High-Performance Adaptive Routing in Multicomputers Using Dynamic Virtual Circuits †

Yuval Tamir and Yoshio F. Turner
Computer Science Department
University of California
Los Angeles, California 90024

Abstract

A message transport mechanism which provides high-bandwidth low-latency interprocessor communication is the key to the ability of multicomputers to achieve high performance. The system should adapt to changing conditions by routing packets around congested areas and failed links or nodes. We introduce a new message transport mechanism, called *Dynamic Virtual Circuits*, that combines the best features of circuit switching, packet switching, and static virtual circuits. Routing through intermediate nodes usually requires only a single lookup in a small table, packets include minimal control information, and are delivered in FIFO order. Nodes in the middle of a Dynamic Virtual Circuit can break it and later re-establish it through a different physical path, thus supporting adaptive routing while maintaining the semantics of virtual circuits. We present the basic algorithms for Dynamic Virtual Circuits and the required hardware support in the context of a VLSI communication coprocessor for multicomputers.

I. Introduction

Multicomputers, composed of thousands of computing nodes interconnected by point-to-point links, achieve high performance at a low cost by exploiting parallelism [1, 5]. Such systems rely on a *message transport* mechanism that supports high-bandwidth low-latency interprocessor communication.

Low latency communication requires minimizing the delay of forwarding packets through intermediate nodes on their way to their destinations. This is achieved using communication switches with buffers that support *virtual cut through* [12, 18] and a routing mechanism that quickly determines the output port to which the packet should be forwarded [5]. The routing scheme should direct packets through the lowest latency paths from source to destination. It should take into account the topology of the network and *adapt* to the current workload and resource availability to route packets around congested or faulty areas [16, 13, 5]. It is important to minimize the addressing and control information that must be sent with each packet as well as to maximize the availability of network resources for active connections.

With message transport based on *virtual circuits* [15, 3], many of the desirable properties described above are realized. Packets are sent through pre-established logical paths, thus minimizing packet routing time and overhead while maintaining efficient link utilization and first-in-first-out (FIFO) packet ordering on each virtual circuit. The problem with conventional virtual circuits is that the paths are static and cannot be easily changed in response to congestion or failure.

This paper introduces a new message transport mechanism, called *Dynamic Virtual Circuits*, that has the advantages of virtual circuits but allows individual nodes to make a local decision to break or reroute an existing circuit. Each node maintains sufficient information to re-establish broken circuits while preserving the FIFO ordering of packets. Since routing with Dynamic Virtual Circuits is based on tables, the scheme does not depend on a regular system topology. Hence, efficient routing can be performed even after a large number of system nodes or links have failed.

The research described in this paper is part of the UCLA ComCoBB (**Com**munication **Co**processor **B**uilding-**B**lock) project, whose focus is the design and implementation of a high-performance communication coprocessor for VLSI multicomputers. A critical aspect of the ComCoBB chip design is support for efficient routing for a wide variety of network topologies. Hence, schemes based on routing tables [16] must be supported.

Conventional static virtual circuits are described in Section II. The disadvantages of static virtual circuits are discussed in Section III, where the basic technique of Dynamic Virtual Circuits is described. The hardware support for Dynamic Virtual Circuits in the ComCoBB chip is described in Section IV.

II. Static Virtual Circuits

The two fundamental approaches to message routing in multicomputers are *circuit switching* and *packet switching* [15, 3]. With circuit switching, a static physical path is set up between the sender and receiver before communication takes place. Once the path is set, data can be transmitted quickly at nearly the full bandwidth of the links with almost no redundant control information [5]. A disadvantage of circuit switching is that physical links are statically allocated to a particular circuit for the lifetime of the circuit. Even when no information is being sent through the established circuit, the links cannot be used for other circuits.

With packet switching, the data is partitioned into small

† Supported by Hughes Aircraft Company and the State of California MICRO program. Y. Turner is supported by a Hertz Foundation Graduate Fellowship.

packets which are routed independently from source to destination. Link utilization is improved relative to circuit switching since the links along the path chosen by a packet are used only during the time required to send the packet's bytes across the link. A key advantage of packet switching is that the routing can adapt to changes in the system, potentially sending different packets with the same source and the same destination through different paths. Simulations of adaptive routing schemes have demonstrated higher reliability, higher throughput, and lower message latency than can be achieved with fixed routing [13, 5].

One disadvantage of packet switching is that each packet must contain complete addressing information as well as message and packet sequence numbers. Since packets may arrive at their destination in any order, they need to be buffered and reordered before they can be processed by the destination application. Message latency through a packet switching network is significantly larger than the latency through an established circuit since each packet of the message must be delayed at each node long enough to determine the appropriate next step on its path to the destination.

Virtual circuits are used for message transport in an attempt to combine the best features from circuit and packet switching [15, 3]. As with circuit switching, paths are established and stored in routing tables along the way. However, each physical link can be time shared among multiple virtual circuits. The physical link is logically divided into multiple virtual channels, where a field in the header byte of each incoming packet indicates the virtual channel number used by the packet. Virtual circuits are paths through the network consisting of a sequence of virtual channels. At each node, mapping tables describe the established virtual circuits passing through the node.

To establish a virtual circuit, a source node generates a Circuit Establishment Packet (CEP) that is then transmitted, as in a packet switching network, to the destination node. At each node along the way, the CEP arrives on an unused input channel of an input port. Based on the final destination of the CEP, the node routes it to one of its output ports. In the most general case, this routing utilizes large off-chip routing tables, which include information regarding the topology of the system with the cost of links weighted according to recent traffic loads [16, 3]. In addition to choosing an appropriate output port, the routing of the CEP also involves choosing an output channel from among the currently unused channels at the output port. The mapping table is then set to route future packets arriving on the same input channel and input port to the chosen output port and output channel. An example of an established virtual circuit is shown in Figure 1.

After sending the CEP, the source node sends data packets along the circuit. Each data packet contains in its header byte the input channel number on which it arrives. This is the only overhead associated with the packet, representing a significant reduction from a pure packet switching mechanism. At each node, the data packets arrive on input ports and input channels and are routed according to the entries in the mapping table to output ports and output channels. This routing is much

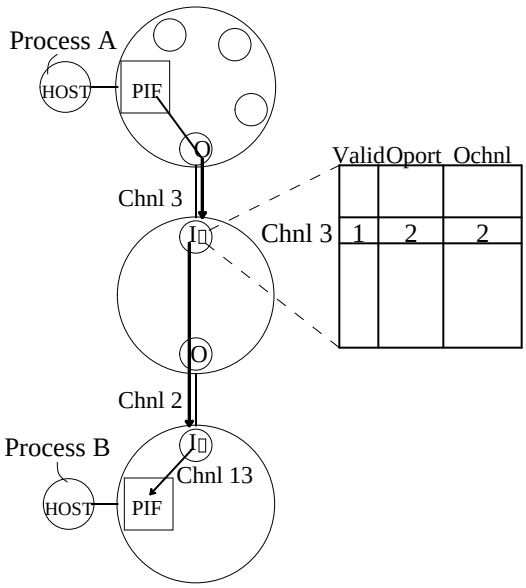

Figure 1: A virtual circuit from process A to process B. The circuit uses channel 3 of the first link and channel 2 of the second.

faster than the routing of the CEP since all it requires is a single lookup in the small on-chip mapping table. Since all packets are transmitted through the circuit along the same physical path, it is possible to guarantee FIFO order of delivery of packets at the destination. Once the virtual circuit is no longer needed, the source node removes it from the system by sending a Circuit Destruction Packet (CDP) that traverses the entire circuit, invalidating the appropriate mapping table entries at each node.

III. Dynamic Virtual Circuits

The problem with the traditional virtual circuits scheme is that the paths are static and cannot be changed in response to changes in the system (congestion or failure). Throughout a virtual circuit's lifetime, it occupies the same resources, namely one virtual channel on each physical link the circuit uses. These resources are allocated for the exclusive use of the virtual circuit, even if the circuit is idle for long time periods. In addition, circuits may permanently occupy resources if nodes or links fail or if processes terminate without tearing down their circuits. Since the number of virtual channels per link is limited, new virtual circuits may be prevented from becoming established on desired links even if idle circuits are holding resources unnecessarily.

The problem of idle circuits preventing new circuits from being established can be overcome by allowing existing circuits to be torn down *on demand*. Circuits do not necessarily persist until they are explicitly torn down. Instead, they are only *cached* in the network for as long as possible [2, 14]. When a channel is needed in a link where all the channels are allocated, a *victim* channel is selected and the entire circuit associated with this channel is torn down [9]. Disestablishing (tearing down) an existing circuit releases resources (channels) that can then be used for new circuits. With the scheme proposed in [9], once a victim channel is picked, the source

node (sender) of the corresponding circuit is signaled to tear down (delink) the circuit. This is done only when the circuit is not *active*. If a message is in the process of being transmitted through the circuit, the circuit tear down must wait until the entire message is transmitted. A long message can thus lead to long delays between the request for circuit disestablishment at an intermediate node in the circuit and the actual release of resources. Another important factor that increases this delay is the latency of transmitting the delink request to the circuit source node followed by the latency of the delink packet from the source node to the node where the channel release is needed.

Dynamic Virtual Circuits (DVCs) overcome the limitations of static virtual circuits and of the *cached circuits* scheme described above. As with cached circuits, the establishment of new circuits is guaranteed to succeed even when there are no free virtual channels on a desired link. Resources allocated to idle DVCs are eventually deallocated and used for active DVCs, as needed. With DVCs, a circuit can be disestablished from an intermediate node, without involving the source node. Hence, the circuit fragment from the source node to the intermediate node remains intact for possible use if the complete circuit needs to be reestablished. Needed channels are released immediately following a local decision at an intermediate node. Hence, DVCs eliminate the delays incurred in cached circuits due to the need to wait for disestablishment requests to be processed through the circuit source node. The delays in releasing resources due to waiting for long messages to be transmitted are also eliminated with DVCs. Messages are partitioned into short (32 byte) packets and a circuit can be disestablished at an intermediate node following transmission of the current packet. With the DVC mechanism, if a particular circuit becomes slow or blocked, due to congestion or failure, a node can make a *local* decision to break the circuit and reestablish it using an operational, less congested route. Adaptation can occur quickly since: (1) it is not necessary to wait for the source node to reroute the circuit, and (2) as the circuit is being reestablished, busy channels needed along the new path are released faster.

In the simplest case, the Dynamic Virtual Circuit mechanism is identical to the static virtual circuit mechanism. When a CEP arrives at a node, it is routed to determine the desired output port. For a regular network topology, this routing may be algorithmic [4]. For irregular networks, a more complex scheme, based on large routing tables, may be used [16]. If there is a free output channel on that output port, it is used by the new circuit and the mapping table is set accordingly.

If a CEP arrives at a node and is routed to a link with no free virtual channels, the static virtual circuit mechanism cannot be used. Instead, the Dynamic Virtual Circuit mechanism chooses an established DVC on the desired link as a victim for temporary destruction. The victim DVC is, ideally, the circuit whose next packet will enter the node furthest in the future. Once a victim is chosen, the node generates and sends a CDP along the victim channel. The CDP is marked as *nonterminal* to inform the destination node that the circuit is being torn down temporarily from an intermediate node, as opposed to permanently from the source node. After the generated CDP is sent through the output port, the CEP can be sent, establishing the new DVC.

When a DVC is first established, information regarding the ultimate destination (node identifier) of the new circuit is kept at each intermediate node. When a packet on a DVC arrives at a node where the DVC was previously cut, the information regarding the ultimate destination of the cut circuit is used to reestablish the DVC. The node chooses an output port on which to reestablish the cut circuit, creates a CEP, updates the mapping tables, and sends the CEP, reestablishing the circuit on the new path to its destination. The data packet that triggered the reestablishment of the circuit as well as future packets on the same circuit can then be sent along the new path.

Since there are multiple input and output ports operating and interacting concurrently at each node, the DVC mechanism description above, though accurate at a high level of abstraction, is overly simplistic. Care must be taken to prevent on-chip components of the chip from entering inconsistent states due to improper ordering of events or from becoming stuck forever waiting for each other to complete some operation. These issues are discussed further in Section IV.

Although DVCs provide most of the advantages and overcome the difficulties of static virtual circuits, they do not guarantee the physical FIFO packet arrival at the destination node as with static virtual circuits. For example, a circuit that has been torn down from an intermediate node may be reestablished on a different path before the nonterminal CDP reaches the destination. In this case, the packets on the reestablished branch of the circuit arrive at the destination before all the packets on the torn-down branch arrive. Since proper packet ordering is vital, some mechanism must be provided to allow the destination node to determine the order in which packets were sent by the source node. Two such mechanisms, one based on packet sequence numbers and one based on logical timestamps, are described and compared here. Both mechanisms rely on providing enough information in CDPs and CEPs to allow the destination node to identify and order arriving branches belonging to the same circuit.

With the packet sequence number mechanism, for each circuit established at a node, a packet count register records how many packets have arrived at the node on the circuit. A sliding window protocol can be used to bound the maximum value of the packet counters. Also stored at each intermediate node is other information needed to uniquely identify the particular DVC, such as source and destination process and processor identifiers. When a circuit is reestablished, the packet count at the node initiating the reestablishment is sent to the circuit destination, together with the information originally used to establish the circuit (source process id, destination process id, etc). The destination node accepts packets on reestablished circuits only after its local packet counter indicates that all previous packets have already been received. Packets which cannot be accepted are buffered at the destination node until they can be accepted. The main disadvantage of this scheme is that it requires dedicated

hardware at each input port to store and increment the packet counters.

The timestamp mechanism avoids the use of dedicated packet counters for each incoming circuit at each input port. The mechanism requires only a small amount of information to uniquely identify branches, and it updates circuit information at the intermediate node only when a circuit is reestablished or disestablished from that intermediate node. Because the circuit information is rarely updated, it is not necessary to store it on-chip or provide dedicated hardware for updating. Each node maintains a count of the number of nonterminal circuit destructions the node has initiated. This count serves as a logical timestamp that, in conjunction with the identifier of the node where the circuit was broken, uniquely identifies the circuit destruction event. The counter has to be sufficiently large (40-64 bits) so that there would be no danger of the counter ''wrapping around'' leading to possible incorrect packet ordering. When the torn down DVC is to be reestablished, the stored destruction timestamp and the node identifier are sent with the CEP. When a circuit branch CEP arrives at the destination node, a matching circuit branch CDP must be found with the same timestamp and node identifier values. The only such CDP is the one terminating the branch to be ordered just prior to the CEP's branch.

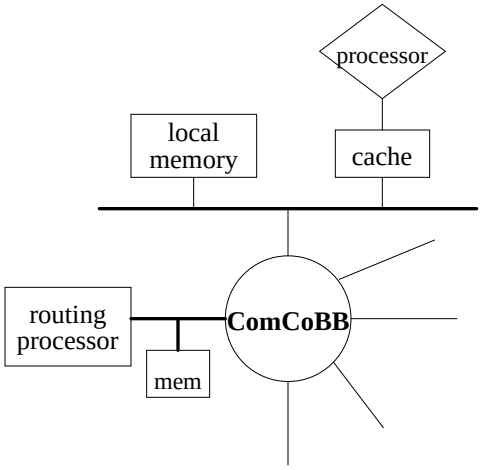

Figure 2: A multicomputer node.

IV. Implementation of Dynamic Virtual Circuits

Figure 2 shows a multicomputer node with the application processor, local memory used by the application processor, the ComCoBB chip, and a special routing processor with its memory. The routing processor is a general-purpose processor which is used as a dedicated controller to perform some of the infrequent but complex operations that are needed to support DVCs. Frequent operations, such as routing and forwarding of a packet on an established circuit, are handled entirely within the ComCoBB chip, using dedicated hardware. The routing processor handles tasks such as, initiating circuit destruction, reestablishing a circuit, updating of global routing tables [16], and resolution of deadlocks. It should be noted that, on a large chip, the routing processor and its memory may be implemented as a dedicated programmable controller on the same chip with the communication switch.

Tables which are accessed for every packet forwarded through the switch are stored at the input and output ports of the ComCoBB chip (see Figures 3 and 4). However, less frequently accessed tables are stored in the private memory of the routing processor. The tables in the routing processor's memory are:

- The *Circuit Destruction Table*: One entry per incoming channel to the node. Each entry contains a timestamp identifying the teardown time and the destination node identifier for the torn-down circuit. This information is placed in the CEP generated for reestablishing the circuit at a later time. The table is indexed by input port and channel numbers.
- The *Inverse Output Mapping Table* (INV): One entry per output channel from the node. Maps each output channel to the corresponding input port and input channel number. The mappings are recorded upon circuit establishment. Each entry also contains a single *reserved* bit, which the Routing Processor sets when the circuit is picked to be torn down and resets when the teardown is completed.
- The *Routing Table*: One entry for each destination node identifier. Each entry contains the output port on the shortest delay path to the destination node. This table is accessed during circuit establishment.

In addition to the tables above, for DVCs which originate in each node, the node maintains a single Source Table, which maps logical DVC identifiers to channel numbers for the first hop of the DVCs. At each node, for DVCs whose destination is the node, there is a single Destination Table, which maintains the information (CEP and CDP timestamps) necessary for ordering packets arriving over different paths of the same DVC. On its way from the source node to the destination node, each packet requires one access to a Source Table and one access to a Destination Table. Hence, the ComCoBB chip's processor interface must support fast access to these tables.

The ComCoBB chip consists of four input ports, four output ports, the Processor Interface (PIF) to the application processor, and the Routing Processor Interface (RPI). The input and output ports each consist of eight data lines and one flow control line. The input port uses the flow control line to stop the output port from sending data when, for example, the buffer at the input port becomes full. The RPI translates read and write requests by the routing processor to both read/write operations on storage elements inside the ComCoBB chip and commands affecting the behavior of ComCoBB modules. The RPI also fields interrupt requests raised by ComCoBB modules and passes them on to the routing processor.

A. Input Port Hardware and Operation

When a packet arrives, it is placed in the input buffer and routed to determine the desired output port. Once the packet reaches the head of the buffer, the buffer makes a request to the crossbar for a connection to the desired output port. After the request is granted, the packet is removed from

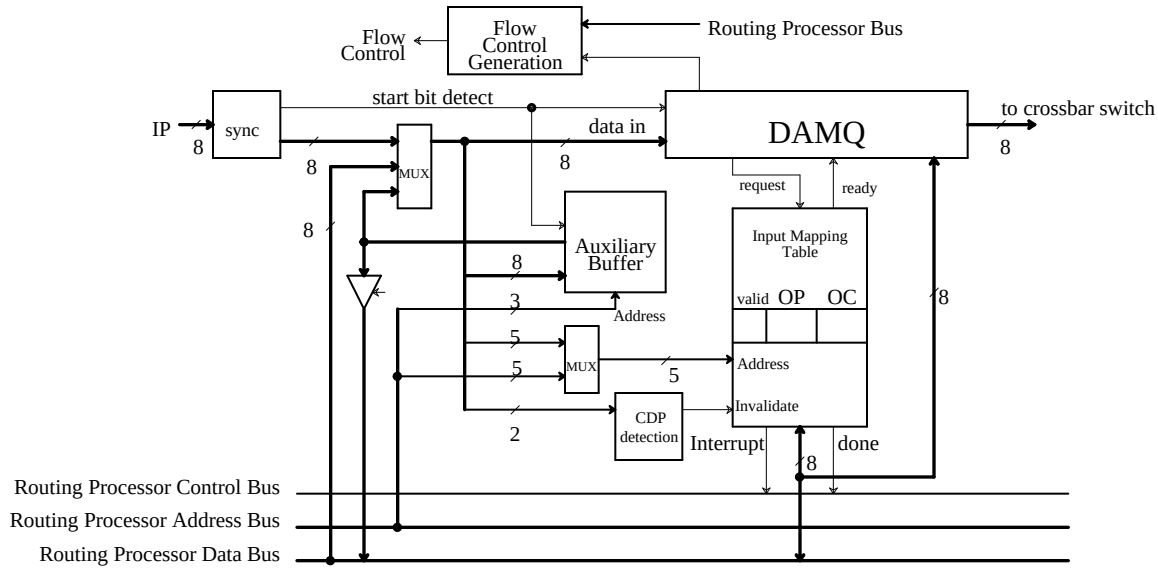

Figure 3: Input port routing hardware.

the buffer and sent through the crossbar switch and the output port to the neighboring ComCoBB chip.

Figure 3 shows a block diagram of the hardware located at each input port of the ComCoBB. There are four main components: the Synchronizer [17], the dynamically-allocated multi-queue (DAMQ) input buffer [18], the Auxiliary Buffer, and the Input Mapping Table (IMT). The Synchronizer produces eight bits of data synchronized to the local clock. These signals are input to both the DAMQ buffer, which is the main packet buffer at the input port, and to the Auxiliary Buffer, which is a much smaller FIFO buffer and usually holds the first eight bytes of the most recent packet arriving through that input port. In normal operation, as packets arrive they are placed in both the DAMQ buffer and the Auxiliary Buffer. In addition, the header byte of each incoming packet is forwarded to the Input Mapping Table for lookup. If the lookup references a valid entry, the header is modified to contain the output channel number, and the new header is latched into the DAMQ buffer. If the access references an invalid entry, the Input Mapping Table raises an interrupt for the routing processor and causes the DAMQ buffer control to use the flow control line to stop traffic into the input port. A packet arriving on an input channel that has no valid mapping is either a Circuit Establishment Packet or a packet arriving on a circuit that has been disestablished from this node. In either case, routing processor intervention is required and incoming packet flow must be stopped.

The DAMQ buffer normally takes its input from the output of the Synchronizer, but it can also take its input either from the Auxiliary Buffer or from a packet buffer located at the RPI via the routing processor data bus. The DAMQ input comes from the RPI when, for example, the routing processor needs to insert a CDP into the circuit. The DAMQ input comes from the Auxiliary Buffer when forwarding a CEP which was held in the buffer while the corresponding IMT entry was set up. Care must be taken to ensure that flow from the input port is halted when the input to the DAMQ buffer is taken from one of the other sources. Otherwise, packets arriving on the input port will be lost.

In some cases, the routing processor requires information contained in the body of the packet. For example, when a CEP arrives, the header byte indicates the input channel, and subsequent bytes of the packet indicate the desired final destination. To access these bytes, the routing processor can read the Auxiliary Buffer whenever it is not being written by the input port.

The CDP that destroys a circuit is generated by the routing processor. However, circuit destruction originating in a remote node can be handled without the intervention of the routing processor. To support this fast handling of CDPs at the input port, an *Invalidate* input is added to the Input Mapping Table. This signal causes the table lookup to mark the entry referenced as invalid. A CDP automatically triggers this operation.

B. Output Port Hardware and Operation

Figure 4 shows a block diagram of the hardware at each output port. This hardware consists of a table and logic for picking victim channels. The Output Port Table (OPT) is used to keep track of valid and invalid output channels and to maintain output channel use information. There are 32 entries in the OPT, one per output channel. The OPT is normally accessed when packets arrive at the output port from the crossbar — the entry corresponding to the channel number of the packet is updated. In addition, the table is accessed by the routing processor when a DVC is established. Each table entry consists of two bits: *valid* and *use*. The *valid* bit specifies whether the output channel is part of a circuit. The *use* bit indicates whether a packet has been sent on the corresponding output port recently.

The information in the Output Port Table drives the circuit that selects a victim when there is a need to find a free

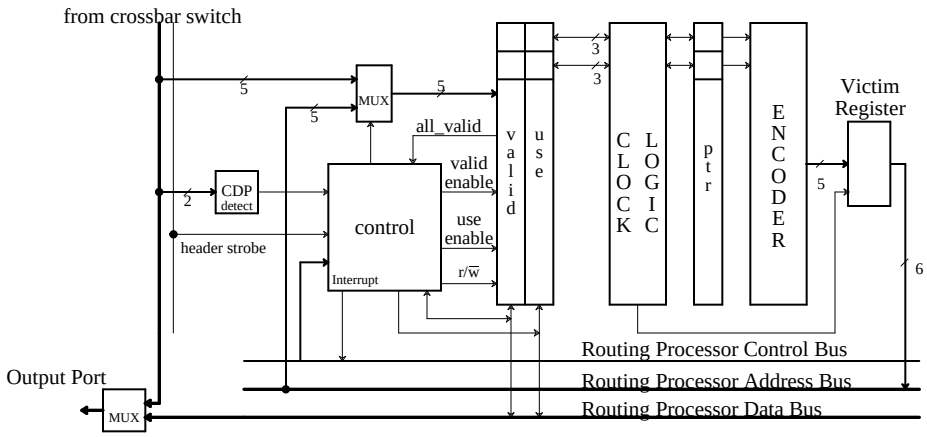

Figure 4: Output port logic. Invalidates circuits and picks victim output channels.

channel for use in establishing or reestablishing a DVC through the output port. The victim selection module is a combinational circuit that continuously computes the victim output channel number. The victim number is placed in the Victim Register, from which it can be read by the routing processor. If there are any invalid output channels (*i.e.*, channels not on established circuits), these are picked by the victim selection logic. If all the output channels are allocated to established DVCs, one of those channels is picked and the corresponding DVC is disestablished, starting from this node. The ''clock'' replacement algorithm, commonly used for page replacement in virtual memory [7], is used to pick the victim established DVC.

C. Sequencing of ComCoBB Operations

Some of the operations performed by the ComCoBB involve several sequential steps. As mentioned in Section III, proper ordering of on-chip events is crucial for avoiding inconsistent states. For example, in order to establish a new DVC, it is sometimes necessary to tear down an existing DVC to free an output channel. A straightforward but incorrect procedure for performing this operation would have the routing processor reset the *valid* bit for the victim channel at both the IMT and at the OPT as soon as a victim is selected. Then, the routing processor would create a CDP and send it directly out the output port. Once the CDP is sent, the pending circuit establishment request would be fulfilled.

The procedure above is wrong because there may be packets, belonging to the victim circuit, that are enqueued in the DAMQ buffer at the input port used by the victim circuit. If the mapping tables are changed and the CDP sent before these enqueued packets exit the node, the packets will be forwarded out the same output port and output channel as the packets on the new circuit being established, thus mixing packets of different circuits. Also, one of the enqueued packets may be a CDP, rendering the creation of a CDP by the routing processor redundant.

The correct procedure incorporating these considerations is shown in Figure 5. This figure shows the sequence of low-level operations required when a data packet arrives. If the IMT entry is not valid, the routing processor must reestablish the DVC to the circuit's destination. As shown in step 2 of Figure 5, the routing processor accesses the Circuit Destruction Table to determine the destination node id for the data packet. Next, in steps 3 and 4, the routing processor picks a victim output channel for use in the new DVC. To do this, it reads the Victim Register, that contains the selected channel number as well as the corresponding *valid* bit. The routing processor then checks the Inverse Output Mapping Table, stored in its private memory, to ensure that this channel had not already been reserved for a different DVC destruction. If the Inverse Mapping Table indicates that the selected channel is reserved, the routing processor reads the Victim Register again to get a different victim.

Assuming that the output port table entry for the chosen victim is valid (i.e., the test in step 5 succeeds), the following procedure correctly tears down the victim circuit and reestablishes the circuit for the data packet:

1. The victim input port and input channel numbers IP′ and IC′ are found in the Inverse Output Mapping Table. The routing processor writes a command to the ComCoBB which causes the ComCoBB to assert the flow control line, thus stopping packet flow from the neighbor node to the victim input port (steps 6 and 7).
2. If the IMT entry corresponding to the victim circuit is valid, this means that no disestablishment of the victim circuit is in progress at the local node. If this is the case, the routing processor creates and enqueues a CDP at the victim input port DAMQ buffer and restarts packet flow (steps 8 through 10);
3. At this point (between steps 10 and 11), while the CDP is waiting for its turn in the DAMQ buffer, the routing processor returns to its normal mode of waiting for interrupts from the ComCoBB chip. When the CDP is finally sent from the DAMQ buffer through the crossbar to an output port, the routing processor is interrupted again by the output port logic (step 11);
4. At this point the victim circuit has been torn down, freeing the victim output channel for use by the DVC being reestablished. The routing processor sets up the mapping tables for this DVC, creates a CEP, and inserts the CEP at

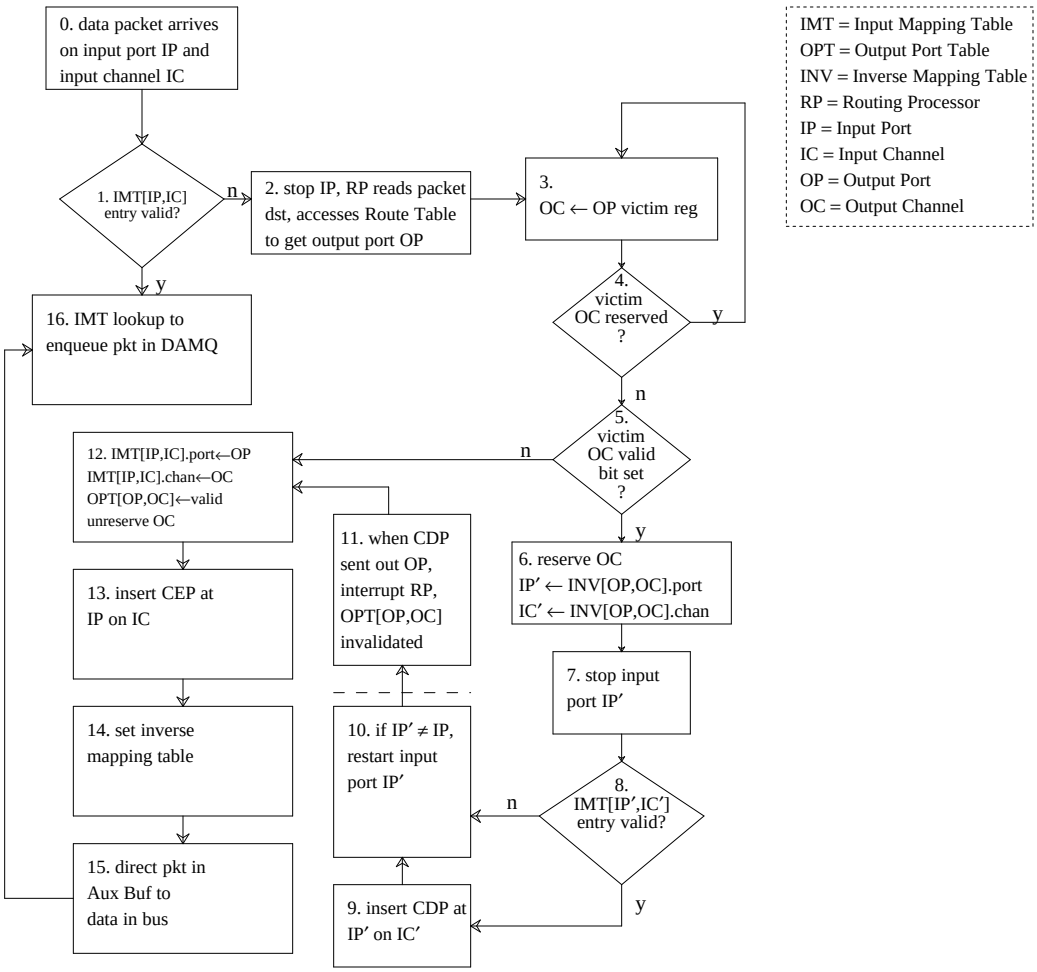

Figure 5: Handling of data packets arriving at an input port.

the data packet's input port. Once this is done, the data packet can be directed to the DAMQ buffer (steps 12 through 16).

D. Support for Deadlock Resolution

In general, multicomputer interconnection networks are susceptible to *deadlock* situations when no messages can advance towards their destinations due to full buffers in one or more cycles of communication switches [15]. With a global view of the interconnection topology, it is possible to guarantee deadlock-free routing by restricting the routing algorithm and the use of the buffers at each node [8]. When deadlocks do not occur, deadlock-free routing results in inefficient use of system resources. An alternative approach to dealing with deadlocks is to detect and resolve them when they occur [11, 6]. With this approach there is no restriction on the routing and all network resources are used for improved performance rather than reserved for eliminating deadlocks. Since Dynamic Virtual Circuits are designed to support distributed adaptive routing in arbitrary topologies [16], the choice of the latter approach to dealing with deadlocks is clear.

When an input DAMQ buffer remains full for some time without transmitting any packets, a deadlock may exist. Following the algorithm proposed in [11], when a node decides that it *may* be in a deadlock, it first attempts to determine if there is a cycle of nodes with full buffers which may form a *potential* deadlock. If such a cycle is found, the packets are rotated along the cycle so that they move towards their destinations. Eventually, this process leads to one of the buffers becoming not-full, thus resolving the deadlock. Further details of the basic algorithm [11] will not be described here.

The cycle detection and deadlock resolution algorithms require, at each switch, a fixed amount of storage that is available even when the normal buffer is full [11]. This *Auxiliary Buffer* is used to receive control packets as well as for receiving a data packet during the rotation phase of the algorithm. The control packets and fragments of data packets can be read by the routing processor from the Auxiliary Buffer. In addition, it is necessary for the routing processor to be able to send a control packet directly to a specified output port. All the connection and controls necessary for these operations are provided in the ComCoBB design (Figures 3 and 4).

The cycle detection and deadlock resolution algorithms require the switch to receive packets even if the input buffer is full. Thus, a simple flow control line that inhibits new packets

from being transmitted by the neighbor whenever the buffer is full is insufficient. For cycle detection or deadlock resolution it must be possible for the routing processor to allow transmission of one control packet or one 8-byte fragment of a data packet. To support this feature, a second flow control line can be added to handle enabling and disabling when the input buffer is full. Alternatively, a protocol requiring only one flow control line can be used with very little loss in overall performance. Specifically, when an input buffer is full, but the node is ready to receive eight bytes to be placed in the auxiliary buffer, the node sends a special control message to its neighbor. This control message is not placed in the DAMQ buffer *or* the auxiliary buffer of the receiver. Instead, it simply sets a single-bit state variable. This technique relies on the ability of the routing processor to transmit a packet to the output port without using the crossbar. The special control message can be sent only when the output port is not otherwise occupied. Hence, it may have to wait for completion of the current packet transmission.

V. Summary and Conclusions

Dynamic virtual circuits combine the best features of the traditional circuit switching, packet switching, and static virtual circuits. This new message transport mechanism minimizes the addressing and control information sent with each packet and the latency of forwarding packets through each intermediate node. Unlike static virtual circuits, DVCs can adapt to congestion or failures in the system by allowing nodes to make local decisions regarding the possible need to reroute the circuit through a different physical path. We have described the basic techniques for allowing circuits to be broken and reestablished while maintaining the semantics of traditional virtual circuits.

We have presented an overview of the hardware necessary to support DVCs in the context of a complete communication coprocessor for multicomputers. Dedicated hardware is used on-chip to handle the critical frequent case, while an off-chip routing processor is used for more complex but less frequent tasks. Based on the costs of handling the complex cases, it is clear that high performance requires that the network message traffic exhibit high spatial and temporal locality, leading to a low rate of involuntary circuit teardowns and subsequent reestablishments. Fortunately, recent measurements of multicomputers indicate that such locality can be expected in real applications [10]. An additional feature of the hardware proposed in this paper is that it supports a deadlock resolution algorithm that does not require limiting the flexibility of the routing scheme. This leads to efficient utilization of system resources, and thus to high performance, during normal operation.

Acknowledgements

Some of the ideas presented in this paper were explored by Gregory Frazier and Tiffany Frazier in course term projects at UCLA in 1988. Tiffany Frazier suggested the idea of using logical timestamps for maintaining packet ordering. Helpful suggestions regarding this work were also made by David Rennels, Bill Smith, and Vance Tyree.

References

1. W. C. Athas and C. L. Seitz, ‘‘Multicomputers: Message-Passing Concurrent Computers,’’ *Computer* **21**(8), pp. 9-24 (August 1988).
2. H. G. Badr, D. Gelernter, and S. Podar, ‘‘An Adaptive Communications Protocol for Network Computers,’’ *Performance Evaluation* **6**(1), pp. 35-51 (March 1986).
3. D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall (1987).
4. M.-S. Chen, K. G. Shin, and D. D. Kandlur, ‘‘Addressing, Routing, and Broadcasting in Hexagonal Mesh Multiprocessors,’’ *IEEE Transactions on Computers* **39**(1), pp. 10-18 (January 1990).
5. E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, ‘‘Hyperswitch Network for the Hypercube Computer,’’ *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 90-99 (May 1988).
6. I. Cidon, J. M. Jaffe, and M. Sidi, ‘‘Local Distributed Deadlock Detection by Cycle Detection and Clustering,’’ *IEEE Transactions on Software Engineering* **SE-13**(1), pp. 3-14 (January 1987).
7. F. J. Corbato, ‘‘A Paging Experiment with the MULTICS System,’’ Project MAC Memo MAC-M-384, MIT, Cambridge, MA (July 1968).
8. W. J. Dally and C. L. Seitz, ‘‘Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,’’ *IEEE Transactions on Computers* **C-36**(5), pp. 547-553 (May 1987).
9. J.-M. Hsu and P. Banerjee, ‘‘Hardware Support for Message Routing in a Distributed Memory Multicomputer,’’ *1990 International Conference on Parallel Processing*, St. Charles, IL (August 1990).
10. J.-M. Hsu and P. Banerjee, ‘‘Performance Measurement and Trace Driven Simulation of Parallel CAD and Numeric Applications on a Hypercube Multicomputer,’’ *Proceedings 17th International Symposium on Computer Architecture*, Seattle, WA, pp. 260-269 (May 1990).
11. J. M. Jaffe and M. Sidi, ‘‘Distributed Deadlock Resolution in Store-and-Forward Networks,’’ *Algorithmica* **4**(3), pp. 417-436 (1989).
12. P. Kermani and L. Kleinrock, ‘‘Virtual Cut Through: A New Computer Communication Switching Technique,’’ *Computer Networks* **3**(4), pp. 267-286 (September 1979).
13. J. Y. Ngai, ‘‘A Framework for Adaptive Routing in Multicomputer Networks,’’ Computer Science Technical Report 89-09, California Institute of Technology, Pasadena, CA (May 1989).
14. D. A. Reed, P. K. McKinley, and M. F. Barr, ‘‘Performance Analysis of Switching Strategies,’’ *Proceedings 1987 Symposium on the Simulation of Computer Networks*, pp. 130-141 (August 1987).
15. D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, The MIT Press (1987).
16. W. D. Tajibnapis, ‘‘A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network,’’ *Communications of the ACM* **20**(7), pp. 477-485 (July 1977).
17. Y. Tamir and J. C. Cho, ‘‘Design and Implementation of High-Speed Asynchronous Communication Ports for VLSI Multicomputer Nodes,’’ *International Symposium on Circuits and Systems*, Espoo, Finland, pp. 805-809 (June 1988).
18. Y. Tamir and G. L. Frazier, ‘‘High-Performance Multi-Queue Buffers for VLSI Communication Switches,’’ *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 343-354 (May 1988).