88

Factors in the Performance of the AN1 Computer Network

Susan S. Owicki and Anna R. Karlin

June 15, 1992

Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center (SRC) and three other research laboratories are committed to filling that need.

SRC began recruiting its first research scientists in l984—their charter, to advance the state of knowledge in all aspects of computer systems research. Our current work includes exploring high-performance personal computing, distributed computing, programming environments, system modelling techniques, specification technology, and tightly-coupled multiprocessors.

Our approach to both hardware and software research is to create and use real systems so that we can investigate their properties fully. Complex systems cannot be evaluated solely in the abstract. Based on this belief, our strategy is to demonstrate the technical and practical feasibility of our ideas by building prototypes and using them as daily tools. The experience we gain is useful in the short term in enabling us to refine our designs, and invaluable in the long term in helping us to advance the state of knowledge about those systems. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

SRC also performs work of a more mathematical flavor which complements our systems research. Some of this work is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. The rest of this work explores new ground motivated by problems that arise in our systems research.

DEC has a strong commitment to communicating the results and experience gained through pursuing these activities. The Company values the improved understanding that comes with exposing and testing our ideas within the research community. SRC will therefore report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

Factors in the Performance of the AN1 Computer Network

Susan S. Owicki and Anna R. Karlin

DEC Systems Research Center
130 Lytton Ave.
Palo Alto, CA 94301

June 15, 1992

Abstract

AN1 (formerly known as Autonet) is a local area network composed of crossbar switches interconnected by 100Mbit/second, full-duplex links. In this paper, we evaluate the performance impact of certain choices in the AN1 design. These include the use of FIFO input buffering in the crossbar switch, the deadlock-avoidance mechanism, cut-through routing, back-pressure for flow control, and multi-path routing. AN1's performance goals were to provide low latency and high bandwidth in a lightly loaded network. In this it is successful. Under heavy load, the most serious impediment to good performance is the use of FIFO input buffers. The deadlock-avoidance technique has an adverse effect on the performance of some topologies, but it seems to be the best alternative, given the goals and constraints of the AN1 design. Cut-through switching performs well relative to store-and-forward switching, even under heavy load. Back-pressure deals adequately with congestion in a lightly-loaded network; under moderate load, performance is acceptable when coupled with end-to-end flow control for bursts. Multi-path routing successfully exploits redundant paths between hosts to improve performance in the face of congestion.

1 Introduction

AN1 (formerly known as Autonet)[SB90] is a local area network composed of crossbar switches interconnected by 100 Mbit/second, full-duplex links. Switch ports can be cabled to one another and to hosts in an arbitrary topology. Software in the switches automatically builds routing tables and rebuilds them whenever switches or links fail or recover. AN1 has been in operation at Digital's Systems Research Center (SRC) since January, 1990.

The purpose of this paper is to evaluate the performance impact of certain choices in the AN1 design. These include the use of input buffering in the crossbar switch, the deadlock-avoidance mechanism, cut-through routing, back-pressure for flow control, and multi-path routing. The performance metrics include network capacity, packet latency, and burst-transmission time.

The performance goals in the AN1 design were to provide low latency and high bandwidth in a lightly loaded network. In this the design is quite successful. However, some choices in the design lead to problems as the load increases. We examine each choice across a range of loads to determine how well the decisions scale. We also consider a number of topologies, since AN1 performance is strongly influenced by topology. Since most of the interesting performance questions are analytically intractable, we rely primarily on simulation.

Let us consider questions that arise concerning the design choices mentioned above. A synchronous switch with FIFO input buffers is well known to saturate at an output link utilization of 58% under a uniform workload. In fact, the situation with an asynchronous switch like AN1's is even worse: the link utilization saturates at 50% under uniform workload. FIFO input buffers are the most serious drawback to using AN1 in conditions of high load.

The AN1 deadlock-avoidance technique is known as up*/down* (pronounced up-star down-star) routing. It prevents deadlocks by restricting the set of paths that a packet can follow from source to destination. We show that the impact on performance is highly topology-dependent. Some common topologies, like the torus and hypercube, fare badly under up*/down* routing; other topologies suffer no loss in performance. We briefly examine other methods for dealing with deadlock and conclude that none of them are well suited to the constraints of the AN1 design.

AN1 uses cut-through routing, in which a packet can be transmitted from a switch as soon as its header has been received and the destination address extracted. We compare cut-through to store-and-forward routing, in which the entire packet must be received before any bytes can be transmitted. As expected, cut-through gives much lower packet latency when there is no contention for the outgoing link. It might be suspected that the advantage would be less in a congested network, where it is likely, even with cut-through, that a packet will have completely arrived before it can be transmitted on a congested link. However, simulation shows that the advantage of cut-through actually increases with load.

AN1 uses back-pressure by means of explicit start and stop signals to prevent buffer overflow at a receiving link. When a host sends too much traffic into a congested part of the network, back-pressure eventually propagates to the host and slows its transmission rate. We examine back-pressure as a means of dealing with congestion, and conclude it is inadequate by itself. When combined with an end-to-end flow control mechanism like windows, back-pressure can provide a tolerable degree of congestion control. However, more sophisticated congestion-avoidance techniques would give better performance under heavy load.

Finally, AN1 allows multiple paths to be used for routing packets between hosts. The particular path followed by a packet is determined by decisions made as it passes through each switch. These decisions are made in a way that is intended to route traffic away from congestion. We consider how successful the AN1 routing mechanism is in circumventing congested parts of the network, finding that it does quite well in some cases and less well in others. The AN1 strategy is compared to static routing and to some slightly more sophisticated forms of dynamic routing. The results are ambiguous. In some situations sophistication pays off well. However, over a range of topologies and a uniform client-server workload, the routing policy did not have much effect on average packet latency.

The rest of the paper is organized as follows. Section 2 gives background information on AN1 and on the workloads and topologies used in simulation. Section 3 examines each of the design decisions, identifying its effect on performance and comparing it to other alternatives. Section 4 presents our conclusions.

Note that this paper only considers performance during normal operation, i.e. when components are not failing or recovering. The performance of techniques for network reconfiguration are reported by Rodeheffer and Schroeder[RS91].

2 Background

2.1 AN1 Overview

AN1 consists of a number of switches and host controllers connected by point-to-point 100 Mbit/sec full-duplex links. A packet generated by a source host travels through one or more switches to reach a destination host. Switches contain logic to forward packets from an input port to one or more output ports, as directed by the destination field in the packet header. A crossbar connects the input and output ports.

Switches can be interconnected in an arbitrary topology. The topology can change with time, as new switches and links are added to the network, or as switches and links fail. A processor in each switch monitors the state of the network. Whenever the topology changes, all switch processors execute a distributed reconfiguration algorithm. This algorithm determines the new topology and loads the forwarding tables of each switch to route packets using all currently operational switches and links.

There is a 4 Kbyte FIFO buffer at each switch input port. A start/stop flow control mechanism is used to ensure that these buffers do not overflow. When the receiving FIFO is more than half full, a stop signal tells the transmitter to stop sending data; when the FIFO is half empty, a start signal causes transmission to resume. Limited buffering implies that a switch must be able to start forwarding a packet without having the entire packet in the local buffer. Consequently, switches forward packets using a cut-through technique that minimizes latency.

Switches in AN1 can be interconnected arbitrarily, so the network can assume any topology. The network can simultaneously handle multiple packets along different routes. This fact together with the unconstrained topology allow a great deal of flexibility in establishing routes that avoid broken components.

Since AN1 uses flow controlled FIFOs for buffering, assumes an arbitrary topology and does not discard packets in normal operation, deadlock is possible if packets are routed along arbitrary paths. In AN1, deadlock is avoided by restricting the paths to a set of deadlock-free routes, based on a loop-free assignment of direction to the operational links. Consider the graph whose vertices are switches and hosts, and whose edges are operational links between them. A breadth-first spanning tree is constructed, from a specified root. Each link is assigned a direction based on this spanning tree, with "up" meaning "toward the root". Ties are broken by comparing switch UIDs. With this assignment, the directed links do not form loops. A legal route is then defined to be one that never uses a link in the "up" direction after it has used one in the "down" direction. This up*/down* routing guarantees the absence of deadlocks. AN1 can route a packet along any of the shortest up*/down* paths between its source and destination. The path is determined dynamically as the packet passes from switch to switch.

2.2 Methods

Most of the analyses presented here are based on simulation, since many of the interesting features of AN1 are not analytically tractable. We used an event-driven simulator coded in Modula2+.

Simulations were typically run until 1 million messages were received (6 million for bursty workloads). For most of the runs, the standard error of the mean was less than 2%. In some high load cases, where the network was approaching saturation, the error of the mean was as large as 7%. We did not perform longer runs to reduce this error, because this is not an interesting range for network operation.

Much of what we want to study is sensitive to topology and workload. Each section of the paper studies a different aspect of the design. Often we construct a topology and workload specifically to illustrate the impact of that design choice. This is followed with more generic topologies and workloads to assess the impact in more realistic situations.

The topologies used will be described in the next section. In most of our simulations we used a client/server workload. A small number of hosts in the network are servers; the rest are clients. All traffic is between clients and servers.

We considered three types of traffic.

- Fixed length : all messages 1000 bytes.
- Bimodal : 80% of the packets are 100 bytes long; 20% are 1500 bytes long.
- Bursty : 80% of the packets are 100 bytes long; 20% are 1500 bytes long and occur in 100 Kbyte bursts.

The packet frequencies in bimodal and bursty workloads are based on observations of ethernet traffic at SRC; these apply to AN1 since it currently carries encapsulated ethernet packets. In the bursty workload, short packets represent RPC, while bursts represent activities like file transfer. In each simulation we use the simplest workload that suits the relevant phenomenon. For example, the capacities of different topologies can be compared with fixed-length messages, while congestion can best be studied with bursty traffic.

The bursty workload uses end-to-end flow control on the long messages, via a window of unacknowledged packets. A packet from a long message is generated at the source host when the appropriate acknowledgment is received. Window size is varied in the simulations when it has an impact on performance.

The performance metrics of interest are *packet latency* and *burst transmission time*. Packet latency is the time between generation of a packet at a source host and arrival of the first bit at the destination. Burst transmission time is the time between generation of the first packet at the source host and arrival of the last packet at the destination host.

Offered load is the percentage of link capacity that a host attempts to send. For the large topologies, we generally refer to the offered load at a server. For smaller topologies we indicate the offered load at each link.

2.3 Topologies

AN1 is designed to function correctly regardless of how switches and hosts are linked together, so long as there is a path between every pair of hosts. However, the performance of the network depends on the connection topology. In this section we describe the six topologies used in this study. We used a range of topologies both to explore the impact of topology on overall performance and to assess the significance of design alternatives in the context of possible installations. Of course, we make no claim to have spanned the range of possible topologies.

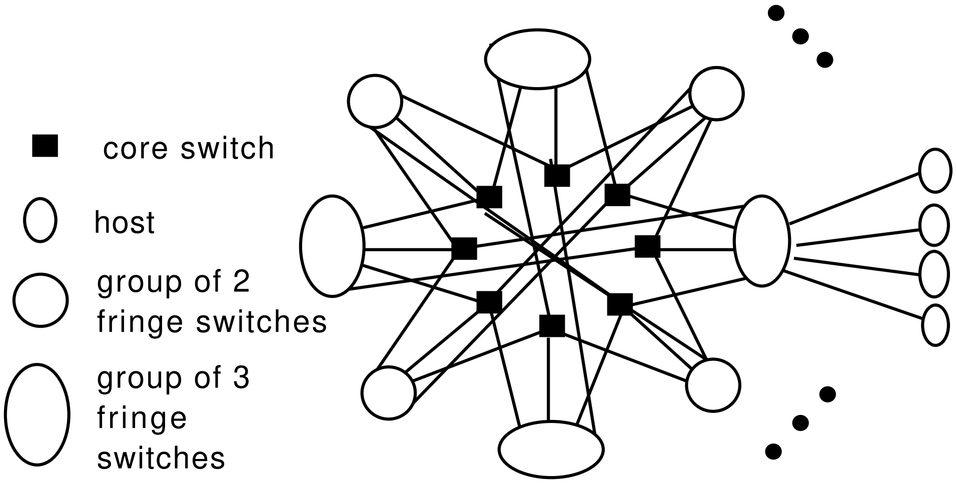

Figure 1: The Corefringe Topology

Table 1 gives the number of switches and hosts in each topology. The topologies are comparable in size to the network installed at SRC. The cost of an AN1 installation is proportional to the number of switches, so the host to switch ratio is included as an indicator of cost.

Topology	Switches	Hosts	Servers	Hosts/ Switch
Torus	20	80	8	4
Hypercube	32	96	9	3
Corefringe	28	80	8	2.9
Mstage	24	88	8	3.7
Floors	24	80	8	3.3
Floors+	24	80	8	3.3

Table 1: Topologies

All the topologies contain redundant connections to give a degree of fault tolerance. In each case, failure of a single switch cannot partition the network. Hosts have two connections to the network, but use only one of them at any time. For simplicity, we show only one connection per host. However we have dimensioned the topologies with enough switches to allow for redundant host connections. Each switch can have at most 12 connections.

The first two topologies are standard ones: the torus and hypercube. *Torus* is a 5 by 4 rectangular array of switches, with four hosts at each switch. Switches are connected

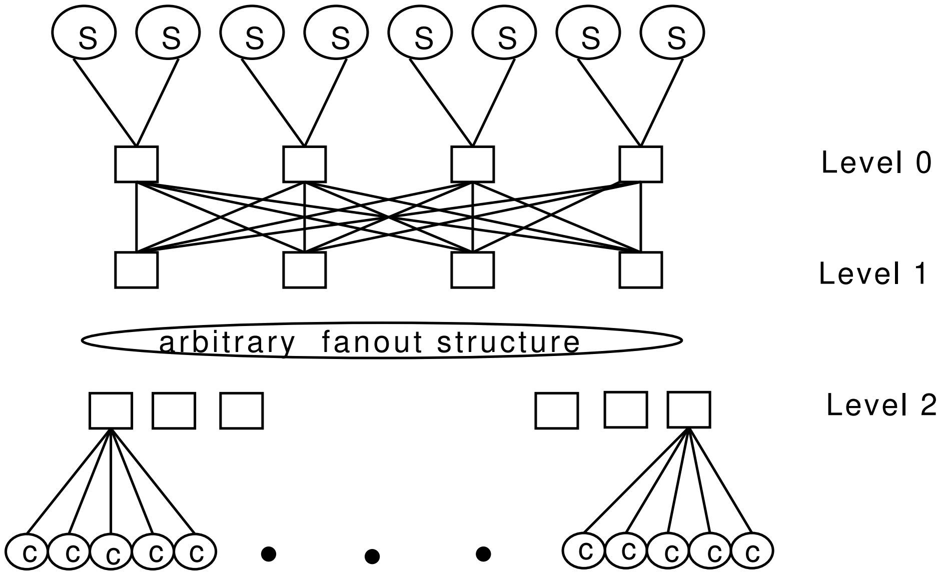

Figure 2: A Multi-Stage Topology

to their neighbors to the north, south, east and west, with wraparound at the edges of the rectangle. *Hypercube* is a 5-dimensional hypercube with $32 = 2^5$ nodes. A node with binary representation x is connected to the 5 nodes whose binary representation differs from x in exactly one bit.

Experiments with these topologies show that they are badly affected by up*/down* routing, as discussed in Section 3.2. We next consider two topologies, Corefringe[1] and Mstage, that are more suited to up*/down* routing. Both have the property that all shortest paths are up*/down* paths. They also give good performance in the face of switch failures, although those results will not be presented here.

Corefringe (Figure 1) has the best performance under all workloads considered. It consists of two levels of switches, 8 in the core and 20 in the fringe, plus a single root. All hosts are attached to fringe switches, four per switch. Each fringe is connected to four core switches. The connections are such that every pair of fringe switches is connected to one or two common core switches. Most messages flow from a source host, to a fringe switch, to the core, back to a different fringe switch, and finally to the destination host. (If source and destination are attached to the same fringe switch, the path does not go to the core.) The connections between core and fringe switches can be described precisely as follows. The fringe switches are divided into 8 groups. Group $2i$, $0 \leq i \leq 3$, has size 2, and group $2i + 1$, $0 \leq i \leq 3$, has size 3. Each fringe switch in group i has links to core switches $i - 1, i, i + 1$ and $i + 4$ (all taken mod

[1]Designed by Jim Saxe

8). Additional links (not shown in the figure) provide fault-tolerance; they are not used in normal operation. An additional switch (not shown) is designated as the root of the spanning tree. The root is connected to all core switches. No traffic goes through the root; it is merely a device to establish the desired up*/down* routing.

Multi-Stage (Figure 2) is a family of topologies designed for a client-server workload. In general, a multi-stage topology consists of levels (stages) of switches, where all links between switches connect switches on adjacent levels. All servers are connected to the switches in the lowest level, denoted level 0, and all clients are connected to the switches in the highest level. Between level 0 and level 1 there is a complete bipartite graph. From level 1 to the highest level there can be any sort of fanout structure. The multi-stage topology we call MStage has 3 levels, with 4 switches in level 0, 4 switches in level 1, and 16 switches in level 2. There are two servers connected to each level 0 switch and 5 clients connected to each level 2 switch. Switches at level 1 are connected to 8 switches at level 2 in such a way that each level 2 switch has links to 2 different level 1 switches.

Finally, since some of the above topologies might be hard to wire, we consider a topology whose structure could easily map to a multi-floor installation. In *Floors*, most connections are between switches and hosts on a single floor. The structure on each floor (Figure 3) consists of a star with 2 center switches and 4 arm switches. Each of the arms has 5 hosts connected to it. There is one server host on each of two arms; the rest of the hosts are clients. This structure is replicated 4 times. In addition, the center switches are connected in the pattern shown in Figure 4.

The *Floors+* topology is like Floors except for four additional links between center switches on each floor (shown in dotted lines in Figure 4). The significance of these extra connections is discussed in Section 4.2.

Figure 5 shows packet latency under a fixed-length workload for the six topologies. There is a wide range in the load at which the topologies saturate, running from roughly 40% to 70%, and in latency, under moderate to heavy load. Note that Floors performs substantially worse than Floors+, even though they are nearly identical. This illustrates the not surprising fact that the interconnection pattern has a substantial effect on performance, and the more surprising fact that what appear to be small differences in topology can lead to large performance differences. Factors in the performance of the topologies will be discussed where appropriate in later sections.

3 Design Decisions

3.1 FIFO Input Buffering

An AN1 switch consists of n input links connected to n output links by a complete crossbar. A FIFO queue is provided at each input port. There, packets wait until they can be transmitted to the desired output port. There is a well-known performance problem

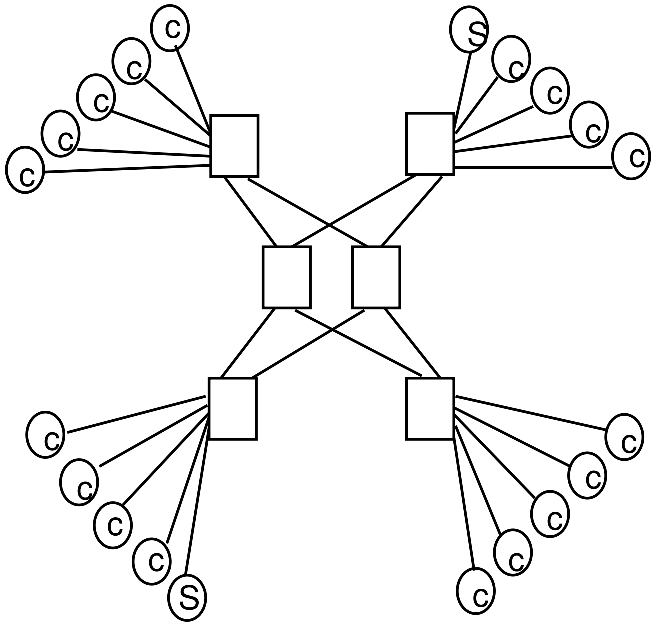

Figure 3: Star Portion of Floors Topology

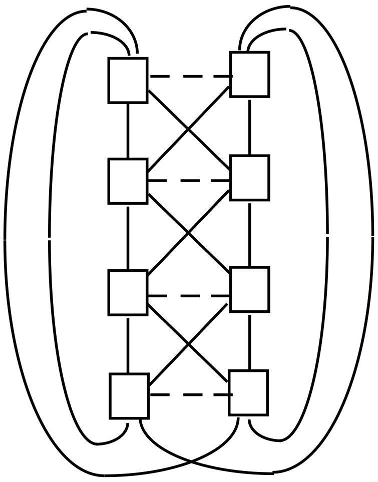

Figure 4: Interconnect Portion of Floors and Floors+ (Dotted lines represent links in Floors+)

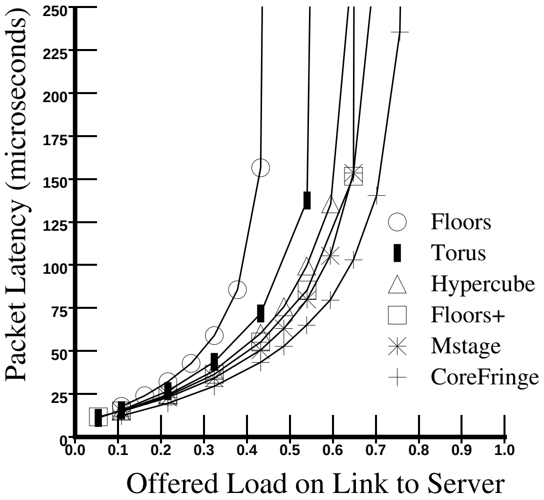

Figure 5: Effect of Topology on Packet Latency

associated with input queueing, known as *head-of-line blocking* (HOL blocking). HOL blocking occurs when a blocked message at the head of an input queue prevents a message behind it from being transmitted over an idle output link.

This problem has been extensively studied. Baskett and Smith [BS76] and Karol et al. [KHM87] showed that for an n by n synchronous switch with fixed-size messages and uniformly distributed destinations, the limiting saturation throughput of a switch with input queueing is 58%. Performance degrades further if the traffic is unbalanced, as shown in [LL89, Li90].

For asynchronous n by n switches with exponentially distributed message lengths and uniformly distributed destinations, the saturation throughput is $n/(2n-1)$. The links of a saturated switch can be modeled as n servers in a closed queuing network with n customers. After completing service, a customer branches with equal probability to any server. The solution of this queueing system gives the stated result. As n tends to infinity the saturation throughput tends to 0.5.

This limitation on the throughput of an input queueing switch is a serious impediment to good performance under heavy load in AN1. The effect of HOL blocking can be mitigated by topology; for example by having multiple connections between a pair of switches. However, in all the topologies considered here, the bottleneck under client/server workloads is a switch, not a link. In switches without HOL blocking, the bottleneck, if any, is a link. It would be possible to design AN1 topologies which performed better at high load, at the cost of using more switches. The alternative is to use a switch that has no HOL blocking.

There are several ways to avoid HOL blocking with a crossbar switch. They include random access input buffers, a faster switch fabric with output buffers, input smoothing, and shared output buffering. The performance of various alternatives are studied by Hluchyj and Karol [HK88], Iliadis and Denzel[ID90], Chen and Stern [C90], and Pattavina [Pa90, Pa91]. All of these alternatives to simple FIFO input queueing entail greater hardware complexity and hence greater cost.

It would also have been possible to base the switch on a shared bus rather than a crossbar, thus avoiding the HOL problem. This option was not pursued because AN1 is a prototype for a gigabit network, and a shared bus is not feasible at gigabit speeds.

3.2 Up*/Down* Routing

Deadlock is avoided in AN1 by restricting the paths of packets to a set of deadlock-free routes. As described in section 2, one direction of each link is denoted "up", and the other "down". All routes are restricted to be of type up*/down*.

This scheme has a number of obvious advantages. It is simple, in that the deadlock-free routes can be set up very efficiently, and universal, in that it works for any topology and buffer size. There are also problems, however. Up*/down* may reduce the number of links available for traffic between certain source-destination pairs, and it may cause the lengths of the paths between hosts to exceed the shortest path length. Most seriously, it can introduce severe bottlenecks.

We illustrate the bottleneck problem with two examples, the hypercube and the torus. Consider a hypercube with $2^d = n$ switches. Suppose that each switch has a single host connected to it, and that each host sends a single message to a random host in the network. It can be shown that the expected number of messages that must go through the root switch in this case is $n^{\log_2(3/2)}$. Therefore, the average latency is $\Omega(n^{0.58})$ using up*/down*, whereas there are routing schemes for which this traffic pattern has latency $O(\log n)$[VB81]. This discrepancy remains the same even under a client/server workload, where each client sends a message to a random server.

For an n by n torus, the situation is not much better. Indeed, a random permutation can be routed on a torus with average (and maximum) latency $O(n)$[Le92], whereas with up*/down* routing, the bottleneck at the root of the underlying spanning tree results in an average latency $\Omega(n^2)$.

Simulation was used to compare up*/down* routing to arbitrary shortest paths, under a client/server workload. To avoid deadlock with arbitrary shortest paths, infinite buffers were assumed.

Figure 6 presents the results of these simulations for Torus, Floors, and Hypercube. All three topologies suffered reduced performance when routes were restricted to up*/down*. For the hypercube, the latency at an offered load of 75% jumped from

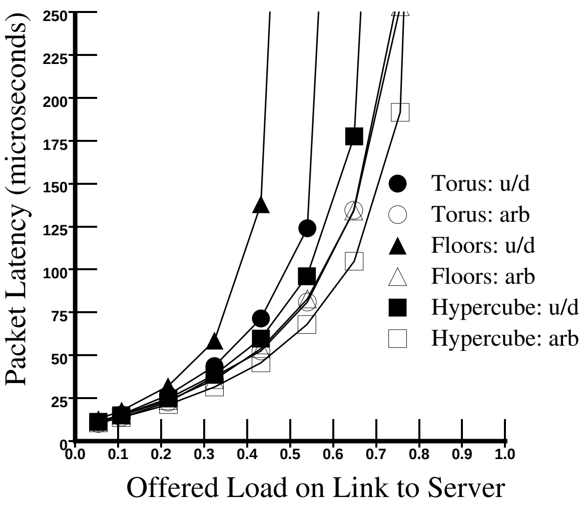

Figure 6: Up*/down* (u/d) vs. arbitrary shortest paths (arb)

roughly 200 usec using arbitrary shortest path routes to 700 usec using up*/down* routes. Note that the hypercube performance using arbitrary shortest paths is similar to that of CoreFringe.

Figure 6 illustrates that up*/down* routing hurts performance for some standard topologies. However, it is possible to design topologies whose performance is unaffected by up*/down* routing; in Mstage, CoreFringe, and Floors+ all shortest paths are up*/down*.

Perhaps the most disturbing aspect of up*/down* routing is that apparently minor changes in a topology can have a substantial effect on performance. This can be illustrated with Floors and Floors+. After Floors was designed, it was discovered that up*/down* routing prevents any traffic from passing through the center switch on the same floor as the root. Thus the root becomes a bottleneck, since all traffic to and from hosts on that floor must pass through it. Adding links between the center switches, as in Floors+, changes the level assignment in the spanning tree and eliminates the bottleneck. Latency and network capacity improve dramatically, as shown in Figure 5. Note that no traffic actually flows through the added links, since they are not on any shortest paths. The improvement in performance caused by this change is certainly non-intuitive.

We have seen that up*/down* can reduce the number of available paths and introduce bottlenecks. It is reassuring that there are topologies that yield good performance with up*/down* routing, but it does not seem desirable to use a scheme in which minor changes in topology can degrade performance in such a major way.

It is natural to ask whether there are more effective deadlock avoidance schemes. Many deadlock-free routing algorithms have been developed, primarily for store-and-forward networks [MS80]. These algorithms are based on *structured buffer pools*. Buffers are partitioned into classes, and the assignment of buffers to messages is restricted in a way that prevents cycles. Unfortunately, structured buffer pool approaches require enough buffer capacity at each link to store multiple packets. A buffer-pool scheme would require considerably more buffer space than the current 4K bytes/link, even with a maximum packet size smaller than AN1's 64 Kbytes. (Note that, with cut-through, allowing packets bigger than the buffer cause no problems.)

A more relevant comparison is to the deadlock avoidance scheme of Dally and Seitz [DS87], which was designed to work with cut-through and requires less buffer space. The Dally and Seitz scheme (DS) restricts routes in order to break deadlock. Physical channels are split into virtual channels, and routes are defined with respect to the virtual channels in such a way that no cycles can result.

There are several problems with this approach in the context of AN1. Constructing deadlock-free routes for an arbitrary network would be a time-consuming addition to reconfiguration. (Note that the scheme was proposed in the context of fixed regular topologies.) In addition, the switch needed to implement this scheme would be more complex. It would require separate buffers for each virtual channel, some means of connecting each virtual channel to the crossbar, and separate back-pressure for each virtual channel. Most seriously, the number of virtual channels per physical channel cannot be bounded *a priori*. For example, with the shuffle-exchange topology, DS introduces $log(n)$ virtual circuits per link, where n is the number of switches in the network.

The main advantage of DS, is that it tends to avoid the bottleneck problems of up*/down*. Therefore, for many standard networks (k-ary n-cubes, cube-connected-cycles and shuffle-exchange graphs), performance should be better than with up*/down*.

In conclusion, up*/down* is well suited to efficient automatic reconfiguration in the presence of failures. It prevents deadlock for any topology and with minimal buffer capacity. However, the performance penalty can be substantial unless the topology is chosen carefully. There doesn't seem to be any way to do better with arbitrary topologies and small buffers. If the problems of up*/down* are unacceptable, then one of these constraints should be relaxed.

3.3 Cut-through Routing

AN1 uses cut-through routing. This means that instead of receiving the entire packet before beginning transmission to the next switch, the head of a packet can be advanced directly from incoming link to outgoing link.

As soon as the switch logic examines the header of a message, it selects a next link on the route (provided that an idle outbound link is available) and begins forwarding the packet down that link. As the packet is forwarded at each switch, its bits can simultaneously occupy all of the links between the source and the destination. If a packet arrives at a switch when there is no idle outgoing link, the bits of the packet are buffered until a link becomes available. At that point the packet can begin transmission, whether or not all of its bits have been received.

Consider two networks, identical except that one uses cut-through (CT) and one uses store-and-forward (SF). If identical messages arrive when the networks are idle, the difference in latency is precisely $h \times tb$, where h is the hopcount between source and destination, and tb is the time to transmit the message body. It is not obvious, however, what happens when there is conflicting traffic in the network. Suppose each network is offered the same load. How do packet latencies differ between the two networks?

These question have been studied to a limited extent. Kermani and Kleinrock [KK79, KK80] compare the delay performance of circuit switching (CS), store and forward message switching (SFMS) and virtual cut-through switching (VCTS), where traffic is traveling in a tandem path with no cross traffic. Under VCTS, a message arriving in an incoming link is permitted to utilize an idle outgoing link as soon as its header has been received. If, however, the outgoing channel is busy, the message must be received completely before being sent out toward the destination node. Kleinrock and Kermani present analytic delay models for each of these switching mechanisms, using the Kleinrock independence assumption [Kl64]. They found that VCTS out-performed SFMS at low loads, but that the advantage disappeared at higher loads. Abo-Taleb and Mouftah [AM87] obtain similar results for another variant of cut-through, which is also different from the form used in AN1.

An exact analysis for the average packet delay in a store and forward network (a tandem array of switches, with a Poisson source of messages of exponentially distributed length) is not known. Approximations have been obtained using the Kleinrock independence assumption. Such an approximation is quite inaccurate since interarrival times are strongly correlated with packet lengths. (The independence assumption is a good approximation when several packet streams merge, so that some independence is restored. This is not the case for a tandem array with a single source.) Rubin presents a more accurate approximation [Ru76], and gives an exact solution for the case where packets have fixed lengths [Ru75].

We use simulation to study these questions in more detail. The first case we consider, shown in Figure 7, is a tandem array of switches with two sources, s1 and s2, and all traffic destined for host d. Most of the interesting phenomena we see in the comparison between CT and SF arise in this simple situation. Table 2 gives the latencies at low load (0.05) and at high load (0.9) for both CT and SF. The workloads considered are fixed length messages and bimodal traffic. Because cut-through and store-and-forward have somewhat different buffer requirements, unbounded input buffers were used in this simulation.

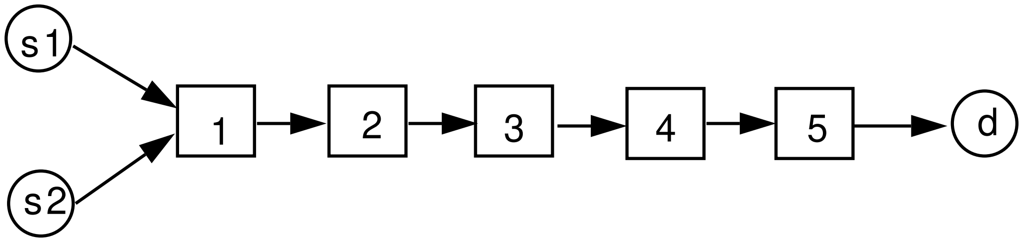

Figure 7: Topology for Comparison of Cut-through and Store-and-Forward

Message Length	ct at 0.05	ct at 0.90	sf at 0.05	sf at 0.90
Fixed	13.4	530	413	912
Bimodal: short	13.7	1030	113	1740
Bimodal: long	13.7	1030	614	1720

Table 2: SF and CT Latencies for Tandem Switches

For fixed length messages, the difference between SF latency and CT latency is close to $h \times tb$. (It is 4.99 $\times tb$ at an offered load of 0.05, and 4.77 $\times$ tb at an offered load of 0.9. In this case $h = 5$.) Note that the advantage of cut-through is consistent over all loads, in contrast with Kermani and Kleinrock's findings. This is entirely due to the difference between cut-through and virtual cut-through, which is actually a hybrid of cut-through and store-and-forward.

For bimodal messages, the difference between SF latency and CT latency remains at least $h \times tb$. It is very close to this value for long messages. For short messages, however, the difference in latency is even more pronounced (more than twice this value at an offered load of 0.05, and more than 17 times this value at an offered load of 0.9). The following gives some intuition for this result. Suppose a short message follows a long message across a link. With CT, the long message can be completely transmitted before the short message arrives. With SF, only part of the long message can be transmitted before the short message arrives. Thus the short message has to wait for the long message to complete transmission before it can start.

With bursty traffic, new phenomena appear. Table 3 gives the burst transmission time for CT and SF as a function of window size at an offered load of .05. As the window size increases, the performance of SF improves. Packets in the burst fill up the pipeline, so that there is parallelism in the transmission of burst packets. Improvement continues until the window size reaches the hopcount plus one. For cut-through, on the other hand, there is no pipeline to fill, since long packets tend to be strung out over all the links. Increasing the window size beyond 2 does not improve burst transmission time.

Similar behavior is observed with full topologies. Figure 8 shows packet latency for

Window Size	CT	SF
w=1	102	521
w=2	86	261
w=5	86	113

Table 3: Burst Transmission Time for CT and SF, in msec.

Mstage with the bimodal workload. For both short and long messages the difference in latency exceeds $h \times tb$.

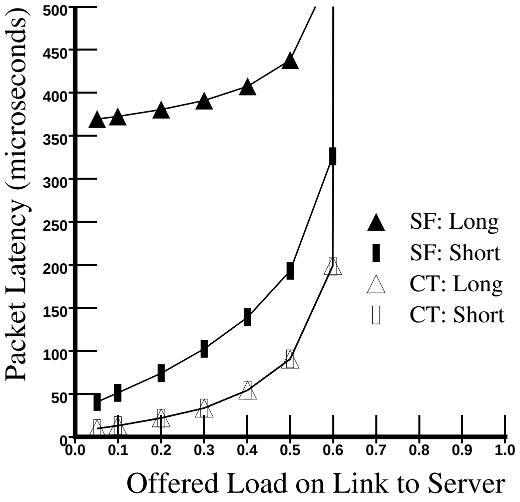

Figure 8: CT vs. SF on Mstage Topology, Bimodal Traffic

3.4 Back-pressure and congestion

Congestion occurs when a link or switch receives packets more quickly than it can transmit them. Congestion may be momentary or sustained. Sustained congestion can delay packet transmission and sometimes reduce throughput. In some networks, though not AN1, it causes packet loss. Congestion can arise even in a lightly-loaded network, for example, if two bursts pass through the same link. AN1 uses back-pressure to deal with congestion without packet loss.

Figure 9 shows a simple topology used to illustrate congestion handling in AN1. Suppose sources s1 and s2 simultaneously send bursts to destinations d1 and d2,

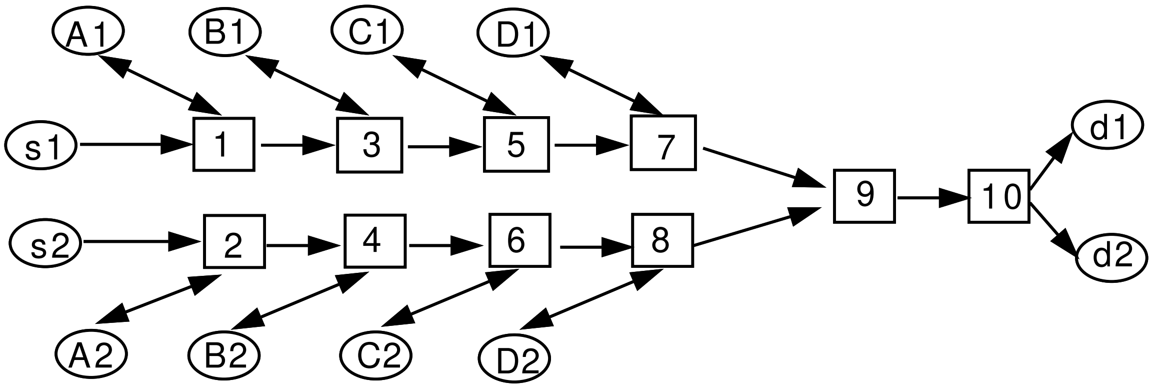

Figure 9: Topology Used to Illustrate Back Pressure: parallel lines with cross traffic

respectively, with a large window size. Packets arrive at the bottleneck link (9, 10) faster than they can be transmitted. As the buffers at switch 9 fill, stop signals are sent to switches 7 and 8. Buffers at these switches fill as well, causing stop signals to be sent to switches 5 and 6. Eventually stop signals propagate back to hosts s1 and s2. Then the hosts can only send packets into the network at the rate at which they can reach their destination.

How effective is this as a way of dealing with congestion? First, it allows the network to continue operating in the presence of congestion. No packets are dropped in the network (although, if congestion is bad enough, timeouts may cause higher-level protocols to drop packets). Once back-pressure reaches the sources of the congestion, it limits their ability to send traffic into the network.

However, there are several ways in which this approach is less than ideal. Consider traffic from C1 to D1. Although this traffic does not pass through the bottleneck, it is still slowed down by the back-pressure. Table 4 shows the average delay experienced by cross traffic (from Ai to Bi, Bi to Ci, and Ci to Di) as it competes with bursts. For a 1-packet window, cross-traffic experiences no delays due to congestion. With a window of 2, cross-traffic close to the bottleneck experiences some delay. As the window size increases, cross-traffic further from the bottleneck encounters congestion.

The behavior with large window sizes illustrates that back-pressure does not work well as the sole means of congestion control. The control it exerts is too little and too late: too late, because it does not take effect until congestion is already a serious problem, and too little, because the hosts are allowed to continue sending at a rate that keeps the network congested. An additional problem is that back-pressure is unselective: it applies to all traffic equally, whether or not it will pass through the bottleneck.

Jain[Ja90] has observed that back-pressure is well suited to dealing with short-lived congestion, but not the sort of long-lived congestion in the example above. He recom-

window size	Ai - Bi latency	Bi - Ci latency	Ci - Di latency
1	36	36	36
2	35	36	44
3	35	36	246
4	36	204	399
5	94	422	400
6	296	463	400
7	401	466	400

Table 4: Cross Traffic Latencies

mends the combination of back-pressure with other mechanisms to slow transmissions from the sources of congestion. A number of such mechanisms have been proposed (e.g. [Ja88], [RJ90]) although none that we are aware of are designed to work in the presence of multiple paths between hosts. Although AN1 does not include this sort of congestion avoidance, it does reasonably well in an environment where bursts are transmitted with small windows.

Figure 10 shows the impact of window size in the Mstage topology under the bursty workload. In this graph, latency is the delay experienced by short packets from the time they reach the head of the host queue until the time their first byte is delivered. We deliberately exclude the time spent waiting behind other packets at the host because this is artificially inflated for large window sizes by the early entry of burst packets into the queue. By using latency from the time a packet reaches the head of its queue, we focus on the way window size affects latency through network congestion. Even at a reasonably light load, like 30%, a large window size results in latency more than four times that of window size 1. Burst transmission time was not substantially affected by window size because of cut-through. Thus small window sizes are preferable for AN1.

In summary, back-pressure combined with window-based flow control for bursts provides an acceptable level of congestion control at moderate loads. The operating points achieved in this way are not likely to be optimal, since the window size is statically determined. A more sophisticated scheme might give better performance.

3.5 Multi-path Routing

AN1 topologies typically contain redundant paths for fault tolerance. Given that multiple paths exist, there must be some means of determining what path a particular packet will follow. Other networks with multiple paths use strategies that include static assignment of paths, dynamic updating of routing tables, and dynamic choice among the entries in a static routing table. AN1 follows the latter strategy in choosing among the set of shortest up*/down* paths. In this section we will assess the performance

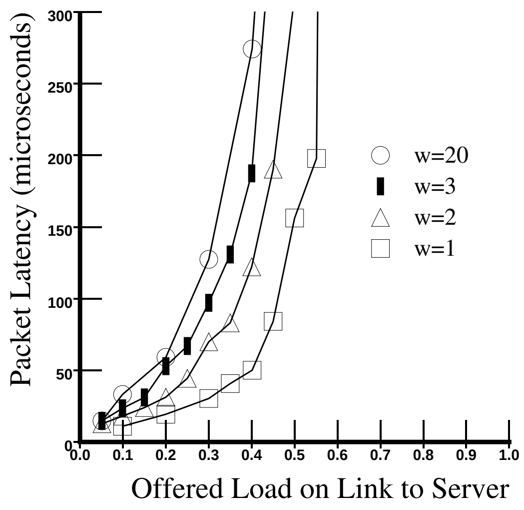

Figure 10: Effect of End-to-End Flow Control Window (window size = w).

of AN1 multi-path routing. After reviewing relevant literature, we describe the AN1 mechanisms. Their behavior is illustrated with some small examples. Finally, we compare the performance of AN1 to static routing and to some mechanisms that offer the possibility of better performance.

In the Arpanet, [MW80] routing tables are periodically recomputed based on information about the load on links in the network. Such techniques would not work in the current AN1 implementation, where the time to reload the routing table is very large compared to the message switching time. It is not clear whether the switch design could be altered to accommodate this sort of adaptive routing.

Reeves et al. [RGC] and Ke and Eager[KE] evaluate the performance of certain locally adaptive routing strategies. The strategies evaluated are different from those we examine here, and in both cases the network design differs in a fundamental way from that of AN1. Both papers found that adaptive routing policies outperformed static routing strategies, particularly with increasing network size and increased nonuniformity in the traffic patterns.

In AN1, when a packet arrives at the head of its input buffer, the switch's routing table is consulted to determine the possible outgoing links for the packet. If there is more than one, the choice is determined as follows. First, if any of the links are idle, a random idle link is selected.[2] Second, if all of the links are busy, the packet enters a

[2] Although the original AN1 design called for random selection, it was eliminated in the implementation

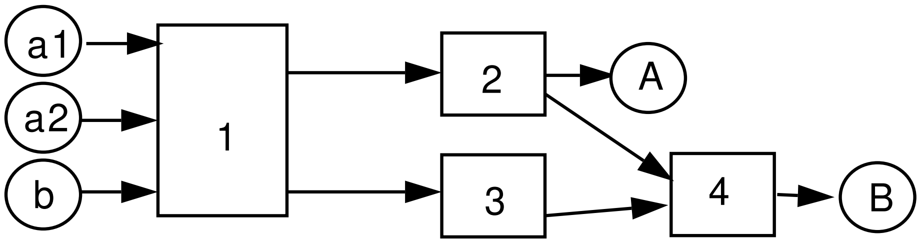

Figure 11: Adaptive Routing: Congestion at Decision Point

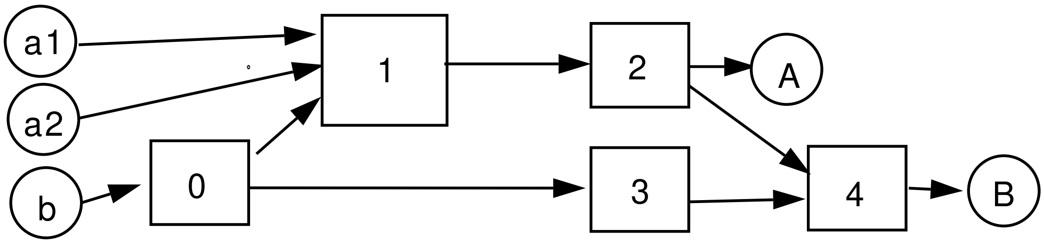

Figure 12: Adaptive Routing: Congestion Downstream of Decision Point

queue of blocked packets. When an output link becomes free, the first eligible packet in the queue is transmitted.

Both of these mechanisms exert an adaptive effect in that they tend to direct traffic away from congested parts of the network. We can see how this occurs in the following examples.

First, consider the topology in Figure 11. There are three sources, $a1$, $a2$ and b. Both $a1$ and $a2$ want to transmit 50% of a link bandwidth to A, and b wants to transmit 100% to B. There is enough capacity in the network to transmit this load, if packets from b pass through switch 3. However, if some of those packets follow the route through switch 2, there will not be enough bandwidth on the (1, 2) link to handle the traffic for A. In this example, AN1's routing mechanism works extremely well. Simulation confirms that packets from b almost always find link (1,2) busy and so use link (1,3). This allows the entire offered load to be transmitted. Note that the AN1 strategy works well in this case because the routing decision is made at the same switch where congestion arises.

In the next example, the routing decision is made upstream of the congestion point. Figure 12 shows a slightly different topology; the offered load is the same as in the previous example. Once again, the network has adequate capacity to handle the offered load, but only if all traffic from b to B goes through switch 3. But in this case, AN1

because of space constraints. In the implementation, the lowest-numbered available link is taken. We have chosen to evaluate multi-path routing using random selection because it increases the symmetry of topologies and thus makes them easier to understand.

is not able to use the available capacity. Random choice of outgoing links cause 1/2 of the packets from b to be routed to switch 1. Contention there means that only 1/3 of link (1,2)'s bandwidth can be used by traffic from b, so back-pressure is exerted on link (0, 1). Typically, the back-pressure will stop transmission in the middle of a packet. When the subsequent start signal is sent, and b finishes transmitting the stopped packet, both outgoing links are once again free. Therefore, on all of b's routing decisions, each of the outlinks is chosen with equal probability. Link (0, 1) can only transmit at 1/3 of the link bandwidth, and the load offered to (0, 3) can be no greater. Thus a load of 2/3 is transmitted from b to B. Sources $a1$ and $a2$ are limited to a load of 1/3 because of contention for link (1, 2).

Examining the reasons for this difficulty suggests two possible modifications to AN1. The first is to have back-pressure take effect on packet boundaries. This can be accomplished by using two kinds of stop signals.[3] The first, a *stop when ready* signal, is sent somewhat before the buffer is full. The link receiving the stop signal can continue to transmit its current packet, but not a new one. An *urgent stop* is sent when the buffer is nearly full, and its receipt causes transmission to stop at once. The signals can be set up so that only very large packets require an urgent stop.

This modification improves AN1's performance in the current example. When link (0,1) is not exerting back-pressure, b's traffic is still split equally among the two outlinks. On the other hand, when link (0,1) is stopped on a packet boundary, source b is not stopped and the next packet coming in will be routed on link (0,3). Therefore, b will be able to transmit its entire load. However, 1/3 of b's traffic goes through switch 1. As a result, each of a_1 and a_2 will only get a third of link (1,2).

A final modification is to use more information in making the routing decision. Since lines that have recently been stopped are likely to lead to congested parts of the network, one option is to choose the output link that has been least recently stopped. In the current example, this strategy leads to optimal performance. Once link (0,1) receives a stop signal, b sends all of its traffic on link (0,3). Thus all hosts are able to send their full offered load.

These examples suggest that applying back-pressure on packet boundaries, as well as load-based decision making (such as least-recently-stopped) might be useful improvements. However, it is not obvious that similar situations arise in real topologies.

To see that these situations do arise, consider the subgraph of the Mstage topology shown in Figure 13. We focus on two clients transmitting to two servers. Here packets from c1 to s1 can go through switch 3 or 4, and those from c2 to s2 can go through 4 or 5. If too many packets are sent to the shared switch 4, performance will be degraded. Figure 14 presents the performance of alternative routing possibilities. Here single-path routing (spr) refers to a static routing choice which assigns the same link to both transmissions. (Any static routing strategy must do this for some such pairs.) Note that least-recently-stopped routing (lrs routing) achieves latency as low as on a completely

[3] This technique was suggested by Jim Saxe.

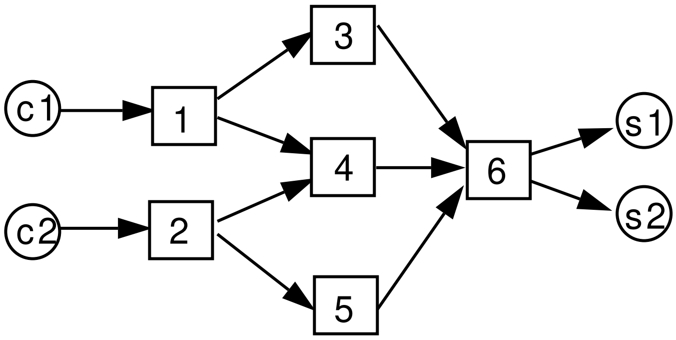

Figure 13: Subgraph of Mstage

uncongested path (no cong). AN1 routing is not quite as good, but it is substantially better than single-path. Back-pressure on packet boundaries (on pb) does not provide much of an improvement over AN1.

This example suggests that adding least-recently-stopped routing to AN1 could substantially improve performance in some cases. However, experiments over a range of topologies with bursty client-server workloads failed to show a substantial improvement in average latency with this modification. Where an improvement was observed, it was visible only as the networks came close to saturation. For these topology/workload combinations, the AN1 strategy, or random selection of static paths at configuration, worked as well as the more sophisticated techniques. It may be that a workload with more pronounced hot spots would benefit more from the proposed modifications. Still, present evidence suggests that they are not worth increasing the complexity of the switch.

4 Conclusions

We have studied a number of the design decisions from AN1. Our findings are summarized below.

The most significant impediment to good performance under medium to heavy load in AN1 is the head-of-line blocking problem in the switch. In all experiments, across the range of topologies, we found that the bottleneck to performance was in the switches, and not in the links. The HOL blocking problem can be mitigated by designing topologies specific to the desired workload, or by using extra switches so that each carries a lower load.

Up*/down* has a substantial negative impact with some topologies, and is undesirable because it makes topology design difficult. It is well suited, however, to the design goals of AN1. These goals include the use of arbitrary topologies, fast reconfiguration,

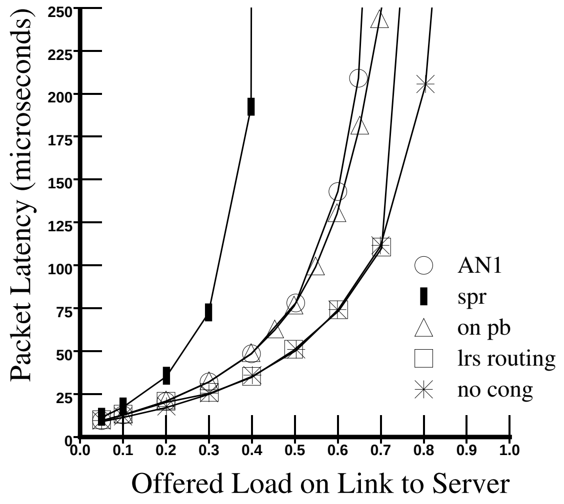

Figure 14: Performance of Adaptive Routing on a Subgraph of Mstage: comparison of AN1, single-path routing (spr), back-pressure on packet boundaries (on pb), least-recently stopped routing (lrs routing), and uncongested routing (no cong).

accommodation of large packets, small buffers, no packet loss and no deadlock detection delay. There does not seem to be any reasonable alternative to up*/down* that satisfies these goals.

The performance of cut-through switching exceeded our expectations. It provides much lower latency than store-and-forward switching not only at light loads, but also under very heavy load.

Back-pressure works reasonably well in moderately loaded networks with end-to-end flow control. It could profitably be supplemented with a congestion avoidance protocol, although it is not clear how to integrate such a protocol with multi-path routing.

Adaptive routing was found to be useful in some situations. Perhaps surprisingly, static routing performed nearly as well on the large topologies. Using static routing would simplify the switch hardware, although choosing static paths might increase the time required for reconfiguration. We did not find the more elaborate adaptive routing schemes, (such as using the least-recently-stopped link) to have a substantial performance impact for the workloads and topologies we considered, although larger differences might show up under less uniform workloads.

The impact of topology on performance is striking. Many standard high-flux networks (such as the hypercube) are adversely affected by up*/down* routing. To provide good performance for a given workload, a topology must have two properties. First, it must provide adequate link and switch bandwidth to carry the load. (HOL blocking forces separate consideration of link and switch bandwidth). Second, it must not lose capacity due to up*/down* routing. Unfortunately, the combination of HOL blocking, up*/down* routing and adaptive multi-path routing make it hard to analyze the capacity of a given topology, thus complicating topology design.

In summary, we have found that AN1 performs well at low loads, in the region for which it was designed. To operate well at increased loads, the head-of-line blocking switch should be replaced. Cut-through and adaptive routing work well. Back pressure with small flow-control windows gives acceptable performance, but in a network where substantial congestion is anticipated, back-pressure should be supplemented with a more sophisticated congestion avoidance protocol.

Acknowledgements

Mike Goguen, Jim Horning, Hal Murray, Jim Saxe, Mike Schroeder and Chuck Thacker contributed helpful comments on an earlier version of this paper.

References

[AM87] A. Abo-Taleb and H. Mouftah. Delay Analysis Under A General Cut-

Through Switching Technique in Computer Networks. *IEEE Transactions on Communications*, Vol. C-35, No. 3, March 1987.

[BS76] Forest Baskett and Alan Jay Smith. Interference in Multiprocessor Computer Systems with Interleaved Memory. *Communications of the ACM*, Vol. 19, No. 6, June 1976.

[C90] J.S.-C. Chen and T.E. Stern. Throughput Reduction due to Nonuniform Traffic in a Packet Switch with Input and Output Queueing. IBM Research Report RC 16354 (72610), December 1990.

[DS87] W.J. Dally and C.L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, Vol. C-36, No. 5, May 1987.

[HK88] M.G. Hluchyj and M.J. Karol. Queueing in High-Performance Packet Switches. *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 9, December 1988.

[ID90] I. Iliadis and W.E. Denzel. Performance of Packet Switches with Input and Output Queueing. *Proc. SIGCOMM '88*, Stanford CA, August 1988.

[Ja88] V. Jacobson. Congestion Avoidance and Control. *ICC/SUPERCOM 1990*, Atlanta, GA, April 1990.

[Ja90] Raj Jain. Myths about Congestion Management in High-Speed Networks. Digital Equipment Corporation, DEC-TR-724, October, 1990.

[KHM87] M.J. Karol, M.G. Hluchyj and S.P. Morgan. Input Versus Output Queueing on a Space-Division Packet Switch. *IEEE Transactions on Communications*, Vol. C-35, No. 12, December 1987.

[KE] Y. Ke and D. Eager. Locally Adaptive Routing in Mesh-Connected Networks.

[KK79] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, Vol. 3, pp. 267-286, September 1979.

[KK80] P. Kermani and L. Kleinrock. A Tradeoff Study of Switching Systems in Computer Communication Networks. *IEEE Transactions on Computers* Vol. C-29, No. 12, December 1980.

[Kl64] L. Kleinrock. *Communication Nets: Stochastic Message Flow and Delay*. New York, McGraw-Hill, 1964.

[Le92] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, Inc. 1992.

[Li90] S. Li. Nonuniform Traffic Analysis on a Nonblocking Space-Division Packet Switch. *IEEE Transactions on Communications*, Vol. 38, No. 7, July 1990.

[LL89] S. Li and M.J. Lee. A Study of Traffic Imbalances in a Fast Packet Switch. *Proc. of the 1989 Infocom Conference*, pp. 538-547, 1989.

[MW80] J.M. McQuillan and D.C. Walden. The ARPA Network Design Decisions. *Computer Networks*, Vol 1, August 1977, 243-289.

[MS80] P.J. Merlin and P.J. Schweitzer. Deadlock Avoidance in Store-and-Forward Networks – I: Store-and-Forward Deadlock. *IEEE Transactions on Communications*, Vol. C-28, No. 3, March 1980.

[Pa91] Achille Pattavina. Performance Evaluation of a Batcher-Banyan Interconnection Network with Output Pooling. *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 1, January 1991.

[Pa90] Achille Pattavina. Performance Evaluation of ATM Switches with Input and Output Queueing. *International Journal of Digital and Analog Communications Systems*, Vol 3. 1990.

[RJ90] K. K. Ramakrishnan and Raj Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Trans. on Computer Systems*, Vol. 8, No. 2, May 1990.

[RGC] D.S. Reeves, E.F. Gehringer and A. Chandiramani. Adaptive Routing and Deadlock Recovery: A Simulation Study.

[RS91] T.L. Rodeheffer, M.D. Schroeder. Automatic Reconfiguration in Autonet. *Proc. 13th Symposium on Operating Systems*, October, 1991.

[Ru75] Izhak Rubin. Path Delays in Communication Networks. *Applied Mathematics and Optimization*, Vol. 1, No. 3, 1975.

[Ru76] Izhak Rubin. An Approximate Time-Delay Analysis for Packet-Switching Communication Networks. *IEEE Transactions on Communications*, Vol. C-24, No. 2, February 1976.

[SB90] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, C.P. Thacker. Autonet: a High-speed, Self-configuring, Local Area Network Using Point-to-point Links. *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 9, October 1991.

[VB81] L. Valiant and G. Brebner. Universal Schemes for Parallel Communication. *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, pp. 263–277, May 1981.