The DECNIS 500/600 Multiprotocol Bridge/Router and Gateway

1 Abstract

The DECNIS 500/600 high-performance multiprotocol bridge/router and gateway are described. The issues affecting the design of routers with this class of performance are outlined, along with a description of the architecture and implementation. The system described uses a distributed forwarding algorithm and a distributed buffer management algorithm executed on plug-in linecards to achieve scalable performance. An overview of the currently available linecards is provided, along with performance results achieved during system test.

The DEC Network Integration Server 500 and 600 (DECNIS 500/600) products are general-purpose communications servers integrating multiprotocol routing, bridging, and gateway functions over an evolving set of local and wide area interfaces. The product family is designed to be flexible, offering a wide range of performance and functionality.

The basic system consists of a Futurebus+ based backplane, a management processor card (MPC), and a packet random-access memory (PRAM) card with a centralized address recognition engine (ARE) for forwarding routed and bridged traffic. Network interface cards or linecards are added to provide network attachment. The DECNIS 500 provides two linecard slots, and the DECNIS 600 provides seven linecard slots. The applications run from local memory on the MPC and linecards. PRAM is used to buffer packets in transit or destined to the system, itself.

The system was developed around distributed forwarding on the linecards to maximize performance. Software provides forwarding on the linecard for internet protocol (IP), DECnet, and open systems interconnection (OSI) traffic using integrated IS-IS (intermediate system to intermediate system) routing, along with bridging functionality for other traffic. The management processor controls the system, including loading and dumping of the linecards, administering the routing and bridging databases, generating routing and bridging control traffic, and network management. X.25 functionality, both for routing data and as an X.25 gateway, and routing for AppleTalk and IPX are supported on the management processor. Performance measurements on a system configured with 14 Ethernets have demonstrated a forwarding performance of 80,000 packets per second as a router or a bridge.

This paper discusses the issues involved in the design of a fast bridge /router. It presents the processing considerations that led us to design the distributed forwarding system used in the DECNIS 500/600 products. The paper then details the hardware and software design and concludes with a

performance summary.

2 Fast Bridge/Router Design Issues

There are a number of conflicting constraints on the design of a bridge /router. It must simultaneously forward packets, participate in the process of maintaining a global view of the network topology, and at all times be responsive to network management. This requires a sophisticated hardware and/or software design capable of striking the correct balance between the demands imposed by these constraints.

The need to make optimum use of the transmission technology is emphasized by the high link tariffs in Europe and the throughput demands of modern high-performance computing equipment. Therefore, the router designer must find methods of forwarding packets in the minimum number of CPU instructions in order to use modern transmission technology to best advantage. In addition to high performance, low system latency is required. The applications that run across networks are often held up pending the transfer of data. As CPU performance increases, the effects of network delay play an increasingly significant role in determining the overall application performance.

Another aspect of forwarding that requires attention is data integrity. Many protocols used in the local area network (LAN) have no data protection other than that provided by the data link checksum. Thus careful attention must be paid to the design of the data paths to minimize the periods when the data is unprotected. The normal technique in bridging is to leave the checksum intact from input to output. However, more advanced techniques are needed, as this simple approach is not possible when translating between dissimilar LAN types.

Two particular operations that constrain the performance of the forwarding process are packet parsing and address lookup. In a multiprotocol router, a variety of address formats need to be validated and looked up in the forwarding table. The most powerful address format in popular use is the OSI NSAP (network service access point), but this is the most complex to parse, with up to 20 octets to be analyzed as a longest-match sequence extracted from padding fields. In a bridge, supporting the rapid learning of media access control (MAC) addresses is another requirement. To provide consistently high performance, these processes benefit from hardware assistance.

Although the purpose of the network is the transmission of data packets, the most critical packets are the network control packets. These packets are used to determine topological information and to communicate it to the other network components. If a data packet is lost, the transport service retransmits the packet at a small inconvenience to the application. However, if an excessive number of network control packets are lost, the apparent topology, and hence the apparent optimum paths, frequently

change, leading to the formation of routing loops and the generation of further control packets describing the new paths. This increased traffic exacerbates the network congestion. Taken to the extreme, a positive

feedback loop occurs, in which the only traffic flowing is messages trying to bring the network back to stability.

As a result, two requirements are placed on the router. First, the router must be able to identify and process the network control packets under all overload conditions, even at the expense of data traffic. Second, the router must be able to process these packets quickly enough to enable the network to converge on a consistent view of the network topology.

As networks grow to global scale, the possibility emerges that an underperforming router in one part of the world could cause incorrect network operation in a different geographical region. A bridge/router must therefore be designed to process all network control traffic, and not export its local congestion problems to other parts of the network: a "good citizenship" constraint. To achieve this, the router needs to provide processing and filtering of the received traffic at line rates, in order to extract the network control traffic from the data traffic under worst-case conditions. In some cases, careful software design can accomplish this; however, as line speeds increase, hardware support may be required. Once the control traffic has been extracted, adequate processing power must be provided to ensure that the network converges quickly. This requires a suitable task scheduling scheme.

Another requirement of a bridge/router is that it remain manageable under all circumstances. If the router is being overloaded by a malfunctioning node in the network, the only way to relieve the situation is to shut down the circuit causing the overload. To do this, it must be able to extract and process the network management packets despite the overload situation. Cobb and Gerberg give more information on routing issues.[1]

3 Architecture

To address the requirements of a high-performance multiprotocol bridge /router with the technology currently available, we split the functional requirements into two sets: those best handled in a distributed fashion and those best handled centrally.

The data link and forwarding functions represent the highest processing load and operate in sufficiently local context that they can be distributed to a processor associated with a line or a group of lines. The processing requirements associated with these functions scale linearly with both line speed and number of lines attached to the system. Some aspects of these per-line functions, such as link initialization and processing of exception packets, require information that is available only centrally or need a sophisticated processing environment. However, these may be decoupled from the critical processing path and moved to the central processing function.

In contrast to the lower-level functions, the management of the system and the calculation of the forwarding database are best handled as a centralized function, since these processes operate in the context of the bridge/router as a whole. The processor workload is proportional to the

size of the network and not the speed of the links. Network protocols are designed to reduce the amount of this type of processing, both to minimize control traffic bandwidth and to permit the construction of relatively simple low-performance routers in some parts of the network.

These processing considerations led us to design the DECNIS 500/600 as a set of per-line forwarding processors, communicating on a peer-to-peer basis to forward the normal packets that comprise the majority of the network traffic, plus a central management processor. Although this processor behaves, in essence, like a normal monoprocessing bridge/router, its involvement in forwarding is limited to unusual types of packet.

Having split the functionality between the peer-to-peer forwarding processors and the management processor, we designed a buffer and control system to efficiently couple these processors together. The DECNIS 500 /600 products use a central PRAM of 256-byte buffers, shared among the linecards. Ownership of buffers is passed from one linecard to another by a swap, which exchanges a full buffer for an empty one. This algorithm improved both the fairness of buffer allocation and the performance of the buffer ownership transfer mechanism. Fractional buffers much smaller than the maximum packet sizes were used, even though this makes the system more complicated. The consequential economy of memory, however, made this an attractive proposition.

Analysis of the forwarding function indicated that to achieve the levels of performance we required, we would need hardware assistance in parsing and looking up network addresses. Considerations of economy of hardware cost, board area, and bus bandwidth led us to a single ARE shared among all linecards. This address parser has sufficient performance to support a DECNIS 600 server fully populated with linecards that support each link with a bandwidth of up to 2 X 10 megabits per second. Above this speed, local address caches are required.

Distributed Forwarding

In understanding the distributed forwarding process used on the DECNIS 500/600, it is convenient to first consider the forwarding of routing packets, and then to extend this description to the processing of other packet types. In the routing forwarding process, as shown in Figure 1, the incoming packets are made up of three components: the data link header, the routing header, and the packet body.

The receive process (RXP) terminates the data link layer, stripping the data link header from the packet. The routing header is parsed and copied into PRAM unmodified. Any required changes are made when the packet is subsequently transmitted. The information needed for this is placed in a data structure called a packet descriptor, which is written into space left

at the front of the first packet buffer. The packet body is copied into the packet buffer, continuing in other packet buffers if required.

The destination network address is copied to the ARE, which is also given instructions on which address type needs to be parsed. The RXP is now free to start processing another incoming packet. When the address lookup process has completed, the RXP is able to read from the ARE the forwarding parameters needed to complete the processing of the packet. These parameters contain information about the output port and channel to use, the destination data link address for the next hop, and any translation information. The RXP combines this information with some information saved from parsing the packet to build the packet descriptor in PRAM.

The RXP builds a set of ring vectors for the packet, one for each buffer used. Each ring vector contains a pointer to the PRAM buffer used, plus some additional information used to decide on which queue the buffer should be stored and to determine its relative importance to the system. During congestion, this information is used by the linecards to discard the least important packets first. These ring vectors are then exchanged with the transmit process (TXP) on the output linecard, which queues them ready for transmission. Before the TXP starts to process a packet for transmission, it reads the descriptor from the first PRAM buffer. From the information in the descriptor, the TXP is able to build the data link header, determine the routing header translation requirements, and locate a number of fields in the header (such as the OSI segmentation and quality of service fields) without having to reparse the header. The TXP builds the data link header, reads the routing header from PRAM, makes the appropriate modifications, and then completes the packet by reading the packet body from PRAM.

Since the transmit packet construction follows the packet transmission order byte for byte, implementations can be built without further intermediate transmission buffering. Linecards need only provide sufficient transmit buffering to cover the local latency requirements. In one instance, a linecard has significantly less than a full packet buffer. This small buffering requirement implies reduced system latency and makes available a number of different implementation styles.

If the RXP discovers a faulty packet, a packet with an option that requires system context to process, or a packet that is addressed to this system (including certain multicast packets), it queues that packet to the management processor in exactly the same way that it would have queued a packet for transmission by a TXP. The MPC contains a full-function monoprocessor router that is able to handle these exception cases. Similarly, the MPC sends packets by presenting them to the appropriate TXP in exactly the same format as a receiver.

The bridge forwarding process operates in a fashion similar to the routing forwarding process, except that the data link header is preserved from input port to output port, and only the data link header is parsed.

Buffer System

The design of an efficient buffer transfer scheme is an important aspect of a high-performance multiprocessor router. We solved this problem by using a set of single writer/single reader rings, with one ring associated with each pair-wise exchange of buffers that can take place in the system. Thus each TXP has associated with it one ring for each of the RXPs in the system (including its own), plus one for the management processor. When an RXP has a buffer to swap, it reads the next transfer location in its ring corresponding to the destination TXP. If it finds a free buffer, it exchanges that buffer with the one to be sent, keeping the free buffer as a replacement. The transferred information consists of a pointer to the buffer, its ownership status, and some information to indicate the type of information in the buffer. This structure is known as a ring vector. A single-bit semaphore is used to indicate transfer of ownership of a ring vector.

The buffer transfer scheme schematic shown in Figure 2 illustrates how this works. Each transmit port (TXa or TXb) has a ring dedicated to each of the receivers in the system (RXa and RXb). RXa swaps ring vectors to the "a" rings on TXa and TXb, and RXb swaps ring vectors to the "b" rings on TXa and TXb.

During buffer transfer, the TXP runs a scavenge process, scanning all its rings for new buffers, queuing these buffers in the transmit queues (TXQs) specified by the ring vector, and replacing the entries in the ring from the local free list. The buffer type information enables the transmit linecard to quickly determine the importance of the buffer. Thus if the linecard runs short of buffers due to congestion, it is able to discard less important packets in preference to those packets required to preserve the stability of the network.

Through judicious optimization of the ring vector encodings, we were able to condense this ring swap transaction into a single longword read followed by a single longword write for each buffer swap, for all unicast traffic. For multicast traffic, a second longword was required. To reduce the amount of bus traffic and the processor time associated with the scavenge process, the random-access memory (RAM) that holds the rings is physically located on the transmit linecard. Hardware is used to watch the rings for activity and report this to the TXP.

Analysis of the traffic patterns indicated that considerable economies in PRAM could be made if we fragmented long packets over a number of buffers. We achieved a satisfactory compromise between the processing overhead associated with buffer management and memory efficiency through the use of 256-byte buffers. With this buffer size, a large fraction of the packets are contained within a single buffer. When a linecard is driven into

output congestion, it is no longer certain that a complete set of packet buffers will be swapped. We therefore had to introduce a simple protocol to ensure that a packet was queued for transmission only if it had been fully transferred to the transmitting linecard. To cope with dissimilar swap and scavenge process speeds, we had to stage the transfer of buffers. Thus,

the TXPs collect a complete set of buffers from an RXP before queuing the packet for transmission; this process is called binning. In this way, a partial transfer due to congestion or a slow receiver does not block the progress of other ports in the system.

Bridging needs a mechanism to support the distribution of flooded and multicast packets to multiple output ports. In some distributed systems, this function is handled by replicating the packet via a copy process. In other systems, the packet is handled by a central multicast service. The use of a central multicaster gives rise to synchronization issues when a destination address moves from the unknown to the learned state. Replication by the linecards is not practical in this architecture since the linecards do not hold a local copy of the buffer after it has been copied to PRAM. We therefore use a system in which multicast buffers are loaned to all the transmit linecards. A "scoreboard" of outstanding loans is used to record the state of each multicast buffer. When a buffer is returned from all its borrowers, it is added to the multicast free queue and made available for reuse. The loan process and the return process are similar to the normal swap and scavenge process, but the ring vector is extended slightly to include the information needed for rapid dereferencing.

Centralized Resources

Three central resources are used in the DECNIS 500/600 products: MPC, PRAM, and ARE. Centralizing these resources reduced both the cost and the complexity of the system. There are two ways of building a distributed processing router. In one method, the router consists of a federation of full-function routers, each a separate network node. An alternative method is to employ a partially centralized design in which only one processor is the router in the traditional sense. The central processor is the focus for network management, calculating the forwarding table and being a central repository for the context of the router, and the peripheral processors undertake the majority of the forwarding work. An analysis of the cost and complexity both from a system and a network perspective led us to choose the latter approach. Thus the MPC provides all the software functionality necessary to bind the collection of forwarding agents located on the linecards together to form a router. To the rest of the network, the system appears indistinguishable from a traditionally designed router. The processing capability and memory requirements of the MPC are those associated with a typical medium-performance multiprotocol bridge/router.

We had a choice of three locations for the PRAM: distributed among the receiving linecards, distributed among the transmitting linecards, or located centrally. Locating the buffering at the receiver would have meant providing the maximum required transmitter buffering for each transmitter at every receiver. Locating the long-term packet buffering

at the transmitters would have meant staging the processing of the packets by storing them at the receiver until the transmit port was determined and then transferring them to the appropriate transmitting linecard. This would have increased the system latency, the receiver complexity, and its

workload. An analysis of the bus traffic indicated that for a router of this class, there would be adequate bus bandwidth to support the use of a centrally located, single shared packet buffer memory. With this approach, however, every packet crosses the bus twice, rather than once as in the other approaches. Nevertheless, we chose to base the system around a single packet memory, and win the consequential economies in both linecard cost and board area.

An analysis of the processing power needed to parse and look up a network address led us to conclude that the linecards would need some form of assistance if the processing power associated with each line was to be constrained to a reasonably cost-effective level. This assistance is provided by the ARE. Some advanced development work on the design of hardware search engines showed that it was possible to design a single address parser powerful enough to be shared among all the linecards. This search engine was adaptable enough to parse the complex structure of an OSI NSAP, with its two right-justified padded fields and its longest-match semantics. In addition, the engine was able to cope with the other routing protocol address formats and the learning requirements of bridging. By centralizing the forwarding database, we also avoided the processing and bus overhead associated with maintaining several distributed forwarding databases and reduced the cost and board area requirements of the linecards.

The bus bandwidth and lookup rate needed to support multiple fiber distributed data interface (FDDI) linecards would place an excessive burden on the system. For FDDI, therefore, we equip the central lookup engine with a linecard-resident address cache.

4 DECNIS 500/600 Hardware Design

There are three primary systems in the DECNIS 500/600, the backplane, together with its interface circuitry, the system core functions contained in the MPC and the PRAM, and the various linecards. In this section, we describe the hardware design of each of these.

Backplane and Interface Logic

The DECNIS 500/600 backplanes are based on the Futurebus+ standard using 2.1-volt (V) terminated backplane transceiver logic (BTL).[2,3] Although all current cards use 32-bit data and address paths, the DECNIS 600 backplane has been designed to support 64-bit operation as well.

Common to all current modules except the PRAM card, the basic backplane interface consists of two applications specific integrated circuits (ASICs), BTL transceivers, and a selection of local memory and registers, as shown in Figure 3. The two ASICs are a controller and a data-path

device. The controller requests the bus via central arbitration, controls the transceivers, and runs the parallel protocol state machines for backplane access. The data-path device provides two 16-bit processor interfaces (ports T and R), multiple direct memory access (DMA) channels

for each processor port with byte packing, unpacking, frame check sequence (FCS) and checksum support, and backplane address decode logic.

On the backplane, four DMA channels are provided per processor port. Two channels offer full-duplex data paths, and the other two are double buffered, configurable to operate in either direction, and optimized for bulk data transfer. DMA write transfers occur automatically when a block fills. Similarly, DMA prefetch reads occur automatically on suitably configured empty blocks. The double-buffered channels allow bus transactions to happen in parallel with processor access to the other block. All data transfers between the processor and the DMA channel are done under direct control of the processors, with the processors reading or writing every byte of data to or from the DMA streams. This direct control arrangement makes the design of the hardware simpler, avoiding the need for ASIC DMA support on the processor buses. More important, the use of processor read and write cycles makes the behavior of the system deterministic and ensures that the processor has the correct context at the completion of all operations, regardless of the outcome.

The data-path ASIC also provides command/status registers (CSRs) and a local bus containing the control interface for the second ASIC, ring vector memory (RVMEM), the geographical address, boot read-only memory (ROM), and nonvolatile battery-backed RAM (BBRAM) for error reporting. The RVMEM and some of the registers are accessible from the backplane. All resources can be accessed from either processor port. The device arbitrates internally for shared resources and has several other features designed to assist with efficient data transfers, e.g., a summary register of write activity to the RVMEM.

The data-path device can be driven from a single processor port (port T) for use in simpler, low-speed linecards. In addition, the architecture supports two data-path devices (primary and secondary) served by a common controller connected to the local bus of the primary device. Each data-path device adopts a different node identifier in the backplane address space.

Dedicated lines on the backplane are provided for power status, temperature sensing, and other system requirements.

Processor and Memory Modules

The MPC has two processors, a main processor and a uniprocessor version of the common backplane interface. The main processor, a VAX device, is in overall command of the system and provides all the management and forwarding services found in a monoprocessor router. The 16-bit, processor-based backplane interface frees the main processor from time-critical backplane-associated tasks.

A block diagram of the memory module is shown in Figure 4. Separate dynamic RAM (DRAM) arrays are used for data buffering and the forwarding database associated with the ARE. Ring structures in static memory are used to allow the linecards to post requests and read responses from the ARE, which is based on the TRIE system originally developed for text retrieval.[4,5]

An ASIC was developed for the ARE; it was extended to include some of the other module control logic, e.g., PRAM refresh control and the synchronous portion of the Futurebus+ backplane interface.

Network Interface Cards-Linecards

The DECNIS 500/600 products currently offer synchronous communications, Ethernet, and FDDI adapters, all using variants of the standard backplane interface.

Two synchronous communication adapters are available: a two-line device operating at up to 2.048 megabits per second, and a higher fan-out device supporting up to eight lines at a reduced line rate of 128 kilobits per second. All lines are full duplex with modem control. The lower-speed adapter uses a uniprocessor architecture to drive three industry-standard serial communications controllers (SCCs). The data and clocks for the channels, along with an extra channel for multiplexed modem control, are connected to a remote distribution panel using a 2-meter umbilical cord. Panels are available to support eight lines using the RS232, EIA422, or V.35 electrical interface. A four-line multistandard variant allows mixed electrical interfaces from a single adapter at a reduced fan-out. The multistandard panel uses a 50-pin cable common to other communication products from Digital.

The two-line device uses a four-processor interface as shown in Figures 3 and 5. The SCC is an ASIC device designed specifically for the data-flow style of processing adopted in the system architecture. It is closely coupled to the data-path ASIC and processors for optimal throughput. The hardware design has minimal dependency between the transmit and receive tasks, recognizing the limited coupling required by acknowledged data link protocols such as high-level data link control (HDLC). State information is exchanged between processors using a small dual-ported RAM in the SCC. In addition, each SCC and associated processors operate as a separate entity, resulting in consistent performance when forwarding both on and off the module. Two 50-pin multistandard interfaces (EIA422 and V.35 only) are provided on the module handle.

Several Ethernet adapters are available. A single-port thick-wire adapter uses a dual-processor architecture (primary R and T ports in Figure 3), along with a discrete implementation, to interface the Ethernet and its associated buffer (tank) memory. This design was reengineered to put the tank memory interface (TMI) into an ASIC, resulting in a dual-port (full implementation of the interface shown in Figure 3 plus two Ethernet interfaces) adapter derivative. This adapter is available in two variants supporting thick-wire, and ThinWire wiring schemes.

As shown in Figure 6, the FDDI adapter (DEC

FDDIcontroller 621) is a two-card option designed to cope with the high filtering and forwarding rates associated with FDDI. The hardware includes a filtering engine closely coupled to the FDDI chip set, a synchronous interconnect between the two cards, and a multichannel DMA engine for data transfer through the device. The DMA engine maintains tank memory under

reduced instruction set computing (RISC) processor control, and can be set up and monitored with minimal processor overhead. Data is transferred to or from buffers in PRAM to the tank memory, where complete packets are kept in contiguous address space. A second DMA channel transfers complete packets in a single burst to or from the buffer memory on the line interface card.

Traffic processing between buffer memory and the ring is done in hardware. A third DMA path is used to prefetch and then burst transfer packet header information from tank memory into the RISC processor subsystem for packet processing. The DMA engine, which includes tank memory arbitration, can queue multiple commands and operate all DMA channels in parallel. The 32-bit RISC subsystem provides the linecard processing, communicating with the bus interface processor using dual-ported RAM. Modular connectivity is offered for different physical media. The module also supports dual-attach and optical-bypass options.

5 DECNIS 500/600 Software Design

This section describes the software design of the DECNIS 500/600. The structure of the management processor software is first described. The structure of the linecard receiver and transmitter is then discussed, followed by details on how we expanded the design to forward multicast packets.

Management Processor Software

The DECNIS 500/600 MPC software structure, as shown in Figure 7, consists of a full-function bridge/router and X.25 gateway, together with the software necessary to adapt it to the DECNIS 500/600 environment. The control code module, which includes the routing, bridging, network management, and X.25 modules, is an extended version of Digital's WANrouter 500 software. These extensions were necessary to provide configuration information and forwarding table updates to the DECNIS 500/600 environment module. This module hides the distributed forwarding functionality from the control module. In this way, the control module is provided with an identical environment on both the MicroServer and DECNIS 500/600 platforms.

The major component of the DECNIS 500/600 environment module contains the data link initialization code, the code to control the linecards, and the code to transform the forwarding table updates into the data structures used by the ARE. A second component of the environment module contains the swap and scavenge functions necessary to communicate with the linecards. Because of the real-time constraints associated with swap and scavenge, this function is split between the management processor on the MPC and an assist processor.

The control code module was designed as a full-function router, thus we

are able to introduce new functionality to the platform in stages. If a new protocol type is to be included, it can be initially executed in the management processor with the linecards providing a framing or data link service. At a later point, the forwarding components can be moved to

the linecards to provide enhanced performance. The management processor software is described in more detail elsewhere in this issue.[1]

Linecard Reception

The linecard receiving processes are shown in Figure 8. The receiver runs four processes: the main receive process (RXP), the receive buffer system ARE process (RXBA), the receive buffer system descriptor process (RXBD), and the swap process.

The main receive process, RXP, polls the line communications controller until a packet starts to become available. The RXP then takes a pointer to a free PRAM buffer from the free queue and parses the data link header and the routing header, copying the packet into the buffer byte-by-byte as it does the parse. From the data link header, the RXP is able to determine whether the packet should be routed or bridged. Once this distinction has been made, the routing destination address or the destination MAC address is also copied to the ARE, together with some information to tell the ARE which database to search. The ARE provides hardware assistance to the bridge learning process. To prevent this hardware from inadvertently learning an incorrect address, the ARE is not allowed to start a MAC address lookup until the RXP has completely received the packet and has ensured that the checksum was correct. This restriction does not apply to routing addresses, which may be looked up before the packet has been completely received, thus reducing latency.

In the case of a routing packet, the data link header is discarded; only the routing header and the packet body are written to the buffer in PRAM. The source MAC address or, in the case of a multichannel card, the channel on which the packet was received is stored for later use. A number of other protocol-specific items are stored as well. All this information is used later to build the descriptor. The buffer pointer is stored on the pre-address queue until it can be reconciled with the result of the address lookup. In the case of an acknowledged data link such as HDLC, the receiver exports the latest acknowledgment status to the transmit process.

The receive buffer system ARE process, RXBA, polls the ARE for the result of the address lookup and stores the result in an internal data structure associated with its corresponding packet. The buffer pointer and the buffer pointers for any other buffers used to store the remainder of a long packet are then moved onto the RX-bin queue. Since the RXP and RXBA processes, the ARE search engine, and the link transmission process are asynchronous, the system is designed to have a number of pending ARE results, which are completed at an indeterminate time. This means that the reconciliation of lookup results and buffers may happen before or after the whole packet has been received. Because of the possibility of an error in the packet, no further action can be taken until the whole packet has actually been

received and all its buffers have been moved to the the queue labeled RX-bin.

If this staging process were not used, we would need to provide a complex abort mechanism to purge erroneous packets from the swap, scavenge, and transmit processes. Under load, the rate at which we poll the ARE has been engineered to be exactly once per lookup request. A poll failure will increase the backlog in the pre-address queue, which should not grow above two packets. This algorithm minimizes the Futurebus+ bandwidth expended in unsuccessful ARE poll operations. When the receiver is idle, the poll rate increases and the outstanding packets are quickly processed to clear the backlog.

The receive buffer system descriptor process, RXBD, writes the packet descriptor onto the front of the first PRAM buffer of the packet. The descriptors are protocol specific, requiring a callback into the protocol code to construct them. After the descriptor has been written, the buffer pointers are passed to the source queue, ready for transfer to the destination linecard by the swap process. The buffer is then swapped with the destination linecard as described in the section Buffer System, and the resultant free buffer is added to the free queue.

As an example of the information contained in a descriptor, Figure 9 shows an OSI packet buffer together with its descriptor as it is written into PRAM. The descriptor starts with a type identifier to indicate that it is an OSI packet. This is followed by a flags field and then a packet length indicator. The ARE flags indicate whether packet translation to DECnet Phase IV is required. The destination port is the linecard to which the buffer must be passed for transmission. The next hop physical address is the MAC address of the next destination (end system or router) to which the packet must be sent if the output circuit is a LAN; otherwise, it is the physical or virtual channel on a multiplexed output circuit. The segmentation offset information is used to locate the segmentation information in the packet in case the output circuit is required to segment the packet when the circuit comes to transmit the packet. This is followed by the byte value and position of the quality of service (QOS) option, the field used to carry the DECbit congestion state indicator.

The transmitter requires easy access to these fields since their modified state has to be reflected in the checksum field, near the front of the routing header. The source linecard number, reason, and last hop fields are needed by the management processor in the event that the receiving linecard is unable to complete the parsing operation for any reason. This information is also necessary in the generation of redirect packets (which are generated by the management processor after normal transmission by the destination linecard).

Linecard Transmission

The linecard transmitter function consists of five processes: the scavenge

rings process, the scavenge bins process, the transmit buffers system select process (TXBS), the main transmit process (TXP), and the TXB release process. These are shown in Figure 10.

The scavenge rings process scans the swap rings for new buffers to be queued for transmission, replacing them with free buffers. Buffers are queued in reassembly bins (one per destination ring) so that only complete packets are queued in the holding queues. The process tries to replenish the destination rings from the port-specific return queues, but failing this it uses the free list. The primary use of the port-specific return queues is in multicasting (see the section Linecard Multicasting).

The scavenge bins process scans the reassembly bins for complete packets and transfers them to the holding queues. Since different protocols have different traffic characteristics, the packets are queued by protocol type.

The TXBS process dequeues the packets from these holding queues round-robin by protocol type. This prevents protocols with an effective congestion control algorithm from being pushed into congestion backoff by protocol types with no effective congestion control. It also allows both bridged and routed protocols to make progress despite overload. The scavenge bins and TXBS processes between them execute the DECbit congestion control and packet aging functions. By assuming that queuing time in the receiver is minimal, we are able to simplify the algorithms by executing them in the transmit path. New algorithms had to be designed to execute these functions in this architecture.

The TXP process transmits the packet selected by TXBS. TXP reads in the descriptor, prepending the data link header and transmitting the modified routing header. When transmitting a protocol that uses explicit acknowledgments, like HDLC, the transmitted packet is transferred to the pending acknowledgment queue to wait for acknowledgment from the remote end. Before transmitting each packet, the transmitter checks the current acknowledgment state indicated by the receiver. If necessary, the transmitter either moves acknowledged packets from the pending acknowledged queue to the packet release queue, or, if it receives an indication that retransmission is required, moves them back to the transmit packet queue.

The TXB release process takes packets from the prerelease queue and separates them into a series of queues used by the swap process. Simple unicast packets have their buffers returned to the transmitter free pool. The multicast packets have their buffers placed on the port-specific queue for the source linecard, ready for return to their originating receiver. Packets intended for return to the management processor are also queued separately.

Linecard Multicasting

A bridge multicast or flooded buffer must be transmitted by a number of linecards. This is achieved by swapping a special type of ring vector, indicating that the buffer is only on loan to the transmitting linecard

and must be returned to its owner upon completion. In addition to the normal packet type, fragmentation, and buffer identification information, the ring vector contains local referencing information indicating where it is stored on the multicast heap. The receiver keeps a record of which multicast buffers are on loan to which transmitters. The scavenge process

notes in which ring it found the ring vector. After transmission, the TXB release process places the ring vector on the corresponding port-specific return queue. These ring vectors are then preferentially returned to their owner via the swap process. As the receiver gets these buffers back, it checks them off against a scoreboard of issued buffers. When a buffer is received from all destination linecards to which it was loaned, the buffer is moved back on the free list. For this to work successfully, some buffers must be set aside specifically for use by the multicast process.

6 Debugging the System

Extensive simulation was performed during system development. A model based on VHDL (a hardware description language used for simulation and logic synthesis) was built to simulate the queues, processes, bus accesses, and bus latency for the fast forwarding paths. Models were developed for the different styles of linecards, and many different traffic scenarios (packet size, packet type, packet rates) were simulated to verify the original thinking and architectural assumptions. In addition, simulation was performed on the software to measure code correctness and execution times. Gate arrays and modules were both functionally simulated and timing verified; analog modeling techniques were used to verify signal integrity of the backplane and selected etches.

The linecard processors used have a serial port and masked ROM embedded in the device. The internal ROM was programmed with a simple boot and console procedure. Provisions for a debug console via a ribbon cable to the module were developed, allowing a terminal connection to be made from the management processor to any linecard processor. Each processor on a module is software selectable from the console, which allows limited access functions to peek and poke memory maps, set break points, and step through the code. The system was enhanced by developing a breakout box and workstation environment that could connect to multiple linecards, offering multiple windows to different modules in parallel. The code executed under this regime ran at full speed. The environment allowed remote access, which proved useful between the two main module development sites in England and Ireland when problems required close cooperation between the two groups.

7 Performance

Performance measurements have been made on the DECNIS 500/600 products for DECnet Phase IV, DECnet Phase V (OSI), IP, and bridged traffic. For a detailed description of the measurement methodology and a comparison between the performance of the DECNIS 500/600 and competing bridge/routers, the reader is referred to independent test results compiled by Bradner.[6]

A summary of the LAN performance measured by Bradner and the WAN performance measured by ourselves is shown in Tables 1, 2, and 3. Table

1 shows the Ethernet-to-Ethernet forwarding throughput for minimum-sized packets. These measurements show the maximum forwarding performance with no packet loss. The use of a no-loss figure for comparison between different

designs is important because this represents the maximum throughput usable by a network application. If the applications attempt to run at more than the loss-free rate, the packet loss causes the transport protocols to back off to the loss-free operating point. The Ethernet-to-Ethernet figures indicate the near linear scalability of performance with number of lines. Ethernet forwarding performances of this magnitude are well in excess of those required to operate on any practical LAN. The correctness software ensures the reception of any routing packets for a significant period after these rates are exceeded.

Table 1: 64-byte Ethernet-to-Ethernet Packet Throughput

Protocol	Number of Ports		
	1	4	6
Bridge	13,950	48,211	80,045
IP	13,362	51,960	79,452
DECnet	9,330	34,164	53,746
OSI	6,652	25,891	38,837

Table 2: FDDI-to-FDDI Throughput

	Packet Size	
	64 Byte	2048 Byte
Throughput	16%	76%
Maximum pps	56,869	4,352
Bandwidth		85.5 Mb/s

Note: pps = packets per second

Table 3: WAN-to-WAN Performance for Routed Traffic

	Measured Percentage Line Utilization

NPDU Size	DECnet Phase IV	DECnet Phase V (OSI)	IP
46	96%	95%	93%
128	99%	99%	98%
512	100%	100%	100%
1450	100%	100%	100%

Note:_NPDU_=_network_packet_data_unit_____

Measurements also indicated that the unidirectional and bidirectional forwarding performances are substantially the same, which is not the case for all router designs. This is of more than academic significance. Poorly designed Ethernet subsystems do not provide adequate transmit processing

power under conditions of receive overload. Such subsystems suffer from a condition known as "live-lock." In this condition, the receiver uses up all the processing cycles, thus preventing the transmitter from attempting the transmission that would force a collision on the Ethernet and thereby restore fair operation.

The FDDI forwarding performance is shown in Table 2. These measurements were also taken at the zero-loss operating point and indicate industry-leading performance results.

The performance of the WANcontroller 622 running at 2 megabits per second is shown in Table 3. These measurements were taken using HDLC (with acknowledgments) as the data link, with a packet overhead of +19 octets for Phase IV and +6 octets for OSI and IP. These results indicate that the lines were running close to saturation.

8 Acknowledgments

The DECNIS 500/600 project has involved a great number of people located around the world. The authors wish to recognize everyone's contribution to the largest project undertaken by the Reading and Galway network engineering groups. Special thanks are extended to Mick Seaman for his leadership and guidance throughout the advanced development and early implementation phases of this project.

9 References

1. G. Cobb and E. Gerberg, "Digital's Multiprotocol Routing Software Design," Digital Technical Journal, vol. 5, no. 1 (Winter 1993) in press.

2. Futurebus+ Logical Layer Specification, IEEE Standard 896.1-1991 (New York: The Institute of Electrical and Electronics Engineers, 1992).

3. Futurebus+ Physical Layer and Profile Specifications, IEEE Standard 896.2-1991 (New York: The Institute of Electrical and Electronics Engineers, 1992).

4. E. Fredkin, "TRIE Memory," Communications of the ACM, vol. 3 (1960): 490-499.

5. D. Knuth, The Art of Computer Programming, Sorting and Searching vol. 3 (Reading, MA: Addison Wesley Publishing Co., Inc., 1973): 481-490.

6. S. Bradner, "Testing the Devices," Proceedings of Fall Interop 1992 . Available on the Internet by anonymous FTP from hsdndev.harvard.edu in /pub/ndtl.

10 Biographies

David L.A. Brash David Brash, a consultant engineer, joined Digital's Networks Engineering Group in 1985 to lead the hardware development of the MicroServer communications server (DEMSA). As the technical leader for the DECNIS 500/600 hardware platforms, David contributed to the architecture, backplane specification, module and ASIC designs and monitored overall correctness. He was an active member of the IEEE Futurebus+ working group. He is currently leading a group supporting Alpha design wins in Europe. David received a B.Sc. in Electrical and Electronic Engineering from the University of Strathclyde, Glasgow, in 1978.

Stewart F. Bryant A consulting engineer with Networks and Communications in Reading, England, Stewart Bryant worked on the advanced development program that developed the DECNIS 600 architecture. During the last six months of the program, he was its technical leader, focusing on implementation issues. Prior to this work, Stewart was the hardware and firmware architect for the MicroServer hardware platform. He earned a Ph.D. in physics from Imperial College in 1978. He is a member of the Institute of Electrical Engineers and has been a Chartered Engineer since 1985.