Chapter 6

Interconnects and Interfaces

This chapter investigates how USD channels may be implemented between user-space applications and peripherals on a number of different network interconnect types, from conventional I/O buses to LANs. The effects that different network link-layers have on the design of higher-level protocols, and thus on the complexity of devices is considered.

Interconnects used in conventional workstations are typically unable to provide the QoS guarantees that applications require. A number of interconnects that attempt to address this issue are examined.

6.1 Introduction

Devices in conventional workstations are typically *memory-mapped* into the address space of their device driver.[1] I/O buses support *memory-addressed* transactions, enabling the driver to access the device using loads and stores, just as it would main memory. Frequently, devices are also capable of issuing transactions on the bus, enabling them to transfer data to and from main memory without processor interaction.

Some I/O networks support memory-addressed transactions in a manner similar to conventional I/O buses. Such networks are typically designed to connect components of a single host. They are normally extensions of bus-based systems

[1] Programmed I/O is still frequently used on PCs, but its use for new devices is deprecated [PCI95].

(such as switched PCI or RaceWay), or designed as shared-memory multiprocessor interconnects (e.g. SCI).

USD channels can often be implemented very efficiently on memory-addressed interconnects. Section 6.3.1 describes how the Memory Management Unit can often be used to provide the protection necessary to allow applications memory-mapped access to USDs.

Rather than supporting memory-addressed read/write transactions, most networks designed for connecting multiple hosts provide only simple send/receive message functions. Protocols may be built on top of these messaging functions to provide a wide range of different communication paradigms, such as Remote Procedure Call (RPC), Active Messages [Culler], reliable byte streams, datagrams, and Distributed Shared Virtual Memory (DSVM)[Delp91].

It will be appropriate for some devices, particularly those expected to be accessed from far afield, to communicate using standard protocols such as TCP or UDP over IP. The more common case will be devices located on a single homogeneous network, such as a desktop, cluster or LAN. In such an environment, the required communication properties can often be achieved more efficiently through the use of 'custom' protocols.

6.2 Reliable Transport

Interconnects that natively support memory-addressed transactions normally deal with issues of error detection, flow control and transaction ordering in hardware. On such systems, bit errors and loss are an indication of faulty hardware. On networks where the physical layer is not error-free, a number of techniques can be used to provide reliable transport emulation in software and/or interface hardware.

6.2.1 Error Control

Interconnection networks that are external to the workstation box are more vulnerable to physical-layer bit errors. Networks using Unshielded Twisted Pair (UTP) cable (such as DeskNet, SSA, 100/1000baseTX Ethernet and SerialBus) are particularly at risk from Electro-Magnetic Interference (EMI). Other networks have chosen to use shielded multi-core cable to provide a measure of protection (Myrinet, SCI) while some have opted for optical media (Fibre Channel, VuNet, 100/1000baseFX Ethernet, some ATM and SCI).

Whereas internal interconnects such as PCI can detect errors using parity bits and signal a serious error condition (often resulting in a kernel panic), errors on external networks need to be detected and dealt with more gracefully. Typically, Cyclic Redundancy Checks (CRCs) are used for error detection, as they are less likely to be fooled by bursts of errors than simple schemes such as parity.

Some networks, for example SSA and DeskNet, detect errors at the link layer and automatically cause a hardware retransmit.[2] Thus, simple errors are hidden from higher-level layers (except by the additional message latency). Not all errors can be hidden in this way. Persistent errors, perhaps due to either device or link failure, must be handled further up the protocol stack.

6.2.2 Flow Control

Many local and wide area networking technologies do not provide any form of flow control.[3] Under high-load conditions the network is unable to apply 'back-pressure' to throttle transmitters. This can result in buffer overflow and hence data discard, and is often a far more common cause of data loss than bit-errors.

Flow control is perhaps more important in the small-scale networks used to connect most NAPs than in Wide Area Networks: There are likely to be fewer individual streams and hence less statistical multiplexing gain. Individual streams can be expected to occupy a large percentage of the total link bandwidth, and links are likely to be operated at high utilization. Without flow control, data-loss can occur in a number of places:

- In switched networks, contention for output ports in switches can lead to buffer overflow.
- Loss can occur at network endpoints due to insufficient receive buffers.
- Within network endpoints it is possible that queues associated with individual clients (e.g. application socket buffers) can overflow.

[2] Forward Error Correction might be used to similar affect.

[3] Forms of flow control have very recently been standardized for Fast Ethernet (IEEE standard 802.3x [IEEE97]) and ATM networks (ATM Forum Available Bit Rate (ABR) [ATMF96]).

Most of the network types proposed for connecting NAPs provide some kind of flow-control mechanism. Some network topologies, such as SCI rings, Fibre Channel Arbitrated Loops and SSA loops do not suffer from problems of contention for switch output ports because they use a shared media. Each node is able to see other traffic on the ring, and thus control when it transmits.

Networks containing switches require explicit flow-control. Some, such as Myrinet and SSA, provide a simple hop-by-hop scheme, stopping and starting the link as buffers become available at the next-hop receiving node. Fibre Channel provides a more advanced credit-based mechanism which operates between endpoints. In addition to preventing loss in switches, Fibre Channel's scheme avoids buffer overrun at the destination endpoint. It also prevents the head-of-line blocking on inter-switch links that can occur with hop-by-hop methods.

Both DEC SRC's AN2 and Sun's DeskNet are ATM type networks that have been augmented with proprietary per-virtual circuit flow-control schemes. This provides all the benefits of Fibre Channel's scheme with the additional advantage that it is possible to identify the individual client within the destination end-station.[4] Thus, it is possible to apply back-pressure when an individual client's buffer queue is full without causing head-of-line blocking of other clients' connections.

6.2.3 In-order Delivery

Messages can be delivered out-of-order to a recipient due to having taken different paths through the network, perhaps as a result of load-balancing or because a new link has come on-line. Although this can, and does, happen as a result of Internet routeing policy, few small-scale networks make use of load-balancing; data is generally only re-routed as a result of link or switch failure.

For desktop, cluster and local area networks, out-of-order delivery is likely to be as a result of data loss and corresponding retransmission. Sequence numbers provide a means to re-assemble packets into the original order. Some networks, for example SCI and Fibre Channel, include such sequence numbers as part of the link-layer packet-format specification.[5] This enables network interfaces to

[4] Fibre Channel does support lightweight virtual circuits (service class 1), but current host interfaces do not use the circuit identifier to demultiplex messages.

[5] A number of ATM networks designed in Cambridge before ATM standardization had two small sequence numbers in each cell header [Temple84, Greaves90, Black93]. This identified the position of the cell in the packet, and the position of the packet in the data stream.

assemble data into the correct order before passing it up.

Assembling messages into the strict original order before passing them up for processing is quite reasonable on a network with low round trip latency. Such networks might be expected to have a low message loss rate anyway (perhaps due to flow-control and low bit error rates), so occurrences where processing is stalled waiting for a message to be resent will be rare.

If messages are being sent over long distances, it may be useful to allow some mechanism for messages to be processed out of order. However, it must be possible for the sending endpoint to insist on ordering constraints when necessary, for example, to ensure consistency when updating meta-data on a disk. This could be indicated by inserting 're-ordering barriers' in the command stream. All commands *before* a barrier must be executed first.

Re-ordering barriers cannot be sent as normal command messages; they could be delayed, allowing commands occurring after them to go ahead. Instead, message ordering constraints can be indicated by an additional kind of sequence number in the message transport header, known as a message *group number*. Messages with the same group number can be delivered and executed in any order. Whenever the client wishes to insert a barrier, it increments the group number used for following messages. When a receiver encounters a message with a new group number, it should not process it until all messages from the previous group have arrived. It can track this using the sequence number in the normal way.

This scheme is robust, but slightly conservative: The first message of a new group must be received before other group members can be processed (since it is not known whether the missing message is actually the last member of the previous group). If necessary, the scheme could be enhanced through a 'first message in group' bit, or another sequence number specifying the position of each message in the group.

In some situations, it may be useful for an endpoint to be able to indicate that messages do not require reliable delivery. For example, if a message containing part of a video frame is lost, it may not be worth requesting a re-send since the effected area will soon be updated anyway. Furthermore, the sender does not need to keep a copy of messages that were sent marked as unreliable, because it will not be asked to re-send them. This may reduce the hardware complexity of some devices.

Unreliable delivery could be negotiated out-of-band for all messages on a channel, or selectable on a per-message basis. Messages could by marked as not requiring reliable delivery by setting a bit in the header, and deliberately not advancing the transport layer sequence number. If the message is lost en route, the receiver will never know it existed.

6.2.4 Protocol Implementation

Many of the networks to which NAPs will attach will not support reliable, in-order message delivery. It shall be necessary to emulate such functionality, through the use of a higher-level transport protocol. This protocol will be implemented within devices (possibly in hardware), and will also be used by operating system and application code that communicates with devices.

It is important that the protocol is kept as simple and efficient as possible. This can be achieved by exploiting as much information available from the link-layer as possible. For instance, many networks use relatively strong CRCs to ensure packet integrity, thus no further check at the transport layer may be necessary.[6] DEC SRC's AutoNet [Schroeder90] used a strong 64 bit CRC. Most other networks only use 32 bit CRCs, but in conjunction with a packet length test it may be considered sufficiently robust for many applications, particularly if the underlying physical layer uses a 'codebook' (e.g. 4B/5B, 8B/10B) that is likely to detect bursts of errors as a codebook violation.

Further assistance can be given to the transport protocol if it is informed about the reception of packets that fail CRC checks. These can be be used as a hint to speculatively send a Negative ACKnowledgement (NACK) to the likely source.[7] This may hasten a retransmission by the source.

Networks for which the link-layer specification does not contain sequence numbers must use them in the transport protocol to enable detection of packet loss and re-ordering. The sequence numbers can also be used as part of a simple window-based flow-control scheme. Complex rate-control features found in general-purpose protocols such as TCP are unnecessary, as far more is known about the underlying network.

[6]Some protocols, for example TCP, still use an independent transport layer checksum to provide true end-to-end protection. This also guards against errors introduced in fragment re-assembly, host memory DMA, cache consistency etc.

[7]The source address may not be reliably known as it may have been corrupted.

A simple transport protocol for communicating with USDs connected to a ATM LANs has been devised. It has been implemented in a number of devices,[8] and also a user-space shared library to which device clients may link. The protocol encapsulates messages in ATM AAL-5 PDUs. An 8 byte header is used in every packet to hold sequence numbers and flags. The protocol uses a fixed window size, chosen to be sufficiently large to hide the round-trip latency.

A number of experiments were performed using the user-space protocol implementation to perform half-duplex transfers between two HP700 series workstations equipped with FORE HPA-200 TAXI ATM cards. Using a window size of 64KB, the sender was able to achieve the same performance as raw AAL-5 between the two machines: 60Mb/s for 8KB packets. For comparison, TCP performance is around 40Mb/s.

The relatively large window size was required not because of speed-of-light or network buffering delays, but due to latency caused by the network interface and operating system scheduler. Despite this, the window is sufficiently small to avoid buffer overrun in either the network or receiver. Even at Gigabit speeds, a 64KB window would enable a network diameter of several kilometers, which would be suitable for many NAP applications.

The protocol could easily be implemented directly in hardware. However, where it is used in current devices, it is executed as software on the device's microprocessor for development convenience.

6.3 Source Authentication and Privacy

USDs need to be able to identify the sender of a message in order for them to determine that the sender is a both a bona fide client, and is authorized to perform the operation requested. Since some user-applications may have malicious intents, it is important that this is done in a secure fashion. A variety of different methods of achieving this are possible. This section examines how properties of the interconnect can often be exploited to provide a solution.

[8] These devices are described in chapter 7

6.3.1 Memory-Mapped Devices

Nodes connected to a memory-addressed interconnect are allocated one or more contiguous address ranges. A node will respond to transactions it receives that are within one of its address ranges, and ignore all others. All nodes may manipulate each other's memory areas; they are all trusted not to violate system integrity.

Different networks have different transaction routeing schemes. RaceWay requires all transactions to be source routed. SCI was originally designed for ring-based implementations, so each node scans the ring and 'hooks off' any transactions addressed to it.

The processor can access memory-mapped devices simply by reading and writing to the appropriate address. On most systems, user-space applications do not have access to I/O address space by default. They must first request the operating system to map some range of physical addresses into their virtual address space. This protection is enforced by the Memory Management Unit (MMU).

In such a system, the MMU can be used to implement the source authentication and data privacy functions required of communication channels between clients and USDs:

Devices can be designed such that their internal registers and memory are accessible at a number of different physical address ranges. Each physical address range should occupy a different set of physical MMU pages. Each individual copy of the device address space can be mapped into the virtual address space of a different client. Thus, when a client accesses a device, the device is able to identify the client by inspecting the physical address with which it is being accessed.[9]

One problem with this technique is that the address space range which the device occupies will be significantly bigger than a traditional device. A device supporting 64 clients will occupy at least 64 times the address space range of a conventional device, possibly more due to the overhead of page-alignment of each register set.

If the 'register set' is large, e.g. a 4MB memory-mapped framebuffer, the total address space requirement will be very large (256MB in this example). Many

[9] Giving different clients different virtual memory translations for accessing devices is unlikely to affect TLB performance: Even if the mappings were the same it is unlikely that the TLB entries could be shared across a context switch.

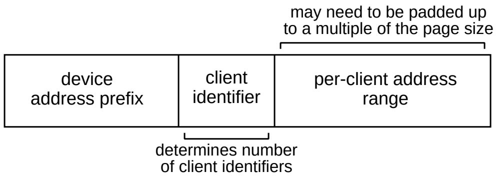

Figure 6.1: Encoding Client Identifiers in physical addresses

systems are limited to 32 bit addresses,[10] which may be insufficient to map all the peripherals in the system.

One solution to this problem is to map the device register set into the physical address space some small number of times, (fewer than the number of supported clients), and treat the copies as a kind of cache. Currently active clients will be given memory translations to enable them to access one of the sets. Other clients will not have mappings. To identify the client, the device will need to be kept informed of which clients have which mappings. This can be done by a privileged process accessing suitably protected registers on the device. This scheme could be used to provide a separate copy of the register set to each CPU in a multiprocessor system. The operating system could be made responsible for updating the device's client identity registers whenever the running process is changed.

6.3.2 Virtual Circuit Networks

Networks supporting *Virtual Circuits* (VCs), such as ATM, DeskNet, the Cambridge DAN and VuNet, are particularly convenient for attaching USDs. VCs are generally sufficiently lightweight that an individual one can be established for each client-device channel. Avoiding the need to multiplex multiple channels down a single VC has performance benefits, and can be very useful in providing source authentication.

Hosts equipped with connection-oriented network interfaces typically only allow applications to send and receive messages on VCs that have been allocated to them. This protection is normally provided by the operating system, but can be provided by the network interface device (see section 6.4). Thus, in a network where hosts are trusted to enforce these restrictions, a device receiving a message

[10] Although the PCI standard specifies a scheme for 64 bit addressing, it is largely unused—even by a well-known manufacturer of 64 bit processors.

on a particular VC can be sure that it originated from the corresponding client.

Virtual Circuit Identifiers (VCIs) are stamped on the front of messages to identify the virtual circuit to which they belong. Different networks use different schemes for VCI allocation. It can be particularly convenient if the device is responsible for allocating the VCI value that is used for receiving messages from a particular client: It can allocate a value such that it can be used to index the array containing client state directly, without need for indirection or hashing. This method was adopted by the Cambridge DAN project, and has been found to be useful in reducing the complexity of devices attached to it.

6.3.3 Datagram Networks

When the underlying network is connectionless, some means of identifying the originating client must be encoded within messages sent to devices. Without resorting to cryptographic techniques (which would incur significant extra complexity in devices), two techniques are possible:

- Clients identify themselves using a *capability*. This would be a unique, unguessable bit pattern assigned to each client by the device manager. A 128 bit random number would probably be sufficient to prevent one application from guessing another's.

- Messages from clients have the identity of the sender certified by some trusted entity, either software or hardware.

6.3.3.1 Capabilities

The capability approach is attractive in that it does not require modification to any operating system code to send and receive packets. Clients can identify themselves by placing their capability bit pattern in all messages which they send to the device.

In addition to being assigned a capability, the device manager could also assign the client a *client number*. Both the capability and the client number should be sent in the header of all messages to the device. The client number enables the device to validate the capability without performing an associative lookup: The device can simply use the client number as an index into an array containing the correct capability value.

There are, however, a number of drawbacks to this scheme. Capability keys owned by one client may fall into the hands of another application, enabling it to masquerade to the device as the other client. On some networks, this could happen as a result of an untrusted host snooping the network. This could be prevented through cryptographic techniques, but this would require significant amounts of extra hardware in devices.

There is also the possibility that a capability could be stolen from a client through exploiting any bugs it may have. It may be possible to trick the application into passing out data values containing the capability. Unlike in a 'real' capability machine (e.g. the Cambridge CAP [Wilkes79]), capabilities will be treated just like other data values, and can be offered no special protection. An attacker in possession of a valid capability may be able to perform significant damage. Furthermore, the source of the attack will be difficult to trace.

6.3.3.2 Certification

Another way of ensuring that messages bear the originator's true identity is to have the operating system 'stamp' a value into the message before it is sent (or check that the value already inserted by the application is valid). This kind of operation is commonly performed as part of protocol processing. For instance, packets sent by applications over UDP and TCP have headers prepended to them containing the IP source address and port number. These could be used by a device to uniquely identify a message's sender, providing all machines on the network can be trusted to use their real IP address.

Rather than using standard protocols, many NAPs will prefer to use more lightweight communication methods. A simple protocol can be added to the operating system that stamps an identifier into the data to be sent. Similarly a value can be used by the operating system to demultiplex received packets to the destination application.

The value that is written into the packet by the operating system can take a number of different forms. It could be a value allocated by the device manager to be unique to the client (like VCIs on the Cambridge DAN), or alternatively, it could be an identifier allocated by the host that is used in conjunction with the source host address. For example, it would be possible to use the application's Process ID (PID), and the host's Media Access Control (MAC) address.[11]

6.3.4 Trust

Many departmental networks are configured such that hosts in the same domain can make use of each other's services. Hosts that provide services trust clients to implement the departmental security policy. This enables them to 'believe' information passed to them by client machines, and avoids the need for more complicated security checks. There is usually a security hierarchy, in which some machines are considered more secure than others. The most secure machines are likely to only have accounts for systems administrators, be physically secure, and provide only essential services.

This hierarchical model works well in many environments, and is a useful model for many NAP installations: The most secure host might run the device manager, while other hosts will be trusted to perform different classes of operation.

The implications of a 'rogue host' attached to the network should be considered. This could happen if a host has its operating system compromised (e.g. the attacker gains root access) or if an attacker connects a machine to the network on which they already have privileged access.

An attacker with root access is likely to be able to exploit the full range of services to which the host has access. It will be able to control all applications running on the machine, snoop on their data, and forge messages that appear to originate from them. The attacker would be able to exploit any access to USDs that the host was entitled to. The extent to which the attacker can go beyond this and imitate other hosts and make use of their privileges is dependent on the mechanism used to implement host-device channels.

A switched VC network (e.g. ATM) in which the VCI is used to identify the client is perhaps least affected by a rogue host, providing control over the switches is maintained. An application that has managed to breach operating system security may be able to send and receive on arbitrary VCIs. It may impersonate other applications running on the same machine, but not applications running on

[11] In the case of Ethernet this is a 48 bit factory-assigned unique number. For other networks it may be be a smaller field allocated during network initialization.

other machines; no VC will be configured through the switch to enable it to do this, and messages sent with invalid VCIs are normally discarded. Provided that the machine on which the device manager runs is not compromised, the attacker will be restricted to the access-control and QoS levels of the host that has been taken over. Furthermore, due to the switched nature of the network, snooping on messages to or from other hosts will not be possible.

Schemes for providing source authentication that rely on the operating system to certify the sender may allow an attacker to impersonate another host more easily. On some networks, it may be possible for a device to discover the MAC address from which a message was sent, and use this to identify the sending host. Not all networks can provide this information securely. For example, messages sent on SSA are source routed, and by the time the message is received the route bytes are zeroed. The receiver cannot identify the sending station, and must rely on information encoded in the message data.

MAC addresses are easily forged on some networks. For example, despite the fact that Ethernet interfaces have a unique factory-assigned MAC address stored in hardware, most will actually allow the operating system to send packets containing arbitrary source addresses. A rogue host could forge packets such that they would appear to have originated from other hosts. It could also put the interface into *promiscuous mode*, enabling it to snoop on other network traffic. As well as violating their data privacy, this could be used by an attacker to discover the client identifiers in use by other device clients.

Furthermore, on a switched Ethernet network, sending a packet with another host's MAC address is likely to lead to the switches believing that the 'victim' host has been moved to a new Ethernet segment.[12] The switches will thus forward any packets addressed to the victim towards the attacker; thus enabling it to fully impersonate the other host.

In a departmental network, it is often convenient to allow some services to be accessed from outside the immediate network. It is also important to provide protection for the services that are to be kept internal. A *firewall router* can be used to control the packets that can pass between the two networks, thus local machines do not need to trust machines on the far side of the firewall. A similar device could be used to connect, for example, a cluster network to a LAN.

[12] Some new Ethernet switches and hubs have a special 'secure' mode, in which new machines cannot be connected or existing ones moved without informing the switch via the management interface.

The Cambridge DAN uses such a device to connect to an external ATM LAN. Before a host outside of the DAN can communicate with a DAN node, it must establish a VC through the DAN's LAN interface. The LAN interface prevents cells which are not on an established VC from passing through onto the DAN. Since external hosts are unable to inject arbitrary messages onto the DAN fabric, DAN nodes may continue to use a message's VCI to identify its sender. The DAN LAN interface is described further in the next chapter.

In situations where there are untrusted hosts on the local network, it will be necessary to use cryptographic techniques to implement communication between hosts and devices. This is likely to require significant extra hardware, and will add to the cost of the device. A suitable scheme for distributing the cryptographic keys will also have to be found.

6.3.5 Manager-Device Privileged Channels

Even if the operating systems on other hosts are trusted, there can still exist a "start of day" security problem. A channel must be established between the device and its manager. It is important to ensure that only a bona fide device manager may attach to this privileged interface, otherwise an attacker can easily gain access to the entire device. This is complicated by two factors:

- Many devices will not contain stable storage. After power-on, they cannot know the identity of their manager, so must wait to be contacted by it.[13]
- For reasons of redundancy, it is often useful to have a choice of hosts on which the device manager can be run.

On a system where devices are memory-mapped, protection of the control interface can be achieved simply by allowing only privileged applications to map devices into their address space. Where devices are communicated with through message-passing, other techniques must be employed.

One potentially flawed method is to rely on the device manager to attach to the device early in the boot sequence, before other applications would get a chance to. In a system with multiple network hosts, an attacker might deliberately

[13]Some devices may broadcast messages informing other hosts of their presence and willingness to be managed.

reboot the machine on which the manager normally runs in order to attempt to impersonate the manager from another host.

Another approach is to partition the client identifier space in a manner similar to that used for TCP/UDP port numbers: TCP and UDP port numbers below 1024 are considered to be privileged. An application can only bind to a port number below 1024 if it is privileged. Thus, another application (or device) receiving messages from a source port in the 0–1023 range can be sure that the message originated from a privileged application. Similarly, the operating systems on all hosts could be configured to only allow privileged applications to create channels with client identifiers less than a certain value. On ATM networks, for example, it is common to prevent user applications from opening circuits with VCI values less than 16. Some operating systems (e.g. Linux) allow this value to be raised.

On a switched VC network, Permanent Virtual Circuits (PVCs) could be configured to connect devices to hosts that are trusted to manage them. At the device end, these PVCs can be terminated with VCIs in a range defined by the manufacturer for communicating with the manager. At the hosts, the PVC should be terminated with a VCI that is not accessible to user applications running on the host. After a reset, a device can watch for messages on its management VCIs, and choose a host to manage it.

On some networks, it will be difficult to prevent unprivileged applications from communicating with the device's management port. In these situations, cryptographic techniques should be used to ensure that the device only responds to a 'real' device manager. Once a channel is established with a manager, it may be considered sufficiently secure that protection of messages sent along the channel is not required. Thus, computationally expensive cryptographic techniques are only required during connection setup.

The use of cryptography requires the device to store a secret. This could be assigned by the site system administrator, or set to some unique value at the factory. A bit-serial ROM or non-volatile RAM memory chip could be used to store the secret relatively cheaply. If the exchange between the device and manager is to be particularly secure, it may be necessary to augment the device with a good random number source, or real-time clock, in order to prevent against replay attacks.

6.4 Application-Accessible Network Interfaces

Invoking the operating system to send and receive network packets can incur considerable overhead, and might seriously impair the performance with which an application could access a NAP. This overhead can be reduced by using network interface device drivers that allow messages to be sent and received using fast operating system traps rather than full-blown systems calls. The Nemesis OPPO and the Cornell University U-Net/FE (Fast Ethernet) [Welsh96] drivers fall into this category.

An alternative even lower overhead method is to develop network interfaces that can be accessed directly from applications, without operating system interaction: Messages are sent and received either by using special CPU instructions, or through memory-mapped I/O. A number of such interfaces have been developed by the message-passing multiprocessor community, for whom reducing message latency is very important.

Many of these interfaces have not been designed for use in a multi-programmed environment, and thus provide scant protection between applications; they allow clients to send and receive arbitrarily formed packets. Such interfaces would be unsuitable for implementing the communication channels between applications and USDs, as the identity of a message's sender could not be securely established.

There are, however, a number of application-accessible interfaces that do provide protection:

6.4.1 SHRIMP

The SHRIMP-II memory-mapped network interface [Blumrich94] was developed at Princeton as part of the SHRIMP project to build a multicomputer. The interface connects Pentium PCs to an Intel Paragon interconnect [Intel91].

Communication between two hosts is connection-oriented, in that two processes on different hosts must first agree to create a *mapping* between an area of memory they each own. The memory areas do not need to be at the same physical or virtual addresses on the two hosts, but must be of the same size. Physical pages occupied by the areas are *pinned* to prevent them from being paged out. Each mapping is simplex, and may be created in either *automatic update* or *deliberate update* mode.

In automatic update mode, any stores the sender makes to a mapped memory region will be snooped by the network interface, and sent as messages across the interconnect, where they will cause the equivalent value to get updated in the receiver's mapped region. This can provide the illusion of shared memory between processes running on different hosts, providing cache coherency is maintained.

In deliberate update mode, messages are not automatically sent when a process updates areas of mapped memory. Instead, the process must explicitly ask the interface to transmit an area of memory from the mapped buffer. The message will be received into the corresponding position in the mapped region of the destination process. Applications can cause messages to be sent by writing to a *command page*. Every page of physical memory has a command page associated with it, the address of the two pages typically being different by only the top couple of bits. Command pages are not backed by real memory, but the physical addresses are decoded by the network interface. Writing a value x to command page address y causes the x bytes pointed to by the physical memory address associated with y to be transmitted over the interconnect. The destination is determined by the mapping associated with the physical page.

Protection between applications is maintained by the processor's memory management unit. An application can only transmit from an area of memory if the corresponding command page is mapped into its virtual address space. This prevents it from sending data belonging to other processes. The destination of the message is determined by the mapping to which the data area belongs, thus, applications are unable to forge message headers. The operating system must ensure that if a process' access to a physical page is revoked, access to any command page is also removed. The network interface must ensure that individual messages do not cross page boundaries.

When a message is received from the interconnect, the interface must decide where to put it in host memory. It must check that the mapping specified in the message is still valid, and then translate the offset within the mapped region into a physical address. This is done through a *network interface page table*, which is stored on the interface, and updated by the operating system to reflect the current state of mapped regions.

The Shrimp-II interface provides good low-latency communication between co-operating processes. Its use as a general-purpose network interface is limited however, due to the inability of the receiving node to choose where received data is placed. The use of command pages provides a very efficient means of requesting the interface to send a message, but does require a very large physical address space, and may potentially double the TLB entry requirement of a process.

6.4.2 Hamlyn

Like the SHRIMP project, the Hewlett Packard Hamlyn project [Wilkes92, Buzzard96] developed an application-accessible network interface to enable workstations to be connected together to form a multicomputer. The current version of the interface allows HP PA-RISC workstations to be connected via their high-speed graphics bus to a Myricom Myrinet interconnect [Seitz95].

Messages are sent from, and delivered into *message areas*. Message areas can be of arbitrary size and alignment, and must be backed by pinned physical pages, though the pages do not need to be contiguous. They are established via an operating system call, which stores information about the message area into a *message table slot*. The slot number identifies the message area. When a message area is created, as well as being allocated a slot number, it is assigned a *protection key*, a 64bit random number. This key value must be known by any process that wishes to send messages into a message area.

Processes that wish to send messages must have a *send terminus* (a copy of the interface control registers), mapped into their virtual address space.The terminus can be operated in two modes, DMA mode and direct-mode. Direct-mode is intended for use sending small packets, or when latency is critical. The process uses store instructions to write the message body into a FIFO register in the terminus. Other registers are used to set the message destination, which is specified by a $\langle$*remote node ID*, *message area slot number*, *offset in message area*$\rangle$ 3-tuple. Since the sender determines where the data is placed in the receiver's message area, this is known as *sender-directed communication*.

A sender cannot push data into arbitrary message areas. The protection key sent in the header of every message must match the key of the receiving message area, otherwise the packet is discarded. Likewise, the message must fit within the destination message area. Processes cannot change the protection key used by their sending terminus; they must request the operating system to do this for them.

DMA mode is useful for sending longer messages as it absolves the CPU from storing the data to be sent to the interface. Sending processes simply write *work* *descriptors* to the interface. Each descriptor contains the source message area and offset, the message length, and the destination 3-tuple. Associated with each terminus is a list of message areas to which the sending process has access. Thus, the DMA engine can check that the process is not trying to send data belonging to other applications.

In addition to the *send* function, Hamlyn enables a sending terminus to issue a *remote-get*. This is processed in a similar way to *send* messages, but causes an area of memory to be read from the remote message area and written to the local one.

Each terminus has a *transmit priority* associated with it. This is assigned by the operating system, and is used to resolve contention for resources such as the DMA engine and interconnect when multiple termini have work outstanding.

When a message is received by a node, the interface writes a *notification record* into memory, and optionally generates an interrupt. A separate notification record queue is maintained for each application, enabling applications to busy-wait if required. Alternatively, they may be unblocked or sent a signal by the operating system demultiplexing the interrupt.

Hamlyn provides a very flexible high-performance communication mechanism, but does require a relatively complex hardware implementation. The Hamlyn application library [Buzzard96] does support stream and datagram protocols, but this is not the interface's natural mode of operation. The sender-directed communication makes it very well suited for message-passing between closely co-operating processes.

6.4.3 Osiris

The OSIRIS interface [Davie93, Druschel94] was developed at the Bellcore as part of the Aurora Gigabit Test-bed [Clark92]. It is an OC-12 (622Mb/s) ATM interface for DEC TurboChannel machines. The card consists of two relatively independent halves, (transmit and receive), each controlled by its own Intel i960 processor. From the host's perspective, the interface looks like a 128KB region of memory. This memory is effectively dual-ported, being shared with the on-board processors to enable buffer descriptors to be passed to and from the host.

The Osiris interface supported *Application Device Channels* (ADCs), a mechanism that enabled applications to send and receive packets without operating system interaction. Applications using an ADC had two 4KB pages of the device's SRAM mapped into their virtual address space. One was used during transmission, the other for reception. Since the interface had 128KB of RAM, up to 16 ADCs could be accommodated.

When an ADC is created, the OS passes various protection information to the card. It associates the ADC with a set of VCIs on which the application is allowed to transmit and receive. It also passes a list of pinned physical pages that the application can use to transmit from, and receive into.

Applications initiate a transmit operation by writing one or more *fragment descriptors* into a circular buffer stored in its 4KB page of interface memory. Fragment descriptors each contain a physical pointer and a length. The transmit buffers are polled by one of the interface's i960s. The processor will validate each descriptor against the list of pages that it knows the application has access to, and then initiate a DMA transfer. If the application asks the interface to DMA from a page from which it has not been authorized, the interface raises an interrupt to alert the OS, which may send a signal to the application.

Receive operation is similar to transmit. The application writes fragment descriptors for *free buffers* to the interface. When a PDU is received on one of the VCIs associated with the ADC, the interface takes the first buffer descriptor from the ADC's free list and DMAs the packet into it, after first checking that the physical pages referred to in the descriptor are in the list assigned to the ADC by the OS.

After the DMA is complete, the interface moves the fragment descriptors from the free queue to the *receive queue*. The ADC receive queue is stored in host memory rather than interface memory, as it is more efficient for the application to access it there. This is particularly important if the application is polling for a new packet.

If the receive queue is empty when a packet arrives, an interrupt will be signalled to the host. Thus, an interrupt will not be generated for every packet when the system is under load and has a backlog. The OS interrupt handler is responsible for determining the ADC that caused the interrupt condition and signalling the appropriate application.

Osiris does make some considerations towards QoS. Unlike many ATM interfaces, complete packets are not DMAed onto the adaptor and then processed.

Packets are DMAed in 88 byte two cell chunks.[14] This prevents small, latency-critical packets from getting stuck behind a large PDU.

Each ADC is also assigned a *transmit priority*. This is used to determine the order in which the ADCs are serviced, and hence can be used to provide some measure of QoS amongst co-operating processes.

Osiris succeeded in providing over 500Mb/s of application-to-application throughput over ATM. This is something which many commercial ATM vendors are still struggling to do four years on.

6.4.4 U-Net

The U-Net project at Cornell is investigating the feasibility of using conventional networks and network interfaces to connect workstations for use in parallel and distributed computing. The original work, U-Net/ATM [von Eicken95] took a FORE Systems Inc. SBA-200 140Mb/s TAXI ATM adaptor for Sun Sbus, and rewrote the firmware for the on-board i960 CPU to enable user-level network access. The result was a dramatic improvement in the communication latency and throughput available to applications.

The U-Net/ATM firmware enables the SBA-200 to offer applications *U-Net channels*, which are much like the ADCs introduced by Osiris. Applications have an area of the adaptor's on-board SRAM mapped into their virtual address space, into which they write descriptors for the transmit and free buffer queues. The i960 writes descriptors into a host-memory receive queue whenever a PDU is received on the VCI associated with a channel. All transmit and free buffer descriptors must point into a *communication segment* associated with the channel. Communication segments are held in contiguous, pinned, physical pages.

The U-Net SBA-200 firmware reduces the per-message overhead and hence latency with which applications can send messages. The application-to-application 'ping-pong' round trip time is reduced to $65\mu s$. Although this is larger than that achieved by most dedicated message-passing interconnects, using U-Net for message-passing multi-processing is certainly possible. [von Eicken95] contains a number of benchmark results that show a cluster of SuperSparc machines connected via U-Net holding their own against some similarly configured commercial

[14] Osiris supports AAL-3/4, with 44 byte data payload cells rather than the now more common 48 byte AAL-5

parallel processing machines.

One problem with original U-Net implementation was that it required applications that used the channel interface to have a communication segment of contiguous, pinned physical pages. Although not all applications use channels, (non critical applications can be multiplexed down a single channel by an OS driver), there can be contention for physical memory, particularly contiguous chunks. This has been addressed by U-Net/MM (Memory Management) [Basu97].

U-Net/MM for ATM currently runs on PCs equipped with a FORE PCA-200 PCI OC-3 ATM adaptor and the Linux operating system. This hardware configuration was chosen over the Sun platform of the original work due to the more convenient kernel development environment provided by Linux. U-Net/MM allows applications to send and receive messages from anywhere in their virtual address space. This is achieved by locating a software TLB on the network interface, an NI TLB.

During a transmit operation, the interface must convert the virtual address specified in the descriptor into a physical address. It does this by looking up the $\langle \textit{process ID}, \textit{virtual address} \rangle$ tuple in the NI TLB. If there is a corresponding entry in the TLB, the interface can be sure that the physical page referred to in the entry is owned by the process and currently paged-in. If the buffer spans multiple pages the process is repeated.

Should a NI TLB 'miss' occur, the interface informs the host kernel. The kernel will examine the host page tables and inform the interface about the current state of the requested page. If possible, it will supply a new TLB entry and transmission will continue. If the virtual address is not legal it will inform the interface so that it may return an error response to the client. If the virtual page referred to in the descriptor is not currently resident, the kernel will initiate a page-in and notify the interface to suspend transmission of the packet until the page fault is resolved.

On the receive side, the network interface pre-translates a number of descriptors in each channel's free buffer queue. When a packet arrives, the VCI is used to determine which channel it is destined for. It is then DMAed into the first pre-translated buffer. The interface will then attempt to pre-translate the next item on the free queue. NI TLB misses are dealt with as per transmit. If a buffer has not been paged-in by the time it is needed to receive a message, it will be returned to the user unfilled. Hence, page faults will not stall reception on a channel.

U-Net/MM ensures consistency between the OS page tables and the NI TLB by pinning down all physical pages for which the NI TLB has mappings. When a mapping is evicted from the NI TLB the kernel is informed and it is un-pinned. Thus, the kernel can limit the amount of memory it is prepared to have pinned by reducing the TLB size. Care is taken to ensure that pages for which message transmission or reception are already in progress are never evicted, as this could result in an illegal DMA. Work is continuing to assess the success with which U-Net/MM is able to reduce the amount of memory that needs to be pinned at any one time.

As well as U-Net/ATM, there is also U-Net/FE [Welsh96]. This uses a Fast Ethernet (100Mb/s) interface based on the DEC 21140 controller. The interface does not contain an on-board CPU like the SBA-200 ATM card, so they were unable to take the same approach of re-writing the device firmware. Unable to turn the 21140 into a true application-accessible network interface, they opted to produce an efficient kernel driver that is able to provide U-Net channels to application endpoints.

Ethernet is not a connection-oriented network, so there is normally no VCI that can be used to demultiplex received packets to the appropriate channel. The demultiplexing problem can be solved by using packet filters to delve inside the Ethernet frame data to examine higher-level protocol headers [Yuhara94, Engler96]. The Nemesis Ethernet driver [Black97] uses a packet filter in the transmit direction as well, to ensure that applications are sending messages with properly formed headers i.e. they are not attempting to spoof their IP source address or such like.

U-Net/FE uses a different approach. It takes a block of unallocated Ethernet protocol type numbers, and uses the protocol type field as a VCI. This makes identifying the recipient channel of an Ethernet frame very efficient. Unfortunately, it does limit the amount of inter-working that is possible with other protocol stack implementations. This is not a problem for message passing between hosts on the same network forming a multi-computer cluster, or for that matter, communication between a host and a USD. However, it is a problem for running standard protocols over U-Net channels. Although the U-Net team have impressive results for TCP and UDP performance, it is should be noted that their implementations could not inter-work with standard stacks, due to the different protocol type fields.

The 21140 was not designed to demultiplex packets to different clients. It delivers all received packets into a circular buffer within the kernel driver. The kernel interrupt handler examines the protocol type field, takes a buffer off the appropriate channel's free list, copies the packet into it, and then adds it to the channel's receive queue. For transmit, applications push a buffer onto their transmit descriptor queue and issue a fast kernel trap to the U-Net service routine. The kernel trap does not incur the full overhead of a system call. The service routine examines the channel's transmit queue and pushes appropriate buffer descriptors onto the 21140's transmit ring.

The U-Net project has been successful in demonstrating that providing applications with protected, yet low level access to network interfaces can yield significant performance advantages—even on conventional interface hardware.

6.4.5 Discussion

A number of the network interfaces described above have many of the qualities that would be required of a memory-mapped User-Safe Network Device. One issue that has not been addressed is that of QoS. Hamlyn and Osiris do support transmit priority. Unfortunately, this is not sufficient to perform traffic shaping, or provide the QoS guarantees required by soft real-time applications. Providing such guarantees should be easier with application-accessible interfaces, where multiplexing occurs very low down in the protocol stack [Tennenhouse89].

Low-latency, high-bandwidth communication from user-space is essential for accessing NAPs. Traditional operating system protocol stacks seem unable to provide this. Application-accessible interfaces can.

6.5 Interconnect Scheduling

QoS crosstalk can occur whenever applications compete for a resource for which they have not been guaranteed any QoS. The I/O interconnect is no exception. If the required interconnect resources are unavailable QoS crosstalk can manifest itself in two ways:

- **CPU resource crosstalk.** Applications performing memory-mapped or programmed I/O operations will find they make less progress than expected in their CPU time-slice. I/O read operations will experience long access-latencies, resulting in the CPU stalling until the data is returned. I/O write operations can be 'posted' in write buffers, but if the write bandwidth required exceeds that available on the interconnect, the processor will be stalled.

- **Device resource crosstalk.** Applications may find that QoS reservations they have made with I/O devices are not satisfied. Insufficient interconnect bandwidth can prevent the device from transferring data to and from host memory. If the situation persists, it could result in the device being forced to discard the client's data. Conversely, if availability of interconnect bandwidth can be guaranteed, it may be possible to reduce the device's buffer memory requirements and therefore cost.

Conventional workstations provide little support for providing interconnect resource guarantees. To complicate matters, many workstations have an I/O bus hierarchy. A modern high-end workstation might be sold with one or more PCI bus segments, an ISA or EISA bus for 'legacy' peripherals, two IDE buses (for disks and CD-ROMs), a SCSI bus (for high-performance disks) and a Universal Serial Bus (USB, for keyboards, modems and conference-quality video cameras). A single I/O transaction is likely to need to traverse more than one bus. Predicting the QoS a transaction will receive can be very difficult, since all the buses have different arbitration and access policies.

Most buses attempt to arbitrate the next bus master in parallel with the current data transfer. The mechanism used is generally fixed in hardware, beyond operating system control. Some buses, such as VME, use a strict priority scheme, where devices have an order of precedence determined by the wiring of their request and grant lines. PCI uses a round-robin scheme, as this reduces the possibility of deadlock due to transaction ordering rules. Often the arbitration is 'weighted' in favour of the host-bridge, in order to reduce the likelihood of the CPU stalling during I/O transactions.

PCI, along with most other buses, uses a per-transaction accounting scheme. The bus is granted to a device for the period of a transaction; no account is taken of either the number of bus cycles used or the number of words transferred during the transaction. The number of bus cycles consumed by different types of transaction can vary greatly. On PCI, a single 32-bit write can take five cycles. A long DMA can take many more, and can be slowed down by either target or initiator stall cycles. Thus, even with a 'fair' arbitration scheme, the share of the bandwidth that each device receives can be very different. Devices making lots of short transactions (such as CPU MMIO writes to a frame buffer) are likely to be starved in favor of longer DMAs.

Some buses attempt to bound the maximum latency a device can experience whilst waiting to be granted the bus, by placing a restriction on the maximum transaction size. The DEC TurboChannel bus limits a single transaction to a maximum of 128 32-bit words. PCI uses a slightly more sophisticated scheme, wherein each device has a Master Latency Timer (MLT). When a device becomes bus master it initializes the MLT to a value determined by the BIOS (the same value is normally used for all devices). If another device wishes to use the bus, the arbiter will take the 'grant' signal away from the current master. When this happens, the master must start to decrement its MLT every cycle. When the timer expires it must relinquish the bus.

This scheme enables a PCI bus master to have unrestricted use of the bus during otherwise idle periods, but in the presence of load, the bus will be shared out with the granularity of the MLT initialization value. Assuming a fair round-robin arbitration scheme, the latency a device will experience will be bounded by: (no. devices − 1) $\times$ MLT value. This MLT initialization value may be reduced to reduce the maximum latency, but this will be at the expense of increased overhead. For example, a value of 64 can result in 7% of cycles being lost to overhead, 32 in 14% and 16 in 25%.

In a conventional workstation, the operating system has very little influence over how the I/O interconnect is used. It may be able to control when some DMA-type transfers are initiated (others will be unsolicited), but this only provides a very coarse-grained form of control, and is unsuitable for soft real-time multimedia systems.

6.5.1 QoS Scheduling

The performance required of an interconnect scheduling algorithm is very demanding: There are likely to be fewer individual data streams than on a traditional LAN or WAN. Traffic on some of the streams is likely to be 'bursty', and closely correlated with other traffic. Furthermore, on an idle system, a single data stream might expect to be able to utilize most of the available bandwidth. Statistical multiplexing is unlikely to be able to provide much benefit in such a system. Flow control is essential to avoid loss.

Some of the data streams in a multimedia workstation will have characteristics that are known in advance. For instance, an audio stream may behave like a Constant Bit Rate (CBR) source. An uncompressed video stream is likely to have periodic properties, determined by the horizontal and vertical scan rates. Some newer interconnects enable this information to be exploited, allowing some degree of QoS to be guaranteed to clients.

Many of these interconnects allow individual transactions to be prioritized, as opposed to the per-device prioritization provided by most traditional workstation interconnects. For example, ANSI/VITA Raceway [Cyp95], supports three different priority levels. When two transactions wish to use the same switch output port, strict priority is used to determine the 'winner'. If two transactions of the same priority compete, the winner is determined by examining the source port number, for which there is also a strict priority order.

A single Raceway transaction can transfer up to 2KB of data. For the hard real-time systems for which Raceway was designed, this would provide a rather coarse-grained sharing of the interconnect. Thus, Raceway provides the ability for high priority transactions to preempt or 'kill' lower priority ones. Nodes at either end of a Raceway link can assert the kill signal to indicate that they have a higher priority transaction to send than the one currently using the link. Upon receiving the kill signal, the master must voluntarily terminate its transaction. It may immediately begin to request the arbiter for use of the link again, whereupon it will be successful only after all higher priority traffic has passed.

Using strict priority can lead to total starvation of other data streams. SCI [ANSI95b] avoids this by using a scheme in which 90% of the link bandwidth is reserved for pending transactions of the current highest priority, while the remaining 10% is shared fairly between all other transactions. Transactions of the same priority are scheduled in a round-robin fashion. Four different priority levels are supported.

SCI is designed for use as an interconnect in a cache-coherent multiprocessor. Providing low-latency access to the interconnect is very important. Enabling read operations on which the processor is stalled to have priority over writes, or speculative pre-fetches, is likely to provide performance benefits. In order to maintain SCI's transaction ordering rules, lower priority transactions that are blocking higher priority transactions inherit a higher priority.

The maximum SCI transaction size is limited to 256 bytes, thus limiting the amount of time that high priority transactions may be blocked. At 1GB/s, a 256 byte packet plus headers takes less than 300ns to send.

IEEE P1394 (FireWire) [IEEE94] fixes the maximum packet size in units of time rather than in packet length. Packets can be at most $62\mu s$ long, corresponding to 512 bytes on a 12.5MB/s link. Such a scheme enables media-access jitter to be bounded, yet reduces the per-transaction overhead on higher bandwidth links. This can be beneficial to both switches and end-stations, where longer packets and the corresponding lower packet arrival rate can reduce hardware complexity.

P1394 can support up to 64 guaranteed bandwidth data streams by means of its isochronous access mode. Isochronous data streams have strict priority over all other data. The channel management protocol should ensure that the link is not over-contracted.

Every $125\mu s$, a signal is sent to all devices to indicate that an isochronous access period has started. Any device that has data outstanding on a guaranteed bandwidth stream should send it during this period. When all devices have sent all isochronous data, the device returns to its normal arbitration mode, which enables each device to become bus master in a round-robin fashion.

6.5.2 ATM Interconnects

Rather than limiting the maximum transaction size or providing a pre-emption mechanism, some interconnects use Asynchronous Transfer Mode (ATM) techniques to bound the media-access jitter. Packets are split up and sent as a number of independently-scheduled fixed-size cells. Due to the ITU decree that ATM cells carry 48 byte payloads, most cell-based interconnects follow suit in the hope of easier inter-working with ATM LANs. In principle, there is no reason why an interconnect should not pick a different cell size: A cluster interconnect is likely to be several times faster than a LAN, so converting between different cell-sizes at LAN rates should not be particularly difficult.[15] Dealing with the LAN's different flow control and error detection properties is likely to be more troublesome.

As well as reducing media access-latency, small, fixed-size cells make band-

[15]Bridging between the 32 byte cell CFR network [Temple84], and 48 byte cell Fairisle [Black94a] network was commonplace in the University of Cambridge Computer Laboratory during the period where both networks were operational. This was achieved in software using a simple ARM3-based device equipped with two network interfaces.

width accounting easier. The bandwidth used by a data stream is simply the number of cell slots it uses; there is no need to keep track of the size of each transaction. Since the size of each transaction may not be a multiple of the cell size, some of the interconnect bandwidth will be wasted. Due to the relatively small size of cells this is unlikely to be of great concern in most circumstances.

Each individual cell needs to be routed across the network. Fortunately, cells contain a Virtual Circuit Identifier (VCI) in their header, enabling the destination to be found using simple table lookup. This enables a very high forwarding rate to be achieved.

Some data streams will consist of a flow of discrete cells. For example, an audio stream might be sent as a CBR stream in which each cell is independent and can be processed separately. However, most data streams have a packet structure, and thus most end-stations must be capable of segmenting packets into cells for transmission, and re-assembling received cells into packets.

Often, it will be necessary for the receiver to assemble cells into complete packets before any useful processing of the data stream can occur. This is because on most networks, the receiver can only be sure that all cells have arrived intact when the CRC has been computed over the entire packet. Cells from packets on different VCs will often arrive at the receiver in an interleaved fashion.[16] Cell interleaving can increase the latency with which a particular packet can be sent over the interconnect.

Furthermore, the receive interface must be capable of assembling packets from multiple VCs concurrently. It must have sufficient memory to store the current 'context' of each active VC, and be capable of switching between them on a cell-by-cell basis. This results in extra hardware complexity not incurred by packet-based interconnects.

Some device's transmit interfaces will also need to perform operations on multiple VCs concurrently. This can enable urgent data to be sent without having to queue waiting for the current packet to be sent completely. If some of the VCs are to be rate controlled, it is particularly important that the segmentation unit should be capable of processing different VCs concurrently. For some devices, the jitter introduced by waiting for transmission of a complete packet will be acceptable, and thus these devices may have simple single-threaded segmentation

[16] The scheduling algorithms used in many switches deliberately interleave cells from different input ports that are heading for the same output port.

units. The transmit interface for such devices will be no more complex than for packet-based networks.

The following sections outline a number of experimental cell-based interconnects.

6.5.2.1 The Desk Area Network

The University of Cambridge Computer Laboratory's Desk Area Network (known as *The DAN*) [Barham95, Hayter93b] was designed as a replacement for the system bus of multimedia workstations. It is built around a 16-port switch fabric developed by the Fairisle [Black94a] ATM LAN project. A number of different nodes have been designed for the DAN, and are discussed in the next chapter. Here, the design of the switch fabric is considered.

The 16-port switch is a delta network constructed from eight 4×4 crossbar elements. The crossbar switches are implemented using Xilinx FPGAs, and can achieve a per-port bandwidth in excess of 150Mb/s. Cells injected into the fabric consist of 2 bytes of source routeing information, 4 bytes of cell header, and 48 bytes of data payload. The format of the cell header is opaque to the switch, but a convention exists for devices to interpret it. The header contains the VCI, and packet-level framing information.

All nodes are synchronized to present cells they wish to send to the fabric input ports at the same time. The switch decodes the source routeing information, arbitrates between cells competing for the same resources, and configures the crossbar matrix. The fabric contains no internal buffering. Nodes are informed whether their cells were successfully delivered or not by means of an ACK signal. Cells that were rejected may be retried repeatedly.

Cells may be rejected because of competition for the same output port, blocking in the internal delta network, or because the receiving node was not prepared to accept the cell (perhaps because its buffers are full). The ACK signal provides end-to-end flow control, ensuring cells will not be dropped by either the fabric or the end-stations. Also, the switch is engineered to ensure that cells are not corrupted in transit. Thus, a reliable transport of cells from one node to the other is assured (in the absence of hardware failure).

The combination of end-to-end flow-control and reliable cell transport provides a number of advantages for devices. Often it is possible for a device to begin processing cells belonging to a packet as they arrive, rather than waiting to assemble and verify the whole packet. Thus, some devices avoid the need for a special memory area into which packets are assembled. For example, the DAN Framestore (section 7.3) is able to transfer image data contained in cells directly to the frame buffer as they arrive; if reliable cell transport were not assured, the Framestore would have to wait until the entire packet contents had been verified in order to avoid the possibility of putting corrupt data into the frame buffer.

The timing of the ACK signal is such that it enables a receiving node to examine the contents of a cell header before it decides whether it wishes to accept the cell. This enables it to reject cells belonging to client VCs that are currently 'out of quota' of device resources. A sending node can distinguish between its cell being rejected by the switch and the destination node by examining the timing of arrival of the negative ACK signal. If the cell was NACK'ed by the destination node, it should probably not retry it immediately. A sophisticated transmitter might schedule the retransmit behind cells bound for other destinations (though no current devices do this).

The DAN's space division switch topology eases some of the interconnect scheduling problems encountered on traditional workstation buses, by reducing contention for shared resources: Transactions only compete if they are bound for the same destination node, or use the same internal delta network link.

The routeing tag on the front of every cell includes a priority bit. Cells with the bit set always take priority over cells with the bit clear whenever there is contention for an output port (internal or external). Cells with the same priority are arbitrated using a simple round-robin mechanism. Each output port stores state to indicate which input port it last accepted. This is used to implement rotating priority between input ports.

The arbitration algorithm used in the Fairisle switch fabric is very simple. An algorithm that allowed each input port to select from a number of input cells would result in greater crossbar utilization. Examples of such algorithms include Parallel Iterative Matching (PIM) [Anderson93] and SLIP [McKeown95].

The priority mechanism provided by Fairisle is used to provide a mixture of guaranteed and best-effort service. Individual VCs can be allocated a share of the interconnect bandwidth by a software manager. Devices are responsible for controlling the rates that they send on each of their VCs. Cells that are sent as part of the VC's guaranteed bandwidth are sent with the priority bit set. If no cells are waiting to be sent on VCs that have guaranteed bandwidth outstanding, the device can choose to send cells on a best-effort basis with the priority bit clear. The mechanism used to implement the guaranteed bandwidth and to share out the best-effort is device-dependent. A scheme using a pre-computed schedule has been experimented with to provide CBR guarantees [Kahn95]. Alternatively, a leaky bucket algorithm could be used to provide a VBR-type guarantee.

6.5.2.2 VuNet

MIT's VuNet [Houh95] is a cluster network for interconnecting hosts and NAPs. The basic component is a 6-port crossbar switch that operates up to 700Mb/s per port. Hosts and devices connect to the switch via short cables providing a 32 or 64-bit data path. Like the DAN, the network uses cells with a 48 byte payload and 4 byte header with a source routeing tag prepended. Arbitration is round-robin; there is no priority bit. Unlike the DAN, there is no end-to-end flow-control. The switch provides over 256 cells worth of buffering per port to absorb bursts. Nodes are expected to be able to 'keep up' with the maximum data rate sent to them.

A number of VuNet nodes exist, such as a TurboChannel host interface for DEC Alpha workstations, a video capture device (known as the Vidboard), and a multiprocessor DSP device called the Vid40. The host interface node is capable of DMAing cells directly to and from host memory. Thus, host memory is used to augment the buffering present in the switch. The interface performs no packet segmentation or assembly. The device driver is left to do this and to demultiplex packets to clients.

The Vidboard supports digitization and simple processing of video into a cell stream. It is controlled by a 'command cell' sent on a special VC. The single cell message contains a number of parameters to configure the device, such as the VCI to send the video on, the pixel format to use, and the image size. The device echoes the command cell back to the sender to indicate successful reception (the command cell also contains the VC on which to send the echo reply). Each command cell causes the next video frame to be digitized and output as a stream of AAL-5 packets. A number of parameters in the command cell control the shaping of the cell stream produced, enabling the peak rate and inter-packet gap to be controlled.

VuNet switches may be connected together to form a larger cluster network. For distances of a couple of hundred metres, a VuNet *G*-link interface may be used to provide connectivity at 500Mb/s. For longer distances, a Sonet OC-3c interface is available (155Mb/s). Both interfaces support header re-mapping. When a cell is received on the serial link, the 16-bit VCI in the cell header is used as an index into a table. The table supplies a new VCI and a switch routeing tag. The new VCI is used to replace the old one. No other bits in the header are modified. The routeing tag is prepended onto the front of the cell before it is injected into the fabric. The re-mapping table is configured by sending cells to the interface over the switch fabric on a special VCI.

Re-mapping VCIs between every VuNet switch avoids the need for a global VCI allocation policy. Like the DAN, the cell header format is not the same as that used in B-ISDN. The VuNet Sonet interface must translate the 16-bit VCIs used internally into the 12-bit B-ISDN VCIs. Since only a single bit of packet-level framing is used per cell, this can also be sent over the B-ISDN link.

VuNet addresses many of the same issues as the Cambridge DAN. The high level of trust between nodes belonging to the cluster enables devices with very simple control interfaces to be used. Because of its lack of flow-control, and potentially unreliable inter-switch links, VuNet does not guarantee reliable delivery of cells. This can lead to greater complexity in the higher-level protocols operated over the network. Unlike the DAN, the VuNet switch fabric makes no distinction between guaranteed and best-effort traffic. Providing QoS guarantees whilst allowing nodes to make use of 'spare' bandwidth is likely to be difficult: nodes will need to be assigned new transmit rate parameters at frequent intervals.

6.5.2.3 DeskNet

DeskNet [Lee95] was developed as prototype desktop network. The workstation box may be located up to 300m away from the user peripherals, and is connected via optical fibre. Devices on the desktop are daisy-chained together using UTP cables of up to 5m in length. A slotted-ring MAC protocol is used, operating at 1Gb/s. Each of the slots holds a 48 byte cell, a header and a trailer. The header contains a 16 bit VCI, and a number of bits for ring maintenance and media access control.

A combination of source and destination-slot release protocols are used. Source release is used for the first and last cells of a packet, to provide flow-control and acknowledgment. Destination release is used for other cells, enabling the slot to be reused by other devices on the ring, hence potentially increasing the available ring bandwidth. Stations determine whether a cell is addressed to them by examining the VCI in the header (and the full/empty bit to determine whether the slot is in use). Thus, a global VCI allocation algorithm is used.

The first cell of every packet will have the *Acknowledge Request* MAC bit set, indicating that the cell requires source release. The destination will verify that it has received the cell correctly, using the 16 bit CRC stored in the trailer, and decide whether it is prepared to accept the cell. If both conditions are true, it will set the trailer 'ACK' bit, thus notifying the sender that everything is well and that it may continue to send the rest of the packet. The final cell of a packet will also have the *Acknowledge Request* bit set. The destination node will reply with a positive ACK only if verification of the entire packet is successful, e.g. after checking an AAL-5 trailer CRC and length.

A node may choose to reject the first cell of a packet (and hence delay the packet), if it currently has insufficient resources. This could be because the particular client VC is (currently) using all its allocation, or because the device as a whole is out of some resource. For example, a low-cost device might be designed such that it is only able to assemble packets on some small number of VCs concurrently. The source-release protocol provides back-pressure to signal to other nodes that the device is busy. The DeskNet specification does, however, expect all nodes to be able to receive single-cell messages at all times, as these are frequently used for control purposes,

The DeskNet MAC protocol attempts to provide clients with a guaranteed share of bandwidth, yet also allow best-effort use of spare capacity. Each node is assigned parameters for a guaranteed rate, of the form x slots in y slots. They are also assigned a peak rate—the maximum short-term rate at which they send in either best-effort or guaranteed mode. One of the MAC bits in each cell header is independent of the cell, and can be set by any node as the cell passes. This bit is used by nodes to indicate whether or not they are getting their guaranteed bandwidth. A node may fail to get its guaranteed bandwidth if insufficient slots are being left empty by upstream devices. By setting the bit, the node requests that other devices back off to their guaranteed rates. Nodes should only transmit above their guaranteed rates if they have observed the bit being clear for a full ring revolution. This provides some hysteresis to prevent oscillation.

DeskNet's flow-control and bandwidth-sharing properties enable many of the features of the DAN to be realized on a ring topology network.

6.5.2.4 Switcherland

ETH Switcherland [Eberle97] is a cluster network for interconnecting workstations and devices. The network is built from switch elements, and is scalable to a network diameter of around 10 switches. The prototype's links operate at 265Mb/s full-duplex, over UTP cable runs of up to 50m.

Although Switcherland professes not to be ATM interconnect, it does use small, fixed-size cells and virtual circuits. Cells consist of a 60 byte data payload and 4 byte cell header, of which 16 bits are effectively the VCI. The other 16 bits are used for miscellaneous in-band signalling between endpoints.

The current switch implementation is a four port bus-based design. 128KB of fast SRAM is used to provide internal buffering to avoid cell loss when there is contention for switch output ports. In the absence of contention, cells may be rapidly cut-through from input to output port, resulting in a latency of less than one third of a cell time ($1.2\mu s$).

Virtual circuits are classified as either CBR or VBR, dependent on the high-order VCI bit. Separate CBR and VBR queues are maintained for each output port, enabling up to 256 cells to be buffered in each queue. When servicing the queues, output ports give strict priority to the CBR queue. Each CBR stream is allocated a bandwidth guarantee that allows it to transmit x cells every y cell slots. Endpoints are trusted to abide by the bandwidth allocation, thus no policing is required within the network. To avoid over-committing the network, the system should ensure that the sum of all guaranteed rates on a link is less than the link bandwidth.[17]

Any stream which requires more flexibility than is allowed by the strict CBR regime must use the VBR service. VBR streams use an end-to-end credit-based flow-control mechanism. When cells are received at the destination endpoint, an acknowledgment cell is sent back to the source. Transmitters are allocated a credit which determines the maximum number of cells they may have outstanding in the network. This number of cell buffers is logically reserved in the VBR queue of every switch output port that the VC passes through en route to the destination. Since Switcherland networks have a relatively short round-trip time, only a small number of credits are necessary to enable a VC to utilize the full link bandwidth (typically less than 10). When allocating credits, the system must ensure that it

[17] It is also necessary to ensure that the y parameter in the rate guarantee is less than 128 to avoid overruning the CBR queue.

does not over-commit the VBR buffer queue in any switch.

As well as preventing cell loss, the credit-based flow-control scheme enables simple bandwidth guarantees to be made. VCs may be allocated more credits than they need to mask the round-trip latency. The number of credits a VC has been allocated is likely to determine the share it receives of any bottleneck link. This is because many of its outstanding cells are likely to be resident in the upstream switch's output port queue. The effectiveness of this scheme in providing bandwidth guarantees in a network consisting of several switches requires further study.

One benefit of the Switcherland approach is that the switch elements are simple, since they require no flow-control or policing functions. However, there are a number of drawbacks: The rigid partitioning of the buffer memory into CBR and VBR halves is rather restrictive. Furthermore, for a VBR stream with n credits it is necessary to reserve n cell buffers in every switch on the VC. This is rather wasteful of the buffer memory, but since high-speed SRAM is now a commodity product (for use in caches) this may not be important. Switcherland requires that an acknowledgment cell be sent back to the sender whenever the destination receives a cell on a VBR stream. Although for some full-duplex streams it may be possible to piggy-back data with the acknowledgment, this scheme will often be wasteful of network bandwidth.

Despite these observations, Switcherland remains an interesting ongoing project, and a good platform for experimentation with NAPs.

6.6 Summary

This chapter discussed how link-layer properties may be exploited to reduce the sophistication required of higher-level protocols providing reliability and authentication. This can result in reduced device complexity, and hence cost.

Application-accessible network interfaces enable user-space applications to send and receive messages without the overhead of operating system calls. Such interfaces are essential to provide the low-latency communication required between applications and NAPs.

Finally, the chapter focused on interconnect scheduling, observing how lack of interconnect resources can cause QoS crosstalk, and how conventional interconnects provide little support for such scheduling. A number of new interconnects designed for real-time use were described, including three that use ATM cells to assist bandwidth multiplexing and accounting.