Communication Models for a Free-Space Optical Cross-Connect Switch

David Er-El[1] Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

Abstract

We design and analyze communication algorithms for an optical switch based on the electroholographic effect, that provides capabilities for configuration changes and a limited multicast. As configuration changes incur a high overhead, we suggest the use of a high-speed forwarding unit as part of the network interface. We develop broadcast algorithms that trade off the number of configuration changes and forwarding operations in order to achieve the minimal completion time. For general point-to-point communication, we use the limited multicast to create a spanning topology along which messages can be forwarded. This requires care to avoid collisions, but is still preferable to using configuration changes if the topology is rich enough but still has a low degree.

1 Introduction

High-speed communications are a basic requirement for parallel computing. This has led to much interest in optical networks, as they boast a very high bandwidth, orders of magnitude greater than the bandwidth of copper wire or coax [6]. Optical networks also have a packing advantage, due to the small diameter of optical fibers and the option of free-space propagation; this may be important in systems with thousands of links. Such considerations have led to several suggestions for using optical interconnects in parallel systems [9, 3, 12, 13].

In large systems fully connected networks are not feasible, so some sort of switching must be used. In optical networks there are two main options: to use electronic switching or to use optical switching. In electronic switching the laser signal is transformed into electronic form, switched, and then transformed back into a laser signal. In this way the full potential bandwidth of fiber links is largely unused due to the "electronic speed bottleneck" imposed by the relatively slow electronic switching and modulation techniques [11]. The other option is to use optical switches, which do not require optoelectronic conversions of the data. Our work is based on such an all-optical design. This also implies the use of circuit switching rather than packet switching, as the parsing of a packet's header to decide on the next hop imposes technical difficulties with current technology [5]. Circuit switching avoids this need, as routes are created in advance.

The most common form of optical switching used today is Wavelength Division Multiplexing (WDM), in which each node has a tunable transmitter and a fixed receiver. The nodes are connected by a passive star coupler, and use a "broadcast and select" mechanism: the information-bearing light beams emanating from each node are split equally and delivered to all other nodes, but only the desired destination node has the correctly tuned receiver. The down side is that this wastes optical power, since each transmitted signal is divided evenly between all the nodes in the network. In addition, each node requires a distinct reception wavelength, so the number of nodes is limited to the number of available wavelength channels. Even so, extra control is needed so that two nodes do not transmit at the same wavelength at the same time [4].

Our work is based on a novel optical switch based on the electroholographic effect [10]. This effect allows prerecorded holograms to be activated as desired in order to steer light beams to their destination. Thus it does not suffer from the drawbacks of WDM as mentioned above, although it still does need a similar control network. These switches have performed well in a laboratory setting (in terms of bit-error-rate and crosstalk), and a small prototype network is now being built. The switch and its properties are described in Section 2.

Our work is concerned with extending the understanding of this novel optical switch beyond the device level, to the system level. In particular, we investigate the effect of the novel switching technology on the design and implementation of a communications library that will use them. To this end we design and evaluate broadcast algorithms in Section 3, and support for arbitrary point-to-point communication in Section 4.

2 Modelling the Switch

In order to develop a communication system we need a model of the capabilities of the underlying switch and of the cost of various operations. We start by describing the switch and its place in the system, and then go on to develop a more abstract model.

2.1 System Overview

The system consists of computing nodes that are connected via optical fiber to a central optical switch. All the nodes are also connected by an electronic control network such as an Ethernet. Apart from the compute nodes, this net-

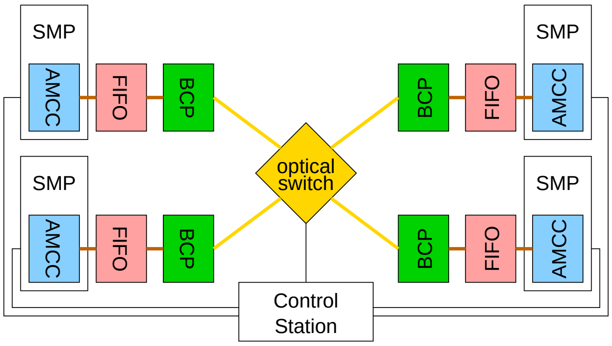

Figure 1: Schematic of the system configuration.

work also connects to a distinguished control station that controls the configuration of the switch.

The connection between the nodes and the optical subsystem is done in three steps:

- AMCC: a PCI card that functions as a device that can read/write data from the optical network.
- FIFO: an FPGA that performs buffering and adjustments of the clock between the PC and the BCP.
- BCP: a transceiver that performs the conversion between the electronic media and the optical media.

A schematic sketch of the system (with 4 nodes) is shown in Figure 1.

2.2 Switch Physical Properties

The switch is based on a KLTN crystal in the paraelectric phase, which allows pre-recorded volume holograms to be switched on and off by applying a voltage to the crystal [1]. The recorded holograms, when active, cause light beams that impinge upon the crystal to be diffracted in selected directions. When the holograms are not activated, the crystal is transparent and light beams pass through it without any change in direction.

Switching the holograms is in principle very fast, but in practice, it can take several milliseconds for the new connection to stabilize. Messages cannot be sent during this period of time as they may not reach their destination, and the result of such a transmission would be unpredictable. Hence configuration changes are a costly operation that should be avoided as much as possible.

A single crystal has two configurations: light beams either continue straight ahead, or else they are diffracted. This basic structure is used as a building block in the construction of networks that can support switching between many nodes [10]. The topology used is typically either a multistage network or a crossbar. In either case, the switches can potentially be set up in such a way that one node can receive messages from several other nodes. These nodes are effectively on the same bus, and a bus mastering method is needed to ensure that no node will receive more than one message at a time. Without proper control, transmissions may collide resulting in garbled data.

2.3 The Model

The components of our prototype optical system are limited by the current hardware purchased under financial limitation, and by the amount of time that could be spent on their development in an academic setting. The model we describe here is free of these limitation. We have based it on the physical one but extended it with many features. We believe these features can be implemented with the proper resources. We have also made several simplifying assumptions to ease the evaluation as described below.

In the model each node is connected to the optical switch through two links, one dedicated for transmit and the other dedicated for receive. A node is capable of doing the following: local computations, sending a request for a switch configuration change on the control network, and sending packets on the optical media to other nodes. The nodes that receive such transmissions depend on the current switch configuration. The optical network supports both unicast and limited multicast packet delivery. The control network supports broadcast as well.

Terminology

The **switch configuration**, denoted by $C = (\alpha_1, \alpha_2, ..., \alpha_i, ..., \alpha_N)$, indicates that for each i a packet sent by node i will be directed by the switch to node α_i. Note that this may or may not be a permutation. When we use the multicast capability with multicast size m, the α_i are actually vectors: $\alpha_i = (\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,m})$. In this case, a packet sent by node i will be directed by the switch to all these nodes, which should be different. Naturally, as there are N nodes and mN destinations, each node can potentially receive messages from multiple other nodes. It is the responsibility of the communication protocol to synchronize the senders so that this does not happen at once.

Nodes can change the switch configuration by requesting a **local configuration set**, denoted $CSet(i,k)$. This changes the destination of node i to be k, leading to the configuration $C_{new} = (\alpha_1, ..., \alpha_{i-1}, k, \alpha_{i+1}, ..., \alpha_N)$.

Consider a configuration in which $\alpha_i = j$ and $\alpha_j = k$. In this configuration node i can only send packets directly to node j and node j can only send packets directly to node k. In the case where node i wants to send a packet to node k, it can either change the switch configuration or use node j to receive the packet and deliver it to node k. The latter is called **forwarding**. Forwarding can be done with any number of intermediate nodes.

Parameters

- N: the number of nodes in the system. We will typically assume this is a power of two (i.e. $N = 2^k$ for some k), and focus on the range from 64 to 2048.
- m: the multicast size. In most cases we will use $m = 1$. In any case this is a small constant number, due to optical power limitations.
- T_c: the time it takes to change a configuration. We will assume this is relatively long, on the order of sev-

eral milliseconds, as it is in our prototype. However, we will also check the sensitivity of our algorithms to improvements in this parameter.

- T_b: the time it takes to transmit/receive 1 byte, including the overhead of the application. Thus T_b is measured from the time a transmitted byte is still in the application layer until the time that same byte gets to the optical wire. We assume that transmit and receive are equivalent in this respect.
- T_f: the time to transmit/receive 1 byte using forwarding. Forwarding and its implications on overhead are discussed at length below.
- T_d: the difference between a regular byte transfer to a forwarded one ($T_b - T_f$).
- S: the message size. To avoid issues of packetization to fit into buffer space, we will assume messages are limited to about 1KB.

Assumptions

Setting a configuration is an operation that, once started, ends after a fixed period of time. The operation consists of the request (T_r), the setting itself (T_s) and a waiting period until the switch stabilizes and the receiver synchronizes (T_w). Therefore $T_c = T_r + T_s + T_w$. We assume that the configuration set itself is the only atomic action, since it involves setting the switch's hardware. The other two actions can be overlapped in time with similar actions made by other nodes. For example, if the switch gets several requests for changes then each requesting node should waits for it's local configuration change to stabilize, and these occur simultaneously. Second, we assume that T_s is smaller by several orders of magnitude than T_r and T_w, and therefore $T_c \approx T_r + T_w$ which are both not atomic. We conclude that setting a global configuration which consists of k local configuration changes takes about T_c time instead of $k \cdot T_c$ time.

The control network is a broadcast/select network like Ethernet. Each node is capable of listening to all the traffic on the network, even if that traffic is not sent to it. Since configuration changes are sent over the control network we can assume that every node continuously monitors these requests and stays fully synchronized with the current switch configuration.

Finally, we assume there exists some external synchronization method that can be used to start an operation (such as a broadcast) almost simultaneously on all nodes. Likewise, we sometimes assume the existence of synchronization pulses that are used to prevent the danger of transmission collisions. These mechanisms can be based on using the control network.

2.4 The Forwarding Unit

Forwarding is assumed to be a very quick action in comparison to regular send and receive. This assumption depends on the use of a hardware **forwarding unit** (FU) that is part of the network interface, as shown in Figure 2.

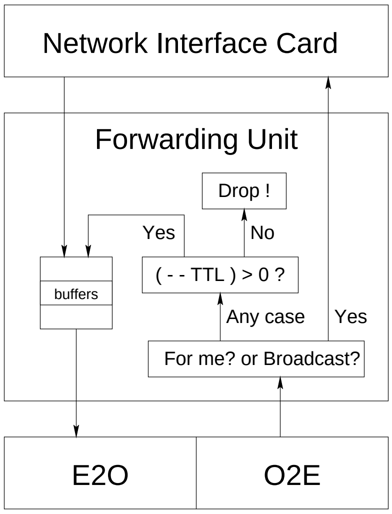

Figure 2: The forwarding unit.

The node's network interface card (the AMCC card in the prototype) will be referred to as the NIC, and the unit which performs optoelectronic conversion (the BCP) will be represented by the O2E and E2O units. We will place the FU between the node's NIC and the E2O and O2E units. Every received packet is passed from the O2E unit to the FU. The FU's job is to examine the packet header and decide on the fly whether to deliver it to this node or to return it back to the network. A packet that is delivered to this node is passed directly to the NIC. A forwarded packet is moved to outgoing buffers on the FU itself, and then forwarded to the E2O unit and through it back to the optical switch. The outgoing buffers are needed to coordinate the forwarding with sending new packets by the NIC. The FU has priority over the NIC, since we must first handle packets that are already on the network before we produce more packets. The NIC therefore has its own buffers so whenever it is blocked, outgoing packets are saved to be later transmitted when the bus is free.

In addition to the basic forwarding functionality, we propose that the FU support some extra functionality for efficient broadcast. To this end we add a special broadcast bit in the header of each packet. If this bit is on then the incoming packet is both passed to the NIC *and* at the same time forwarded along the outgoing optical link. In this way, a message can be forwarded from one node to another, and also get delivered to all the nodes in the forwarding route. Obviously we need some method to stop the forwarding at some stage. This is done by a "time to live" (TTL) field in the header. In each hop the TTL value is decreased by one, and when it reaches 0 the packet is dropped. The initiator of the broadcast should put the correct value of the TTL so the message would get to the correct nodes and not beyond them.

An additional useful mechanism is to retain the last packet sent out of the FU until a new packet arrives. This mechanism enables the corresponding node to forward the same message to different nodes with only part of the operating system overhead. Without this mechanism, a node A which wants to forward a message to both B and C would have to do deliver the full message to the networking subsystem, which would then request a configuration change, and only after the configuration change occurred pass the whole message again down the stack and forward it. With the proposed mechanism, once the header of a message is delivered up the stack a request for the configuration change is made. When it is done, a directive is sent back down the stack to instruct the FU to forward the last message again, but this time, since the switch configuration is different, it will be forwarded to a new node.

From the behavior of the FU we can assume T_f will be in the order of few microseconds at most. A regular send/receive of a byte that should pass all the network stack of the operating system should take at least two orders of magnitude longer then forwarding. Therefore $T_f \ll T_b$. The switch takes about 10 milliseconds to stabilize so we can conclude that $T_c \gg T_f$. Assuming the size of a message is up to 1K, we estimate that in the time that a configuration is set a message can be forwarded about 10 times.

3 Broadcast Algorithms

In this section we use the broadcast operation to investigate the tradeoff between configuration changes and forwarding. We present several algorithms that span the spectrum of possibilities, from the exclusive use of configuration changes to the exclusive use of forwarding. We assume no multicast for these algorithms. The performance metric is the time to complete a single broadcast operation.

3.1 The Algorithms

3.1.1 Use Configuration Changes

In the naive method a node broadcasts a message by doing N-1 local configuration sets and sends. This method does not use forwarding. The algorithm is performed by the source of the broadcast, which we take to be node 0 (code done by a specific node is denoted by an underlined tag, and "node i" denotes all nodes):

```
node 0:
for ( i = 1 .. N ) {
    CSet( 0, i )
    send
}
```

The running time is $(N-1)(T_c + ST_b)$. We need $N-1$ stages and in each stage we perform one configuration set and one send.

3.1.2 Forwarding in a Ring

This algorithm is at the other end of the spectrum: it only uses forwarding, and no configuration changes. The switch is configured to create a ring topology, in which each node is connected to the next one. The message is then forwarded along this ring, and also delivered to all the nodes along the way. The sender sets the TTL to $N-1$. As forwarding is done automatically by the FU, it is not listed in the code.

```
node i: CSet( i, i+1 )
node 0: send
    // all nodes forward and deliver locally
```

The running time is $T_c + (N-1)ST_f + 2ST_d$. There is only one T_c, because all the configuration changes are done in parallel. Each forwarding step takes ST_f time. In addition, the first and the last nodes must go through the software stack at an additional cost of $2ST_d$.

3.1.3 Forwarding in a Tree

Forwarding in a ring is sequential, and therefore has a running time of $O(N)$. This can only be reduced by a parallel algorithm that performs multiple sends simultaneously. The obvious candidate is a tree topology. Specifically, in each stage all the nodes that have already received the message forward it to a new set of nodes, as shown in the following code:

```
node 0:
CSet( 0, 1 )
send
node i:
for ( stage = 1 .. lg N ) {
    if ( have message ) {
        CSet( i, i+2^stage )
        forward
    }
}
```

The running time is calculated as follows. In each stage the number of nodes that got the broadcast message is doubled, leading to a logarithmic number of stages. In each stage we have a configuration set and one forwarding of the message. Finally, we need to cross the software stack at the end. The total is therefore $(\lg N)(ST_f + T_d + T_c) + 2ST_d$.

3.1.4 Forwarding in a Tree Embedded in a Ring

The previous algorithm performs a configuration change each time it needs to access a new node. During these operations no productive work can be done. Given our assumption that $T_c \approx 10ST_f$ or more, it is more efficient to use this period of time to forward messages without the need for more configuration sets. The idea is to start with a ring topology, and then to recursively work on increasingly smaller fragments of the ring. In each stage, forward the message to the next k nodes along the ring. After it has left the current node, make a configuration change to some node q in the current ring fragment, such that when node q will receive the message, the number of nodes left in the fragment after node q will be about equal to the number of nodes that didn't receive the message between node k and node q. The value of k can therefore be calculated as $\frac{T_d+T_c}{ST_f}+1$: this is the number of consecutive message forwarding operations that can be overlapped with a configuration change. Figure 3 (left) shows the progression of the algorithm in the first two stages. The k light-gray nodes are the ones that benefit from the overlapped forwarding.

The description of the algorithms uses the notion of "end nodes": the two nodes that are reached at the end of each overlapped forwarding and configuration change pattern (one is at the end of the sequence of the k forwarding operations, and the other is reached on the new arc after the configuration change). Initially, node 0 is the only end node.

```
node i: CSet( i, i+1 )
node 0: send
node i:
for ( i = 0 .. lg N − 3 ) {
    if (i is an end node) {
        // forward automatically and deliver locally
        CSet( i, i + (N+k)/2^(stage+1) )
        forward
    }
    // else forward automatically and deliver locally
}
```

The running time is calculated as follows. The number of nodes that have received a message in stage i is $k(2^i-1)$, so N nodes are covered in $i=\lg\frac{N}{k}+1$ stages. Substituting k leads to $i\approx\lg(\frac{NST_f}{T_d+T_c})=\lg N+\lg(\frac{ST_f}{T_d+T_c})$. Assuming that $T_c\gg T_d$ and that a configuration set takes more than a forwarding by about one order of magnitude, the second term becomes a small negative number (around -3.3). Therefore the number of stages is slightly less than $\lg N$. The amount of work done in each stage is by definition equal to the time for one forwarding, a configuration change, and another forwarding. The total running time is therefore $T_c+(\lg N-3.3)(2ST_f+T_d+T_c)+2ST_d$.

An obvious optimization is to allow all previous nodes to create arcs into the remaining area of the ring, leading to an alternating pattern of forwarding to k adjacent nodes and branching to k new nodes (Figure 3 right). Again distinguishing between end nodes e and other nodes o (where end nodes are those whose next ring-neighbor did not yet receive the message, so they can forward to the next k nodes), we get the following recursive equations describing the number of nodes reached in stage i:

$$
\begin{aligned}
e(i) &= 2\cdot e(i-1)+o(i-1)\\
o(i) &= k\cdot e(i-1)+o(i-1)
\end{aligned}
$$

As $o(i)$ is the dominant term, we can say that the number of nodes reached in i stages is at least $k^{i/2}$, leading to an upper bound of $2\log_k N=\frac{2}{\lg k}\lg N$ on the number of stages. For $k>4$ this is smaller than $\lg N$ by a multiplicative constant, due to the k-ary branching factor. The cost of each stage is the same as above.

3.2 Algorithm Comparisons

The first algorithm is linear and costly, as it uses multiple configuration changes. Its only redeeming aspect is that it does not depend on the existence of a forwarding unit. The second is also linear, but uses forwarding rather than configuration changes. Comparing their runtimes leads to the inequality $(N-1)(T_c+ST_b)<T_c+(N-1)ST_f+2ST_d$. simplifying leads to $(N-2)T_c+(N-3)ST_d<0$, which is impossible unless $N<3$. Thus the runtime of the second algorithm (forwarding) is always lower.

The third algorithm introduces a tree structure in order to achieve a logarithmic cost. Comparing the second and third algorithms as above, the requirement for the second to be better is $T_c+(N-1)ST_f+2ST_d<(\lg N)(ST_f+T_d+T_c)+2ST_d$. This essentially compares a linear running time with a small constant with a logarithmic time that has a larger constant. Omitting the T_c term on the left, the T_d on the right, and using the assumption that $T_c\approx 10ST_f$, we can reduce the expression to $\frac{N-1}{\lg N}<11$, which holds for $N<68$. For larger N, the logarithmic algorithm wins.

The last algorithm introduces the idea of overlapping the configuration change with The forwarding to nearby nodes. This reduces the logarithmic number of stages by a small constant. The expression becomes (under the same assumptions) $\frac{N-1}{\lg N-3.3}<12$, which holds for only $N<10$. Based on figure 3 we expect the number of stages in the second version to be even lower. This is not seen from the equations because we just presented a crude upper bound on the number of stages, and not an exact count. However, this is indicative of the complexity that would be involved in a detailed implementation of this last improvement.

3.3 Sensitivity to T_c

A broadcast operation running time is mainly affected by the number and timing of configuration sets (noted by T_c in our model). In the current prototype T_c is high, motivating the design of algorithms that avoid configuration changes as much as possible. But with proper investment it seems that T_c can be reduced significantly. We therefore need to evaluate the sensitivity of our results to T_c.

The results in previous sections have expressed the time to perform a broadcast as a function of the model parameters, including T_c. Let us now assume a hypothetical optimal case in which $T_c=0$, i.e. configuration changes are for free. If we plug this value into the equations, we get an optimal value for the time of a broadcast, which we shall denote T^{opt}. But in a real system T_c is not zero, and the time is therefore higher than T^{opt}. We can solve the equations for T_c, and find the value of T_c for which the running time is twice the optimal. Denote this value by $T_c^{0.5}$. The significance of this critical value is that if the actual value of T_c is smaller than $T_c^{0.5}$, then we are assured that the performance of the broadcast is within a factor of 2 of the

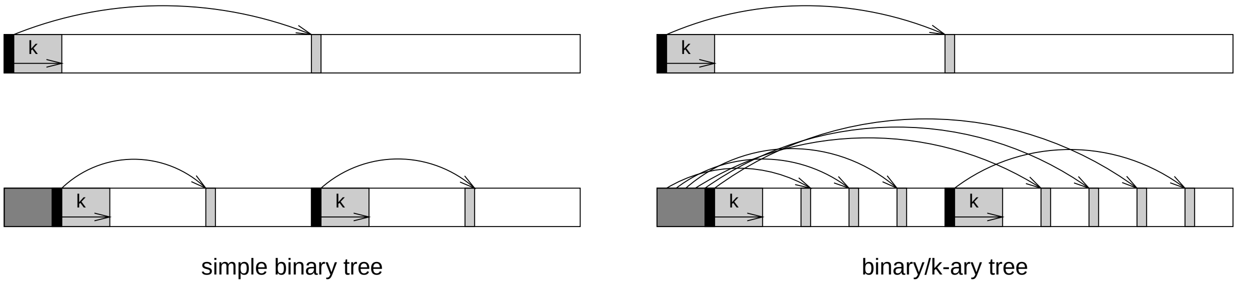

Figure 3: First two stages of forwarding in a tree embedded in a ring: black denotes end nodes, gray is nodes that received the message in a previous round, and light gray are the nodes that receive in the current round.

best possible (for the hypothetical case of no overhead for configuration changes).

Consider the algorithm for forwarding in a tree as described in Section 3.1.3. For this algorithm, $T = (\lg N)(ST_f + T_d + T_c) + 2ST_d$. It follows that $T^{opt} = (\lg N)(2ST_f + T_d) + 2ST_d$. The equation for $T_c^{0.5}$ is then

$$\begin{aligned}(\lg N)(ST_f + T_d + T_c^{0.5}) + 2ST_d = \\ 2(\lg N)(ST_f + T_d) + 4ST_d\end{aligned}$$

Solving leads to

$$T_c^{0.5} = \frac{2(\lg N)(ST_f + T_d) + 2ST_d}{\lg N} - ST_f - T_d$$

which can be simplified to yield

$$T_c^{0.5} = ST_f + T_d + \frac{2ST_d}{\lg N}$$

Using our common assumptions that $S \approx K$, $T_d \approx 10T_f$, and $N \approx K$ leads to $T_c^{0.5} \approx 3000T_f$. Our estimates of T_c and T_f for the prototype indicate that the current T_c is about 3 times higher than this critical value. Thus we conclude that even with the current value of T_c the performance of our algorithm is within a small factor of the optimal. Reducing T_c will at best improve the performance by a factor of 6.

3.4 All-to-All Broadcasts

An immediate extension of one broadcast is N broadcasts from different nodes at the same time. Of course, this can be implemented as N consecutive broadcasts. However, it is more efficient to overlap them, possibly using pipelining.

Two known solutions for the all-to-all broadcast operation are hypercube dimension based and tree based. The first is based on a hypercube pattern. In stage i, each node sends all the messages it received so far to its hypercube neighbor along dimension i. Therefore, in every stage each node sends twice the number of messages it did in the previous stage. The process takes $O(\lg N)$ stages, but the number of messages sent by each node is still $O(N)$. There are also $O(\lg N)$ configuration changes.

The second is based on building a full (binary) tree. Initially each node propagates its message and those received by it to the root of this tree. Then the root broadcasts all the messages to its direct children, and recursively to all the nodes in the system. Gathering the information to the root takes $O(\lg N)$. Scattering the information back to all the nodes also takes $O(\lg N)$. As in the previous algorithm, when the messages are propagated down the tree in the second phase, every node sends $O(N)$ messages. This algorithm also needs one configuration change to build up the tree for the information gathering and additional ones for the broadcast tree.

The algorithm that we suggest to be optimal is surprisingly based on a ring topology. In the first stage each node sends its message to the next node. In every other stage, each node forwards the message it received in the previous stage to the next node, and in parallel passes it up to the local communication stack. This process finishes after $N-1$ stages, in which each node sends one message. The whole process takes $O(N)$ messages that are sent by each node and one configuration change (or more precisely, $T_c + (N-1)ST_f + 2ST_d$.

The reason that this version is optimal in our model is the need to take both message sends and configuration changes into account. But when accounting for the cost to send a message, we have the option of using forwarding rather than an explicit send from the application level. Forwarding avoids the extra overhead of starting a new message, which dominates most communication models. Therefore the common tradeoff that favors a smaller number of messages (only $O(\log N)$) that may each be bigger ($O(N)$) does not hold. By definition of the all-to-all broadcast operation each node must receive $(N-1)S$ bytes of data. In the ring algorithm, this is multiplied by the smallest possible cost factor which is T_f, and added to the smallest possible terms: one configuration change and one pass through the software stack at each end. It is therefore optimal.

4 Algorithms for a General Point-to-Point Communication Model

Parallel applications often use patterns of collective communication, such as broadcast. But sometimes they use patterns composed of multiple pairwise communications, as in FFT or the stencils of image processing and CFD.

And arbitrary communications are also possible, especially in input-dependent and irregular computations. A *point to point (P2P)* communication model is defined to be a model in which there is no knowledge of any global operations. Each node operates on its own, sending messages to other nodes. This can be modeled as sending to destinations chosen from a certain distribution.

Given this model of application behavior it is clear that no switch configuration will match all the requests. The simple way to support such a scenario in our model is to perform a configuration change for each message, setting the switch to the desired destination. However configuration changes are a costly operation, and moreover, it is difficult to ensure that messages do not clash (that is, that two nodes don't happen to send to the same destination at once). We therefore use an alternative approach, in which a certain network that spans all the nodes is created, and messages are forwarded along this network.

In the following sections we check two candidate topologies: cube-connected cycles and chordal rings, both of which have a low constant degree. This is implemented using the switch limited multicast capability, and then selecting the intended recipient. The performance of arbitrary communications using these networks is evaluated using simulations, with destinations selected at random either uniformly or with hot spots. The performance metric is the average time to transmit a message.

4.1 Evaluation Methodology

Time is divided into discrete periods called rounds. In each round nodes can send messages, receive them, or change the switch configuration. As a configuration change takes longer than sending a message, this implies that multiple messages can be sent in the same round, which is useful anyway as explained below in Section 4.2.3. We assume transmit and receive operations can be overlapped. In each round the simulator processes the existing operations and adds new random operations, according to a given distribution.

A send operation consists of a message that needs to be sent from a given source to a given destination. The processing of a send operation can result in either of two situations: the message is propagated one hop towards the destination, or no action is taken because it is blocked or conflicts with the model's policy. The average running time of a send operation is measured by the average number of rounds (hops and delays) it takes to finish a send operation.

The source and destination of a new send operation are chosen randomly, with the only restriction being that they be different nodes. For uniform traffic, both are chosen at uniform from all nodes. For hot-spot traffic, node 0 is selected to be the hot spot (this is easily extended to a group of hot nodes, but having only one amplifies its effect). A hot spot with parameter p means that there is an "extra" probability of p to select the hot spot as the destination. In other words, the destination is chosen uniformly from *all* nodes (including the hot spot) with probability $1-p$, and the hot spot is chosen with probability p. We used three different parameter values: 1%, 3%, and 5%.

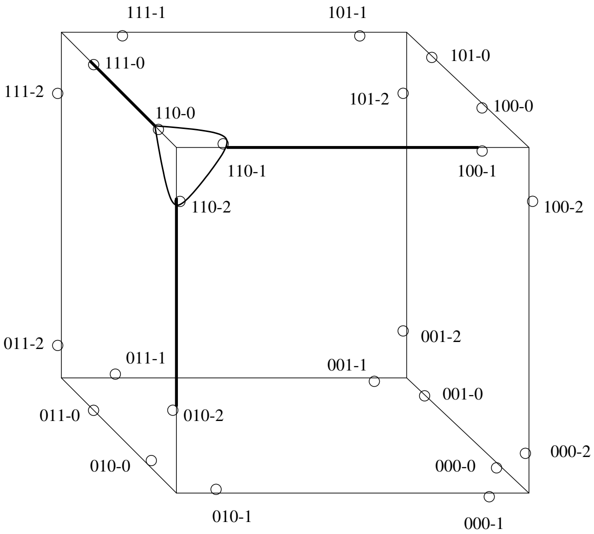

Figure 4: A CCC topology with D=3.

The simulation is run until both the number of requests exceeds 10000 and the number of rounds exceed 1000, after deleting a hundred initial requests in order to avoid cold-start effects. Likewise, messages that are in transit at the end of the simulation are not counted.

4.2 The Cube Connected Cycles Model

4.2.1 The CCC Topology

The Cube Connected cycles (CCC) topology consists of a regular hypercube of dimension D with each node replaced by a ring of D nodes. The degree of each node in a CCC is three. An example of the CCC topology with $D = 3$ is shown in Figure 4. One ring is shown in bold, with the nodes 110-0, 110-1, and 110-2. Their neighbors are also connected with a bold line.

Two out of the three neighbors of any node are on the same ring, and will be called *inner* nodes. The third neighbor is on another ring and will be called an *outer* node. The nodes are numbered by an ordered pair (HN,RN) where HN is the hypercube node number (using the conventional numbering scheme) and RN is their node number in the ring, corresponding to the hypercube dimension along which it is connected to its outer neighbor. The routing scheme in a CCC topology is based on the conventional routing in a hypercube, and is broken into 3 steps [8]:

1. Routing within the ring of the source node s to node a (for example).
2. Routing from node a to a node in the destination node's ring (say node b) via nodes in intermediate rings.
3. Routing within the destination ring from node b to the destination node d.

4.2.2 The Collision Problem

To implement the CCC topology in our model, the switch is statically configured such that a message from any node will reach the node's three neighbors (using the limited multicast feature). However, only one of the neighbors should actually process it, while the other two discard it (based on an inspection of the address). The problem is that if two nodes on the same ring transmit at the same time, even if the intended recipients are distinct, a collision will occur. This will cause all data to be garbled.

To avoid collisions, we need a synchronization method, that will ensure that all nodes will only receive one message at a time. One option is using a central coordinator that will grant permissions for each transmission *on demand*. Each node asks the switch controller for permission to send a message on its outgoing link, starts sending only when it gets an approval, and notifies the controller when it is finished. The controller grants permission only if there are no collisions with other messages being sent to any of the nodes in the sender's multicast group. This is expected to be efficient for low loads, but can become a bottleneck for highly loaded systems.

An alternative is to use a globally synchronized *cycled* approach, reminiscent of a traffic light. A designated controller sends pulse messages on the control network at regular intervals (on the order of several milliseconds). These messages specify which dimension is active in the current cycle, and whether this pertains to inner or outer messages. Each node listens for these messages from the controller, and determines whether its message matches the specification for the current round. If it does, the message is sent. Otherwise it is delayed. Note that this also specifies when a node may expect to receive a message, so unexpected messages can be dropped without inspection.

The simplest way to set the specifications is to cycle through all the possibilities. For example, on a 24-node system that has 3 dimensions the pulse messages will be: (0,inner), (1,inner), (2,inner), (0,outer), (1,outer), (2,outer), and so on. However, this means that nodes are unable to send a large fraction of the time: they only have neighbors in 3 of the $2D$ rounds in a cycle. It is therefore desirable to allow non-interfering sends to proceed in parallel.

In D dimensions, we can actually do all the inner sends in $3+D \bmod 3$ rounds. For example with $D = 6$ we can allow send on dimensions 0 and 3 in the first round, on 1 and 4 in the second round, and on 2 and 5 in the third round. Extra rounds are needed if the dimension is not evenly divided by 3. With outer sends there is no problem for every second node to send/receive data as long as there is one node that is separating between them which knows that the data it receives is garbage and should be disregarded in this round. Hence, the outer sends will be divided into 2 stages. In the first stage sends are legal on even dimensions and in the other stage sends are legal on odd dimensions. For simplicity, we will restrain ourselves to even D from here on.

With these two optimizations, the length of the round robin cycle is fixed. It is decreased from $2D$ to not more than 7 rounds (or only 5 if the dimension is divisible by 3).

4.2.3 The Tick Time

When several messages are queued to be sent from node i to node j, it would be preferable to actually send them all in one round. However, it is not possible to send too many messages in the same round since the length of a round is fixed by the pulse messages. One reason for not making rounds too long are that this will lead to nodes using only a small portion of the round on average, and staying idle the rest of the time. Moreover, messages will be blocked for a longer time until their round comes up. From the simulations we have concluded that the optimal time should allow about 20–40 messages to be sent in the same round. This number will be denoted the MMIR (maximal messages in round).

4.3 Evaluation of CCC

4.3.1 Preliminaries

The diameter of the CCC is $2D + \lfloor \frac{D}{2} \rfloor - 2$ [8]. One D is needed for doing the hops between rings for the D dimensions. Another D is for inner moves from one dimension to the next inside rings. Another $D/2$ is needed for reaching the destination node on the last ring. However, the simulations use a slightly less efficient method in which the dimensions are traversed in order from 0 to D. This requires an additional $D/2$ hops in the origin ring to reach dimension 0, for a total diameter of $3D$.

Using on-demand access with no collisions a request will therefore take exactly $3D$ rounds. With collisions this can rise significantly, as we will see in the simulations. Using the cycled synchronization a request will suffer from collisions only if the MMIR is exceeded, which is expected to happen only under extreme loads. However, due to the synchronization policy, the message may have to wait in intermediate nodes. If we restrict ourselves to even dimensions, and on average $D \bmod 3 = 1$, then there are 2 outer rounds and 4 inner rounds in each cycle. The expected number of rounds a node has to wait is thus 3, leading to an expected time for communication of $3 \times 3D = 9D$ rounds.

On one hand, it is obvious that a system with a very light load should use the on-demand version. On the other hand, the cycled version is more robust in the face of high load. The purpose of our simulation is to find the point where it stops to be profitable to use one method and it starts to be better to use the other method. Note that an adaptive algorithm, that can dynamically switch between these two versions, can yield even better results.

4.3.2 Simulation Results

The simulations were performed for CCC of dimensions 4, 6, and 8, with 64, 384, and 2048 nodes respectively. The maximum number of new events that are spawned each

dim	cycled	on-demand
4	21	10–21
6	28	24–47
8	58	45–82

Table 1: Average number of rounds needed to finish a random send operation.

round is called the *spawn value*. Spawn values were checked in the range from 10% of the number of nodes to 100% of the number of nodes. In each round the simulator creates a random number of new send operations, which is chosen uniformly from 1 to the spawn value. Hence, a spawn value of X% means that on average there will be $\frac{XN}{200}$ new events. We use this absolute count because translating the spawn value to a fraction of the network's capacity depends on the average route length and the synchronization policy. Therefore an absolute count is better in order to compare the alternatives, and expose the capacity differences. A spawn value of 100% means that on average every node generates a new message every other round. The MMIR is assumed to be 35 unless noted otherwise.

Table 1 shows the average number of rounds needed to finish a random event using the cycled and the on-demand versions. The cycled version is almost not affected by the load so it has only one result in it's column. The on-demand version has a range of results starting from very light load on the system and ending with a high load corresponding to a 100% spawn value. As expected, with light load the on-demand version is better then the cycled version. With a very high load on the system the situation is reversed, especially for the larger system sizes. However, the details do not fully fit our expectations. The on-demand algorithm was expected to require $3D$ rounds for low load. The gaps between the formula and the simulation results are -2, 6, and 21 for dimensions 4, 6, and 8 respectively. The gap for $D = 8$ is especially large and remains unexplained.

In the following graphs the X axis represents the spawn value and the Y axis represents the average number of rounds needed to complete a random event. The legend of each curve in a graph describes the test that produced it, and consists of the algorithm (cycled or on-demand), destination distribution (hot spots) if non-uniform, MMIR value, and the dimension. The curves do not tend asymptotically to infinity as the load increases due to the use of a finite number of messages in the simulations. However, the flexing point at which they start to rise sharply gives a good indication of the system capacity, beyond which it starts to saturate.

The results of simulations showing the effect of the MMIR on cycled performance are shown in Figure 5 (the MMIR has no effect with on-demand synchronization). Reasonable values for the MMIR must be in the range of tens of messages per round. With $D = 4$ we can see the performance degrade when the spawn value is above 70% with MMIR=10. With $D = 6$ the same effect happens at 50%

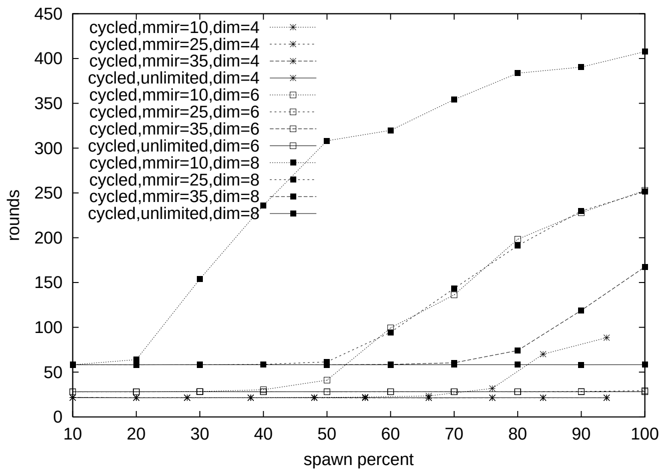

Figure 5: The effect of different MMIR values and cycled synchronization.

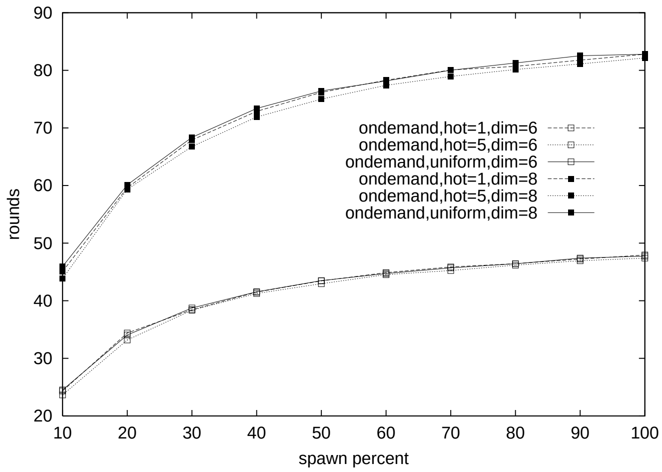

Figure 6: Hot spots have no effect on the on-demand version, or when the MMIR is unlimited.

spawn value, while for MMIR=25 or 35 there is no degradation in performance. With $D = 8$ the effect is even stronger: for MMIR=10 the degradation starts after a mere 20% spawn value, MMIR=25 after 50%, and MMIR=35 after 80%. Thus a value of 35 seems to be the value that relatively to it's size causes the smallest degradation in performance.

Hot spots alone don't have any effect on the performance of both versions, as seen in Figure 6 for on-demand synchronization (cycled has a similar graph, except that the lines are straight). However, the combination of hot spots and a limited MMIR has very dramatic effect on the cycled version (Figure 7). This happens because the hot spots cause a lot of traffic to converge on specific nodes. A limited MMIR then causes these messages to pile up in the network, because they cannot be moved fast enough. Comparing with Figure 5 we see that a hot spot of 5% with MMIR=30 is worse than an MMIR of 10 with uniform traffic. Similar effects are seen for the other dimensions too.

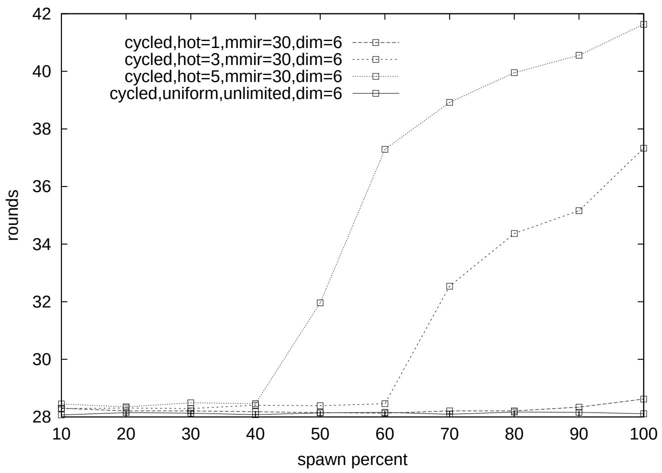

Figure 7: Effect of hot spots with MMIR=30 in the cycled version, with $D = 6$.

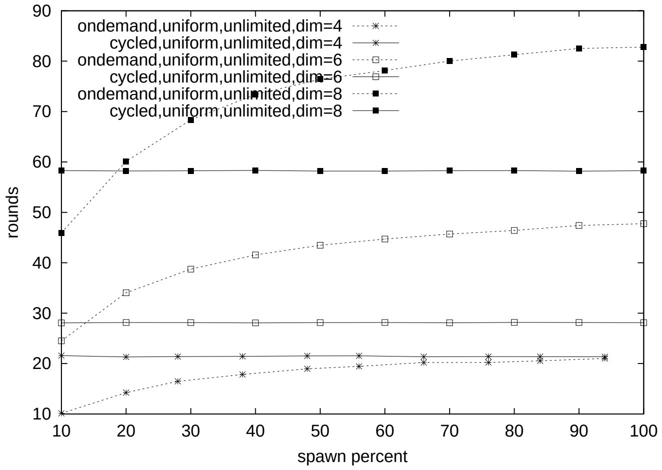

Figure 8: Comparing on-demand and cycled synchronization.

The on-demand version does not suffer as much, because it is more flexible in re-allocating bandwidth.

Figure 8 compares the two synchronization methods, and shows that on-demand is generally better then the cycled version only for $D = 4$. In higher dimensions the cycled version is better, except for very low loads. These results are true with hot spots as well (not shown).

To summarize, both the cycled and the on-demand versions are quite efficient for a general P2P communication model, and generally route messages in a logarithmic number of rounds. The cycled version works better than the on-demand one in large systems (from $D = 6$), but it depends on the MMIR value being high enough (not under 30 or so). Even so, it is vulnerable to hot spots.

4.4 Chordal Ring Model

Regular networks of low degree are easy to implement physically. Rings are an example for such networks. The main drawback of rings is the diameter which is too large for

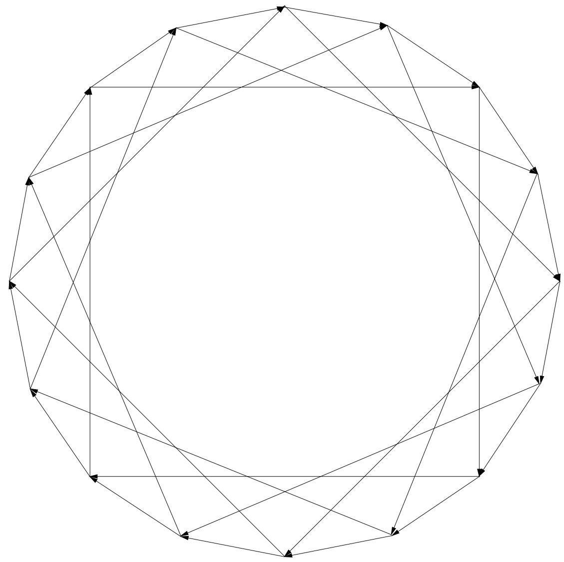

Figure 9: A chordal ring with a fixed assignment.

more than several nodes. This can be overcome by using a *chordal ring*, which also contains internal "shortcut" arcs between nodes [2]. The regular arcs are called *ring arcs* while the internal arcs are called *chordal arcs*. In terms of our model, the implementation will use limited multicast as for the CCC, and maybe also configuration changes to adjust the chordal arcs.

4.4.1 Topology and Routing

The topology is based on a uni-directional ring with each node having one additional chordal arc. The switch is configured such that each node sends messages to two other nodes using limited multicast: the next ring node, and the node at the end of the chordal arc. Likewise, each node can only have two incoming links, from the previous ring node and from a chordal arc. An example of a chordal ring with fixed assignments is shown in Figure 9. In this figure $N = 16$ and for each node i there is a chordal arc to node $i + 4 \bmod N$. This reduces the diameter from 15 to 6.

The ring arcs are statically set such that node i is connected to node $(i + 1) \bmod N$. For the chordal arcs, there are two main options:

- Use a *fixed* assignment, where all the chordal arcs have the same size. Thus for every i node i connects to node $(i + w) \bmod N$ for some w.
- Use a *random* assignment in which each node is connected to some other node at random.

For a fixed configuration the routing algorithm is straightforward. First, use the chordal arcs to jump to the correct segment of the destination node. Since the ring is unidirectional the last hop must still be before the destination. Second, advance on the ring nodes until the destination is reached.

In a random configuration the routing is more complex. We assume each node knows the configuration (this assumption is valid even if we allow reconfiguration of chordal arcs, as configuration change requests are broadcast on the control network). Each node can therefore create (and maintain) a shortest path routing table to all the other nodes in the system. When performing a send operation or forwarding a message, the node sends the data on the first hop of this shortest path.

4.4.2 Avoiding Collisions

In order to ensure that no node will ever receive two messages at the same time from two different nodes we have restricted the chordal connections. We will use chordal links that are only connected to nodes with the same parity. This means that even nodes can *only* be connected to even nodes and odd nodes can *only* connect to odd nodes (where nodes are numbered according to their order in the ring).

Given this restriction, the synchronization is done based on a cycle of two rounds. Each round is valid for a certain period of time, as was done in the CCC model, and can be used to transfer a maximum of MMIR messages. In the first round in the cycle only even nodes are allowed to send data, and in the other round only odd nodes are allowed to send. To see why this works, lets examine an even node with the number of $2i$. In the round where even nodes can send data it can only get messages from the chordal arc, which connects to a node that also has an even number. The previous node, $2i-1$, cannot send it data in this round, since it has an odd number. In the other round, where only odd nodes can send data, the situation is reversed.

The fixed and random topologies described above must abide by this restriction, that is, they must only use chordal arcs that span an even number of nodes.

4.5 Evaluation of Chordal Ring

The parameters for the simulation are the same as those used to evaluate the CCC. The number of nodes is 64 , 384, or 2048. The spawn values are also the same as for the CCC. Unless stated otherwise the spawn value used is 100%, and the MMIR is 35. In a fixed configuration the arc length is $w = \sqrt{N}$, which, under appropriate definitions, is optimal [7].

4.5.1 Preliminaries

In the worst case the distance between any two nodes in a fixed configuration is limited to $2\sqrt{N}$ hops for a system with N nodes. The first $\sqrt{N}$ hops are needed to reach the correct segment on the ring. In the worst case, the destination node would require a full circling of the ring which can be done with $\sqrt{N}$ hops on chordal arcs. Then, in the worst case another $\sqrt{N}$ hops are required to reach the destination node, if it is the last node on this segment. The average distance between two arbitrary nodes is $\sqrt{N}$.

When using a random configuration or a dynamic configuration it is difficult to estimate the distance between two random nodes. We have used the simulation results to measure the average case figure.

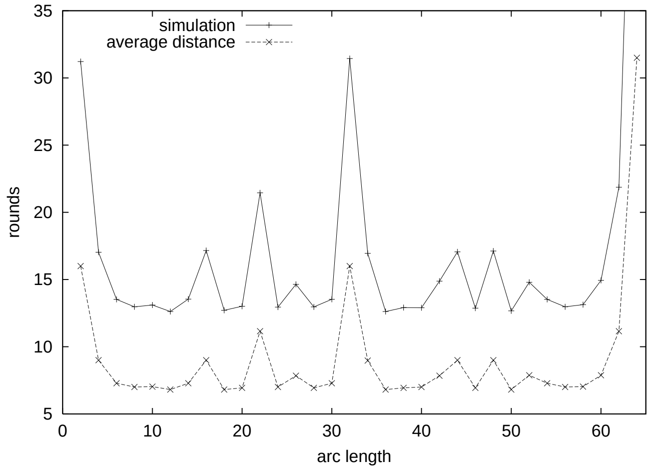

Figure 10: Average number of hops for fixed configuration.

4.5.2 Simulation Results

In order to compare various options, we must first decide on the arc length to use in the fixed configuration. Results of simulations using different lengths are shown in Figure 10. The dashed line represents the average number of hops between any source and destination. This value was calculated in the following manner: First, the arcs lengths were set to the current value. Second, an all-to-all shortest path algorithm such as Floyd was employed on all the nodes, and the average path length calculated. The solid line is the result of simulations with a spawn value of 100%. As expected, this leads to higher values, as messages may be delayed along the way. Remarkably, the shapes of the two graphs are the same, indicating that there is a strong correlation between the average path length and the average performance under load. In addition, the two graphs are symmetrical, indicating that the performance for arcs of length $w = i$ is the same as for $w = N - i$. This can be explained using a simple example. Let's say we examine $w_0 = \frac{N}{4}$ or $w_1 = \frac{3N}{4}$, which satisfy $N = w_0 + w_1$, and try to find the shortest paths from node 0 to nodes $\frac{N}{4}$ and $\frac{3N}{4}$. With w_0 it will take 1 hop to reach the first of these and 3 hops to reach the second. With w_1 it would take 3 hops to reach the first and 1 hop to reach the second. Such arguments can be generalized to show that indeed using complimentary arc lengths leads to symmetrical situations regarding pairs of nodes.

According to Mans [7] the optimal number for w is an integer close to $\sqrt{N}$. From our simulation $\sqrt{N}$ was discovered to be one of the optimal values for w but not he only one, the others being odd multiples of it. Even multiples are clearly not close to optimal in our case, because the number of nodes we consider has an even square root. Therefore using chordal arcs leads to the same nodes being visited after a full circle, while skipping over nodes that are an odd multiple of $\sqrt{N}$. Using an odd multiple therefore gives better coverage of the ring.

An alternative to the fixed assignment is to use a random assignment of the chordal arcs. We have found that

nodes	fixed		random	
	paths	rounds	paths	rounds
64	6	12	4	7
384	18	35	7	10
2048	42	84	9	14

Table 2: Comparison of fixed and random chordal arc configurations.

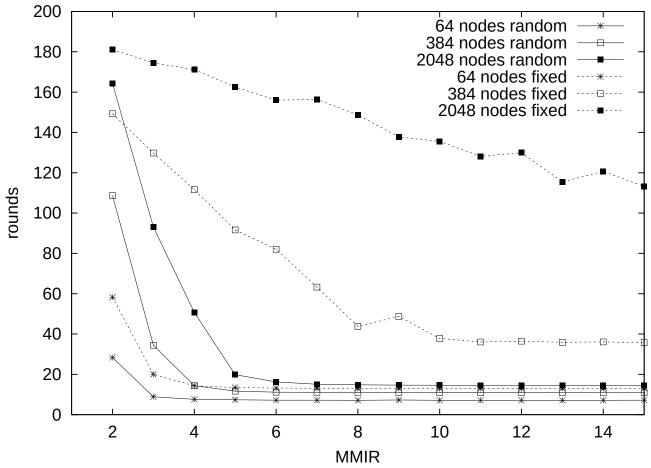

Figure 11: Average rounds needed to complete communication as a function of the MMIR, with a random configuration.

a random assignment of the chordal arcs outperforms the fixed assignment, as shown in Table 2. For each approach (fixed or random) there are two columns, giving the average path length (as calculated above) and the average number of rounds needed to route a message under 100% spawn rate. Clearly the random configuration yields better results than the fixed configuration in both cases.

The superiority of the random version stems from the random mix of short and long chordal arcs, leading to a diameter that is more-or-less logarithmic. This provides a distinct advantage over the fixed assignment where the diameter is always $\sqrt{N}$.

Once the topological options have been chosen, we can investigate the impact of various parameters. Contrary to the CCC model, the effect of changing the MMIR in the chordal ring topology is much smaller, at least for random arc assignments. Simulations show that if the MMIR is not limited than the largest number of messages sent in a single round is usually around 10. Thus we found that using MMIR=20 insures that we will not suffer almost any slowdown due to blocked messages. Nevertheless, in other simulations we will still use an MMIR of 35 to allow comparisons with the CCC model. Also, as shown in Figure 11, when using a fixed assignement larger MMIR values are needed.

Hot spots also have little effect on the randomly configured chordal ring, at least for the low values considered so far. For large values, the degradation in performance is linear with respect to the hot spots percentage. The explanation is that this degradation is caused by messages that are blocked due to exceeding the MMIR value.

To summarize, the chordal ring approach seems to achieve relatively good results. When the chordal arcs are configured at random, the average number of rounds for transmitting a message is proportional to the logarithm of the number of nodes. This is resilient to relatively low values of the MMIR, and relatively high values of hot spots.

4.6 Comparison between CCC and Chordal Ring

Comparing the results contained in Tables 1 and 2, it is clearly evident that the chordal ring model yields better results than the CCC model.

There are several reasons for the advantage of the chordal ring model:

- In the chordal ring the limited multicast is done to two nodes and in the CCC model the limited multicast is done to three nodes. The overhead to synchronize three nodes is much larger than the overhead to synchronize two nodes.
- The random topology in the chordal ring model is more robust, and can better support different kinds of patterns. The static topology of the CCC is less flexible in this sense.

In spite of all these facts, there are also some advantages to the CCC model:

- The routing in CCC is straightforward. In the randomized chordal ring each node must keep the whole topology in a routing table.
- Our discussion was restricted to random communication patterns. However, people that write programs for real systems often exploit the locality between adjacent nodes, basing their process assignment on a well structured topology. In this sense the CCC model has a huge advantage over the chordal ring model, which uses a random assignment.

4.7 Adding Configuration Changes

Until now we have dealt with models which use limited multicast and forwarding. This model has proved itself to be efficient. In this section we consider the addition of configuration changes, and check whether this additional operation can provide any additional gains. The results quoted below indicate that this is not the case, and configuration changes just degrade performance for this model.

The main idea behind dynamically changing the configuration is to adapt the configuration to a certain load on the network. When the load on the network is uniformly distributed between all the nodes, and the system uses a random configuration, it can be assumed that there is not much to gain by changing the current configuration. There could be some gain if the random assignment is accidentally very biased, but this scenario has a very low probability to occur. A real contribution to the network performance can be achieved when the model uses some biased distribution like the hot spots distribution. With hot spots, an adaptive algorithm can detect heavy loads on certain links and try to dynamically reconfigure itself to handle the load better.

4.7.1 Algorithms

We start with some notation:

- $i \to j$ notes that node i sends a message to node j.
- $i \Rightarrow j$ notes that the switch is configured so that node i is connected to node j.
- *strength(i,j)* is the load on the link from i to j. Each message that is sent from i to j increments $strength(k,l)$ by one for all nodes k and l along the path from i to j. Every time one of the strength values reaches a large constant value (we used 100 in the simulations) all the strength values for all the pairs are divided by two. This normalizes the values and ensures that the effect of load surges is decayed.

All the algorithms are based on the same idea but differ in its implementation. Let's assume that in the current configuration a new event is added which includes sending a message $i \to j$. Let's also assume that the current configuration is $i \Rightarrow l$ and $k \Rightarrow j$. The different algorithms consider different options for re-configuration:

1. If $strength(k,j) < C$ for some constant C then it means that not too many messages travel on $k \Rightarrow j$. Then we can afford to break this link and make a direct connection $i \Rightarrow j$ (naturally this can only be done if they have the same parity).
2. A more global approach is to build a shortest-path matrix for all pairs of nodes, and compute a weighted average using $strength(i,j)$ as the weight. Now tentatively change the configuration such that $i \Rightarrow j$, and repeat the calculation. If the new average path length is lower, accept the change.
3. With a hotspots distribution it may not be desirable to connect directly to the destination, as many nodes may conflict with ech other as they all try to do so. To cope with this situation, the previous version was changed to try and connect to the two nodes before j (along the ring and chordal arcs).

We also checked the possibility of using a new random configuration, if it reduced the average weighted path length.

4.7.2 Results

All the versions with dynamic re-configuration performed worse than the static configuration. We do not present the exact results since none of the versions suggested showed any sign of performing well, even on a fraction of the cases. The best we could achieve was to get results which are no worse than the static version on some distinct cases. We conjecture that the reasons for this failure are:

- Any configuration change is harmful in the short term because it may degrade the routing of messages that are in transit: They may need to be re-routed because a link in the original route has been removed.
- Under our assumptions of hot spot distribution there is nothing the network has to adapt to. A dynamic configuration is needed where the load changes. With this in mind we checked the effect of moving hot spots, but found that configuration changes were not useful in this case either.

It seems that under the current model there cannot be a better solution than the static model with random chordal arcs.

5 Conclusions and Future Work

This work is based on a novel optical switching technology. This technology can be a basis for the next generation networks for parallel computers. However, it is quite different from the packet switching networks used today. Therefore there is a need to re-design the communication software, taking the circuit switching nature of the network and the relatively high switch configuration time into account. We have started this work be designing and evaluating algorithms for the collective broadcast operation, and for arbitrary point-to-point communications.

In our model of the switch there are three main elements: forwarding, multicast, and configuration changes. The design of efficient algorithms necessitates a balanced use of these elements, respecting the tradeoffs involved in their use. Specifically, our study of broadcast algorithms exposed the tradeoff between configuration changes and forwarding. For point-to-point communications, we found the combined use of forwarding and limited multicast to be most effective. This was used to create a low-degree network that spans all the nodes in the system, and can be used to route messages.

Future works includes extending the broadcast study to other collective operations such as gather-scatter, global exchange, etc. On the practical side, it is desirable to implement the suggested algorithms on the prototype — when a larger one is completed — fine tune them, and measure their actual performance. This will complement the results presented here, by providing a data point that directly corresponds to the prototype's values of the model parameters.

Acknowledgements

This research was supported by the Ministry of Science and by the Wolfson Foundation for Scientific Research.

References

[1] A. Agranat, V. Leyva, and A. Yariv, "*Voltage-controlled photorefractive effect in paraelectric $KTa_{1-x}Nb_xO_3$:Cu,V*". *Opt. Lett.* **14(18)**, pp. 1017–1019, Sep 1989.

[2] B. W. Arden and H. Lee, "*Analysis of chordal ring network*". *IEEE Trans. Comput.* **C-30(4)**, pp. 291–295, Apr 1981.

[3] A. Barak and E. Schenfeld, "*Embedding classical communication topologies in the scalable OPAM architecture*". *IEEE Trans. Parallel & Distributed Syst.* **7(9)**, pp. 979–992, Sep 1996.

[4] I. Elhanany, J. Nir, and D. Sadot, "*A contention-free packet scheduling scheme for provision of quality-of-service in Tbit/sec WDM networks*". *Optical Networks* **1(3)**, pp. 1–6, Jul 2000.

[5] P. Green, "*Optical networking update*". *IEEE J. Selected Areas Commun.* **14(5)**, pp. 764–779, 1996.

[6] A. Jourdan, F. Masetti, M. Garnot, G. Soulage, and M. Sotom, "*Design and implementation of a fully reconfigurable all-optical crossconnect for high capacity multiwavelength transport networks*". *J. Lightwave Technology* **14(6)**, pp. 1198–1206, Jun 1996.

[7] B. Mans, "*On the interval routing of chordal rings*". In *Int'l Symp. Parallel Architectures, Algorithms & Networks*, Zomaya, Hsu, Ibarra, Horiguchi, Nassimi, and Palis (eds.), pp. 16–21, IEEE, Jun 1999.

[8] D. S. Meliksetian and C. Y. R. Chen, "*Optimal routing algorithm and the diameter of the cube-connected cycles*". *IEEE Trans. Parallel & Distributed Syst.* **4(10)**, pp. 1172–1178, Oct 1993.

[9] A. G. Nowatzyk and P. R. Prucnal, "*Are crossbars really dead? the case for optical multiprocessor interconnect systems*". In 22nd *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 106–115, Jun 1995.

[10] B. Pesach, G. Bartal, E. Refaeli, A. J. Agranat, J. Krupnik, and D. Sadot, "*Free-space optical cross-connect switch by use of electroholography*". *Applied Optics* **39(5)**, pp. 746–758, Feb 2000.

[11] R. Sabella, E. Iannone, and E. Pagano, "*Optical transport networks employing all-optical wavelength conversion: limits and features*". *IEEE J. Selected Areas Commun.* **14(5)**, pp. 968–978, Jun 1996.

[12] B. Webb and A. Louri, "*A class of highly scalable optical crossbar-connected interconnection networks (SOCNs) for parallel computing systems*". *IEEE Trans. Parallel & Distributed Syst.* **11(5)**, pp. 444–458, May 2000.

[13] T. Yoshikawa and H. Matsuoka, "*Optical interconnections for parallel and distributed computing*". *Proc. IEEE* **88(6)**, pp. 849–855, Jun 2000.