line blocking on such frames because they are not yet queued for transfer across the switch fabric.
- **The frame has been inserted into the appropriate output queue(s) and is awaiting transmission:** Once the Lookup Engine has completed its task, it knows the output port(s) to which it wants to transfer the frame. This transfer is affected by linking the appropriate buffer pointer into the output queue for the target port(s). As this can be an atomic operation performed by the Lookup Engine, we never have frames awaiting transfer across the fabric for which we know the target output port. While there may be buffers within the Receive data path to provide time for frame processing, a shared memory architecture needs no input queue to the fabric itself.

17.3.4.2 Input Queues, Output Queues, and Flow Control

Recall from Chapter 8 that the PAUSE flow control mechanism allows a device to request that its full duplex partner stop sending frames for a period of time. Typically, this action would be taken because the instant device (for example, a switch port) sees that there will be imminent buffer overflow if frames continue to arrive.

Once frames are transferred to an output port within a switch, it becomes rather difficult to use the PAUSE mechanism to prevent buffer overflow. Frames on output queues have typically arrived from a multitude of different input ports. When the output queue begins to grow, how does the switch know on which input port to assert flow control? Without a crystal ball (or an amazing traffic-anticipation heuristic, which amounts to the same thing), it is impossible to predict the port on which future frames will arrive, causing a troubled output queue to overflow.

[33]This is a practical application of the ''hurry up and wait'' syndrome. We want to speed frames through the Receive data path processing and across the switch fabric, even if they will have to sit in an output queue waiting for transmission.

PAUSE flow control is effective only with an architecture that maintains input queues to the switch fabric. If an input queue is in danger of overflow, we know a priori that any overrun will be caused by frames arriving from the input port feeding that queue. Thus, we can apply high-water marks on the input queue and assert PAUSE flow control as needed.

Even with an input-queued system, flow control becomes problematic if link aggregation is being used. Because neither the fabric arbitration mechanism nor the aggregator's frame collector will typically know the algorithm being used by the link partner to distribute frames across the multiple links, it is not generally possible to determine which physical link within the aggregation will receive the frames that ultimately cause queue overflow. Unless we are willing to throttle an entire aggregation as a result of a single, traffic-intensive conversation, flow control can only be used to prevent overflow of the individual physical link queues within the frame collector, not the aggregated input queue to the switch fabric. IEEE 802.3x flow control was designed specifically to prevent buffer overflow under transient overload conditions in memory-constrained, input-queued switches. As a result of:

- Rapidly decreasing memory costs
- The emphasis on rapid transfer of frames from input ports to output queues in modern switch designs
- The widespread use of link aggregation

PAUSE flow control has become less useful and effective than originally intended. While it is often provided as a product feature in commercial switches, most users disable the function.

17.4 Switch Data Transmit Path Functions

Ultimately, the switch fabric delivers a stream of frames to each output port. These frames may have entered the switch from a single input or from a multitude of input ports. In any case, we need to perform some additional checks and processing before sending them onto the physical link. Figure 17-20 depicts the logical blocks present in the output port (transmit) data path. The sections that follow look at the operations performed within each of these blocks.

17.4.1 Output Filters

For each frame targeted at a given output port, we need to perform two VLAN-related qualification tests before actually queuing it for transmission. As discussed in Chapter 12, ''Virtual LANs: The IEEE Standard,'' the Egress

Filter determines whether the output port is in the member set for the VLAN to which the frame belongs. A frame should never be sent on a port that is not in the member set for its associated VLAN. Under most conditions, frames will never be forwarded through the switch fabric to output ports not in the VLAN member set. However, it is possible that, because of inconsistent static entries in the Filtering Database or the operation of Shared VLAN Learning, frames will appear at the output that should not be sent on that port. The Egress Filter detects and prevents this condition.

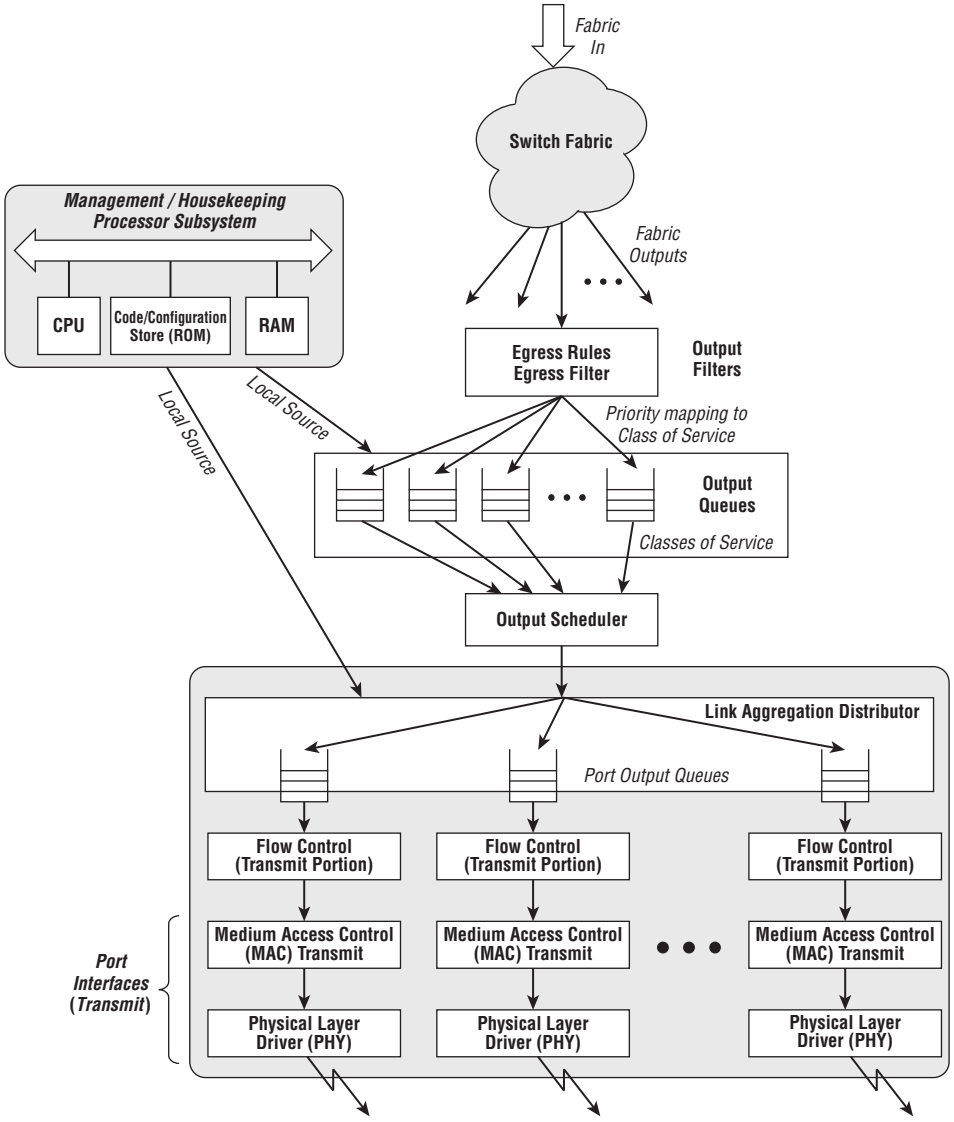
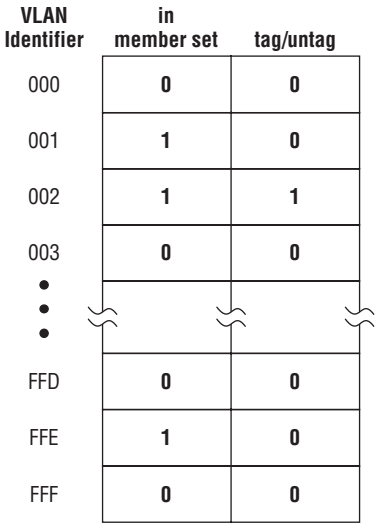


Figure 17-20 Switch output data flow (transmit path)

For each frame that should properly be transmitted on the output port, the Egress Rules function determines whether it must be sent tagged or untagged. This decision is made on a per-frame basis, as a function of the VLAN to which this frame belongs. For each VLAN for which this output port is in the member set, frames will be sent either all tagged or all untagged under administrative control. The setting of the Egress Rule is based on whether any tag-unaware devices must receive frames for this VLAN on the link to which this output port connects.

The Egress Rules and the Egress Filter can be applied simultaneously through the use of a simple output filter table (depicted in Figure 17-21). For each of the 4,096 possible values of the VLAN Identifier, 1-bit indicates whether the port is in the member set (Egress Filter), and 1 indicates whether frames should be sent tagged or untagged (Egress Rules).[34] Note that no complex lookup mechanism is needed; the VLAN identifier in the internal switch header can be used as a pointer directly into the output filter table. In addition, the output filter table requires only 1 Kbyte of memory.



Note: VLAN IDs 000 and FFF are preset to be excluded from the member set for any port, i.e. they are invalid VLAN ID values for frames transmitted by a switch.

Figure 17-21 Output filter table

Once the tagging decision is made, all of the information needed to create the tag (if necessary) is already present in the internal switch header.

[34]For a VLAN for which the port is not in the member set, the setting of the tag/untag bit is irrelevant.

The VLAN identifier, priority, and other information were all determined by the Classification Engine when the frame was received; at this point in the process we are only deciding whether to include those fields in the transmitted frame.[35]

In the steady-state, the configuration of the filtering database should prevent a switch from attempting to forward frames to ports that are not in the forwarding state according to the operation of the Spanning Tree Protocol. However, there may be transient or boundary conditions when the filtering database has not yet converged to a new, stable state; that is, the filtering database cannot always be depended upon to implement spanning tree port states. The Egress Filter can prevent unwanted transmission of data frames onto ports that are in inappropriate states. If a port is defined as not being in the member set of any VLAN, all frames forwarded through the switch fabric will be discarded at the output. The housekeeping processor can still submit BPDUs for transmission on ports that are in the *listening* or *learning* states by inserting them directly into the output queue following the Egress Filter (see the following text).

17.4.2 Output Queues and Priority Handling

Frames passing all of the qualification tests from the output filters are then placed into one of the output queues on that port. It is in the output queues that the various Classes of Service (CoS) offered by the switch are implemented; there will generally be one queue for each CoS. The internal switch header contains each frame's priority as determined by the Classification Engine. This priority is mapped to a CoS (and hence, an output queue) as discussed in Chapter 13, ''Priority Operation.''

The number of output queues is typically in the range of one to eight. This is an implementation tradeoff; more output queues provide greater granularity in the CoS offered at the expense of a more complex memory data structure, and possibly a greater total memory requirement.

Frames are de-queued and submitted for transmission according to the priority policy implemented in the output queue scheduler. Scheduling policies may include strict priority, weighted fair queuing, and so on (see Chapter 13). Regardless of the policy, the end result is a single stream of frames delivered for transmission on the output port in the desired priority order.[36]

[35] Another way to look at this process is to consider every frame as being tagged by the switch upon receipt. The tag used is either one contained explicitly in the received frame or one generated implicitly by the Classification Engine. The Egress Rules decision then becomes whether to strip the tag or to leave it intact.

[36] As discussed in Chapter 2, some commercial switches are capable of implementing proprietary transmission policies, for example limiting broadcast or multicast traffic to a specified maximum number of frames per unit time. Such policies are easily implemented in the output scheduler.

The housekeeping processor should be able to locally source frames at any point in the output process flow. That is, frames may be inserted into one of the priority queues, directly into the single, prioritized output stream, or even into the individual physical link output queues following the Link Aggregation Distributor (see section 17.4.3). The point at which frames are locally sourced is a function of the application submitting the frames:

- Higher-layer application traffic (SNMP, HTTP, Telnet, and so on) may be submitted into one of the normal output queues at a priority level commensurate with the application.
- Spanning tree BPDUs may be inserted directly into the output stream from the priority scheduler, effectively stepping to the head of the queue (see Chapter 5).
- Link Aggregation Control Protocol (LACP) messages may be inserted into the individual physical link queues of an aggregated link as necessary (see section 17.1.1 and Chapter 9, ''Link Aggregation'').

It can generally be assumed that if the housekeeping processor is locally sourcing frames, they do not need to be qualified by the output filter.[37]

17.4.3 Link Aggregation Distributor

If the output port comprises an aggregation of multiple physical ports, we need to take the stream of frames coming from the output scheduler and assign each one to a specific physical interface. The decision is made according to the distribution algorithm implemented within the Aggregator (see Chapter 9). The Distributor determines the conversation to which each frame belongs (based on a set of conversation mapping rules) and assigns it to one of the available physical interfaces.

The Distributor may also need to independently source frames onto the underlying physical ports on its own behalf; for example, it may generate Marker messages when transferring conversations from one physical port to another in the aggregation.

17.4.4 Transmit Flow Control

Under normal conditions, the Transmit Flow Control module transparently passes frames submitted by the Distributor (or output scheduler) to the Transmit MAC. It neither modifies nor interferes with frame transmissions unless the Receive Flow Control module has detected, parsed, and decoded a PAUSE frame containing a non-zero value for the pause_time parameter (see

[37] If you can't trust your own internal management processor, whom *can* you trust?

Chapter 8 and section 17.2.2). In that case, the Transmit Flow initially halts the interface; it simply does not allow frames to be submitted to the MAC until the pause timer expires (or is cleared by receiving a PAUSE frame with a 0 value for pause time).

The method used to halt frame flow is implementation-dependent. Typically, some internal mechanism (for example, a DMA engine or memory controller) will simply prevent additional frame buffer pointers from being linked into the MAC controller's transmit queue.

If the switch needs to assert flow control to its partner, it can always insert a PAUSE frame into the transmit stream, regardless of whether data frame forwarding has been halted or not. The signal to assert flow control might come from the housekeeping processor, or possibly from a queue buffer manager on the receive side of the interface (for example, the Link Aggregation Collector or the switch fabric input queues).

17.4.5 Hey, Kids! What Time Is It?[38]

There is one final qualification test to perform before we can finally send the frame. At the input port, we time stamped each frame upon arrival. We now need to compare that timestamp with the current time to see if the frame has overstayed its welcome.

As discussed in Chapter 2, a bridge must place an upper bound on the transit delay experienced by a frame between receipt and retransmission through an output port.[39] Unless this upper bound can be guaranteed by design, the timestamp mechanism ensures that frames are discarded rather than forwarded with excess delay.

17.4.6 Port Interfaces (Transmit)

We have been examining, qualifying, modifying, and manipulating the frames so much that it's almost a shame to have them leave the switch forever through the output port. Oh, shoot — that's the reason the switch exists in the first place! The Transmit Port Interface implements the MAC and PHY functions appropriate to the technology and medium being used on this particular port.

On a shared (half duplex) channel, the Transmit MAC performs the necessary channel arbitration functions for the LAN technology in use (CSMA/CD, Token Passing, and so on). On a full duplex link, there is no need to contend for channel use with any other station; the MAC can transmit frames whenever they are available to be sent. The Physical interface (PHY) converts the data

[38]Send us an e-mail if you know the answer to this question.

[39]IEEE 802.1D/Q-compliance requires that this limit be not more than 4 seconds, with a default value of 1 second [IEEE98a, IEEE98d].

stream emanating from the MAC into electrical or optical signals appropriate for the medium in use.

In a typical end station application, the Transmit MAC generates the proper FCS for all outgoing frames. In a switch, two possibilities exist:

- **The forwarded frame is not identical (bit-for-bit) to the frame as it was received:** The switch may have modified the contents of the frame between reception and transmission as the result of inserting, modifying, or stripping of a VLAN tag or because of frame format conversions between dissimilar LAN technologies on the input and output ports. If the forwarded frame is not exactly identical to the received frame, a new FCS must be computed before transmission. Typically, this function will be performed in hardware by the Transmit MAC entity.[40]
- **The forwarded frame is identical to the frame as it was received:** If every bit in the forwarded frame is exactly the same as in the received frame, then the original FCS is still valid. Rather than calculate the FCS in the Transmit MAC, the switch should simply forward the frame with the received FCS. This implies that the Transmit MAC entity must be able to enable or disable FCS generation on a frame-by-frame basis. The switch port can determine whether frame modifications have been made by inspecting the flag bits in the internal switch header.

Ladies and gentlemen, the frame has left the switch. It is gone, but not forgotten; a hint of its transitory encounter will live on forever in the network management frame counters.

[40]Methods for preserving FCS integrity when a recalculation is needed are discussed in Chapter 3, ''Bridging Between Technologies.''