Computer Science Department Technical Report #TR050021
University of California, Los Angeles, June 2005.

Deadlock-Free Connection-Based Adaptive Routing with Dynamic Virtual Circuits

Yoshio Turner and Yuval Tamir
Concurrent Systems Laboratory
Computer Science Department
UCLA
Los Angeles, CA

Abstract

Virtual circuits can reduce routing overheads with irregular topologies and provide support for a mix of quality of service (QOS) requirements. Information about network loads and traffic patterns may be used during circuit establishment to utilize network resources more efficiently than is practical with packet routing. Most virtual circuit schemes are static — each established virtual circuit remains unchanged until the connection is no longer needed. In contrast, we propose the *Dynamic Virtual Circuit* (DVC) mechanism, which enables existing circuits to be quickly torn down in order to free up resources needed for other circuits or to re-establish circuits along routes that are better suited for current network conditions. We propose a deadlock avoidance technique, based on unconstrained routing of DVCs combined with a deadlock-free virtual network. We present a correctness proof for the scheme, describe key aspects of its implementation, and present performance evaluation results that explore its potential benefits.

Keywords: adaptive routing, deadlock, virtual circuits, interconnection networks.

1. Introduction

The routing scheme used in multicomputer or high-end cluster interconnection networks 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 [6, 11, 16, 21, 25, 29, 35]. However, multicomputer and cluster networks use backpressure flow control in which packets that encounter congestion are blocked rather than discarded. A sequence of blocked packets can form a deadlock cycle unless sufficient constraints are imposed on packet routes and/or buffer usage [1, 5, 14, 15, 19, 28]. The routing scheme should therefore minimize routing constraints while preserving deadlock-freedom.

The routing scheme should also minimize the overhead costs for routing and forwarding packets. The addressing and control information that is sent with each packet should be minimized to enable efficient utilization of link bandwidth. The processing to interpret and route each packet at each hop should be minimized to provide low latency communication. These goals can be achieved by using *connection-based* routing, in which resources are reserved in advance of communication at each switch along the path from source to destination. This approach reduces the overhead for forwarding each packet but introduces connection setup/teardown operations. Connection-based routing can improve network efficiency so long as these new operations are fast or occur infrequently, e.g., for workloads in which each connection is used by several packets [23].

Virtual circuit switching [7, 12, 23, 34] is a form of connection-based routing which provides end-to-end connections called *virtual circuits*. The network's physical resources (packet buffers, link bandwidth, etc.) are multiplexed among active virtual circuits. Once a virtual circuit is established, packets can be sent without the addressing and sequencing information needed in pure packet switched networks. Packets are quickly forwarded along the virtual circuit's network path by using a single lookup in a small virtual channel table at each hop. This mechanism enables low overhead packet forwarding even in networks with irregular topologies [18]. In these topologies, intermediate nodes between a source and destination cannot use simple algorithmic routing. Instead, routing is based on large tables at each node which are constructed at system initialization and may be changed over time to adapt to changing traffic conditions [1, 19, 29]. With virtual circuits, the routing tables can be used during virtual circuit establishment to determine a virtual circuit's network path. Subsequently, packets are forwarded along the path using the more efficient virtual circuit forwarding mechanism.

Virtual circuit switching can be a valuable extension for current and emerging cluster and I/O interconnects. An important example is the industry standard InfiniBand architecture [1, 17], which is

supported by major systems vendors (e.g., IBM, HP, and Dell) and operating systems (e.g., Linux, and announced support in Microsoft Windows), and is increasingly deployed in clusters (e.g., the NASA Columbia system, which as of November 2004 had the number two position on the TOP500 [2] list of supercomputer sites). InfiniBand matches our system model in its use of backpressure flow control and routing tables at each switch to support irregular topologies. In addition, one of the main communication mechanisms provided by InfiniBand requires establishing a connection between a ‘‘queue pair’’ at a source node and a queue pair at a destination node. However, network switches do not reserve state for connections. Thus each packet must have routing and transport headers specifying source node and queue pair IDs, destination node and queue pair IDs, a service level indicating the scheduling priority for the connection, etc. If InfiniBand switches and protocols were extended to support establishment of virtual circuits for queue pair connections, much of this information could be eliminated, improving transmission efficiency.

Most virtual circuit schemes have the limitation that each established circuit’s route cannot be changed until the connection is no longer needed and the circuit is torn down. This prevents adaptation to changes in traffic patterns, and it prevents establishment of new virtual circuits once all the required resources are assigned to existing circuits. To solve this problem we have proposed the *Dynamic Virtual Circuits* (DVC) mechanism [30, 32] which combines adaptive routing and connection-based routing. With DVCs, a portion of a virtual circuit can be torn down from an intermediate node on the circuit’s path and later be re-established, possibly along a different route, while maintaining the virtual circuit semantics of reliable in-order packet delivery. The DVC mechanism provides fast circuit establishment and re-routing by using only local operations at each node. In addition, packets can proceed on new circuit paths without waiting for end-to-end handshakes.

In this paper, we present the DVC algorithm and propose a hardware/firmware architecture for its implementation. The challenge in devising the algorithm is to simultaneously provide fully adaptive routing, deadlock-freedom, and the low per-hop overhead of static virtual circuit switching. Compared to pure packet switching networks, DVC networks pose a more complicated deadlock avoidance problem because DVCs introduce dependencies among circuit establishment and teardown operations and also dependencies between these operations and packet buffer resources.

The main contributions of the paper are as follows. First, we present the deadlock-free DVC algorithm. The algorithm allows virtual circuits to use any routes and imposes few constraints on packet buffer usage. A dependency cycle-free virtual network is used to avoid packet routing deadlocks, and a second virtual network decouples circuit manipulation and packet routing operations to avoid more

complex deadlocks. Second, we present a description and evaluation of the hardware resources and mechanisms needed to implement the DVC algorithm and show that the scheme is simple to implement with modest hardware requirements. Third, a correctness proof of the DVC algorithm is presented. The proof analyzes the system's state space to show that each packet injected into the network using a DVC is delivered to the DVC destination in order.

The paper is organized as follows. Section 2 reviews approaches for avoiding packet buffer deadlocks in traditional networks. Section 3 describes Dynamic Virtual Circuit (DVC) networks. Section 4 presents the DVC algorithm including the proposed mechanisms for avoiding deadlocks in DVC networks. Section 5 presents a correctness argument for the scheme, and Section 6 describes practical implementation issues. Section 7 presents simulation results that explore the potential performance benefits of DVCs. This is done by considering limit cases, where the more sophisticated (complex) routing possible with DVCs leads to significantly higher performance than can be achieved with conventional packet switched networks, which typically must use simple (e.g., algorithmic) routing. Section 8 summarizes related work in connection-based and adaptive routing.

2. Background: Packet Buffer Deadlock Avoidance

To provide deadlock-freedom in DVC networks, our solution builds on previous deadlock avoidance techniques for traditional networks. Packet routing creates dependencies between packet buffer resources at adjacent switches. Cycles of these dependencies can cause deadlocks which prevent packets from making progress. A simple way to prevent deadlocks is to restrict packet routing such that dependency cycles do not exist, for example by using Dimension Order Routing (DOR) in a mesh network [10]. However, such restricted routing may result in poor performance because of insufficient flexibility to route packets around congested network links or buffers.

A network can provide higher performance by using a less restricted routing function that guarantees deadlocks are avoided despite having buffer dependency cycles. The key is to ensure that packets can escape from any dependency cycles they encounter. This approach was generalized by Duato, who determined necessary and sufficient conditions for deadlock-free routing in cut-through and wormhole networks [14, 15]. Stated informally, there must be a set of packet buffers that can be reached by packets in any buffer in the network, and this set of packet buffers acts as a deadlock-free escape path for the delivery of blocked packets.

An escape path from dependency cycles can be provided by embedding in the network a *virtual network*, consisting of a set of dedicated packet buffers, for which packet routing is free of dependency

cycles [4, 11, 35]. For example, in the Disha scheme [4] the physical network is partitioned into two virtual networks, one which allows fully-adaptive routing with dependency cycles, and a second which provides a dependency cycle-free escape path. A packet that blocks in the fully-adaptive virtual network for a timeout period becomes eligible to be transferred to the dependency cycle-free network, which is guaranteed to deliver the packet without creating a deadlock.

In the following sections we describe the DVC algorithm. The scheme avoids packet buffer deadlocks by using a dependency cycle-free virtual network. It also ensures that new types of dependencies introduced by virtual circuit setup and teardown procedures cannot cause deadlocks.

3. Dynamic Virtual Circuits (DVCs)

With both static virtual circuits and DVCs, a source node initiates communication with a destination by establishing a new virtual circuit on some path to the destination. The source node then transmits one or more data packets over the new virtual circuit. The data packets are forwarded at each intermediate switch with very little processing and carry only a few bits of overhead (packet header) information. Finally, the source terminates the virtual circuit. Each source and each destination may have many virtual circuits established at the same time.

Unlike static virtual circuits, DVCs enable any intermediate switch on a circuit's path to tear down the circuit for the portion of the path from the switch to the destination. The intermediate switch may later re-establish the torn down portion of the circuit in order to forward additional data packets that arrive. The new path chosen for the circuit may be different from the original path. For example, the new path may be chosen because it is less congested than the original.

With DVCs, circuit teardown and re-establishment from an intermediate node are fast, local operations that completely avoid time-consuming synchronization with the circuit's endpoints. However, without synchronization a race condition can develop in which data packets that traverse a new path after circuit re-establishment can arrive at the destination before older data packets on the original path. To preserve the FIFO delivery semantics of virtual circuits, the destination reorders the arriving packets that belong to the same circuit. As discussed below, to facilitate this packet reordering, each time a circuit is re-established one sequence number must be sent through the network. However, the overwhelming majority of the packets do not include sequence numbers [30]. Although the destination needs to reorder packets when circuits are torn down and later re-established, this is a rare event; packets almost always arrive in order. Hence, there is not need to devote resources to optimizing the performance of packet reordering at the receiver.

We next describe the basic steps of DVC establishment, disestablishment, and rerouting. Consider a virtual cut-through network composed of $n \times n$ switches interconnected with bidirectional point-to-point links. At each switch, one or more ports may connect the switch to one or more hosts. The switch is input buffered with an $n \times n$ crossbar connecting the n input ports to the n output ports.

The source host establishes a new DVC by injecting a Circuit Establishment Packet (CEP) into the network. The CEP records the DVC's source and destination addresses, which are used to adaptively route the CEP. For example, CEP routing may be accomplished through the use of routing tables maintained at each switch, such as in the SGI Spider chip [19].

A CEP allocates for a new DVC one *Routing Virtual Channel* (RVC) on each link it traverses on its path from source to destination (including the source and destination host interface links). An RVC is an entry in a table at the switch input port that is connected to the link. Each physical link is logically subdivided into multiple RVCs. Each packet header has a field that identifies the RVC used by the packet. At each switch, the mapping from input RVC to output RVC is recorded in an ''Input Mapping Table'' (IMT) at the input port. The IMT is indexed by the RVC value in the header of an arriving packet. An IMT entry records the following information about the DVC that has allocated the RVC: the output port, output RVC value, source and destination addresses, and sequence number.

Note that we use the term ''RVC'' instead of the more familiar ''virtual channel'' to distinguish it from the same term commonly used to refer to flow-controlled buffers that prevent deadlock and increase performance [10]. We call the latter *Buffering Virtual Channels*, or ''BVCs''. In contrast, ''RVCs'' simply identify DVCs, and they do not require separate flow-controlled buffers.

The source may transmit one or more data packets over a new virtual circuit. Each data packet is quickly routed at each switch, by accessing the IMT entry with the RVC value in the packet header. The RVC value in each packet's header is overwritten with the output RVC value recorded in the IMT entry, and the packet is enqueued for transmission to the next switch.

The source host terminates the virtual circuit by injecting a Circuit Destruction Packet (CDP) into the network. The CDP traverses the circuit path, releasing at each hop the RVC that is allocated to the circuit after the data packets that use the circuit have departed the switch.

With DVCs, an intermediate switch may initiate a teardown by inserting a CDP into the circuit path at the switch input port. The switch may tear down the circuit to free up an output RVC to allocate to a new circuit. Alternatively, it may tear down the circuit to adapt to traffic conditions by shifting the circuit onto a lower latency path. The CDP traverses the circuit's path until it reaches either the destination or another switch that has also torn down the same circuit. The portion of the circuit from the

source to the intermediate switch remains intact, unless the source or a switch along that portion of the circuit also initiates a teardown.

A data packet that arrives on the input RVC of a torn-down DVC triggers DVC re-establishment, in which the switch creates a new CEP from the information retained in the IMT. The CEP is routed to the destination, allocating RVCs on the new path. There are no restrictions on when to reroute DVCs or which new paths to take.

The adaptive rerouting of DVCs requires some packets to be stamped with sequence numbers for reordering at the destination. Specifically, each CEP is stamped with the sequence number for the next data packet of the circuit. Each switch along the path records the sequence number in the CEP as a circuit is established or re-established. The switch increments the sequence number for each data packet that subsequently arrives on the circuit.

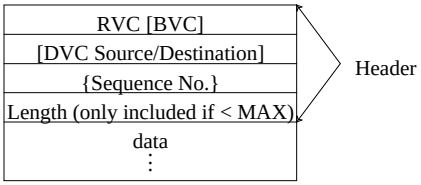


Figure 1: Packet Format. Fields in ‘‘[]’’ have the stated use only for diverted data packets (described in Section 4.1). The sequence number field is used only for CEPs, diverted data packets, and the next non-diverted data packet.

The general packet format is shown in Figure 1. A packet consists of a header followed by data phits. The first phit of the header records the RVC value. An additional four bits of the RVC field indicate packet type and whether the packet is of maximum length. If not, a length field is present in the header. For most packets, the header consists only of the RVC field and possibly the length field. For a minority of packets, the header includes additional fields. These additional fields are required for data packets that are diverted onto deadlock escape paths.

4. DVCs With Deadlock Avoidance

This section shows how the mechanism for tearing down and re-establishing Dynamic Virtual Circuits can be combined with a deadlock avoidance scheme that provides packets with an escape path from potential deadlock cycles. Data packets that take the escape path are routed individually to their destinations, independently of the virtual circuit paths.

4.1. Avoiding Deadlocks Involving Data Packets

The DVC mechanism supports adaptive routing by imposing no restrictions on the choice of path for any circuit, and by enabling circuits to be rerouted adaptively during their lifetimes onto new paths to minimize latency and maximize throughput. The flexibility of adaptive routing comes at the cost of possible deadlocks that involve the packet buffers. Deadlock cycles may also involve RVCs, since those resources are contended for by the circuit establishment and disestablishment operations at a switch.

To avoid deadlocks arising from the unrestricted paths of DVCs, we embed in the physical network two virtual networks: the *primary network* and the *diversion network* [15]. Each virtual network is composed of one *Buffering Virtual Channel* (BVC) per switch input port (i.e. per link). We name the two BVCs the *primary BVC* and the *diversion BVC*. Each is associated with a buffer (the ‘‘primary’’ and ‘‘diversion’’ buffers), which may be a FIFO buffer, or a more efficient Dynamically Allocated Multi-Queue (DAMQ) buffer [31], or a buffer with any other organization.

The primary network supports fully-adaptive routing. This allows data packets to follow the unconstrained paths that are assigned to virtual circuits, but it also creates dependency cycles among packet buffers. In contrast, routing in the diversion network is constrained such that dependency cycles do not exist (e.g., in a mesh topology, Dimension-Order Routing (DOR) could be used in the diversion network). In addition, the diversion network can accept blocked packets from the primary network to provide a deadlock-free escape path [15]. We say that a data packet is *diverted* if it takes a hop from the primary network into the diversion network.

Whereas virtual circuit forwarding is used to route data packets in the primary virtual network, traditional packet routing is used in the diversion network. Hence, while data packet headers in the primary network consist of only an RVC number, headers of packets in the diversion network must include source, destination, and sequencing information (Figure 1). When a data packet in the primary network becomes eligible to be diverted, the switch obtains this information from the IMT entry where this required information is recorded. To enable locating the correct IMT entry, the primary buffer retains each packet’s input RVC value until the packet is forwarded on the output RVC. As long as diversions are rare, the slower forwarding of diverted packets at intermediate switches and the overhead of transmitting the larger headers of diverted packets will not significantly impact network performance. Furthermore, if diversions are rare, small diversion buffers are sufficient (e.g., capacity of one packet).

Data packets that become blocked in the primary network can time out and become eligible to enter the diversion network on the next hop. The routing function used for the diversion network determines which output links the blocked packet in the primary network can take to enter the diversion network.

The packet eventually advances, either by taking one hop on its virtual circuit path within the primary network, or by taking one hop into the diversion network on an adjacent switch. To enable switches to identify which virtual network an arriving packet is using, one RVC is designated as special. A data packet arriving on this special RVC uses the diversion BVC, else it uses the primary BVC.

Since diverted data packets may arrive out of order at the destination, each diverted data packet is stamped with a packet sequence number for use by the packet reordering at the destination. After a packet is diverted, the next data packet on the same circuit is also stamped with a sequence number. At each subsequent switch the non-diverted data packet visits along the circuit path, the switch reads the sequence number and locally updates its record to account for the diverted data packets. As long as only a minority of packets require the escape path, most of the advantages of static virtual circuits are maintained with DVCs.

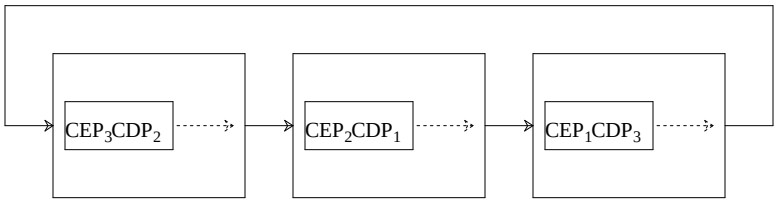


Figure 2: Deadlock involving only control packets in three switches. Packet buffer capacity is two packets. Each CEP_i establishes a unique virtual circuit. The matching CDP_i will disestablish the circuit set up by CEP_i.

4.2. Avoiding Deadlocks Involving Control Packets

Whereas data packets are diverted from their virtual circuit paths to avoid deadlock, control packets (CEPs and CDPs) cannot deviate from virtual circuit paths. Instead, each CEP must traverse the path that is selected for it by the network's fully adaptive routing function, and each CDP must traverse the path used by the virtual circuit it is disestablishing. A deadlock forms when a cycle of packet buffers fills with CEPs and CDPs, as shown by example in Figure 2.

To avoid such deadlocks we develop a mechanism which prevents any buffer from filling with control packets and blocking. The mechanism is derived by analyzing all possible sequences of arrivals of control packets on a single RVC to identify the storage required for each arrival sequence. Deadlocks are prevented by providing switches with packet buffers that are large enough to store the control packets of the worst case sequence without filling and blocking.

We initially examine arrival sequences consisting only of control packets on a single RVC to find the per-RVC storage requirement. Control packets arrive in alternating order (CDP, CEP, CDP, etc.) on

an RVC. If a CDP arrives and the CEP that matches it is present at the switch without an intervening data packet, then the CEP and CDP are deleted from the network (freeing buffer slots). If they were not deleted, the circuit establishment and disestablishment would be wasted operations since the circuit is not used by any data packet. Thus if the first packet in an arrival sequence is a CEP, then the CDP that follows it causes both packets to be deleted. One other case of packet deletion is possible; a CDP is deleted if it arrives at a switch where its circuit is already torn down as a victim. If the first packet of a sequence is a CDP that tears down an existing circuit, then the second packet is a CEP that establishes a new circuit. The third packet is a CDP, which matches the CEP and causes both packets to be deleted. Therefore, in the worst case storage is required for one CDP and one CEP for each RVC. One additional buffer slot is needed that is common to all the RVCs. This slot accommodates the arrival on any RVC of a CDP that will be deleted along with its matching CEP. Hence an input port that supports R RVCs requires storage for R CEPs plus $(R+1)$ CDPs to handle arrival sequences without data packets.

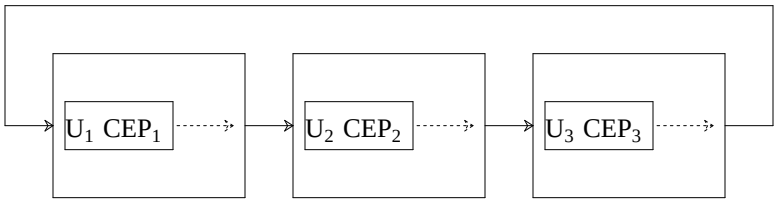


Figure 3: Example deadlock involving three switches. Each U_i is an unmapped data packet that is waiting for CEP_i to establish an RVC mapping. When a CEP transmits to the next switch, the unmapped data packet is converted into a mapped data packet. So long as the unmapped data packets are present at the input ports, subsequent arrivals on the primary Buffering Virtual Channel (BVC) are blocked. Therefore, the CEPs cannot make progress, and a deadlock results.

We next consider arrival sequences that include data packets. A data packet that uses a circuit that is currently allocated an output RVC at the switch is called ‘‘mapped’’. A mapped data packet eventually frees its buffer slot via normal forwarding or via diversion. Therefore, including a mapped data packet in an arrival sequence does not increase the storage required to prevent deadlock. A data packet that uses a circuit that is not currently allocated an output RVC is called ‘‘unmapped’’. It is complex to divert an unmapped data packet because its circuit identification information is not recorded in the IMT entry, as with mapped data packets, but rather in a CEP that is somewhere in the packet buffer [33].

To avoid this complexity, we prohibit diverting unmapped data packets. However, this causes the storage requirement to become unbounded because an infinite repeating pattern of packet arrivals of the following form may arrive to a single RVC: CEP, followed by unmapped data packets, followed by CDP. Matching CEPs and CDPs cannot be deleted because unmapped data packets intervene.

To restore a bound on the storage requirement, we restrict each input to store at most one unmapped data packet at a time (having a larger limit on the number of unmapped packets would also work but would increase the storage requirements). If a data packet arrives and the Input Mapping Table lookup reveals that the RVC is free or allocated to a different circuit, then the primary buffer asserts flow control to block subsequent packet arrivals.

Blocking flow when an unmapped data packet arrives prevents control packets from arriving in addition to data packets. Blocking control packets can cause deadlocks, as shown in Figure 3.

We prevent such deadlocks by introducing a new Buffering Virtual Channel (BVC) for control packets. We call the new BVC the *Control BVC*. The Control BVC is the third and final BVC, in addition to the Primary BVC and the Diversion BVC. The buffering associated with the Control BVC is dedicated to storing control packets. Introducing the Control BVC allows control packets to arrive even when the Primary buffer is full with data packets. The deadlock of Figure 3 does not occur since the CEPs in the figure can all advance using the Control BVC.

Prohibiting unmapped data packets from being diverted and restricting each input port to store at most one unmapped data packet cause the storage requirement for control packets to increase slightly over the case with no data packets. The additional storage is common to all the RVCs at a switch input port and can store one CDP and one CEP. This increases the storage requirement to $(R+1)$ CEPs plus $(R+2)$ CDPs for an input port that supports R RVCs. The two additional control packets arrive after the unmapped data packet and on its RVC. Appendix 1 presents an exhaustive examination of the storage requirements for all possible packet arrival sequences. The worst case sequence of packet arrivals on the RVC that is used by the unmapped data packet is as follows (in order of arrival): mapped data packets, CDP_1 (will release the RVC), CEP_2 (will allocate the RVC for the next data packet), unmapped data packet, CDP_2 (will release the RVC), and CEP_3 (will allocate the RVC for yet another circuit). Since the unmapped data packet cannot be diverted, all four control packets in the sequence are necessary and cannot be deleted. After this sequence of arrivals, CDP_3 may arrive which matches CEP_3. In this case, both CDP_3 and CEP_3 are unnecessary and are deleted.

The control packets created by source hosts (as opposed to switches along the path) trigger critical operations at the destination host. The initial CEP injected by a source to establish a new circuit causes the destination host to allocate memory for delivering packets that use the circuit. The final CDP injected by the source to disestablish the circuit triggers release of resources at the destination host. These control packets may be deleted by a switch if all the data packets that use the circuit are diverted, in which case the final CDP may catch up to the initial CEP at some switch, with no intervening data packet. To

resolve this potential problem, the first data packet that uses a circuit replicates the information in the initial CEP, and the final data packet of a circuit consists only of a header (no data payload). Since data packets are never deleted, they are guaranteed to reach the destination and trigger the actions.

4.3. Algorithm for Handling Control Packets with DVCs

As described in the previous subsections, deadlock-freedom is maintained using data packet diversion through a deadlock-free virtual network. The DVC mechanism imposes two restrictions that simplify the algorithms and their implementations: each input port accommodates at most one unmapped data packet at a time, and unmapped data packets cannot be diverted. Dedicated control packet buffers are used to prevent deadlocks that include control packets. These control packet buffers are large enough to ensure that they can never fill up and cause blocking.

The control packet buffer is organized as follows. For each RVC i, a logical queue is maintained of the control packets in the order they will be transmitted to the next switch. The head of the logical queue for RVC i is comprised of a queue H_i with space for one CEP and one CDP. Whenever an unmapped data packet is present on RVC i, the tail of the logical queue is extended by a queue T_* with space for one CEP and one CDP. Queue T_* is shared by all RVCs at an input port but used by only one RVC at a time.

The algorithm for managing DVCs and control packets is listed in Figure 4. Details of data packet forwarding are not shown because the focus is on circuit manipulation. The algorithm listing shows the key DVC control events that occur at a switch and the actions the switch takes in response. For example, the first event shown is ‘‘Output port Z becomes free’’. The following lines 1 through 15 show how the switch decides which next packet to transmit on output Z and the actions that are taken if the transmitted packet is a CDP or a CEP.

The algorithm listing uses the convention that i and j refer to input RVCs, k is an output RVC, X is an input port, and Z is an output port. Also, N_X is a primary input buffer, D_X is a diversion buffer, and H_i and T_* are control packet buffers.

5. Algorithm Correctness

We consider the algorithm to be *correct* if the following statement is true: *All data packets injected into the network on a DVC are delivered eventually to the DVC destination in the order injected*. Eventual delivery to the DVC destination is guaranteed if packets make progress (the network is deadlock-free) and data packets are never delivered to incorrect destinations. In-order delivery is guaranteed by attaching a sequence number to each packet that may arrive at the destination out of order.

Conditions/Events and Actions (atomic):

1. Output port Z becomes free

```
1       Arbitrate access (assume RVC i of input
        port X maps to RVC k of output port Z),
        but also obey BVC flow control.
        Arbitration priority:

2          A. a packet in some D_X buffer
              waiting for port Z or a mapped
              packet on RVC k from buffer N_X
3          B. a CEP/CDP on RVC k from head of H_i

4       when a CDP from RVC i is transmitted
        on RVC k:
5          delete mapping from i to k
6          if a CEP is waiting at the head of
           some H_j for RVC j {
7            set up new mapping from RVC j
             to RVC k
8          }
9          if required, transfer head of
           T* to H_i

10      when a CEP from input RVC i is
        transmitted on output RVC k:
11         if an unmapped packet on RVC i
           exists {
12            convert unmapped packet to a
              mapped packet
13            unblock flow to primary input
              buffer N_X
14         }
15         if required, transfer head of
           T* to H_i
```

2. CEP arrives on RVC i of input port X

```
16      if H_i is not full {
17         enqueue CEP at tail of H_i
18      } else {
19         place CEP at tail of T*
20      }
```

3. CEP at head of H_i, and RVC i does not map to an output RVC

```
21      record SRC, DST in Input Mapping Table
        entry for RVC i

22      output port Z <-
        route(CEP's destination field)

23      if free RVC k exists on output port Z {
24         set up mapping from i to k
25      } else if the number of RVCs on Z for
        which there are CDPs at the switch is
        less than the number of CEPs that are on
        unmapped RVCs and are routed to output
        port Z {

26         // it is necessary to select and
           // teardown a victim

27         select RVC k (that reverse maps to
           input RVC j != i) such that no CDP
           resides at H_j

28         create CDP, place at tail of H_j

29         (note: the next data packet to
           arrive on RVC j must be considered
           unmapped even though RVC j stays
           mapped to RVC k until the
           transmission of the CDP in H_j)
30      }
```

4. Unmapped packet arrives on RVC i of input port X

```
31      block flow to primary input buffer N_X
32      if H_i has no CEP {
33         create CEP using circuit
           info in RVC i's Input Mapping
           Table (IMT) entry

34         place CEP at tail of H_i
35      }
```

5. CDP arrives on RVC i of input port X

```
36      if CDP is redundant {
37         delete arriving CDP
38      } else if CDP matches a CEP not
        associated with an unmapped packet {
39         delete both the CEP and the
           arriving CDP
40      } else if H_i not full {
41         place arriving CDP at tail of H_i
42      } else {
43         place arriving CDP at tail of T*
44      }
```

Figure 4: Algorithm for Handling Control Packets with DVCs

Section 5.1 below proves that network deadlocks are impossible. Section 5.2 proves that data packets are associated with the same DVC from injection to delivery. Together with the FIFO delivery mechanism described in Section 4, these results show the algorithm satisfies the statement of correctness.

5.1. Proof of Deadlock-Freedom

To prove the network never enters a deadlocked configuration, we examine data packets and control packets separately. Theorem 1 below shows that diverted and mapped data packets cannot be part of a deadlock. Theorems 2 and 3 prove that control packets make progress which in turn guarantees that mappings are eventually provided for unmapped data packets. Together, the theorems show that every

packet, regardless of its type, is guaranteed to make progress.

Theorem 1: Every mapped data packet in a primary buffer N_X and every data packet in a diversion buffer D_X is eventually forwarded.

Proof: This follows from Duato's sufficient condition for deadlock freedom in a virtual cut-through packet-switching network [15]. Deadlock is avoided if there exists a set C of BVCs such that all packets can reach set C in one hop, routing in set C reaches all destinations from all switches, and the set C is free of buffer dependency cycles. The set of diversion buffers D_X meets the definition of set C. A packet in any primary buffer N_X can enter the diversion network by taking one hop and is routed to its destination with a routing policy that is free of dependency cycles (Section 4.1). □

Theorem 2: Control packets do not block in deadlock cycles that involve multiple switches.

Proof: Assume a deadlock involves a control packet and an entity at another switch. By examining all possible chains of dependencies from a control packet to entities at neighboring switches, we show that each chain includes an entity that cannot be in a deadlock cycle, contradicting the assumption.

A control packet may wait directly for one of the following five entities: a buffer slot at the next switch, the output link, a mapped data packet at the same switch, a control packet at the same switch, or an unmapped data packet at the same switch. We examine each entity in turn. First, dedicated control packet buffers at the neighbor cannot be part of a multiple switch deadlock cycle because they have sufficient capacity to avoid filling and blocking. Second, eventual access to the output link is guaranteed for bounded length packets with virtual cut-through forwarding and starvation-free switch crossbar scheduling. Third, mapped data packets cannot participate in deadlock cycles (Theorem 1). Fourth, all control packets have the same set of possible dependencies to other entities, hence a dependence of one control packet on another control packet at the same switch does not introduce the possibility for a deadlock cycle that spans multiple switches. Fifth, a CDP may directly wait for an unmapped data packet at the same switch and on the same RVC. In turn, the unmapped data packet waits for a CEP at the same switch to transmit, which will cause the unmapped data packet to become mapped. The chain of dependencies (from the CDP to the unmapped data packet to the CEP at the same switch) is equivalent to case four above (direct dependence from one control packet to another at the same switch). □

Theorem 3: Control packets do not enter intra-switch deadlocks.

Proof Sketch: (full proof in Appendix 2). Such deadlocks would arise from cycles of dependencies among entities within a single switch. We construct a graph of all the dependencies at a switch that involve control packet buffers. The resulting graph is acyclic, hence intra-switch deadlock is impossible.

The dependency graph is constructed from the control packet handling algorithm (Figure 4) and the enumeration of buffer states in Appendix 1. The graph characterizes a mapped RVC i at some input port X and an unmapped RVC j at some input port Y. RVCs i and j are arbitrary representatives of their classes: mapped and unmapped RVCs, respectively. The set of all dependencies associated with these representative RVCs captures all the dependencies associated with all mapped RVCs and all unmapped RVCs because there are no dependencies between different RVCs in the same class. □

Theorem 4: The network is deadlock-free.

Proof: By Theorem 1, mapped data packets and diverted data packets always make progress. By Theorems 2 and 3, control packets always make progress, hence if a switch creates a CEP to establish a mapping for an unmapped data packet, the CEP is guaranteed to establish the mapping and transmit to the next switch. Once that occurs, the unmapped data packet becomes a mapped data packet. Therefore, all types of packets make progress, and the network is deadlock free. □

5.2. Correct Delivery

To complete the correctness discussion, we show that the deadlock-free network delivers each data packet to the destination of its virtual circuit (instead of some other destination). We also show that the destination can associate each data packet with its virtual circuit. This association is needed to deliver the contents of a data packet to the host application to which the packet's circuit is assigned.

The proof is based on a state transition table derived from an elaboration of the possible states of a switch. From the perspective of some RVC i of input port X of a switch, the switch state is the combination of the states of the buffers H_i, T_*, the primary buffer N_X, and the record in the IMT entry of a mapping to an output RVC. For our purpose and from the perspective of RVC i, the state of buffers at other input ports does not matter: the state of those buffers does not determine the destination to which a data packet on RVC i is forwarded. The state of the diversion buffer D_X is also irrelevant because diverted data packets do not participate in circuit manipulation operations.

Appendix 3 presents in detail the elaboration of the state space as a list of sets of states, where each set of states is classified as either ‘‘reachable’’ or ‘‘unreachable’’. Reachable states arise through error-free operation of the switch, and the switch never enters unreachable states. A set of states is labelled unreachable if it contradicts steps of the control packet handling algorithm (Figure 4) or basic properties of DVCs such as the alternating arrival of CEPs and CDPs on an RVC.

To verify that the manual elaboration of the state space is complete and consistent, we wrote a program that exhaustively generates all combinations of all values each buffer may hold. The program

verifies that each generated state matches at least one set of states in the list and that all matching sets are labelled identically (all reachable or all unreachable).

Using the set of reachable states and the control packet handling algorithm, we next derive the possible state transitions. Each state is represented by the contents of H_i, T_*, and two more fields that capture the relevant state of N_X and the IMT entry for RVC i. H_i can hold one CEP and one CDP on RVC i. T_* can hold one CEP and one CDP on any RVC. The state of N_X and the IMT entry for RVC i are too numerous to list exhaustively, but the relevant states are determined by the mapping status of RVC i, and whether an unmapped packet is present at input port X. That information is represented by using two symbols: map_i and U_X. Table 1 shows the state transitions for all reachable states. The state notation used in Table 1 is described in Figure 5 which lists the possible states of a slot for H_i and T_* and defines the symbols used in columns U_X and map_i. In the state transition table, the actions that trigger a transition are as follows: k (RVC i acquires mapping to output RVC k), D (CDP arrives on RVC i), E (CEP arrives on RVC i), U (unmapped packet arrives on RVC i), T (transmission of packet at head of H_i), D' (CDP arrives on RVC $i' \neq i$), E' (CEP arrives on RVC $i' \neq i$), $U_{i'}$ (arrival of unmapped packet on some RVC $i' \neq i$), $TE_{i'}$ (transmission of CEP at head of $H_{i'}$), and $TD_{i'}$ (transmission of CDP at head of $H_{i'}$). Each entry of the table is derived by tracing through the control packet handling algorithm to determine what the next state would be given each possible action. Note that each empty entry in Table 1 means the column's action cannot occur in the present state for the row. For example, for rows 0–9, there is no packet in buffer H_i. Therefore, action T, transmission of the packet at the head of H_i, cannot happen.

In some cases, two next states are indicated for some transitions, indicating that either of the two next states may become the new present state. This is because the next state depends on the present state of $H_{i'}$, which is not displayed in Table 1 because it is not associated with RVC i.

Table 1 is used below to show that a packet P, which is injected on virtual circuit V, reaches the destination of circuit V, and the host interface at the destination associates packet P with circuit V.

Theorem 5: Whenever packet P is mapped, its input RVC identifies virtual circuit V.

Proof: The proof is by induction on the distance from the virtual circuit's source.

Basis: The theorem is true at the injecting host interface because the host transmits only one CEP for the circuit until it is disestablished.

Induction Step: Assume the theorem is true for P at switch or host S_n (n hops from the source). We now show the theorem is also true at the host or switch that is $n+1$ hops from the source (at host/switch S_{n+1}). Suppose packet P is about to transmit to host/switch S_{n+1}'s primary BVC on RVC k. Since P is about to transmit, P must be a mapped data packet at host/switch S_n. Therefore, the previous control

Present State						Action/Next State									
ID	H_i		T_*		$U_X map_i$	k	D	E	U	T	D′	E′	$U_{i'}$	$TE_{i'}$	$TD_{i'}$
0	–	–	–	–	– F		0	10	12		0	0	2	0	0
1	–	–	–	–	– T		26				1	1	3	1	1
2	–	–	–	–	i′ F		2	14			2,6	4		0	2
3	–	–	–	–	i′ T		27				3,7	5		1	3
4	–	–	CEP′	–	i′ F		4	16			2			0	
5	–	–	CEP′	–	i′ T		28				3			1	
6	–	–	CDP′	–	i′ F		6	18			6	8			2
7	–	–	CDP′	–	i′ T		29				7	9			3
8	–	–	CDP′	CEP′	i′ F		8	20			6				4
9	–	–	CDP′	CEP′	i′ T		30				7				5
10	CEP	–	–	–	– F	11	0		12		10	10	14	10	10
11	CEP	–	–	–	– T		0		13	1	11	11	15	11	11
12	CEP	–	–	–	i F	13	22				12	12		12	12
13	CEP	–	–	–	i T		23			1	13	13		13	13
14	CEP	–	–	–	i′ F	15	2				14	16		10	14
15	CEP	–	–	–	i′ T		3			3	15	17		11	15
16	CEP	–	CEP′	–	i′ F	17	4				14			10	
17	CEP	–	CEP′	–	i′ T		5			5	15			11	
18	CEP	–	CDP′	–	i′ F	19	6				18	20			14
19	CEP	–	CDP′	–	i′ T		7			7	19	21			15
20	CEP	–	CDP′	CEP′	i′ F	21	8				18				16
21	CEP	–	CDP′	CEP′	i′ T		9			9	19				17
22	CEP	CDP	–	–	i F	23	22	24			22	22		22	22
23	CEP	CDP	–	–	i T		23	25		26	23	23		23	23
24	CEP	CDP	CEP	–	i F	25	22				24	22		24	24
25	CEP	CDP	CEP	–	i T		23			31	25	23		25	25
26	CDP	–	–	–	– T		26	31	32	0	26	26	27	26	26
27	CDP	–	–	–	i′ T		27	33		2	27,29	28		26	27
28	CDP	–	CEP′	–	i′ T		28	34		4	27			26	
29	CDP	–	CDP′	–	i′ T		29	35		6	29	30			27
30	CDP	–	CDP′	CEP′	i′ T		30	36		8	29				28
31	CDP	CEP	–	–	– T		26		32	10	31	31	33	31	31
32	CDP	CEP	–	–	i T		37			12	32	32		32	32
33	CDP	CEP	–	–	i′ T		27			14	33,35	34		31	33
34	CDP	CEP	CEP′	–	i′ T		28			16	33			31	
35	CDP	CEP	CDP′	–	i′ T		29			18	35	36			33
36	CDP	CEP	CDP′	CEP′	i′ T		30			20	35				34
37	CDP	CEP	CDP	–	i T		37	38		22	37	37		37	37
38	CDP	CEP	CDP	CEP	i T		37			24	38	38		38	38

Table 1: State Transition Table

packet on RVC k must have been a CEP that identified V.

If S_{n+1} is a host instead of a switch, then the host will correctly associate packet P with virtual circuit V. However, if S_{n+1} is a switch, then there may be a danger that the CEP will be deleted before it reaches the head of H_k. Only if the CEP reaches the head of H_k will the switch S_{n+1} record the circuit identification information for V in the IMT entry for RVC k (algorithm line 21). We will now prove that the danger is unwarranted. The CEP will successfully reach the head of H_k at switch S_{n+1}.

H_i and T_*		T_* only		U_X		map_i	
-	empty	CDP′	CDP on RVC i′ ≠ i	-	no unmapped packet	T	RVC i is mapped to an output RVC k, which also means that N_X may store mapped data packets from RVC i.
CDP	CDP on RVC i	CEP′	CEP on RVC i′ ≠ i	i	one unmapped packet on RVC i is in N_X (note: N_X may hold at most one unmapped packet)	F	RVC i is unmapped, which means any mapped packets in N_X did not arrive on RVC i.
				i′	the unmapped packet is on some RVC i′ ≠ i		

Figure 5: State Notation for Table 1

Switch S_{n+1} may delete a CEP if a CDP arrives from S_n or is generated locally. In our scenario, a CDP may not arrive at S_{n+1} from S_n, since packet P is about to transmit from S_n. It is impossible for a CDP to transmit ahead of a mapped data packet such as P (see arbitration priority rules in lines 1–3 of Figure 4). Therefore, only a CDP that is generated locally at S_{n+1} could trigger the deletion of the CEP.

From Table 1, we can identify all the state transitions in which a CEP is deleted upon introduction of a CDP. That is, we can identify all present states in the table such that the column D transition takes the switch to a next state that has one less CEP than the present state. For some of those states (10, 11, and 14 through 21), the CEP that identifies virtual circuit V is already at the head of the logical queue (i.e. H_k for switch S_{n+1}), hence the IMT entry identifies V. Even if that CEP is deleted, the information is available when packet P arrives at switch S_{n+1}. The remaining qualifying states are 24, 25, 31, and 33 through 36. In those states, a CDP is present in the buffer. According to line 27 of Figure 4, the presence of the CDP prevents the introduction of a locally-generated CDP. Therefore we conclude that it is not possible for a locally-generated CDP to cancel the CEP before the IMT is written to identify circuit V. □

Theorem 6: If packet P is a diverted packet, then P's header identifies circuit V.

Proof: By theorem 5, at the time P is diverted, the IMT entry identifies circuit V. The procedure for diversion attaches that information to packet P's header. After diversion, the packet header is not altered until P is delivered. Therefore, packet P is always associated with circuit V. □

6. Implementation Issues

Efficient implementation of the DVC algorithms presented earlier requires specialized hardware support. To demonstrate the feasibility of using DVCs, this section presents key components of a possible implementation.

6.1. Switch Organization

We assume an $n \times n$ switch is a single-chip device. The switch includes a processor that performs the relatively complex yet infrequent circuit manipulation operations of the DVC algorithm. This processor also executes higher-level routing protocols that identify low latency paths [29]. The switch has packet buffers at each input port. Each input port X has primary buffer N_X, diversion buffer D_X, and control packet buffer T_*^X. The input buffers are connected to output ports through an $n \times n$ crossbar.

Each input port also has an Input Mapping Table (IMT) that records RVC mappings (Figure 6).

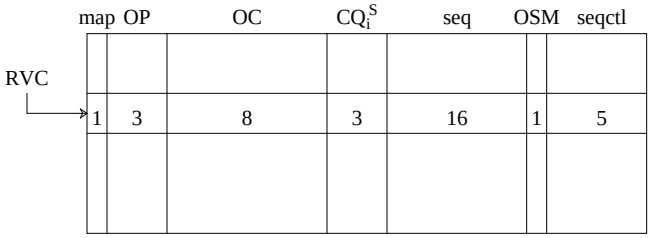


Figure 6: Input Mapping Table (one at each input port). Field widths in bits are shown.

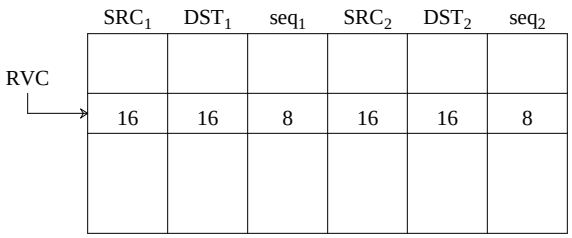


Figure 7: Circuit Information Table (one for each input port). Holds information about mapped and torn DVCs, and CEP information from H_i.

When a data packet arrives to an input port, its RVC identifier is used to access an IMT entry. The entry's ''map'' field indicates whether the input RVC is mapped to an output RVC (i.e., whether a circuit is established). If so, the ''OP'' field specifies the output port to forward the arriving packet, and the ''OC'' field specifies the output RVC value which replaces the value in the packet's header. The input RVC value is saved in N_X with the data packet and is discarded when the packet is dequeued upon transmission. The remaining fields of the IMT are used to keep track of control packet buffering (Section 6.2) and FIFO sequencing (Section 6.3).

Each output port has an Output Mapping Table (OMT). The OMT is indexed by output RVC value and indicates the status of the output RVC. Each OMT entry has 3 bits. The ''map'' bit is set when the output RVC becomes mapped to an input RVC. It is cleared when a CDP transmits to the next switch using that RVC, releasing the output RVC for use by a new circuit. The ''victim'' bit is set when the circuit mapped to the output RVC is chosen as a victim for teardown. It too is cleared when the CDP transmits to the next switch. At that point, the RVC can be mapped to an input RVC that has a CEP waiting to establish a circuit. The ''active'' bit indicates whether the RVC has been used by any recently transmitted packet. The switch uses the ''active'' bit to choose a victim. In particular, the ''clock'' algorithm [9], which approximates LRU, can be used to select a victim.

The IMT and OMT are accessed for each packet and must therefore be stored in high-speed (SRAM) memory. Other information is accessed only when packets are diverted or when circuits are manipulated and can therefore be stored in lower speed dense (DRAM) memory. This latter information consists of tables for routing CEPs, tables that store DVC identification and routing information (these tables are updated upon DVC establishment and read when DVCs are rerouted), and tables used to implement the H_i control buffers.

6.2. Control Buffer Implementation

At each input port, the algorithm in Section 4 requires the ability to store up to four control packets (CEPs, CDPs) for one RVC (in H_i and T_*) and up to two control packets for the rest of the RVCs (in H_j). To minimize the cost of this storage, the storage for each RVC is split into two components: CQ_i^S — a frequently-accessed component stored as part of the IMT in dedicated SRAM at each port, and CQ_i^{CEP} — an infrequently-accessed component stored in the DRAM available on the switch. The CQ_i^S component consists of a compact representation of the state of the logical queue of control packets for input RVC i. From Table 1, the logical queue of control packets for an RVC i can at any time be in one of eight states: empty, CEP, CDP, CEP CDP, CDP CEP, CEP CDP CEP, CDP CEP CDP, CDP CEP CDP CEP. The CQ_i^S field of the IMT specifies the current state as a 3-bit value. Since CDPs carry no information other than the RVC value, the 3-bit state encoding represents all the CDPs in the queue.

The infrequently-accessed CQ_i^{CEP} component is maintained in DRAM in the Circuit Information Table (CIT) (Figure 7). For each input RVC, there is an entry in the table with space to record the information of two CEPs (DVC source, destination, and sequencing information). The first set of fields (SRC_1, DST_1, and seq_1) reflects the current status of the DVC and is set from the contents of the CEP that established the current DVC. The second set of fields (SRC_2, DST_2, and seq_2), if valid, corresponds to the DVC that will replace the current DVC. This second set of fields is the storage of CQ_i^{CEP} — the CEP that is part of H_i (Section 4.3). Storage for information from two CEPs is needed since the switch may contain data packets on the previous DVC when the CEP for a new DVC arrives. If one or more of these data packets needs to be diverted, the information for the previous DVC will still be needed.

Finally, a dedicated buffer for each input port provides storage for the one CEP that is part of T_*.

Performance can be optimized by adding a small, fast write buffer for arriving CEPs. The switch can access the buffer as a cache to forward CEPs quickly without incurring the DRAM access time for writing to the CIT. A write buffer only benefits CEPs that arrive to empty H_i queues. This should be the common case since control packets present only a light load with reasonable traffic patterns.

The switch must transmit the control packets and data packets in a logical queue in FIFO order, even though the packets are stored physically at the switch in various memory buffers (i.e., the primary buffer for data packets, and the control buffers described above for control packets). Each buffer preserves the partial order of the packets it stores from the logical queue. The partial order of mapped data packets of a logical queue is preserved because the primary buffer maintains in FIFO order its data packets that are destined to a single output port. The partial order of control packets in the logical queue is recorded and maintained by using the IMT CQ_i^S field. Although these buffers preserve the correct partial order of transmission, a mechanism is needed to control the total interleaved order of transmissions from these buffers.

The total order of a logical queue is preserved by transmitting all of its mapped data packets before any of its control packets. The mapped data packets must precede the control packets because a CDP that precedes a mapped data packet would delete the current RVC mapping, and a preceding CEP would establish a new mapping before the data packet is transmitted. Transmission of mapped data packets ahead of control packets is enforced through the use of the ‘‘seqctl’’ field in the IMT (Figure 6). The seqctl field has initial value zero and records the number of mapped data packets that are present in the logical queue for an RVC. Control packets in the logical queue are allowed to transmit only when the value in ‘‘seqctl’’ is zero. Mapped data packets can transmit anytime. The ‘‘seqctl’’ value is incremented when a mapped data packet arrives at a switch on the RVC or when an unmapped data packet on the RVC becomes mapped as a result of transmitting a CEP. The ‘‘seqctl’’ field is decremented when a data packet that previously arrived on the RVC is either transmitted to the next switch along its circuit or is diverted.

6.3. Sequencing and Diversion

As explained in Section 4, to support FIFO delivery, one sequence number per input RVC is maintained in the IMT. The IMT entry ‘‘seq’’ (sequence number) field (Figure 6) is incremented whenever a packet is transmitted whose header has no sequence number field (Figure 1). The index used to access the IMT entry is the value of the input RVC on which the packet arrived at the switch. As discussed in Section 6.1, this value is saved in the main data packet buffer N_x. Whenever a packet with a sequence number is transmitted from the primary buffer, that sequence number replaces the value in the IMT ‘‘seq’’ field.

After a data packet is diverted, the next data packet that is transmitted normally on the same circuit is stamped with a sequence number (Section 4.1). The IMT entry’s single-bit ‘‘OSM’’ (Out of Sequence

Mode) field is used as a flag to determine whether a data packet that is transmitted normally should be stamped. The flag is set when a data packet is diverted. Before a data packet begins transmission to the next switch, the OSM flag for the packet's input RVC is read. If the flag is set, then it is cleared and the sequence number is added to the data packet header on-the-fly as it is transmitted to the next switch. With virtual cut-through, a packet is transmitted only if the next switch has sufficient buffer space for a maximum size packet. Hence, lengthening the packet, following diversion, as it is transmitted, will not cause the buffer at the next switch to overflow.

Packet diversion requires forwarding a timed-out packet to an output other than the one associated with the logical queue storing the packet. Hence, when a packet times out, its request from the crossbar arbiter is modified to include access to the switch output that was its original destination as well as access to the switch output(s) on the diversion path(s). If access to the diversion network is granted first, the packet's RVC field is changed to indicate use of the diversion BVC, and DVC information from the IMT and CIT is added to the header.

7. Performance Evaluation

One key advantage of DVCs is the potential for reducing overall network contention by establishing circuits or rerouting existing circuits onto low latency paths. In most systems, traffic patterns change dynamically, and circuits require time to adjust their paths to compensate. For a first-order evaluation of the performance *potential* for DVCs with adaptive routing, we consider simpler limit cases with stable traffic patterns and circuit placements. The actual performance of a system with DVCs will depend on the routing and rerouting algorithms.

We consider Uniform, Transpose and Bit-Reversal traffic patterns. In all cases, we precompute the paths used by DVCs in order to simulate ideal conditions where circuits have settled into a steady-state, low contention configuration. We compare the performance of the resulting configuration against a packet switched network using Dimension Order Routing (DOR), which is known to perform well for Uniform and poorly for Transpose and Bit-Reversal patterns [21]. In our experiments, the routes for DVCs are precomputed using a simple heuristic Bellman-Ford minimum cost path algorithm. The cost of each link is set to an estimate of the delay experienced by packets waiting to use the link. The delay is estimated by modeling the link as a Geo(N)/D/1 queueing system fed by packet arrivals from all DVCs whose routes include the link. Details of this route precomputation procedure are described elsewhere [33].

For our simulation experiments, packet size is 32 phits, and switches have DAMQ primary input

buffers. For DVC simulations, there is also a diversion buffer of capacity 32 phits. The results for DOR and routed DVCs are shown for equal total input buffer capacity, which for DVCs is the sum of the primary input buffer capacity plus 32 phits for the diversion buffer. The interval between packet creations has a geometric distribution. At each switch, the crossbar arbitration policy gives priority to the packet that has resided longest at the switch. The performance metrics are the average latency, the average node throughput and the normalized throughput. Packet latency is defined as the number of cycles from when the first byte of a packet is generated at a sender to when it leaves the network. Normalized throughput expresses throughput as a fraction of the network bisection bandwidth.

With DVCs, a packet is diverted if it is blocked at the head of the queue at a switch and the timeout expires. If the timeout interval is too short, packets will be diverted unnecessarily. If the timeout interval is too long, it will take longer to resolve deadlocks. To evaluate the impact of this effect on performance, our evaluation includes simulations for a variety of timeout values.

7.1. Transpose Traffic Pattern

The transpose traffic pattern sends all packets across the diagonal of the mesh; packets from the source at row i column j are sent to the destination at row j column i. Nodes along the diagonal only forward packets; they do not produce or consume any packets of their own. For this traffic pattern, Figure 8 shows the latency versus throughput for an 8×8 mesh, with input buffer capacity 64, 96, and 288 phits.

These results show that for all levels of buffer capacity, the maximum throughput achieved with DVCs is about twice that achieved with DOR. With DOR, only the horizontal incoming links of the switches along the diagonal are utilized. With routed DVCs, both the vertical and horizontal incoming links of the diagonal are utilized with approximately equal traffic loads assigned to each link. The results also show that the impact of increasing the buffer capacity is higher latency, not higher throughput. Throughput does not increase for either DOR or routed DVCs because it is limited by the bandwidth of saturated links along the mesh diagonal. Finally, the results show that using DVCs with long timeouts for packet diversion results in higher maximum throughput than using short timeouts. Short timeouts increase the frequency of packet diversions, which for the transpose traffic pattern occur before the packet crosses the diagonal, the congested point in the network. Since diverted packets use DOR, they can only use the horizontal incoming links of switches on the diagonal. Hence packet diversions shift traffic from the vertical to the horizontal incoming links at the diagonal. These traffic imbalances reduce performance, thus in this case longer timeouts which minimize packet diversions are better.

Figure 9 shows the fraction of traffic diverted versus normalized throughput for the DVC network with input buffer capacity of 96 phits. For low and medium network loads, as the load increases, the fraction of diverted packets increases. However, past a certain point, the fraction of diverted packets decreases as the load increases. The reason for this is that at these high loads the increase in network throughput is mostly for the subset of circuits using low-contention routes. Other circuits and their diversion paths are saturated and their throughput does not increase as the applied load increases. For the low-contention circuits no diversion occurs so more packets get through the network without a corresponding increase in the number of diverted packets.

The performance of a distributed application is often limited by the performance of its slowest member rather than by the aggregate throughput available to the application. For example, an application whose nodes communicate via the transpose traffic pattern may occasionally need to synchronize to ensure that all sending nodes are in a known state. If some flow is particularly slow, progress of nodes associated with this flow will be impeded and the progress of all other nodes will be throttled upon synchronization. Hence, it is useful to evaluate the fairness of the system by comparing the throughputs achieved by individual senders.

Figure 10 shows the raw (not normalized) throughput achieved by each source node in the 8×8 mesh using the transpose traffic pattern. The throughputs from each sender are displayed, sorted to be monotonic (the first eight sources are along the diagonal and do not generate packets). Throughputs for routed DVCs and DOR are displayed as separate curves. Since fairness in a network decreases as the load increases, comparison of the fairness of the two policies should be done at the same average node throughput. In Figure 10, average node throughput for DOR is at its maximum, 0.233, and average node throughput for routed DVCs is 0.242. Since unfairness increases with average node throughput, the result in Figure 10 is biased slightly in favor of DOR, yet the routed DVCs achieve far greater uniformity of sender throughput than does DOR. As we increase applied load further, the routed DVCs policy also becomes unfair, but only at much higher levels of average node throughput than can be achieved with DOR. This is demonstrated in Figure 11, which shows throughput fairness at saturation, in which each source constantly tries to inject packets.

7.2. Bit-Reversal Traffic Pattern

The bit-reversal traffic pattern sends messages from each source $x_{n-1}x_{n-2}\cdots x_0 y_{n-1}y_{n-2}\cdots y_0$ to destination $y_0 y_1 \cdots y_{n-1}x_0 x_1 \cdots x_{n-1}$. Figure 12 shows latency versus throughput on an 8×8 mesh, for total input buffer capacity 64, 96 and 288 phits. The reported throughput is normalized to the bisection

bandwidth, the upper bound on throughput for the bit-reversal traffic pattern.

The results for bit-reversal show that, as with transpose traffic, routed DVCs significantly outperforms DOR and there is no advantage to increasing the buffer size. Unlike with transpose traffic the latency-throughput results with bit-reversal traffic are nearly independent of the diversion timeout value. With bit-reversal traffic, diverted packets do not necessarily follow poor paths that increase congestion.

Figure 13 shows the fraction of traffic diverted versus throughput with total input buffer capacity 96 phits. These results show, as with transpose, that increasing timeout values greatly reduce the fraction of traffic diverted. Since diverted packets are handled less efficiently than packets on DVCs, these results and the transpose traffic results indicate that long timeout values should be used.

7.3. Uniform Traffic Pattern

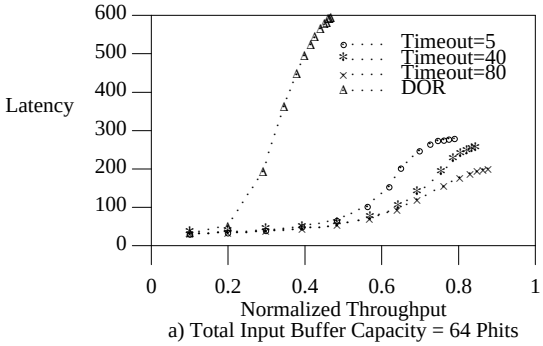
For uniform traffic in a square mesh, DOR distributes load evenly across the links that comprise the network bisection and thus should perform well. In contrast, some adaptive routing schemes tend to steer more traffic toward the center of the network, causing congestion [27].

Figure 14 shows latency versus throughput for uniform traffic with total input buffer capacity 64 and 288 phits. The results show that the performance of routed DVCs is close to that of DOR. Unlike transpose and bit-reversal traffic, with uniform traffic the use of larger buffers improves performance. Increasing the buffer capacity increases the number of flows that can be represented at any instant by the packets that are present at a switch. Larger buffers are therefore more helpful for uniform traffic with $O(N^2)$ flows than for the previous traffic patterns which have only $O(N)$ flows (one from each source node). Performance also improves with the use of smaller timeout values which effectively increase the useful buffer capacity by enabling more packets to take advantage of the 32 phit diversion buffers. With large buffers (288 phits), routed DVCs and DOR have nearly identical performance.

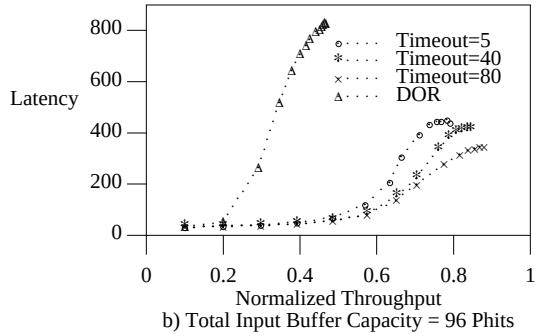
For routed DVCs with short timeouts, as the applied load increases beyond saturation the network throughput decreases slightly. This may occur because congestion in the primary virtual network causes a larger number of packets to enter the diversion virtual network which has limited buffering and therefore limited throughput.

7.4. The Impact of Network Size

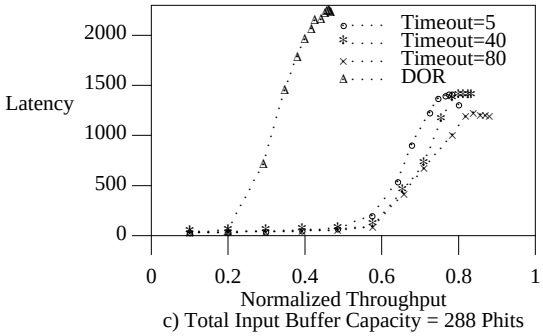
For a larger mesh network of size 16×16 and the same traffic patterns as used previously, Figures 15, 16 and 17 show latency versus throughput, and Figures 18, 19 and 20 show the fraction of traffic diverted versus normalized throughput. For non-uniform traffic, the results show that routed DVCs significantly outperform DOR, but the performance difference is smaller than on the 8×8 mesh. With the larger network, packets travel longer distances, and there are more opportunities for delays and deadlocks. Hence, the fraction of packets diverted tends to be larger than on the 8×8 mesh, resulting in more of the traffic facing congestion as with DOR.



a) Total Input Buffer Capacity = 64 Phits



b) Total Input Buffer Capacity = 96 Phits



c) Total Input Buffer Capacity = 288 Phits

Figure 8: Transpose traffic: Latency vs. Normalized Throughput.

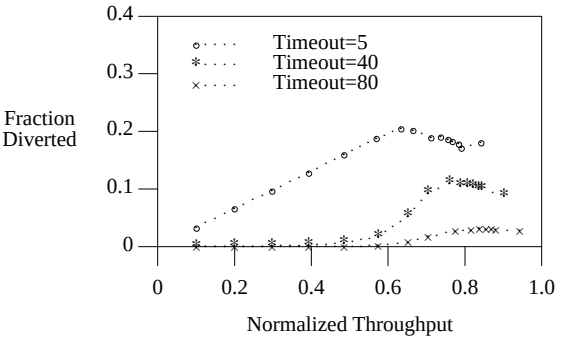


Figure 9: Transpose traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total input buffer capacity = 96 phits.

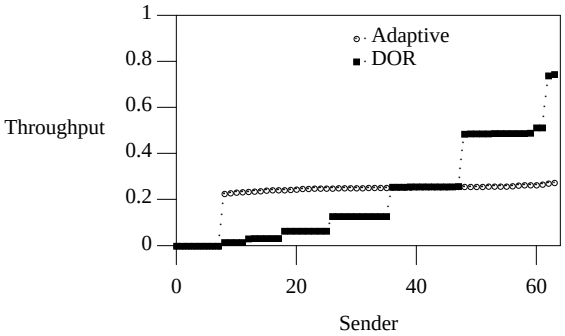


Figure 10: Transpose traffic: Throughput fairness. Throughput vs. sender, sorted. Total input buffer capacity = 64 phits. Average node throughput = 0.241 for DOR (48.2% normalized), 0.242 for routed DVCs (48.4% normalized).

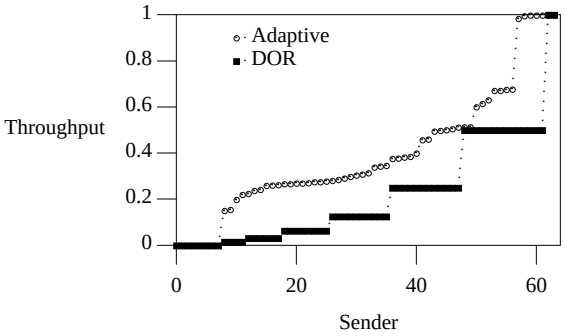
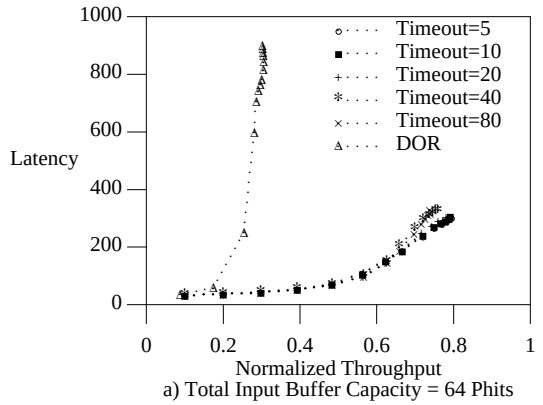
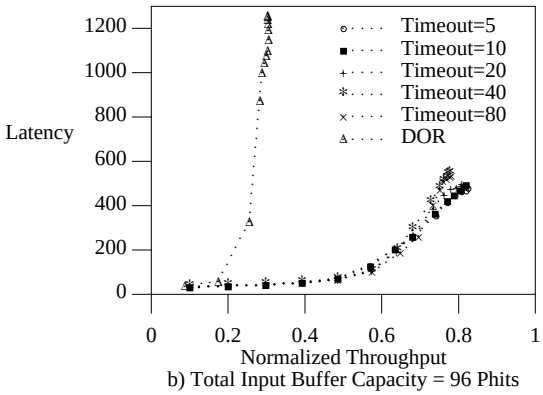


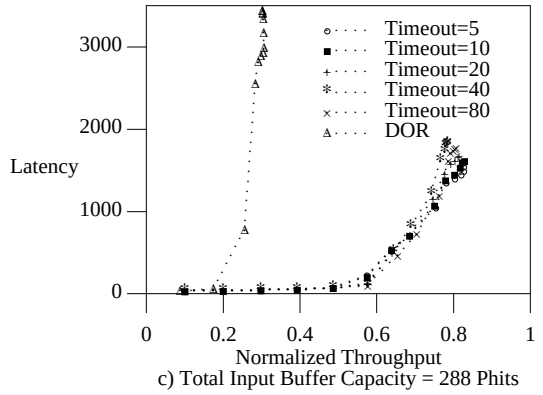
Figure 11: Transpose traffic: Throughput fairness at saturation. Throughput vs. Sender, sorted. Total input buffer capacity = 288 phits. Average node throughput = 0.24 for DOR (48% normalized), 0.47 for routed DVCs (94% normalized)



a) Total Input Buffer Capacity = 64 Phits



b) Total Input Buffer Capacity = 96 Phits



c) Total Input Buffer Capacity = 288 Phits

Figure 12: Bit-Reversal traffic: Latency vs. Normalized Throughput.

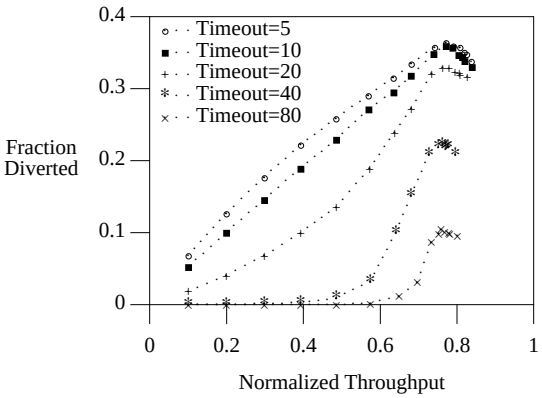
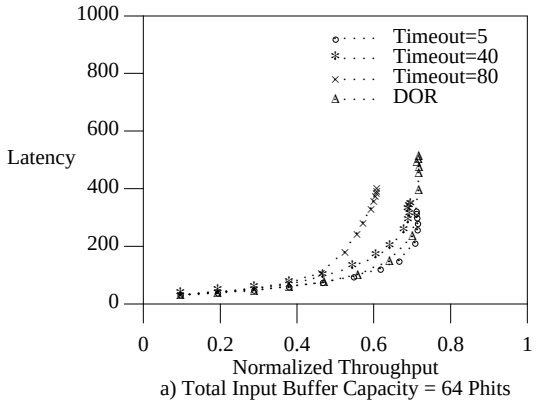
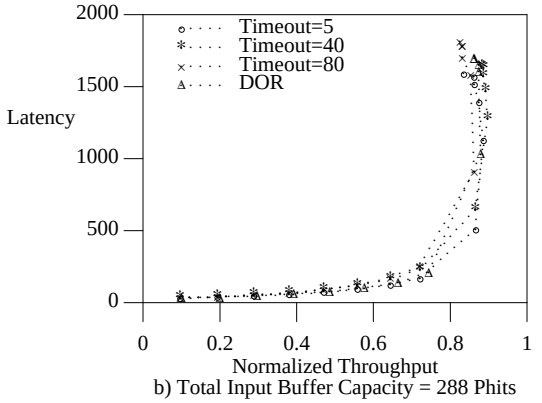


Figure 13: Bit-Reversal traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total input buffer capacity = 96 phits.



a) Total Input Buffer Capacity = 64 Phits



b) Total Input Buffer Capacity = 288 Phits

Figure 14: Uniform traffic: Latency vs. Normalized Throughput.

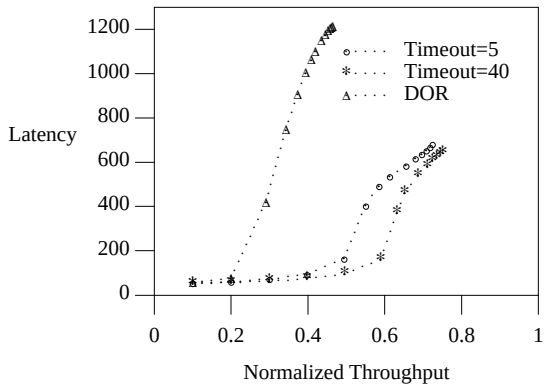


Figure 15: 16×16 Transpose traffic: Latency vs. Normalized Throughput. Total input buffer capacity = 64 phits.

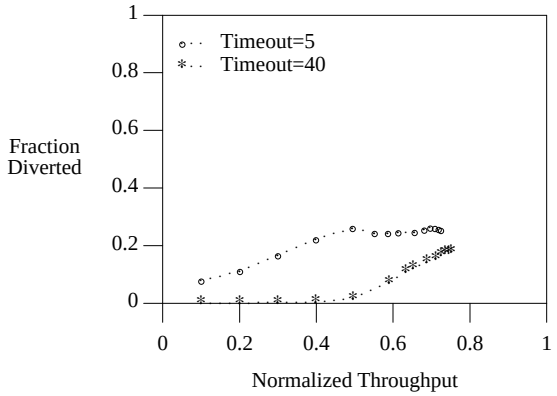


Figure 18: 16×16 Transpose traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total input buffer capacity = 64 phits.

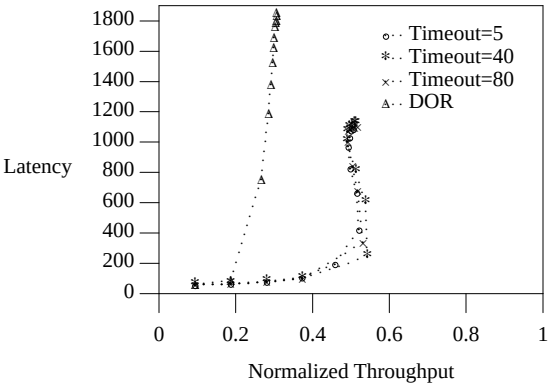


Figure 16: 16×16 Bit-Reversal traffic: Latency vs. Normalized Throughput. Total input buffer capacity = 64 phits.

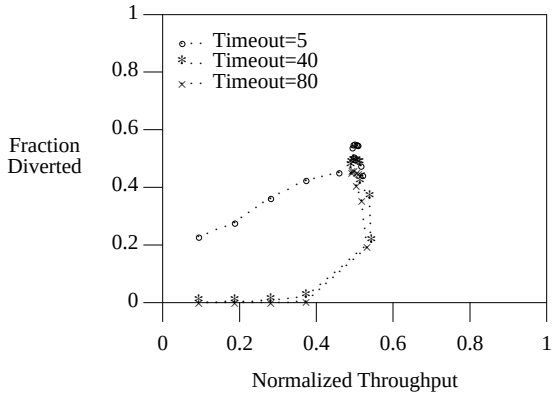


Figure 19: 16×16 Bit-Reversal traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total buffer capacity = 64 phits.

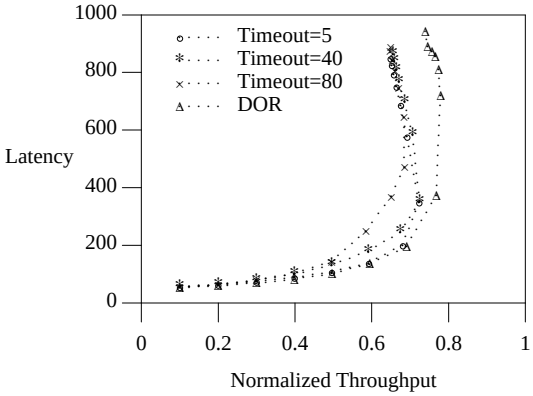


Figure 17: 16×16 Uniform traffic: Latency vs. Normalized Throughput. Total input buffer capacity = 64 phits.

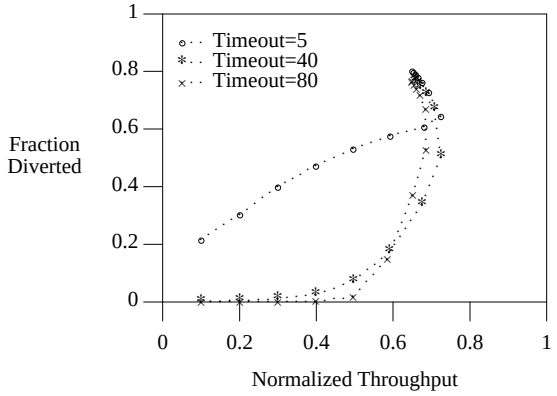


Figure 20: 16×16 Uniform traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total input buffer capacity = 64 phits.

8. Related Work

In this section we describe related work on connection-based routing for multicomputers or high-end cluster networks, and existing techniques for virtual circuits to support adaptive routing and circuit rerouting. We compare these approaches to our proposed Dynamic Virtual Circuits.

An approach to connection-based routing that combines static virtual circuits (for traffic exhibiting locality) and conventional wormhole routing (for other traffic) was proposed by Dao, Yalamanchili, and Duato [12]. In this proposal, a source node tries to establish a static virtual circuit by injecting into the network a circuit establishment probe. The probe is adaptively routed toward the destination; along the way the probe may backtrack as it searches for a path with free resources on each link. If the probe backtracks to the source node, the source either re-injects the probe for another try or gives up on establishing a circuit, in which case it uses conventional wormhole routing for traffic it sends to the destination. Existing circuits are not torn down to free resources for new circuits. Nor are circuits rerouted to adjust to congestion or faults.

A different approach that relaxes the static nature of virtual circuits was proposed by Hsu and Banerjee [22]. They proposed adding logic to support on each switch a small number of virtual circuits, called *cached circuits*. A cached circuit relaxes the static restrictions of virtual circuits; a cached circuit may be torn down in order to free up resources at an intermediate switch. The resources may be needed, for example, to establish a new cached circuit. The intermediate switch selects an existing cached circuit to be a victim, and it sends a request to the source node of the victim circuit to tear it down. The packets in transit from the victim's source must progress past the intermediate switch before the resources held by the victim circuit can be released. Therefore, new circuit establishment may be blocked for an extended period while packets on the victim circuit are being flushed out. In addition, the packets being flushed out are forced to take the existing path of the victim circuit, which may no longer be a desirable path because of the possible prior onset of congestion or faults.

Virtual circuits are used in Asynchronous Transfer Mode (ATM) [13]. ATM supports two types of connections: virtual circuits (VCs) and virtual paths (VPs). A VC is composed of a sequence of VPs from source to destination. Each VP can support 2^{16} VCs. A VC can be rerouted to improve quality of service via a round trip exchange of control messages between the source and destination [8]. A VP that is used by many VCs can be rerouted when a link on its route fails. Rerouting a VP is transparent to the VCs that use it. A VP can be rerouted onto a backup route that is either pre-defined or is selected after a failure is detected [24, 20]. VP rerouting is accomplished through a round trip exchange of control messages on the backup path between the VP endpoints [24]. Alternatives to end-to-end VP rerouting include rerouting

only the portion of the VP between two switches that are adjacent to the failure, and rerouting the portion of a VP from a switch that is upstream from the failure and the VP destination. These strategies differ in the time required to reroute a VP and in the spare bandwidth that is needed to guarantee that all VPs can be rerouted and meet their bandwidth requirements after any single failure [26, 3].

Our Dynamic Virtual Circuits (DVCs) [30] proposal differs from the above proposals by allowing virtual channel resources to be quickly reallocated through local operations at a switch, avoiding the long delays of schemes that require interactions with faraway nodes before releasing local resources. Resource reallocation avoids blocking circuit establishment, and it enables adaptive circuit rerouting.

9. Conclusion

In this paper, we presented the algorithms and hardware/firmware architecture for Dynamic Virtual Circuits (DVCs), a novel technique that enables multicomputer and cluster interconnection networks to combine the benefits of connection-based routing and adaptive routing. In particular, DVCs reduce link bandwidth overheads and packet processing delays at switches compared to pure packet switching networks. A Dynamic Virtual Circuit can be established on any path from source to destination without the possibility of deadlocks involving packet buffer resources or virtual circuit manipulation operations. Unlike prior approaches, DVCs can be rerouted dynamically through the use of fast operations at any switch along a virtual circuit's path without requiring costly delays for coordination with remote nodes. It is practical to implement the DVC mechanism, which has only modest hardware requirements and applies to networks with arbitrary topologies. Emerging interconnect standards such as InfiniBand could be extended to support DVCs, which fit well with the semantics of InfiniBand end-to-end queue pairs connections and could be used to improve transmission efficiency for InfiniBand network fabrics.

To guarantee deadlock-freedom in DVC networks, our solution decouples data packet routing and circuit manipulation operations. To enable data packets to use unconstrained virtual circuit paths, we leverage the existing approach from packet switching networks of allowing data packets that encounter buffer dependency cycles to transition to a dependency cycle-free virtual network, the *diversion network*. To avoid deadlocks involving circuit manipulation operations, we present a new approach, based on an analysis of control packet arrival sequences, which guarantees that control packets cannot experience blocking across switches. We use an elaboration of the state space of a switch to develop correctness arguments showing that the DVC algorithms ensure the network is deadlock-free and that data packets are delivered to correct destinations. Our performance evaluation results show that with virtual circuits, global routing optimization is possible and provides performance superior to fixed routing. Furthermore,

the results show that the use of deadlock-free escape paths is sufficiently infrequent to preserve the bandwidth efficiencies of the DVC mechanism.

For future investigation, an interesting and important question is how DVCs behave with shifting traffic patterns. In particular, there are many alternatives for choosing when and how to reroute existing circuits. Other future investigations could focus on extending DVCs to provide advanced functionalities including improved fault tolerance and multicast capabilities.

REFERENCES

1. *InfiniBand Architecture Specification, Volume 1, Release 1.0*, InfiniBand Trade Organization www.infinibandta.org (October 2000).
2. *TOP500 Supercomputer Sites*, www.top500.org (November 2004).
3. J. Anderson, B. T. Doshi, S. Dravida, and P. Harshavardhana, ‘‘Fast Restoration of ATM Networks,’’ *IEEE Journal on Selected Areas in Communications* **12**(1), pp. 128-138 (January 1994).
4. K. V. Anjan and T. M. Pinkston, ‘‘An efficient, fully adaptive deadlock recovery scheme: DISHA,’’ *Proceedings 22nd Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, pp. 201-10 (22-24 June 1995).
5. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, ‘‘Myrinet -- a Gigabit-per-Second Local Area Network,’’ *IEEE Micro* **15**(1), pp. 29-36 (February 1995).
6. K. Bolding, M. Fulgham, and L. Snyder, ‘‘The case for chaotic adaptive routing,’’ *IEEE Transactions on Computers* **46**(12), pp. 1281-1292 (December 1997).
7. S. Borkar, R. Cohn, G. Cox, T. Gross, H. T. Kung, M. Lam, M. Levine, B. Moore, W. Moore, C. Peterson, J. Susman, J. Sutton, J. Urbanski, and J. Webb, ‘‘Supporting Systolic and Memory Communication in iWarp,’’ *17th Annual International Symposium on Computer Architecture*, Seattle, Washington, pp. 70-81 (May 28-31, 1990).
8. R. Cohen, ‘‘Smooth Intentional Rerouting and its Applications in ATM Networks,’’ *IEEE INFOCOM’94*, Toronto, pp. 1490-1497 (June 1994).
9. F. J. Corbato, ‘‘A Paging Experiment with the MULTICS System,’’ Project MAC Memo MAC-M-384, MIT, Cambridge, MA (July 1968).
10. 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).
11. W. J. Dally and H. Aoki, ‘‘Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels,’’ *IEEE Transactions on Parallel and Distributed Systems* **4**(4), pp. 466-475 (April 1993).
12. B. V. Dao, S. Yalamanchili, and J. Duato, ‘‘Architectural support for reducing communication overhead in multiprocessor interconnection networks,’’ *Third International Symposium on High-Performance Computer Architecture*, San Antonio, TX, pp. 343-52 (1-5 Feb. 1997).
13. M. De Prycker, *Asynchronous Transfer Mode: Solution for Broadband ISDN (3rd edition)*, Prentice Hall, New York (1996).
14. J. Duato, ‘‘A Necessary And Sufficient Condition For Deadlock-Free Adaptive Routing In Wormhole Networks.,’’ *IEEE Transactions on Parallel and Distributed Systems* **6**(10), pp. 1055-1067 (October 1995).

15. J. Duato, ‘‘A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks,’’ *IEEE Transactions on Parallel and Distributed Systems* **7**(8), pp. 841-54. (August 1996).

16. J. Duato, ‘‘Deadlock avoidance and adaptive routing in interconnection networks,’’ *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing*, Madrid, Spain, pp. 359-364 (21-23 Jan. 1998).

17. C. Eddington, ‘‘InfiniBridge: An InfiniBand Channel Adapter with Integrated Switch,’’ *IEEE Micro* **22**(2), pp. 48-56 (Mar/Apr 2002).

18. J. Flinch, M. P. Malumbres, P. Lopez, and J. Duato, ‘‘Performance Evaluation of a New Routing Strategy for Irregular Networks with Source Routing,’’ *14th Int’l Conf on Supercomputing*, pp. 34-43 (2000).

19. M. Galles, ‘‘Spider: A High-Speed Network Interconnect,’’ *IEEE Micro* **17**(1), pp. 34-39 (January/February 1997).

20. A. Gersht and A. Shulman, ‘‘Architecture for Restorable Call Allocation and Fast VP Restoration in Mesh ATM Networks,’’ *IEEE Transactions on Communications* **47**(3), pp. 397-403 (March 1999).

21. C. J. Glass and L. M. Ni, ‘‘The Turn Model for Adaptive Routing,’’ *Journal of the Association for Computing Machinery* **41**(5), pp. 874-902 (September 1994).

22. 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).

23. J.-M. Hsu and P. Banerjee, ‘‘Performance measurement and trace driven simulation of parallel CAD and numeric applications on a hypercube multicomputer,’’ *IEEE Transactions on Parallel and Distributed Systems* **3**(4), pp. 451-464 (July 1992).

24. R. Kawamura and H. Ohta, ‘‘Architectures for ATM Network Survivability and Their Field Deployment,’’ *IEEE Communications Magazine* **37**(8), pp. 88-94 (August 1999).

25. J. H. Kim, Z. Liu, and A. A. Chien, ‘‘Compressionless Routing: A Framework For Adaptive and Fault-Tolerant Routing,’’ *IEEE Transactions on Parallel and Distributed Systems* **8**(3), pp. 229-244 (March 1997).

26. K. Murakami and H. S. Kim, ‘‘Comparative Study on Restoration Schemes of Survivable ATM Networks,’’ *IEEE INFOCOM’97*, Kobe, Japan, pp. 345-352 (April 1997).

27. M. J. Pertel, ‘‘A Critique of Adaptive Routing,’’ Computer Science Technical Report 92-06, California Institute of Technology, Pasadena, CA (June 1992).

28. F. Petrini, W.-C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, ‘‘The Quadrics Network: High-Performance Clustering Technology,’’ *IEEE Micro* **22**(1), pp. 46-57 (February 2002).

29. 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).

30. Y. Tamir and Y. F. Turner, ‘‘High-Performance Adaptive Routing in Multicomputers Using Dynamic Virtual Circuits,’’ *6th Distributed Memory Computing Conference*, Portland, OR, pp. 404-411 (April 1991).

31. Y. Tamir and G. L. Frazier, ‘‘Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches,’’ *IEEE Transactions on Computers* **41**(6), pp. 725-737 (June 1992).

32. Y. F. Turner and Y. Tamir, ‘‘Connection-Based Adaptive Routing Using Dynamic Virtual Circuits,’’ *International Conference on Parallel and Distributed Computing and Systems*, Las Vegas, NV, pp. 379-384 (October 1998).

33. Y. F. Turner, *High-Performance Adaptive Routing in Multicomputers Using Dynamic Virtual Circuits, Ph.D. Dissertation, in preparation*, 2005.

34. L. Tymes, ‘‘Routing and flow control in TYMNET,’’ *IEEE Transactions on Communication* **COM-29**, pp. 392-398 (1981).

35. A. K. Venkatramani, T. M. Pinkston, and J. Duato, ‘‘Generalized theory for deadlock-free adaptive wormhole routing and its application to Disha Concurrent,’’ *The 10th International Parallel Processing Symposium*, Honolulu, HI, pp. 815-21 (15-19 April 1996).

Appendix 1: Packet Arrival Sequences

In order to demonstrate in a rigorous fashion how much buffer space is required for the control BVC to avoid deadlock cycles involving control packets, the following subsections present an enumeration of all possible sequences of packet arrivals on one RVC of a switch input port. We assume in this analysis that for an extended period, competition with data packets for link bandwidth prevents any of the control packets that arrive from being transmitted to the next switch. The control packets that arrive to the switch must be accommodated by the switch input buffers. Otherwise, the switch would block the arrival of subsequent control packets, a condition we wish to avoid in order to guarantee that inter-switch deadlocks involving control packets cannot form. The arrival sequences are partitioned into two major classes: sequences that contain no data packets, and sequences that contain data packets.

Cases Without Data Packets

We first consider all arrival sequences (on a single RVC) that lack data packets. The empty sequence (no arrivals at all) is a trivial example. In that case, no buffer space is required to accommodate the sequence. Non-empty arrival sequences without data packets begin with the arrival of either a CEP or a CDP.

If a CEP arrives first, then the next arriving control packet on the same RVC can only be a CDP that matches the CEP. When the CDP arrives, the switch identifies both the CDP and the CEP ahead of it as unnecessary, and both packets are deleted. We write the arrival sequence as follows (from first arrival to last): CEP [CDP]. The square brackets mean the arrival of the packet inside the brackets causes both that packet and the last packet in the sequence to be dropped. Only buffer storage for the CEP is required in this case, since the CDP is dropped as it arrives.

If a CDP (call it CDP_1) arrives first in the sequence, then the next arriving control packet on the same RVC is CEP_2 for a new circuit. CDP_1 will release the RVC upon transmission, and then CEP_2 will try to allocate it and set up a new mapping. After CEP_2 arrives, the next arrival would be CDP_2, which matches CEP_2. Both packets are unnecessary and are dropped. The sequence is thus the following: CDP_1

CEP_2 [CDP_2]. The maximum buffer requirement in this case is storage for the first CDP and the CEP.

To summarize, the maximum control packet storage required for arrival sequences without data packets is 1 CDP and 1 CEP for each RVC. Next we examine the arrival sequences that include data packets.

Cases With Data Packets

There are three categories of arrival sequences (on a single RVC) that include data packets: sequences with data packets that are all mapped; sequences with data packets that are all unmapped; and sequences with a mixture of mapped and unmapped packets. We show that the sequences with unmapped data packets pose the greatest demand on control packet buffer storage.

Consider the arrival sequences with data packets, all of which are mapped. A data packet is mapped if it is on a circuit that has allocated the RVC. The RVC is allocated by the CEP that precedes the data packet. The RVC is allocated when the CEP transmits to the next switch. Therefore, if a data packet is mapped, the CEP ahead of it is stored at some different switch or has been delivered to the destination. Since by our assumption control packets are unable to transmit downstream for a long time, if a control packet arrives in the sequence ahead of a data packet, then the control packet is present when the data packet arrives. Therefore, the data packet is unmapped, not mapped. That implies that each arrival sequence in this category begins with a sequence of data packets that are followed by a sequence of control packets. Having data packets strictly at the front of the arrival sequence does not change the storage requirement for the control packets at the tail of the sequence. Hence again storage is required for one CDP and CEP on each RVC. The worst case arrival sequence is the following, where $M_1 \cdots M_n$ are mapped data packets: $M_1 \cdots M_n \ CDP_1 \ CEP_2$ [CDP_2].

We now consider arrival sequences on one RVC that include data packets, all unmapped. When the first unmapped data packet arrives, flow on the primary BVC is blocked, preventing subsequent arrivals of data packets. Control packets can still arrive after the data packet.

Suppose the first packet that arrives in the sequence is a CEP. Then, if a CDP follows it, both are deleted. Otherwise, the data packet would follow the CEP. In that case, the sequence that requires the most buffering is the following: $CEP_1 \ U \ CDP_1 \ CEP_2$ [CDP_2]. U in the sequence is the unmapped data packet. The control packet storage required is a single CEP in addition to the usual per-RVC requirement of one CEP and one CDP.

If the first packet in the sequence were a data packet instead of a CEP, then the switch would generate a CEP and insert it ahead of the data packet. Thus this case is equivalent to the previous case

and has the same storage requirement.

If a CDP (call it CDP_0) were the first packet in the sequence, then the worst case sequence is CDP_0 followed by the sequence from the previous case. That is, the sequence is the following: $CDP_0\ CEP_1\ U\ CDP_1\ CEP_2\ [CDP_2]$. The storage requirement is one CDP and one CEP in addition to the usual per-RVC storage requirement.

Finally, we consider arrival sequences that include both mapped and unmapped data packets. By the same reasoning as for sequences with only mapped data packets, the mapped data packets in the sequences in the current category must precede all other packets in the sequence. Again, the presence of the mapped data packets does not affect the storage required for control packets. The worst case sequence is the following: $M_1 \cdots M_n\ CDP_0\ CEP_1\ U\ CDP_1\ CEP_2\ [CDP_2]$.

Appendix 2: Intra-Switch Deadlock Freedom

Theorem 3: Control packets do not enter intra-switch deadlocks.

Proof: Such deadlocks arise from dependency cycles within a single switch. We construct a graph of all the dependencies at a switch that involve control packet buffers. We show that the resulting graph is acyclic. Therefore, intra-switch deadlock is impossible.

From the algorithm listing and the enumeration of buffer states in Appendix 1, we can construct the dependency graph shown in Figure 21. The graph shows all possible dependencies for the buffers associated with two input RVCs: a mapped RVC i at input port X, and an unmapped RVC j at input port Y. Input ports X and Y are at the same switch. RVC i is mapped to RVC k of output port Z.

RVCs i and j are arbitrary representatives of their classes: mapped and unmapped RVCs, respectively. The set of all possible dependencies associated with these representative RVCs completely characterizes all the dependencies associated with all mapped RVCs and all unmapped RVCs. That is because it turns out that mapped RVCs have local dependencies only to packets associated with unmapped RVCs, and unmapped RVCs have local dependencies only to packets associated with mapped RVCs.

Each vertex of the dependency graph is labeled with a 3-tuple (α, β, γ). Component α specifies a buffer slot. Component β specifies the type of packet occupying the buffer slot. The packet may be a CDP, a CEP, a mapped data packet (denoted ‘‘M’’), or an unmapped data packet (denoted ‘‘U’’). Component $\gamma = TRUE$ if the input RVC used by the packet is mapped to an output RVC. Otherwise, $\gamma = FALSE$.

A directed arc from vertex (α_1, β_1, γ_1) to (α_2, β_2, γ_2) represents a potential dependency. The packet in buffer slot α_1 of type β_1 whose RVC mapping status is γ_1 may be blocked by the packet in buffer slot α_2 of type β_2 whose RVC mapping status is γ_2.

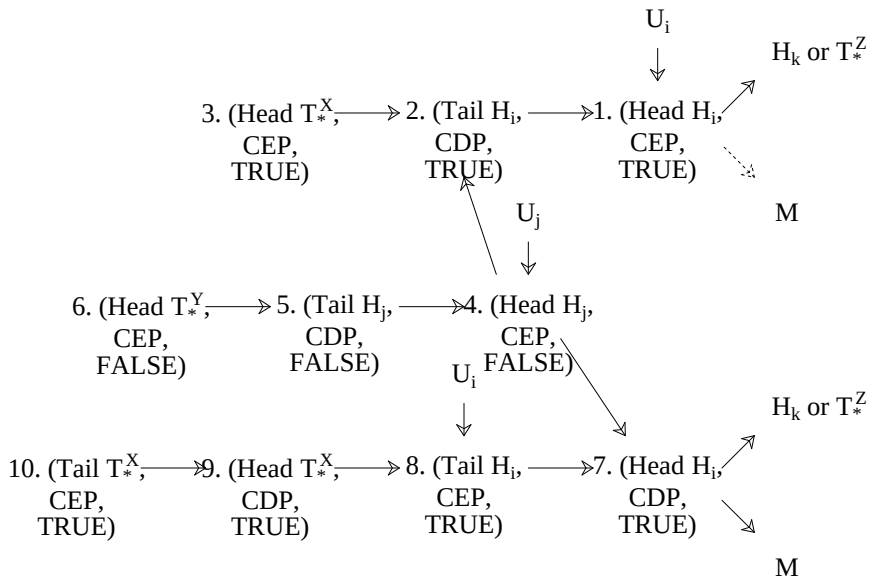


Figure 21: Buffer dependency graph for mapped RVC i and unmapped RVC k.

Figure 21 shows buffers H_i and T_* at input port X (vertices 1–3 and 7–10), and buffer H_j at input port Y (vertices 4–6). Output RVC k is on output port Z, and buffers H_k and T_*^Z are at the neighbor switch.

The symbol U_i represents an unmapped data packet on RVC i that resides in primary buffer N_X. Similarly, U_j represents an unmapped data packet on RVC j at input port Y. Each unmapped packet waits for the transmission of a CEP in order to be converted into a mapped data packet. The CEP resides in the H buffer for the unmapped packet's RVC. Note that U_i is unmapped even though RVC i is mapped to RVC k; according to lines 10–12 of the algorithm listing in Figure 4, U_i will be converted to a mapped data packet simultaneously with the next CEP transmission on output RVC k from buffer H_i.

To construct the graph, consider the packet at the head of H_i. It may be either a CEP or a CDP. Recall that H_i is the head of the logical queue of control packets for RVC i. Vertices 1–3 show the logical queue for RVC i if a CEP is at the head of H_i. If a CDP is at the head, then vertices 7–10 apply. In both cases, each packet not at the head of H_i depends only on the packet immediately ahead in the logical queue. Thus each of vertices 2, 3, and 8–10 has only one outgoing arc directed to the packet immediately ahead in the logical queue.

For the unmapped RVC j, the only packet that can be at the head of H_j is a CEP to establish a new

mapping. The logical queue for RVC j is shown in vertices 4–6. Again, each packet not at the head of the queue depends on the packet immediately ahead.

To complete construction of the graph, we examine the dependencies for each packet at the head of a logical queue (vertices 1, 4, and 7). First we examine vertices 1 and 7. Since RVC i is mapped to RVC k, the packet at the head of H_i blocks waiting for all mapped data packets on RVC i to be flushed out of the switch. That dependency corresponds to the arrows from vertices 1 and 7 to the symbol M. By theorem 1, mapped packets eventually make progress. Hence the arrows to symbol M cannot be part of a deadlock cycle. Once all the mapped data packets have left, the packet at the head of H_i waits for access to output port Z for transmission. At the next switch, the packet will be either discarded or else deposited in H_k or T_*^Z. We represent that in the graph by the arrows from vertices 1 and 7 to the symbols H_k and T_*^Z. Since those buffers are at a different switch, that dependency cannot cause intra-switch deadlock.

Finally, consider vertex 4. The CEP at the head of H_j is blocked from transmission until a mapping can be established from RVC j to an output RVC. Establishing a mapping may first require that an existing mapping be disestablished (algorithm lines 26–30). In that case, the CEP blocks waiting for a CDP on a mapped RVC to transmit and free up an output RVC. This explains the potential dependencies from vertex 4 to vertices 2 and 7.

By inspection, the buffer dependency graph of Figure 21 is acyclic. Therefore, there can be no intra-switch deadlocks involving control packets. □

Appendix 3: Switch State Space

Table 2 describes the state space of a switch. In the Table, each row identifies a set of states. The notation for states is identical to the description in Section 5.2 except that the symbol ‘‘d’’ is ‘‘don’t care’’.

Each set of states is labelled as either ‘‘reachable’’ or ‘‘unreachable’’. Reachable states may arise through error-free operation of the switch. In contrast, the switch never enters unreachable states.

The ‘‘comment’’ column explains the unreachable states. ‘‘Non-FIFO’’ means the state corresponds to a non-FIFO configuration of the buffers (e.g. the head slot of H_i is empty but the tail slot is full). ‘‘T/U inconsistent’’ means the unmapped packet and the contents of buffer T_* are not using the same RVC (Section 4.3). ‘‘Consecutive CEPs’’ and ‘‘consecutive CDPs’’ refers to states that violate the alternating order of CEPs and CDPs on a single RVC. ‘‘U implies CEP’’ refers to states in which an unmapped data packet is present but there is no CEP to establish a mapping for it. That condition is impossible because of lines 10–15 and 32–35 of Figure 4. Note that there is a small time window in

N	Reach	H_i		T_*		U_X	map_i	comment
		head	tail	head	tail			
1	F	–	CEP	d	d	d	d	non-FIFO
2	F	–	CDP	d	d	d	d	non-FIFO
3	F	d	d	–	CEP ∨ CDP	d	d	non-FIFO
4	F	d	d	–	CEP′ ∨ CDP′	d	d	non-FIFO
5	F	d	–	CEP ∨ CDP	d	d	d	non-FIFO
6	F	d	d	CEP ∨ CDP	d	i′ ∨ –	d	T/U inconsistent
7	F	d	d	d	CEP ∨ CDP	i′ ∨ –	d	T/U inconsistent
8	F	d	d	CEP′ ∨ CDP′	d	i ∨ –	d	T/U inconsistent
9	F	d	d	d	CEP′ ∨ CDP′	i ∨ –	d	T/U inconsistent
10	F	CEP	CEP	d	d	d	d	consecutive CEPs
11	F	d	d	CEP	CEP	d	d	consecutive CEPs
12	F	d	CEP	CEP	d	d	d	consecutive CEPs
13	F	d	CDP	CDP	d	d	d	consecutive CDPs
14	F	d	d	CDP	CDP	d	d	consecutive CDPs
15	F	CDP	CDP	d	d	d	d	consecutive CDPs
16	F	d	d	CDP′	CDP′	d	d	consecutive CDPs
17	**T**	–	–	–	–	– ∨ i′	d	
18	F	– ∨ CDP	– ∨ CDP	d	d	i	d	U implies CEP
19	**T**	–	–	CEP′	–	i′	d	
20	**T**	–	–	CDP′	–	i′	d	
21	**T**	–	–	CDP′	CEP′	i′	d	
22	**T**	CEP	–	–	–	– ∨ i	d	
23	**T**	CEP	–	–	–	i′	d	
24	**T**	CEP	–	CEP′	–	i′	d	
25	**T**	CEP	–	CDP′	–	i′	d	
26	**T**	CEP	–	CDP′	CEP′	i′	d	
27	F	CEP	CDP	d	d	– ∨ i′	d	CDP cancels CEP
28	**T**	CEP	CDP	– ∨ CEP	–	i	d	
29	F	CEP	CDP	CEP	CDP	d	d	CDP cancels CEP
30	F	CDP	d	d	d	d	F	CDP must be mapped
31	**T**	CDP	–	–	–	– ∨ i′	T	
32	**T**	CDP	–	CEP′ ∨ CDP′	–	i′	T	
33	**T**	CDP	–	CDP′	CEP′	i′	T	
34	**T**	CDP	CEP	–	–	d	T	
35	**T**	CDP	CEP	CEP′ ∨ CDP′	–	i′	T	
36	**T**	CDP	CEP	CDP′	CEP′	i′	T	
37	**T**	CDP	CEP	CDP	– ∨ CEP	i	T	
38	F	d	d	CEP′	CEP′ ∨ CDP′	i′	d	symmetry w/RVC i

Table 2: Switch State Space

which an unmapped packet can be present before the CEP for it has been created, but this is a transient state that quickly resolves. We ignore such transient states in our analysis.