Communication Models for a Free Space Optical Cross-Connect Switch

October 18, 2000

A thesis submitted in partial fulfillment

of the requirements for the degree of Master of Science

By

David Er-el

Supervised by

Dr. Dror G. Feitelson

August, 2000

School of Computer Science and Engineering

The Hebrew University of Jerusalem

Jerusalem 91904, Israel

Acknowledgements

First, I would like to thank my supervisior Dr. Dror G. Feitelson for his personal guiding, for his patience all along the way, for all the hours we have spent together and for all the usefull comments he has raised during this work.

Second, I would like to thank all the other students in the parallel lab who I have consulted with, regarding various theoretical parts, and various programming/system parts: Dan Tsafrir, Yoav Etsion (etsman), Avi Kavas, Daniel Citron, Uri Lublin and Alex Kremer(Kreso).

Last but not least, I would like to thank my wife Racheli for her support during the 3 years of the masters degree.

Contents

1 Introduction

1.1 Parallel computing

A parallel system is a system which utilizes multiple computing units concurrently to solve a computational problem with each computing unit working on a part of the problem. In order to solve these problems the computing units are engaged in doing calculations locally and in exchanging data with other computing units. In the past, parallel systems used to be built using a proprietary hardware. This hardware was responsible for the communication between the computing units and for the computation itself. The communication was based on a shared memory model or on a distributed memory model. In the shared memory model computing units share a common address space. In the distributed memory model computing units operate in a disjoint address spaces and exchange messages to interact with one another. Today, however there is a tendency to build many parallel systems with standard building blocks. This is done by using a regular PC as the computing unit and using a LAN (local area network) for the communication. This model is also referred to as NOW (network of workstations). The strength of a parallel system is measured as a combination of the strength of the computing units and the strength of the network.

1.2 Interconnecting network

It is trivial that the overall performance of a parallel system will be poor if the computing units are not strong enough. It is also agreed that the performance of the interconnecting network has a very large impact on the overall performance. In many parallel computations vast amounts of data are exchanged between different nodes of the system. Therefore , network performance is a key issue in

the overall performance of parallel systems. The standard building blocks for computing units (usually Pentium CPU's) are quite satisfactory and exponentaly improving according to Moore's law. However, in the network area the situation is different. The standard building blocks for networks do not supply the desired performance in terms of bandwidth. For example, a regular 10/100 Ethernet network can slow down a system that works on large N-body problems. One of the solutions to this low network performance problem is to use optical networks. Optical network have large bandwidth, orders of magnitude greater than the bandwidth of copper wire or coax [6]. Another problem solved by optical networks is the wire diameter. In a system holding thousands of links the wire density becomes more critical. Optical networks use optic fibers which have a much smaller wire diameter in comparison to regular copper coax cables as in Ethernet.

1.3 Optical networks and optical switches

The biggest advantage of optical media is the capability of supplying very large bandwidths between two points. Unfortunately , fully connected networks are too expensive and not feasible, so some sort of switching must be carried out. Switches for optical networks are not as developed as switches for electronic networks. In electronic networks we can use regular switches and routers in some hierarchy to achieve efficient switching. In optical networks there is no standard switch/router available. Instead there are two main options. One is to use electronic switching and the other is to use optic switching.

In electronic routing the laser signal is transfered into electronic form, switched, and then transfered back into a laser signal. In this way the full potential bandwidth of fiber links is largely unused. Large portions of bandwidth are wasted due to the "electronic speed bottleneck" imposed by the relatively slow elec-

tronic switches and modulation technique [9]. Another option is to use optical switches, which do not require optoelectronic conversion of the data and subsequent regeneration. Many of the shortcomings of electronic or optoelectronic networks can be avoided by using all-optical networks, in which data is maintained in optical form throughout the transmission. This work is based on such an optical switch.

In both options of switching the network can be based on circuit switching or packet switching. When using optical switches, circuit switching is more popular. In packet switching the switch's hardware should parse the header of an incoming packet in order to compute the destination of the next hop. Parsing a header of a packet in optical form imposes technical difficulties in the current technology [5].

1.4 WDM (Wavelength Division Multiplexing)

Most optical networks take advantage of the WDM approach. WDM consists of simultaneous transmissions on multiple wavelengths on the same fiber. Each signal travels within its unique color band. One fiber can hold up to 40-80 different wavelengths (as published for example by leading companies Lucent and Nortel networks). Each node holds at least one optical receiver unit and one optical transmitter unit. Either the receiver or the transmitter must be tunable. Usually the receiver unit is tunable. Let's assume from now on that this is the case. An additional control network is required to control the tuning of receiver units in each node. The control network is usually an electronic one since it requires only low bandwidth [4].

1.5 Current optical network topologies

This overview will concentrate on the WDM technology since this technology is conceptually similar to our technology. Another technology worth mentioning is SONET which uses TDM(time division multiplexing) and is pretty much a standard in optical networking [2]. There are several companies who are working on optical switches like Lucent and Nortel networks. Lucent's Aurora switch is an example of a switch capable of optical layer restoration, dynamic wavelength management and network gateway functions. (http://www.lucent-optical.com/solutions/products/aurora_optical_switch). There are several standard topologies for using WDM over optical networks:

1. **passive star coupler**: Each node is assigned a single distinct transmission wavelength, and each node has a tunable receiver. When one node wishes to transmit to another node in the network, the destination node tunes its receiver to the transmission wavelength of the sender, using the control network. The sender transmits at its assigned wavelength, and the passive star coupler broadcasts the signal to all of the nodes in the network. Only the receiver listens for this wavelength and collects the message. Broadcast and select networks of this model are problematic for two reasons. First, such networks waste optical power, since the power of each transmitted signal is divided evenly between all of the nodes in the network. Second, each node requires a distinct transmission wavelength, so the number of nodes is limited to the number of available wavelength channels. Broadcast and select networks are not scalable for this reason.

2. **wavelength routed network**: a signal at a particular wavelength is routed directly to a specific destination, instead of being broadcast to the entire network. This both eliminates unnecessary divisions of the signal

power and also allows a single wavelength to be used simultaneously in multiple, non-overlapping parts of the network. This method can be used with fixed routing without optical switches or with optical switches and a reconfiguration method of the optical transmitters. Such networks require the use of one or more controllers in order to configure the routers, so wavelength routed networks are more complex than their broadcast and select counterparts.

3. **Hybrid**: use wavelength routed network to connect small LANs which are built of passive starts.

1.6 Our work

Our work is based on an optical switch which is developed in the Applied Physics department of the Hebrew University. This switch is a generic switch based on the concept of Electro-Holography(EH). The optical switching is done in a unique way (presented later) which differs from the current market solutions. Most current solution involve WDM with it's drawbacks. Our network does not use WDM, thus overcoming the known problems of limited wavelengths, complexity, loss of light power, etc as explained in section 1.5.

However, this switch has it's drawbacks too. The switching time is relatively large, broadcast can not be used and multicast is limited to a small number of nodes. In this work we design network topologies and communication algorithms that can make the most out networks based on this switch.

2 The physical model

2.1 Overview

The system consists of computing nodes that are connected via an optical fiber to one central optical switch. All the nodes are also connected together by an electronic control network such as Ethernet. The switch is electronically controlled by the control station. The switch is based on a KLTN crystal in the paraelectric phase, which enables to turn on and off a hologram by applying different voltages to the crystal. The switch configuration is determined by which hologram is activated. Writing the holograms is done in advance when the switch is created.

The system sketch is shown in figure 2.1. Following is a description of each part of the system:

- Nodes: a PC, and preferably an SMP. If the node is an SMP, one processor can be dedicated to the communication and the other can be dedicated to the actual computing task. These nodes are also connected via a regular network card to an Ethernet network.
- AMCC: a PCI card that functions as a device that can read/write data from the optical network.
- FIFO: a component that performs buffering and adjustments of the clock between the PC and the BCP.
- BCP: a component that performs a translation between the electronic media and the optical media. It can translate electronic signals to optical signals and vise versa.

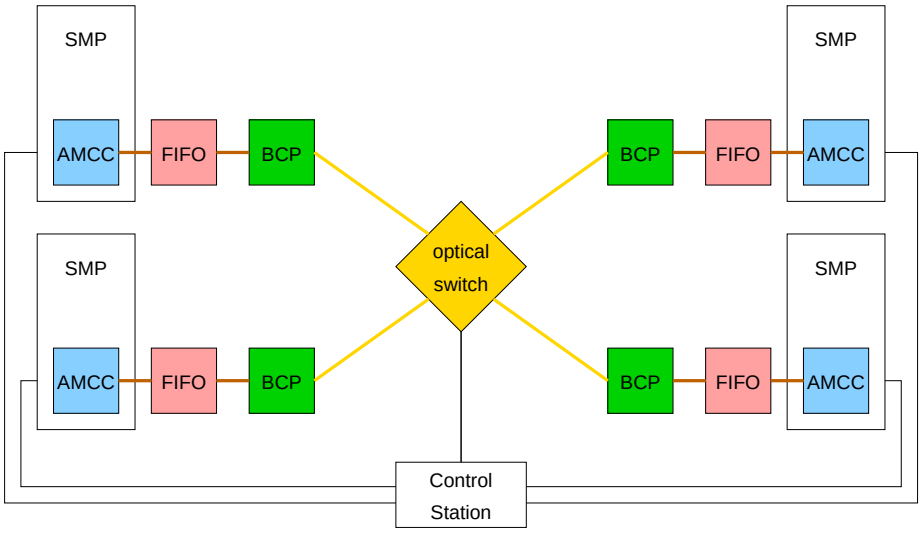

Figure 2.1: Overview of the physical system

- Optical switch: a crystal that has holograms written on it. Light that passes through this crystal is routed according to the current hologram configuration.

- Control station: A PC that is responsible for controlling the switch. It can change the configuration of the switch by sending commands to the switch using an RS232 connection. This station is also connected to the nodes' Ethernet.

- Ethernet network: the control network. The nodes can either request configuration changes from the control station or receive the new configuration information from the control station as an Ethernet broadcast message.

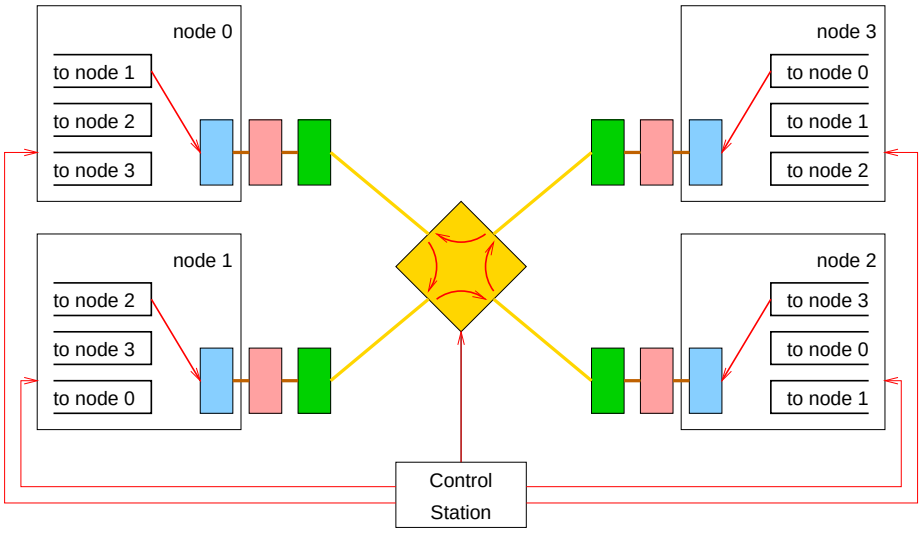

Figure 2.2: The configuration when node i sends to node $i+1$

2.1.1 Examples for configuration changes

Following are examples of possible configurations of the switch. In figure 2.2 the switch is configured such that each node can send messages to the next node. In figure 2.3 each node is configured to send data to the node that is after the next node.

2.2 Specific physical system properties

This system differs from other system that have optical switching in the way the switch configuration is controlled. In this system the switching is controlled by applying different electric voltages to the crystal. These voltages turn on and off holograms written on the crystal. The holograms define the switching. Unfortunately, this electroholographic effect has an undesired side effect. The voltage can be applied instantly on the crystal, but the new hologram is not instantly ready. It takes some time for the new hologram to stabilize on the

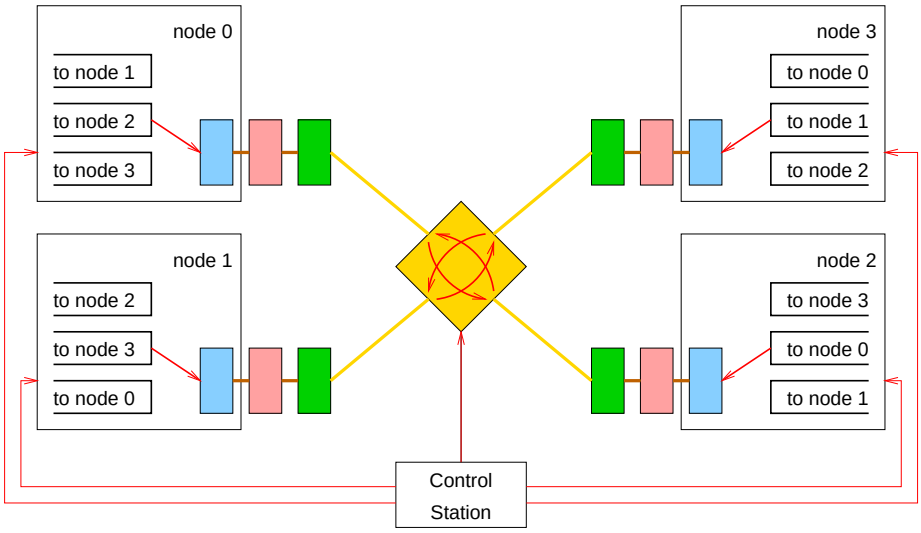

Figure 2.3: The configuration when node i sends to node $i+2$

crystal. Messages that are sent in this period of time using the routing that has just changed might not reach their destination. The final result of such a send operation is unpredictable. The amount of time it takes for the switch to stabilize is in the scale of several milliseconds (up to 10 ms) and is the main drawback of this system. As a result of this fact, configuration changes are a costly operation that should be avoided as much as possible. Our work concentrates on this issue of avoiding configuration changes.

Another issue that is connected to the optical media is that the system needs some bus mastering. Potentially, a switch configuration can be set in such a way that one node can receive messages from several other nodes. In this context, those nodes are on the same bus. The bus mastering method should insure that no node will receive more than one message at a time. This requirement is mandatory since in an optical media messages are received as light signals. In case two messages are sent to the same node on the same time, the receiver will get a combination of the light signals of both messages. It is impossible for

the receiver to decode such a message. Each node can only send one message at a time so from this aspect there is no problem. It is left to synchronize different nodes not to send messages to the same node. Our work also includes synchronization methods of this sort.

Finally, there are two main advantages to this system over other systems. The first is the bandwidth of this network. Optical media have a very large bandwidth in the scale of many giga bit per second for each fiber. The second advantage is the optical switching. The switching is done solely in the optical medium. No translations to electronic form back and forth are needed. These two factors are the main advantage of this network. It should be noted however, that the switching is done based on a configuration and not on the contents of a specific packet. Therefore, this network is only appropriate for circuit switching and not for packet switching.

2.3 Switch structure

In this discussion and in the whole work the switch is treated as a single "black box" with a defined interface that enables changing its internal configuration. In this section we will provide a brief description of the internals of the switch. In the rest of our work, there will not be any more reference to the exact implementation.

The beam that reaches a single crystal can be either diffracted or sent directly ahead. This is the basic building block of the switch. This basic structure is used as a single unit in the construction of more complex topologies. Hence, these topologies can support switching between many nodes. There are two main alternatives for these complex topologies: multi stage and crossbar. An example for a multi stage topology can be seen in figure 2.4. This figure demonstrates how to construct a multi stage network that can support 64 nodes. A

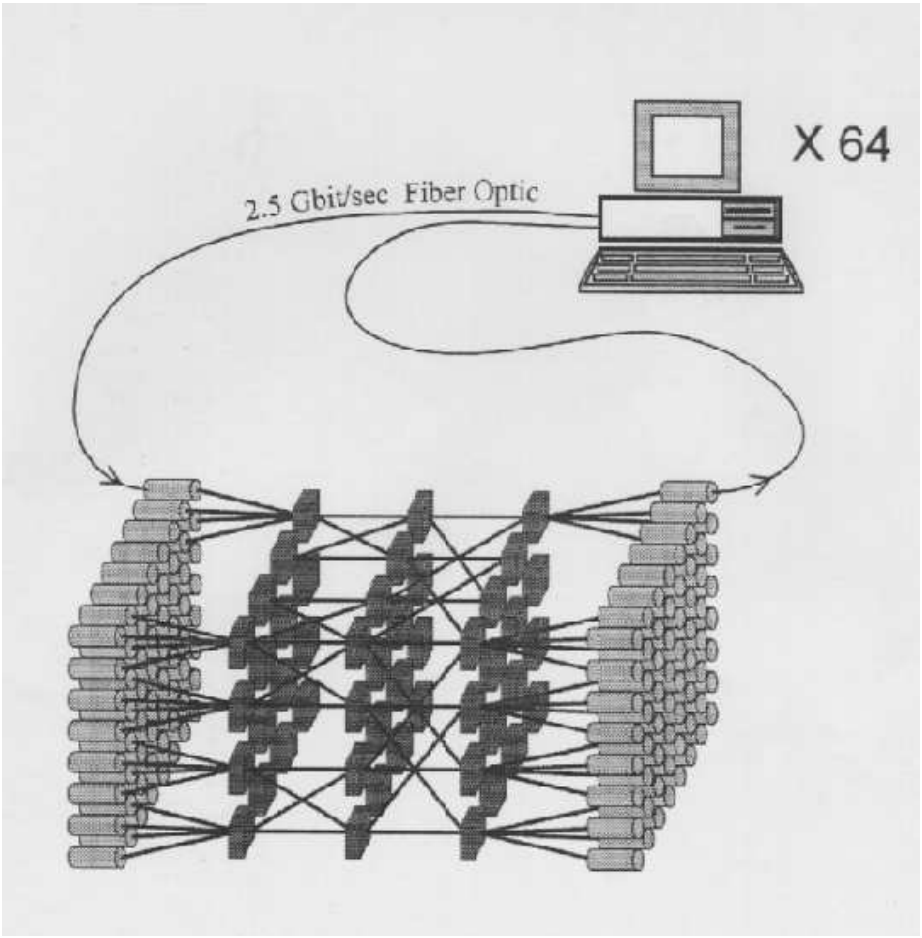

Figure 2.4: Example for building the switch with a multi stage topology

multi stage network requires $O(n*log(n))$ units. In a very large system this is a clear advantage over the crossbar that needs $O(n^2)$ units. Ideally, we would like each node to be able to send messages to any other node in the system. This switch configuration can be represented by a permutation (when we do not use multicast). However, some variants of multi stages network can not support all possible permutations. An example of a multi stages network that can support all the possible permutations is the Benes network. This network is constructed by setting up two butterfly networks back-to-back. Even though the Benes net-

work can support any arbitrary permutation, it is relatively difficult to calculate the exact "wiring" of each switch that matches a certain permutation. Moreover, even a local change in the permutation of the switch might cause a global change in the topology. In this sense a cross bar topology is superior. A cross bar can support every possible permutation and a local permutation change can remain as such. On the other hand, a crossbar is much more expansive in the number of units in large systems.

2.4 The current status of the physical system

The current system is in its prototype stage. It consists of four nodes and a controller machine. The prototype's switch is implemented using a cross bar. Two kind of tests were performed on the system. The first one is a basic test of the optical part. The second one is a full system test. In the first test, the optical switch and the BCP's connecting to it were checked. A bit generator was connected to one of the node's BCP. The bits generated were passed through the switch and directed to another node. The other node displayed the data received using a logic analyzer. An electric voltage was applied to the switch which functioned correctly and redirected the generated bits to a different node. This test worked fine and is fully described in the article [8]. The second test is in its working phase. Until the time of the writing (8/2000) the system managed to transfer a packet of data from one PC to another PC. The switch was configured on a certain permutation, which was set from the control station. The packet that arrived on the other side contained errors that were resolved using a forward error correction scheme. These errors are induced by problems of clocking synchronization and will be hopefully fully resolved in the near future.

3 The formal model

3.1 Model intuition

The components of our real optical system are limited by the current hardware we managed to purchase under financial limitation. They are also limited by the amount of time that the physics department could spend working on this development. The formal model described in this section is free of those 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 let ourselves take several simplifying assumptions to make the model practical for calculations. These assumptions are presented below.

3.2 Model overview

Similar to the physical model the system consists of several independent computing units (referred to as nodes), an optical switch, and a regular control network. Each node is connected to the optical switch through two links, one dedicated for transmit and the other dedicated for receive. The nodes are also connected to each other over some electronic control network. As explained in the previous chapter the optical switch is set with a specific configuration. A node is capable of doing the following tasks: 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 optical network supports both unicast and limited multicast packet delivery. The control network supports broadcast as well.

3.3 Model terms

- **Switch configuration -** noted by $C = (\alpha_0 , \alpha_1 , \alpha_2 , \dots , \alpha_i , \dots , \alpha_j , \dots , \alpha_N)$, such that without multicast support by the switch $\forall i$, 0≤ i≤N: a packet sent by node i will by directed by the switch to node α_i . This will be also noted by : $C(\text{i}) = \alpha_i$. C should also hold $\forall i, j : i \neq j \Rightarrow \alpha_i \neq \alpha_j$. For simplicity matters we will use the same notation to indicate a switch configuration that supports multicast. However, in the multicast version with multicast size m, $\alpha_i = (\text{i}_o, \text{i}_1, \dots , \text{i}_m)$. In this case, a packet sent by node i will be directed by the swith to the nodes $\text{i}_o, \text{i}_1, \dots , \text{i}_m$ and all these nodes should be different.

- **Local configuration set -** a switch configuration change that is triggered by a request of the format CSet (i , k). After the change the new configuration is $C_{new} = (\alpha_0 , \dots , \alpha_{i-1} , \text{k} , \alpha_{i+1} , \dots , \alpha_N)$. A global configuration set consists of several local configuration sets.

- **Local configuration exchange -** A switch configuration change that is triggered by a request of the format CEx (i , j). After the change the new configuration is$C_{new} = (\alpha_0 , \alpha_1 , \alpha_2 , \dots , \alpha_j , \dots , \alpha_i , \dots , \alpha_N)$. A global configuration exchange consists of several local configuration exchanges.

- **Forwarding -** Lets consider C for which $C(\text{i}) = \text{j}$ and $C(\text{j}) = \text{k}$. Under the limitation of C 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 global 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, and is not limited to only one node as in this example.

3.4 Model variables

- **N** : the number of processes.
- **M** : the multicast size. M can hold several values. Usually we will use M=1 but in some advanced algorithms we used M=3 and explicitly stated so.
- $\mathbf{T}_c$: the amount of time it takes to change a configuration.
- $\mathbf{T}_b$: the amount of time it takes to transmit/receive 1 byte. T_b includes the overhead of the application. For a transmit operation T_b is measured from the time the transmitted byte is still in the application layer until the time that same byte gets to the optic wire. For a receive operation T_b is measured exactly in a reverse manner from the transmit.
- $\mathbf{T}_f$: Time to transmit/receive 1 byte using forwarding. (See explanation below)
- $\mathbf{T}_d$: (T_b - T_f). The difference between a regular byte transfer to a forwarded one.

3.5 Model assumptions

- There exists some k such that $N = 2^k$. We have limited our research to systems with number of nodes that is in the scale of from about 50 to about 2000.
- Setting a configuration is an operation that once started, finishes after a fixed period of time. The operation consists of the request (T_r), the operation itself (T_s) and a constant waiting period until the switch stabilizes into the new setting (T_w). Therefore $T_c = T_r + T_s + T_w$. First, 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 got several requests for changes then each requesting node should waits for it's local configuration change to stabilize. Those waiting nodes are waiting in the same time. Second, we assume that Ts is smaller by several scales than Tr and Tw. More formally $T_s \ll T_r + T_w$ 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 $k * T_c$, because all the non negligible operations can be overlapped in time.

- We assume that it takes the same time to receive one byte as it takes to transmit one byte.

- 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/exchanges are sent over the control network we can assume that every node can trap those requests and deduce the current switch configuration. Therefore we assume that in any point in time every node is fully synchronized with the current switch configuration.

- There exists some external synchronization method that can be used to run a command almost simultaneously on all nodes. Such a method can be implemented by using the control network with a barrier for example. Then we can assume that all the global operations, like broadcast, global exchange, etc. are operations in which each node simultaneously calls a library routine to perform this task. Thus, each node already knows at the beginning of the execution of the library routine what it is supposed

to do without having to wait for a packet from the network to trigger an action.

- In several protocols that do not involve global actions we assume a different method for global synchronization. The synchronization is achieved by sending a pulse to all the nodes every fixed time period. This pulse can be sent by a special hardware and wiring. For example using a machine that has direct connections to all the nodes communication ports. This machine can send a special signal on all the communication ports to indicate a synchronization point. The same effect can be achieved without additional hardware. A broadcast message can be periodically sent on the control network by a dedicated machine. This message can indicate the synchronization point.

3.5.1 The Forwarding Unit

Forwarding is assumed to be a very quick action in comparison to regular send and receive. More formally $T_f << T_b$. This assumption is correct in the following scenario. Forwarding is handled by a special hardware which we will refer to as a forwarding unit (FU) [Figure 3.1].

The node's network interface card (in the physical model the AMCC card) will be referred to as the NIC and the unit which translates the optical media to electronic one (in the physical model the BCP) will be referred to as the O2E unit. We will place the FU between the node's NIC and the E2O and O2E units. Every packet that is sent to a specific node is first passed through the node's O2E Unit and only from there it reaches the network. Every packet that is received by a specific node is first passed from the network to the E2O unit and only then it moves to the node's NIC. The FU job during a receive operation is to intercept every packet, examine its header and decide on the fly whether

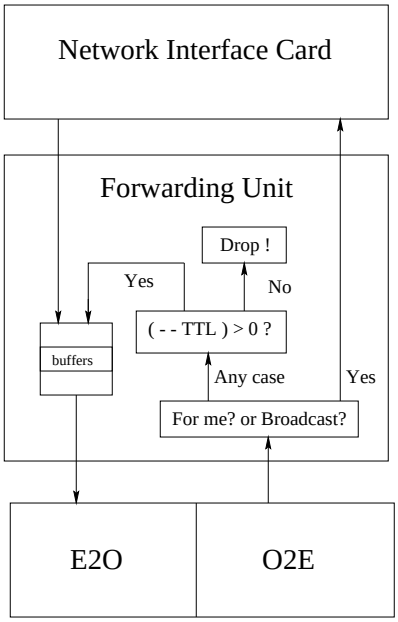

Figure 3.1: The Forwarding Unit

to deliver it to that node or to return it back to the network. A packet that is delivered to that node is passed directly to the NIC. A forwarded packet is moved to outgoing buffers on the FU itself (explained in detail below). That packet is then translated back to optical form and from there to the optical switch. The NIC holds it's own buffers so whenever it is blocked, outgoing packets are saved locally on the NIC to be later transmitted when the bus is free. In the opposite direction packets are delivered from the NIC to the buffers on the FU and from there to the E2O unit which translates the packet to optical form and sends it to the switch. The implication to the overall performance is minimal. A packet that needs to be forwarded is moved back to the network almost instantly, because the comparison is on the fly and because of the higher priority of the FU. From the point of view of the NIC the overhead is also minimal since it only adds a buffering layer which can consume packets much faster then the NIC can produce.

The last packet sent out of the FU is not thrown away 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, for example A, which wants to forward a message to B and C would have to do the following tasks : forward an incoming message and deliver it to the operating system, once the operating system has received the whole message it can request a configuration change, only after the configuration change occurred pass the whole message again down the stack and forward it. In the new mechanism, once the header of a message is delivered up the stack a request for the configuration change is done. In the mean time a direct small command is sent back down the stack to reach in a safe time after the configuration set is over. This new command instructs 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.

We assume extra functionality in the FU to support efficient broadcast. There is a special broadcast bit in the header of each incoming packet. If this bit is off then the functionality of the FU is the same as described above. If this bit is on then every incoming message is also passed to the upper layers and in the same time forwarded ahead (the FU is a hardware component that is capable of doing so). 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. A problematic point in this approach is that we need some method to stop the forwarding of a message. Two methods are suggested. The first is based on the broadcast bit and the node's address. We can instruct the FU to forward a broadcast message only if the message destination is not the current node. For example in a ring topology, we can issue a send from 0 to N with the broadcast bit on. Each node will forward it to the next one, until the last node will stop the forwarding. The

second approach, that was finally chosen for presenting broadcast algorithms, is based on TTL (time to live). In each message we have a TTL field. In each hop the value in the TTL is decreased by one and when the TTL value is 0 the message is not forwarded any longer and 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 it. It is possible to assign a correct value for the TTL field in every algorithm that is deterministic, since we can know the exact route of each message. It does not matter which approach is chosen, they are both correct and analogous one to the other.

One last point that should be noted is the buffer management on the FU. These buffers can be filled by the node's NIC and by the FU itself. Therefore, accessing those buffers will require some initial handshake to insure that in any time only one entity will fill the buffers. An entity filling the buffers will be entitled to fill one entry at a time. Each entry will require a new handshake. The FU has a higher priority than the node's NIC over this bus. If both units (the FU and the NIC) are competing on the privilege to transmit on this bus then the FU will be the one to get this privilege. The NIC will be blocked until the FU finishes. We can of course limit the number of times the NIC is blocked to prevent starvation. The FU has priority over the bus in order to minimize the amount of time a packet is traveling on the network. We must first handle packets that are already on the network before we produce more packets and put them on the network.

3.5.2 Assumptions results

From the behavior of the FU we can assume T_f will be in the order of few microseconds. A regular send/receive of a byte that should pass all the network stack of the operating system should take at least two or more orders of time

longer then forwarding. Therefore : $T_f << T_b$. The switch takes about 10 milliseconds to stablize so we can conclude that $T_c >> T_f$. The size of a message can vary from 100 bytes to 1k and more. Considering these numbers we can assume that in the time that a configuration is set something between 10-100 messages can be forwarded ahead, without any more configuration sets.

3.6 Objectives

We are interested only in the performance of the optical network. Our goal is to find specific network topologies and algorithms that would maximize the throughput of the network. Bandwidth and switching are the two main factors in the performance of the network. The bandwidth of the optical media is very large and an important benefit. On the contrary, the switching time is poor. Therefore, we will concentrate on building network topologies that will need minimum switch configuration changes. We believe that this attitude will eventually improve the overall performance of the network. In our research we have checked topologies with respect to two different approaches. In the first approach we examine a global operation like broadcast and try to find methods to finish this operation as quickly as possible. In the other approach we assume a certain distribution for the traffic on the network and try to minimize the communication time a message travels until it reaches it's destination.

4 Broadcast algorithms

In this section we present several broadcast algorithms. We start by presenting basic algorithms and advance to more sophisticated algorithms, that run faster. We deliberately present all the algorithms, even the trivial ones, in order to show our gradual progression in solving the broadcast problem.

4.1 Assumptions

The following assumption were made to simplify calculations:

1. In all the algorithms we assume that the broadcast is initiated by node number 0 and only one message is broadcasted among all the nodes.

2. A configuration set (that costs T_c) is a non-blocking action which can be overlapped in time with other actions. More specifically, a node can request a new configuration set and until this action is completed send data on it's current outgoing link. Performing these two actions in parallel can save time. However, this scheme is not safe since it relies on the fact that setting a switch configuration takes relatively more time than sending one or more messages on an existing configuration. This assumes that the messages sent on the old settings will get received before the switch stabilizes on the new settings. Taking this assumption can lead us to race conditions, caused by network congestion and operating system delays and might not work well on a real system. Therefore, we will not perform any send/receive operations on a configuration that is currently being changed.

3. On the other hand, it is safe to run several switch configuration changes in parallel. In the time of one T_c a whole new configuration can be set, provided that no other send/receive operations are done at that same time.

4. We assume a receive operation is done overlapped with the corresponding send operation. It is true there is a small gap in time between the starting point of the send operation to the starting point of the corresponding receive. However, in an optical, point to point network, this time gap is negligible. Therefore we will not count the receive operations in our formulas.

5. The system can physically support almost any message size (S). In order to simplify the structure of the FU we will restrict ourselves to S that is in the scale of 1K and not more.

4.2 Semantics

In all the calculations where the first and the last stage can not use forwarding I will still write T_f instead of T_b but also add the difference T_d so the sum will stay the same. Writing so is clearer and more understandable for the reader.

As explained in the previous chapter we are using the FU with the TTL approach. We will assume that every forwarding action is done with the correct TTL value. We can calculate this value for each call but it will make the algorithms descriptions very complex and will withdraw the attention from the main purpose, which is to understand these algorithms.

In the calculation there is a subtle point concerning forwarding. There are several approaches used to describe parallel algorithms. We decided to describe the algorithms in a pseudo high level "code", using two notations. The first notation is to divide the running time of the algorithm into stages and elaborate what each node does in every stage. The possible actions a node can perform in a certain stage are to : send a message / forward a message, request a configuration set or to do nothing. If several nodes are doing some action in

the same stage it means that those actions are done in parallel. The notation of "par:" was added in front of "code" sections that have some parallelism. For example : 'par: for each i send $(i \rightarrow i+1)$' means that in the same time all the nodes are sending a message to their next neighbour (i+1). The second notation, used only in the last algorithm, is to present a function that recursively calls itself on differnet nodes. A stage that contains more than one recursive call in parallel (with the 'par' notation) on several nodes, will imply a parallel execution of the same function in the next stage on those nodes.

4.3 Algorithms evaluation

4.3.1 Mission

Our goal is to find a method to perform the broadcast operation in the minimum amount of time. We define the time of the broadcast operation to be from the first action done until the last node has received the broadcast message. For every algorithm, we present a formula of it's running time. In the last section we compare different algorithms and note their advantages and disadvantages. The success of an algorithm depends on its ability to maintain the right tradeoff between configuration sets and forwarding. Too many configuration sets result in a slower running time, but sometimes it is better to do one configuration set instead of forwarding the message between too many nodes.

4.3.2 The "half optimal" metric

A broadcast operation running time is mainly affected by the number and timing of configuration sets (noted by T_c in our model). Therefore, we have examined an interesting metric on configuration sets, which we will call the 'half optimal' metric. In this metric we consider T_1 to be the running time of the algorithm

with some value of T_c and T_2 to be the running time of the same algorithm with $T_c = 0$. We would like to know for what value of T_c does the following equation hold : $T_1 = 2 * T_2$? We will call this value $T_c^{0.5}$. Note that T_c can accept values from 0 to ∞ so T_2 represents the optimal case in this respect. With this metric we can have a better evaluation for the performance of an algorithm in respect to configuration changes. We would like to ensure that the values we use for T_c in the real system are about equal or lower than the values being calculated in the "half optimal" metric. In order to solve this metric we have to assume some relation between the two values of T_f and T_d . We have assumed that $T_d = i * T_f$ where i is a small integer in the scale of 10.

4.3.3 Forwarding issues

One of the patterns that are frequently used in these algorithms consist of three nodes : A,B and C. First, node A forwards a message to node B and then to node C. We will specifically evaluate the running time of this pattern, as it is so common in the following text. Lets assume node A is already configured to send messages to node B. When A's FU receive a message it is automatically forwarded to node B if the broadcast bit is on. In this case the message is also delivered to the operating system. When the header, or part of it reaches the operating system is it possible for node A to deduce that it also needs to forward the message to node C. The time it takes for a small header to reach the operating system layer is $T_d * i$ where i is the size of this header in bytes. We assume that i is very small so that T_d can be a good approximation for this time . In this point node A issues a request to set the configuration towards node C. Note that until this happens all the data forwarded to node B should have already been received by the FU of node B. We can deduce this due to our assumption number 5 in the assumption subsection, and because this action is

much faster than the corresponding action done by node A. Node A also issues a request to the FU to re-forward the current message, and makes sure this request will reach the FU right after the switch has stabilized it's configuration. Those two action, the configuration set and the command to the FU, occur in parallel and the longer of them which is the configuration set requires T_c . To summarize, the overall time for this pattern is : $S * T_f + T_d + T_c + S * T_f$. The first $S * T_f$ is for the first forwarding to node B and the second one is for the second forwarding to node C. The T_d value is for the time it takes the header to reach the operating system and T_c is for the new configuration set.

4.4 Algorithms

4.4.1 Naive configuration change

In the naive method a node broadcasts a message by doing N-1 local configuration sets and sends. This method does not use forwarding.

Algorithm description:

```
1 :  for ( i = 1 ..  N )
2 :      set_configuration ( 0 → i )
3 :      send ( 0 → i )
```

Running time analysis : $(N-1) * (T_c + S * T_b)$. We need $(N-1)$ stages and in each stage we perform one configuration set and one send. Note that S is the size of the message being broadcasted.

4.4.2 Naive forwarding in a ring.

The previous method didn't use forwarding. In this method we add forwarding on a network topology of a ring. First we will set the ring topology and then

each node will forward the message to the next node.

Algorithm description:

```
1 : par: foreach i :  set_configuration ( i → i + 1 )

2 :  for ( i = 1 ..  N )

3 :      send ( i → i + 1 )
```

Running time analysis : $T_c + (N-1) * S * T_f + 2S * T_d$. The formula is correct since every cycle takes exactly $S * T_f$ of time except of the first and the last one. Those sends can not be forwarded and must reach the node's machine. This takes T_b instead of T_f so adding $2ST_d$ yields the same result. We also added the first T_c to set the ring topology. The 'half optimal metric' : $T_c + (N-1) * S * T_f + 2S * T_d = 2(N-1) * S * T_f + 4S * T_d$. From here we conclude that $T_c^{0.5} = (N-1) * S * T_f + 2S * T_d$. Lets assume $2S * T_d$ is negligible for a large value of N then we can get $T_c^{0.5} \cong N * S * T_f$. The systems that interest us are in the scale of 100-1000 nodes and the message size is about 100-1000 bytes. Therefore $T_c^{0.5} \cong i * T_f$ where $10000 < i < 1000000$. In a typical case the configuration time will be somewhere between 10 to 100 times slower then the forwading time of a message, which means that the configuration time is somewhere between 1000 to 100000 slower then the forwading time of one byte. In this typical case the ratio between T_c and T_f is even worse than the current result of the 'half optimal' metric. In this algorithm T_c is not a problem. It is not surprising since only one T_c is included in the total running time. There can not be a better solution if we only consider T_c .

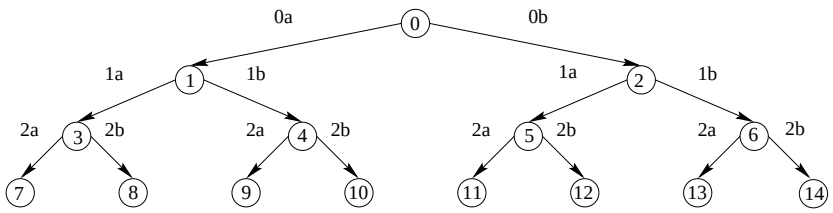

Figure 4.1: Naive forwarding in a tree

4.4.3 Naive forwarding in a tree.

In the previous subsection we have used forwarding in a ring topology. The biggest disadvantage of that algorithm is that it works in sequential manner. Sequential algorithms which have a running time of $O(n)$ can not be optimal for a ring with a large number of nodes. Therefore, a way to deal with this problem might be to use a parallel algorithm that does not work sequentially. We suggest a tree topology to perform the broadcast. In each stage one layer of the tree broadcast the message to the next layer (figure 4.1).

In stage 0 the sends that are done are 0a and 0b. In stage 1 the sends are 1a and 1b for the left branch and also for the right branch. In this way the broadcast proceeds for more stages (only 3 stages are shown here).

Algorithm description:

```
1 : par: foreach i :  set_configuration ( i → 2 * i + 1 )
2 :  while (there exists a node that didn't get the broadcast) {
3 :     par:  foreach i :  If i just got a message then {
4 :        forward ( i → 2 * i + 1 )
5 :        set_configuration ( i → 2 * i + 2 )
6 :        forward ( i → 2 * i + 2 )
7 :     }
```

```
8 :  }
```

Running time analysis : In each stage the number of nodes that got the broadcast message are doubled so the running time is logarithmic. It can be formulated into : $T_c + ((log_2 N) - 1)(2S * T_f + T_d + T_c) + 2S * T_d$. The first T_c is due to the first global configuration set in line 1. $((log_2 N) - 1)$ is the number of levels until all the nodes have received the broadcast. In each stage we do two forwards that cost $2S * T_f$ (line 4,6) and one configuration set that costs T_c (line 5). We also have to add T_d betweem the two forwarding as explained in section 4.3.3. Finally, we add $2S * T_d$ since the first and the last stage can not be done using forwarding. Note that in each stage there is only one configuration set since the first configuration set was done in line 1. The 'half optimal' metric : $T_c + ((log_2 N) - 1)(2S * T_f + T_d + T_c) + 2S * T_d = ((log_2 N) - 1)(4S * T_f + 2T_d) + 4S * T_d$ $\Rightarrow \ldots \Rightarrow T_c^{0.5} = \frac{(log_2 N - 1) * (2S * T_f + T_d) + 2T_d}{log_2 N}$. The value of $\frac{2T_d}{log_2 N}$is negligable and the fraction $\frac{log_2 N - 1}{log_2 N} \cong 1$. Therefore $T_c^{0.5} \cong 2S * T_f + T_d$. According to the assumption in section 4.3.2, we can again approximate $T_c \cong 2S * T_f$. Using the regular value of $S : 100 < S < 1000$, we conclude that $T_c^{0.5} = i * T_f$ when $200 < i < 2000$. This result is not as good as the previous algorithm but is still in the reasonable range of values for T_c.

4.4.4 Smart forwarding in a tree with no idle nodes.

In the previous algorithms we have used forwarding and reached a logarithmic running time limit. However, once a node has forwarded a message it stopped contributing to the broadcast process. In this algorithm we correct this by having all the nodes active in the broadcast process until it is finished. Table 4.4.4 shows the algorithm progress along the first stages. The first row is the stage number and the other rows are the sends that are executed in that stage.

Smart forwarding in a tree

Stage		0	1	2	3	4
Received		0..3		0..15		0..63
Operations	$0 \rightarrow 1$	$0 \rightarrow 3$	$0 \rightarrow 4$	$0 \rightarrow 6$	$0 \rightarrow 16$	$0 \rightarrow 18$
		$1 \rightarrow 2$	$1 \rightarrow 7$	$1 \rightarrow 9$	$1 \rightarrow 19$	$1 \rightarrow 21$
			$2 \rightarrow 10$	$2 \rightarrow 12$	$\vdots$	
			$3 \rightarrow 13$	$3 \rightarrow 15$		
				$4 \rightarrow 5$		
				$7 \rightarrow 8$		
				$10 \rightarrow 11$	$\vdots$	
				$13 \rightarrow 14$	$15 \rightarrow 61$	$15 \rightarrow 63$

Algorithm description:

```
1 :  for(stage=0;exists a node that didn't get the broadcast;stage++){
2 :     par: foreach i :  {
3 :        if (I got a message for the first time in the last round)
4 :           set_configuration_and_forward(i → i + 1)
5 :        else if ( I got a message before previous round ) {
6 :           if ( stage is odd )
7 :              set_configuration_and_forward(i → 3 * i + 2^(stage+1))
8 :           else
9:              set_configuration_and_forward(i → 3 * i + 2^stage + 2)
10:       } // end of else if
11:    } // end of the parallel for each
12: } // end of for
```

Running time analysis : In each stage the number of nodes that got the broadcast message is twice as big as the number of nodes in the previous stage. Formulating we have : Time $= (log_2 N) * (S * T_f + T_d + T_c) + 2S * T_d$ .The number of stages is $log_2 N$. In each stage we have one set of a configuration and one message sending : $(S * T_f + T_d + T_c)$. Finally, we add $2S * T_d$ for the first and the last operations which can not use forwarding. In the 'half optimal' metric we should get a similar result to the previous algorithm since in both algorithm we have $O((log_2 N)T_c)$ in the running time formula.

4.4.5 Smart forwarding in a ring with no idle nodes.

In the previous algorithm all the nodes contributed to the broadcast process in all the stages and it worked in a logarithmic running time. The main drawback is that on each level two configuration sets must be made. During those sets no productive work can be done. Given our regular assumption that $T_c \cong 10S * T_f$ 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 set a regular ring topology and then to recursively work on continuous smaller fractions of the ring, starting with the whole ring. In each stage, forward a message ahead, along the ring. In the mean time set a permutation to somewhere in the current ring fraction, such that when this message will be received, the number of nodes after the receiver node will be about equal to the number of nodes that didn't receive the message before the receiver node. The value of k is taken to be equal to $\frac{T_d+T_c}{S*T_f} + 1$. We reached this figure by comparing the running time of the left box to in the algorithm description to the running time of the right box in the same description. Figure 4.2 shows the progression of the algorithm along three stages. The highlighted nodes are nodes which have already received the broadcast and are in groups of k nodes.

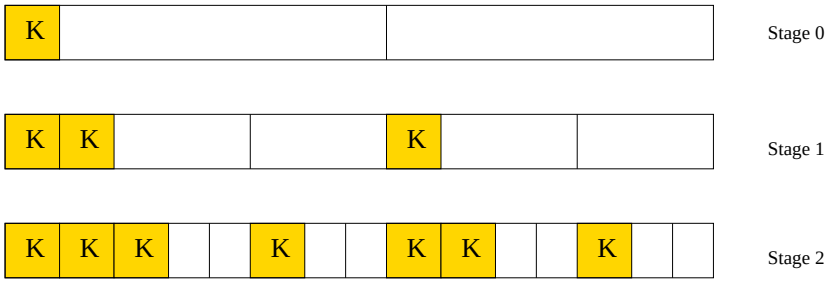

Figure 4.2: Algorithm progress in the first three phases.

Algorithm description:

```
1 :  func broadcast ( left , right ) {
2 :      forward (left → left + 1)
3 :      middle = (right+left+k)/2
3 :      par_section: ( the left code runs in the same time as right
code)
```

4: set_configuration ($left \rightarrow middle$)	for ($i = left + 2...left + k - 1$) {
5: forward ($left \rightarrow middle$)	forward ($i \rightarrow i + 1$)
6:	}
7: broadcast ($left + k, middle$)	broadcast ($middle, right$)

```
8:  }
```

Running time analysis : We can see that the number of nodes that have received a message in stage i is $k * (2^i - 1)$. We would like to know how many stages do we need to finish the broadcast or more formally: for what i does $N = k * (2^i - 1)$. Therefore $i = \log_2(\frac{N}{k} + 1)$ and if we assign k as $\frac{T_d+T_c}{S*T_f}$ and omit the +1 to simplify calculation we get :
$i = log_2(\frac{N*S*T_f}{T_d+T_c}) = log_2 N + log_2(\frac{S*T_f}{T_d+T_c})$. Assuming $T_c \gg T_d$ we get that

$i = log_2 N + log_2(\frac{S*T_f}{T_c})$. The second sum is depended on the ratio between the forwarding time to configuration set time in a logarithmic scale. Our regular assumption is that a configuration set takes more than a forwarding by at least one scale or more. So, $log_2(\frac{S*T_f}{T_c})$ is equal to a small negative number (-2,-3,..,-5) . Therefore, even for a very big system with 2^{10} nodes $i = 10 - 3 = 7$. For any realistic system size we get $i < 8$. For this algorithm the 'half optimal' metric is not meaningful. In this metric we look on $T_c^{0.5}$ in which $T_c = 0$. When $T_c = 0$ this algorithm can not work, since it is strongly based on the fact that during the time it takes for a configuration set, several forwardings are done. Unlike other algorithms, when $T_c = 0$ this algorithm will behave differently, so we will not examine this metric here.

4.5 Algorithm comparisons

No	Algorithm	Running time
1	Naive configuration change	$(N-1)*(T_c + S*T_b)$
2	Naive forwarding in a ring	$T_c + (N-1)*S*T_f + 2S*T_d$
3	Naive forwarding in a tree	$T_c + ((log_2 N) - 1)(2S*T_f + T_d + T_c) + 2S*T_d$
4	Smart forwarding in a tree	$(log_2 N)*(S*T_f + T_d + T_c) + 2S*T_d$
5	Smart forwarding in a ring	$log_2 N - i$ where i = 2..8

The first algorithm is fairly simple but does not use forwarding. Therefore it can be used for a system without a FU or for a system with a small number of nodes. In a system with many nodes this algorithm is clearly inappropriate.

Adding forwarding lead us to the second algorithm. Here we use the basic ring topology to forward the message with a minimum number of configuration sets. We can show that this algorithm is better the the former by asking for what N

does Alg1 $<$ Alg2 : $(N-1)*(T_c+S*T_b) < T_c+(N-1)*S*T_f+2S*T_d \Rightarrow \ldots \Rightarrow N < 2+\frac{ST_d}{T_c+ST_d}$. Since $T_d, T_c > 0$ the whole fraction is positive and smaller then 1 which means $N < 3$. Therefore, we can generally say that the second algorithm is better than the first one. However, it also has a flaw. This algorithm still has a running time of O(n).

We have fixed it in the third algorithm to O(log(n)). The third algorithm uses a tree in which every node in the lower layer that has received the broadcast forwards it to its left and right sons. Comparing the two algorithms: $(N-1)S*T_f < ((log_2N)-1)(2S*T_f+T_d+T_c)$. If we remove all the small values (T_d+T_c) and concentrate on the basis of this formula we can see that this formula is analog to the formula : $O(log(n)) < O(n)$. More formally this formula is about the same as asking for what N does $N < 2log_2N$. Therefore, in a wide perspective the third algorithm is better than the second for about N$>$8. Since we did not count some of the parameters into the final formula it would be safer to claim that this statement is true for N's that are larger than 8, and the exact number is not so important in this case. A disadvantage of the third algorithm is that half of the nodes are idle at any point. Nodes that have finished forwarding to their two sons do not contribute anymore to the broadcast process.

We have corrected this problem of the third algorithm in the fourth algorithm. It is clear to see that the fourth algorithm is always better then the third from the formula : $T_c+((log_2N)-1)(2S*T_f+T_d+T_c)+2S*T_d < (log_2N)*(S*T_f+T_d+T_c)+2S*T_d \Rightarrow \ldots \Rightarrow log_2N < 2 \Rightarrow N < 4$.

The fifth algorithm introduces the idea that during a configuration set other actions such as forwarding can be done. It is difficult to compare these two algorithms since we only have a recursive formula for the fifth algorithm. In spite of that, a good approximation shows that for a system with 32/64 nodes

the fourth algorithm works at least in the same time or even better than the fifth algorithm. However, for larger number of nodes the fifth algorithm is much better. We must also note that the fifth algorithm is a very complicated one, which is difficult to implement in a real system because it requires fine tuning work to find the value of $k = (\frac{T_c + T_d}{S * T_f} + 1)$. Any variation from this exact ideal number will lead to a definite degradation of the results, due to idle nodes.

4.5.1 Conclusion

The algorithms were presented in an oder from the most basic one to the most sophisticated one. In a system with a small number of nodes it would be recommended to use a ring topology with forwarding. In larger systems I would recommend to use the fourth algorithm, which uses a tree structure with no idle nodes. This algorithm is the most realistic and reasonable to implement in a real system. The last algorithm is nice in theory but very complicated to implement, though it is clearly the fastest algorithm presented.

5 Algorithms for a general point to point communication model

5.1 Overview

Parallel systems usually use patterns of collective communication. The most common pattern is broadcast, which was discussed in detail in the previous chapter. There are some other popular patterns like multicast, global exchange, etc. Although many parallel programs use these patterns, not all of the communications in parallel systems is done in a defined collective communication pattern. Many parallel programs send messages between pairs of nodes, or between small groups of nodes, which do not involve all the nodes in the system. Even if a unique pattern exists, we would like to have a system that can handle even unique patterns in a generic manner. For example, in parallel FFT computation or parallel CFD (computation fluid dynamics).

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 it's own, sending messages to other nodes . The frequency of sending messages and their destinations are randomly set according to some given distribution. In reality, there are probably no applications that behave according to this model. However, this model enables us to simulate a generic communication model. Our parallel system must efficiently work in this communication model too.

In contradiction with the broadcast algorithms, presented in the previous chapter, both of the models presented here use limited multicast. This is a necessity when using a P2P communication model. Since the messages are random and distributed uniformly there can not be any specific switch configuration which optimally satisfies all cases. A model that does not use multicast will be forced

to do many configuration changes. We already know that many configuration changes slow down the system due to the physical design of the switch. Therefore, a model without multicast will not be efficient with the P2P communication model.

We have formalized two topology models and two different distribution models, and checked the four possible options for a topology model and a distribution model. The distribution models are: a uniform distribution and a hot spots distributions. The topology models are: a model based on a cube connected cycles topology and a model based on a multiring topology. These two models consists of network topologies and network protocols. These models are much more complicated to evaluate than the models we presented for broadcast. Therefore, in addition to a partial mathematical analysis, we also used simulations to measure their performance.

This chapter is organized in the following manner : first, we present our objective. Second, we present the simulation model used to evaluate both topology models. Finally we describe each topology model in detail, and present the results of the short mathematical analysis and of the simulation.

5.2 Objective

A send operation is defined to be the operation in which a packet is sent from one node to another. The send operation starts when the source node wants to send a packet and finishes when that packet is received by the destination node. Our objective is to find a model in which the average time it takes to perform random send operations is minimal, when these random sends are chosen from a given distribution. We would also like this model to operate successfully in a system with a heavy load of send operations.

5.3 The simulation model

5.3.1 Structure

The running time of the simulation was divided into discrete periods of time, called rounds. The simulation program supports the following operations : send, receive, and set configuration. Each operation starts in the beginning of a round and ends in the end of the same round. This model of division into rounds does not fully reflect a real system since it is less flexible with respect to the exact timing of the operations. However, we feel that in the current context, this model is accurate enough and enables us to write a program that can easily simulate running time scenarios, resulting in a more or less accurate evaluation of our two topology models.

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 random source to a random destination. Both models use forwarding to minimize configuration changes. The specific route a messages passes from source to destination is determined in run-time by the current model. The processing of a send operation can result in one of the two situations: either the message is propagated one hop ahead from the source towards the destination, or that no action is taken. No action will be taken in case this send operation is blocked for some reason, for example : by other sends, or by the model's policy. The simulation is run until both the number of requests and the number of rounds exceed a certain limit. In the results we drop the first constant number of requests and don't include them in the statistics, since we want to evaluate the algorithm in a steady state and not during a cold start where the load on the system is relatively low.

The number of hops between node i and node j is defined to be the number of

nodes on a specific route between node i and node j plus one. This definition implies that there can be several hop values between two nodes if there are several different routes between these two nodes.

5.3.2 Hot spots

When the simulator adds a single send operation it is chosen randomly. The only restriction is that the source node is different from the target node. The nodes are chosen at random according to two distributions. The first is the regular uniform distribution such that each node has the same probability to be selected. The second one is a hot spot distribution. In this distribution a small number of nodes are part of a group called H. This distribution is meaningful only when this group is really small: $|H| \ll |N|$. A hot spot distribution with parameter p means that for every send operation the source and the destination are chosen in the following manner: First, the destination group is chosen to be group H in probability of p, or a group consisting of all the nodes in the system in probability $(1-p)$. Once the destination group is chosen, the probability that the destination would be a specific node in that group is distributed uniformly. Finally, the source is chosen uniformly from all the nodes in the system, except of the already chosen destination. Hence, a hot spot distribution with parameter p means that for every send operation there is $p_H = p + \frac{|H|}{N} * (1-p)$ probability that the destination would be in group H, and $1 - p_H$ probability that the destination would not be in group H. In our case, without loss of generality we chose group H to include only node 0. Our topologies are symmetric so it does not matter which node we choose to be a hot spot. We chose only one node to enlarge the hot spot effect. As the number of the nodes in group H is bigger, the hot spot effect is smaller since there are more different routes to those nodes and less collisions. We used three different parameters for this distribution :

1% , 3% and 5%. A hot spot with 5% is considered to induce heavy load on a general network.

5.3.3 Evaluation

The average running time of a send operation is measured by the average sum of the number of rounds(hops and delays) it takes to finish a random send operation. The sum of these two figures is by definition equal to the number of rounds passed between the time the message started traveling in the network until it reached it's destination. The simulation program measures this number. The result of the simulation is the average number of rounds needed to accomplish a random request according to a given distribution.

We assume that a node can perform a receive operation and a transmit operation at the same time. This assumption complies with the physical model under certain assumptions that are explained in detail in the physical system chapter.

5.4 The Cube Connected Cycles (CCC) model

5.4.1 Overview

The motivation for the CCC model comes from the fact that a configuration change in our model is a costly operation. We would like to use a network topology and a protocol that works with the minimum number of configuration changes. Reducing the number of configuration changes decreases our flexibility and must be compensated with a new feature. The feature that is most appropriate here and also fits our model is limited multicast. With limited multicast and a network topology of CCC it is possible to send a message from every node to another without any configuration changes at all. We will show that

even without configuration changes this model is quite efficient. The CCC network topology with the multicast feature interduces a new problem of message collisions. We will later show how to handle this problem.

5.4.2 Structure and Semantics

Our CCC model consists of a regular hypercube of dimension D with each node replaced by a ring of D nodes. The rank of each node in a CCC is a constant three. The switch supports limited multicast for three nodes. The switch is statically configured such that a message from any node will reach the node's three neighbors. Since the configuration is static, every message is sent with multicast from one node to it's neighbors. This send operation is considered to be a single operation.

An example of the CCC topology can be seen at figure 5.1. The figure illustrates a CCC with $D = 3$. One ring is shown in bold. This ring includes the nodes: 110-0,110-1,110-2. Their neighbors are also connected with a bold line.

Two out of the three neighbors of any node are connected on the same ring. We will call these nodes *inner* nodes. The other neighbor node which is connected to a different ring will be called an *outer* node. The nodes in the CCC are numbered by an ordered pair : (HN,RN) where HN is the hypercube node number and RN is their node number in the ring. To get the HN we assume that each ring in the CCC represents one node in a hypercube. Then we use the conventional hypercube numbering scheme, which I will assume the reader is familiar with. The RN number is the node's index in the ring and corresponds to the dimension number that node is connected to on the outer node. For example in a CCC with $D = 3$ the node (111,2) is node number 7 in the analogous hypercube. This node is connected to the inner nodes: (111,0) and (111,1) on the same ring and also connected to the outer node (011,2) on the ring index which is dimension

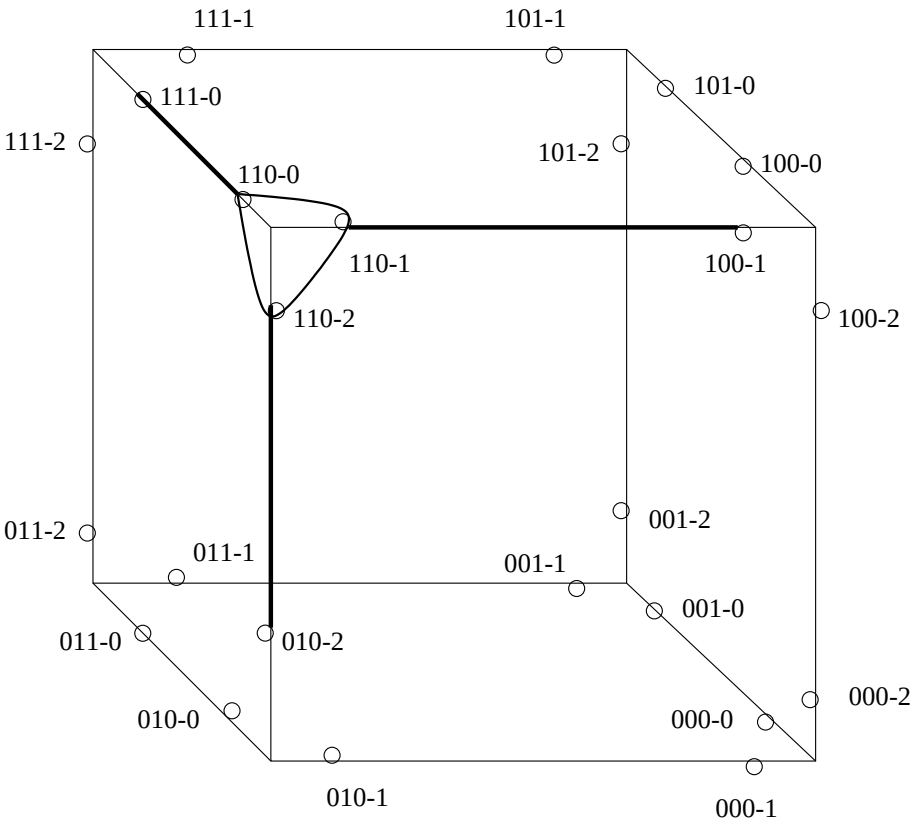

Figure 5.1: A CCC topology with $D = 3$.

2. Note that the outer node has bit number 2 opposite than the original node (bit number 2 is the third bit from the right since the first bit is bit 0).

5.4.3 Routing in CCC

The routing scheme in a CCC topology is based on the naive routing in a hypercube. In a hypercube with N nodes, the distance between any two nodes is limited to D where $D = log(N)$. It is assumed the reader is familiar with this routing scheme. Routing in CCC from a source node s to a destination node d can be broken up into three parts:

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.

A detailed description of the routing in CCC can be found in [3].

5.4.4 The collision problem

The topology presented above is based on the use of multicast. It is possible to connect every node to three other nodes because the switch is configured for a limited multicast from a node to it's three neighbors. A node sends every message to it's three neighbors. Actually only one of them should process the message. When the other two neighbors look at the destination address they understand that this message is not intended for them, and gracefully drop it. This method causes a problem of collisions. A collision happens when more than one node sends data to the same destination node. Due to the physical limitation (as explained in chapter 2), a node that simultaneously receives data from more than one other node gets the combination of the data sent to it. The data received is of course damaged. There isn't any way to get the original data out of the damaged data.

5.4.5 Avoiding collisions

It is clear that any collision is unacceptable in our model. We must ensure that there will not be any collisions at all. To achieve this goal we need a synchronization method, that will make sure that any node will only receive one message at a time. We have suggested two methods:

1. **on-demand** : Let a central manager take care of the synchronization.

   Each node asks the switch controller for permission to send a message on it's outgoing links, start sending only when it got the approval, and notifies the controller when it is finished sending. In detail: node i wants to send a message on it's outgoing link. It sends a request for the controller. The controller checks if there are any other nodes sending to one of the neighbors of node i. If so, it does not grant the request of i and asks it to be blocked. Otherwise, the controller immediately answers node i with the approval to send messages. The controller also saves this fact in it's data bases for future checking. Now, node i can start transmitting several messages in a row. In case a node is blocked, the controller's responsibility is to notify it when it's request becomes valid. Earlier we suggested that after finishing a send operation the sender must notify the controller of this fact, so the controller can update it's records. Another option is to give the send operation a timeout value. After the timeout expires the node loses it's grant to transmit. In this case, it is also the node's responsibility to make sure it does not exceed this time quantum by sending messages after the timeout expires. The on-demand method introduces an overhead of calling the controller at least one time for each series of send operations, but if the load is not high it should work fine. In a very high loaded system this centralized mechanism can slow down the system, until the controller is able to handle all the requests. A large percentage of the nodes might get blocked, thus lowering the utilization of all the system.

2. **Cycled** : Let the send operations be limited to specific cycles that will insure the synchronization.

   Each node will listen on the control network for messages from the controller which I will call *pulse messages*. The *tick time* is defined to be a

small period of time in the scale of several milliseconds . The controller will send a pulse message every tick time with the record: current dimension and a boolean mode flag that can hold *outer* or *inner*. A node will receive this message and according to the record know if it can send messages on the data network or not. Each node has one outer connection and two inner connections to other nodes. Node s can only send data to a node d if the following two requirements are fulfilled: First, s must be connected to d via the dimension received from the controller. Second, d must have the same relation to s as the boolean mode flag received from the controller. If d is an outer node of s than sending is possible only if the mode flag in the pulse message is also outer. The same idea holds for inner mode. The controller will cycle through all the available options for the pulse message record in a round robin fashion. An option can be defined as the ordered couple: $(dim, mode)$ where $dim = 1..D$, $mode = inner\, or\, oute$. For example, on a 24 nodes system that has 3 dimensions the pulse messages will be: (0,inner), (1,inner), (2,inner), (0,outer), (1,outer), (2,outer),.... . This method insures that there will not be any collisions. Therefore, there is no need for the node to contact the controller. The node operates independently, sending data only in the appropriate cycle (that matches the corresponding pulse message) . This method can only work if we choose a pulse time that is larger then sending several messages (at least 1 message) and make sure that the node does not send more messages than it is capable of, with respect to this time limit.

Note that in both methods, messages that are received when a node does not expect them to be received, are immediately dropped. Every node will receive such messages because we use multicast but the messages are intended only for one destination.

5.4.6 Optimization for the cycled version

The cycled synchronization method works well for all dimensions. The main disadvantage is that the nodes are unable to send messages a large percentage of the total time. To be more accurate, since there are D outer cycles and D inner cycles, a node can send data only once every $2D$ cycles. This slowdown is not acceptable for higher dimensions. It is possible to optimize this method and do several sends in parallel that do not interfere with each other.

Optimizing inner sends

In dimension D we can do $3 + D \, modulo \, 3$ parallel inner sends. For example with $D = 6$ we can allow nodes 0,3 to send on the first round, allow nodes 1,4 to send in the second round and allow nodes 2,5 to send in the third round. In dimensions which are divided by 3 with a reminder of 1 or 2, we must use 4 or 5 rounds respectively.

Optimizing outer sends

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 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. In dimension D we can do $\lfloor \frac{D}{2} \rfloor$ parallel outer sends. We will restrain ourselves to even dimensions since otherwise we will have to add special handling for dimension $(D-1)$ and 0 that otherwise, on odd dimensions, are sending in the same time.

With the two optimizations, the length of the round robin cycle is fixed. It is decreased from $2D$ to not more than 7 cycles, and with the best case of $D \, modulo \, 3 = 0$, decreased to only 5 cycles.

5.4.7 The tick time in the cycled version

When several messages are queued to be sent from node i to node j on the same round it would be preferable to actually send them all on one round. Otherwise, the sending queues of the nodes will grow too long and in high load the system will have poor performance. However, in the cycled version, it is not possible to send too many messages on the same round since the number of messages is limited by the tick time. The tick time can not be too small, or else the overall performance will significantly decrease. The tick time should not be too big either. With a large tick time a node will use only a small portion of it for the transmission and will stay idle the rest of the time. Moreover, when a node wishes to transmit it will be blocked for a large period of time in average until it gains access from the controller. From the simulations we have reached that the optimal tick time should allow about 20-40 messages to be sent on the same round. This topic will be discussed in detail later.

Note that in the on-demand version there is no meaning for this tick time. Still, it would be wise to limit the number of messages a node can send once it gains access, to prevent starvation of adjacent nodes.

5.5 CCC evaluation

5.5.1 Theoretical analysis

The distance between any two nodes is limited by $3 * D$. One D is needed for doing the hops between rings from dimension 0 to dimension D. Another D for inner moves inside rings to reach the next outer node. Another D is needed for reaching dimension 0 in the first ring. In this routing scheme, once the message has reached the last ring, the target node can be reached by one step. A tighter limit, $2 * D + floor(\frac{D}{2}) - 2$, can be found in [3]. We can easily slice

$\frac{D}{2}$ off our limit to reach the tighter limit by starting the route from the current dimension and not moving to 0 in the first ring. This optimization is currently not implemented in the simulation.

In the on-demand method without collisions a request will take no more than $3*D = 3*log_2N$ rounds. In an average case with no collisions sending a message should take less, but in case of collisions this number can rise extensively. It is difficult to mathematically estimate it. The simulation is used to estimate the degradation in performance due to collisions.

In the cycled version a request is bounded in any case almost regardless of the load on the system. Above a certain extreme load the maximum number of messages it is possible to send in the tick time is exceeded and the bound is not guaranteed. With more realistic loads a request is bounded by $3 * D$ multiplied by the number of times a node is idle. If we restrict ourselves to even dimensions, and on average $D\, modulo\, 3 = 1$ then there are 2 outer rounds and 4 inner rounds. The expectation for the number of rounds a node has to wait is 3. Therefore in the worst case the limit is $6 * 3 * D = 18 * D$ (the 6 comes from 4 inner rounds + 2 outer rounds). However, this worst case requires a very unique set of occurrences and has a very low probability. In the average case the time is $3 * 3 * D = 9 * D = 9 * log_2N$ 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, a system with a high load would work much better with the cycled version. 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.

Dimension	Average # of round with cycled	Average # of rounds with 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 event

5.5.2 The simulation

The parameters for the simulation are :

- The dimension of the CCC. We have investigated the following dimensions 4 , 6 and 8 which have 64 , 384 and 2048 nodes respectively. These figures represent small, medium and large systems.
- The maximum number of new events that are spawned each round. We will call this number the *spawn value*. Spawn values were checked in the range of from 10% of the number of nodes till 100% of the number of nodes. Each round the simulation chooses a new random number of events to spawn from 1 to the spawn value. Hence, a spawn value of X% means that on average there will be $\frac{X}{2}$% new events.
- The maximum number of messages that can be sent in one round from one node is limited by a number we will call MMIR. When it is not explicitly said we will use MMIR = 35.

5.5.3 Simulation results

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 with 100% spawn value. In general, it is clear that with light load on the system the on-demand version is better then the cycled version. Intuitively it is obvious since if there are very few collisions it is not worth to introduce the overhead of the cycled version. With a very high load on the system the situation is the opposite and it is clear that the cycled version is better. It is also very understandable since in the on-demand version there are too many collisions when the load is very high.

According to the theoretical calculation the on-demand version should take on average about $3 * D$ time to process one request. The simulation results that include the slowdown due to collisions are higher than the theoretical calculation. The gap between the theoretical formula to the simulation results for light load cases are -2,6,21 for dimensions 4,6,8 respectively. Note that the gap is very large for $D = 8$. Unfortunetally, we still have no explaination for this large gap.

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. The legend name consists of the combination of: algorithm cycled or on-demand , MMIR value or the distribution used: "regular" or "hot spots", and the dimension.

MMIR affect of on performance

From the simulation results it is possible to see that reasonable values for the MMIR must be in the range of 5-50 messages per round. A value of 35 was found to be the value that relatively to it's size causes the smallest degradation in performance. Allowing less than 20-30 messages in a round degrade the performance to a large extent. With $D = 4$ we can see a linear degradation in performance from 70% spawn value with MMIR=10. With $D = 6$ the

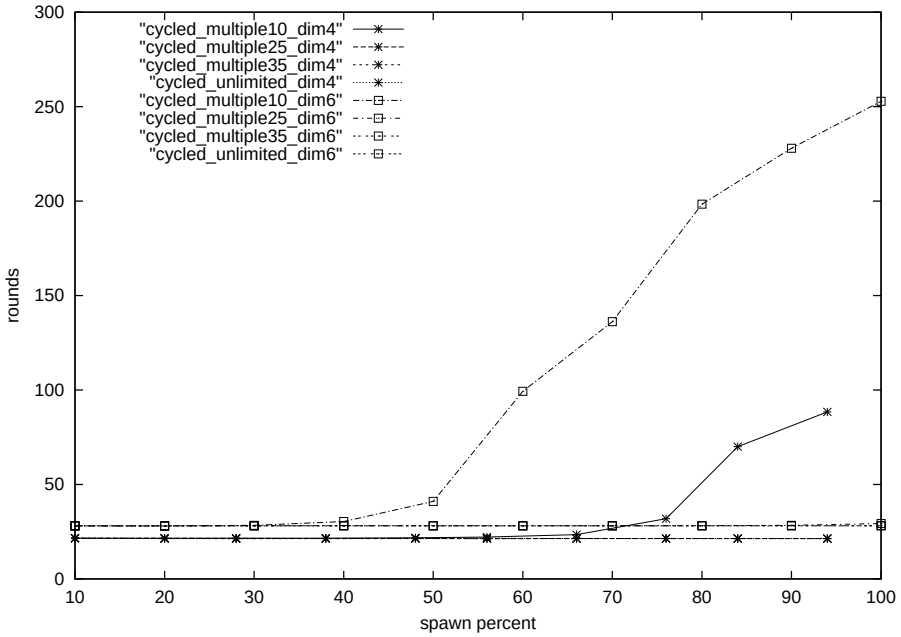

Figure 5.2: Different MMIR values for D=6 with cycled version

same effect happens at 50% spawn value, while for MMIR=25,35 there is no degradation in performance. This results refer to figure 5.2.

With $D = 8$ the same effect is even stronger as seen in figure 5.3.Even after 10% of spawns the performance is very low for not more then MMIR=10. With larger MMIR's (23,35) the degradation in the results is much slower and starts only in states where the system has a very high load.

Note : In the on-demand version the value of MMIR is meaningless.

Hot Spots affect on performance

Hot spots alone don't have any effect on the performance of both versions. This fact is seen in figure 5.4. In this figure the on-demand version was tested with hot spots values of 1,3 and 5 and with $D = 6, 8$ (the cycled version has a similar graph).

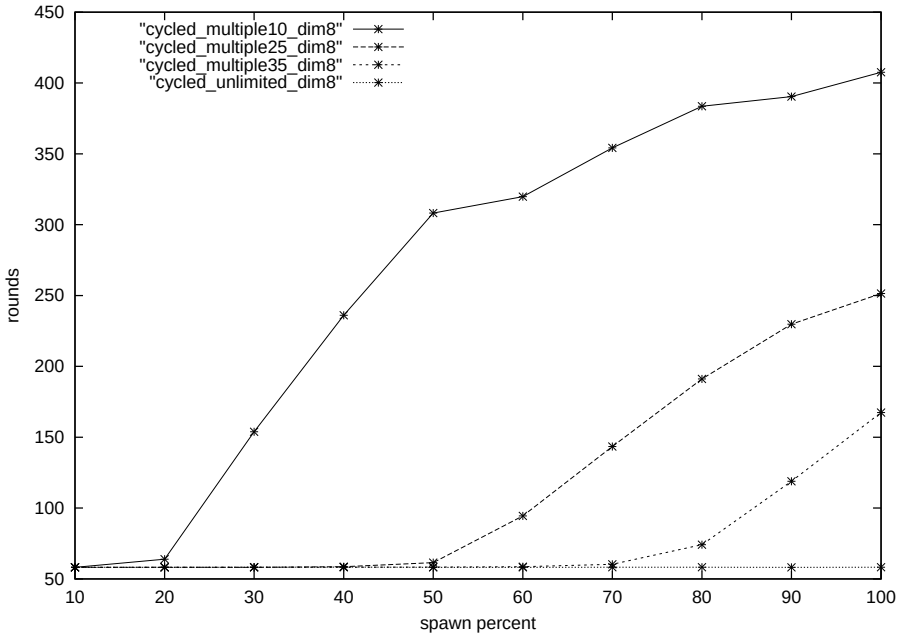

Figure 5.3: Different MMIR values for D=8 with cycled version

The combination of hotspots and limiting MMIR has very dramatic effect on the cycled version. This phenomena can be explained by the fact the hot spots causes a lot of traffic to specific nodes in the topology, and enlarges the send queue for some nodes dramatically. When in this situation the MMIR is limited then the effect is much more drastic. It can be clearly seen in figure 5.5. In this figure the cycled version is shown with $D = 6$ and with several values for hot spots.

When the number of multiple sends is not limited (as in the "cycled_unlimited_dim6", lower line) then the hot spot effect is almost negligible. However, when MMIR is low a large degradation in the results is noticeable. This effect is similar for the other two dimensions.

comparison between the cycled and the on-demand version

The comparison with and without hotspots reveal that only with $D = 4$ the on-

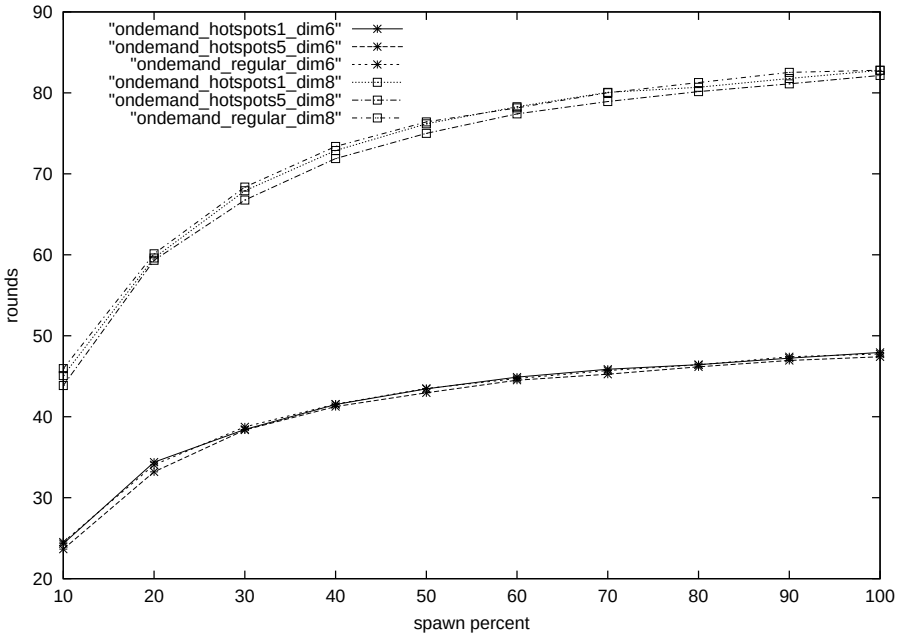

Figure 5.4: Different values for hot spots with D=6,8

demand is generally better then the cycled version. In all other cases the cycled version is better, except for very low loads. Figure 5.7 presents a comparison without hot spots for all dimensions.

With hotspots we get similar results as shown in figure 5.6.

5.5.4 Conclusion

Both the cycled and the on-demand versions are quite efficient for a general P2P communication model. The on-demand version performs well on all cases except when it is encountered with heavy load on the system. The cycled version works better than the on-demand on large number of nodes (from $D = 6$), but it has a main weakness with limiting the MMIR value (under 30-35). Hot spots do not affect both models. In both versions the average time to fulfill a send request is

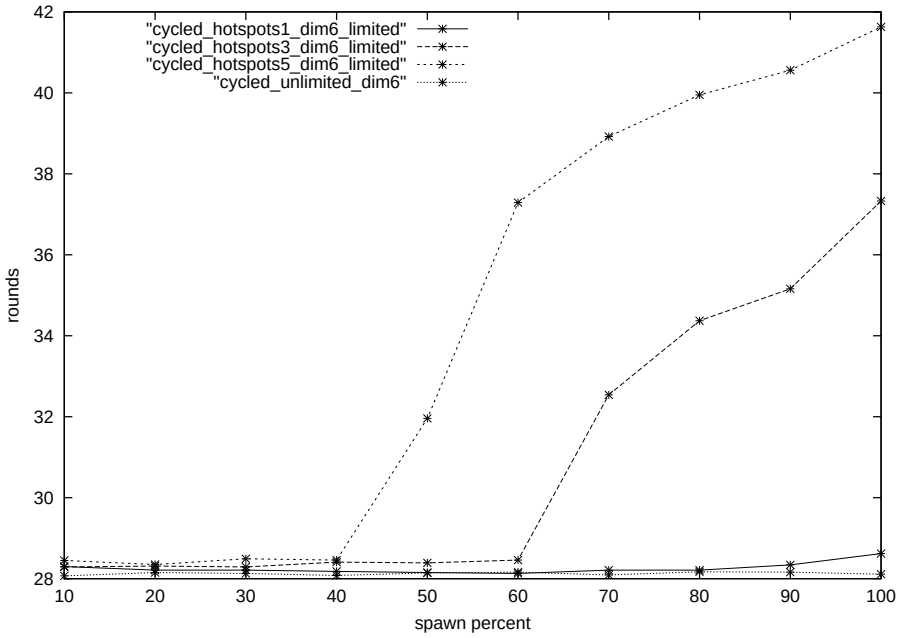

Figure 5.5: different hot spots values with MMIR=30 in the cycled version with D=6

logarithmic to the number of nodes, under certain conditions(MMIR,hotspots values).

5.6 Adaptive Multi Ring model

In the previous section we have designed a topology that uses multicast and forwarding. In this section we present the multi ring topology that also uses these two building blocks. We have also tried to add configuration changes to this model, and by that make it more efficient.

Regulars networks of low degree are easy to implement physically. Rings are an example for such networks. Rings are a very simple structure and are attractive for a network of a parallel computer. The main drawback to rings is

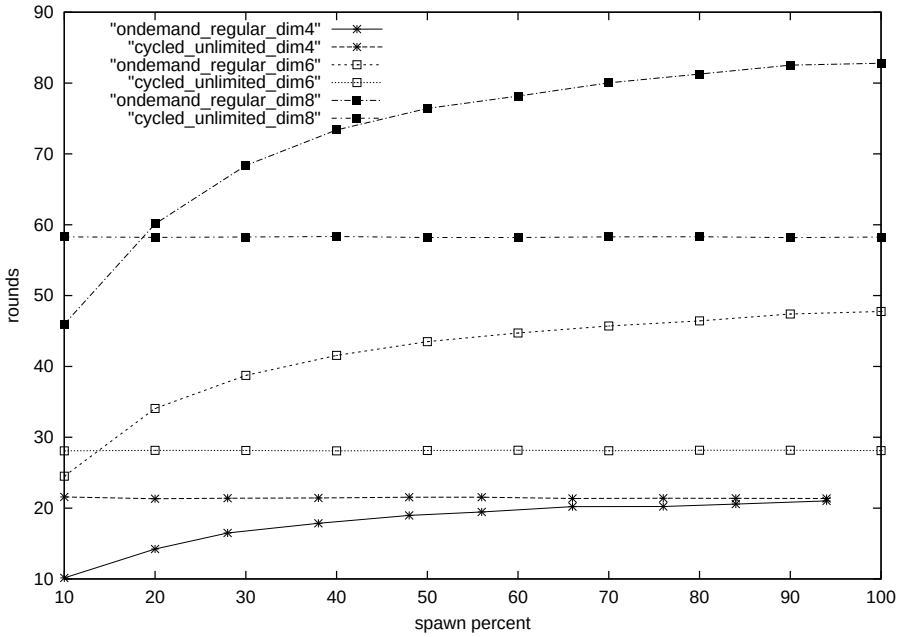

Figure 5.6: comparing on-demand and cycled versions in all dimensions without hotposts

the diameter size which is too large for more than several nodes. In order to overcome this drawback the approach of a *multi ring* is taken. A multi ring is a general name for a ring which also contains internal "shortcut" arcs between nodes. The regular arcs are called *ring arcs* while the internal "shortcut" arcs are called *chordal arcs*. This topology seemed to be a good candidate for having forwarding, configuration changes and limited multicast all together.

There are many works on this subject. These works usually concentrate on two issues. The first issue is to find an optimal static topology under the assumption of a specific communication distribution model. The second issue is to dynamically change the network topology according to a changing network load. This can be achieved by using some smart assignment of the chordal arcs while keeping the ring arcs in the original configuration. Some of these works

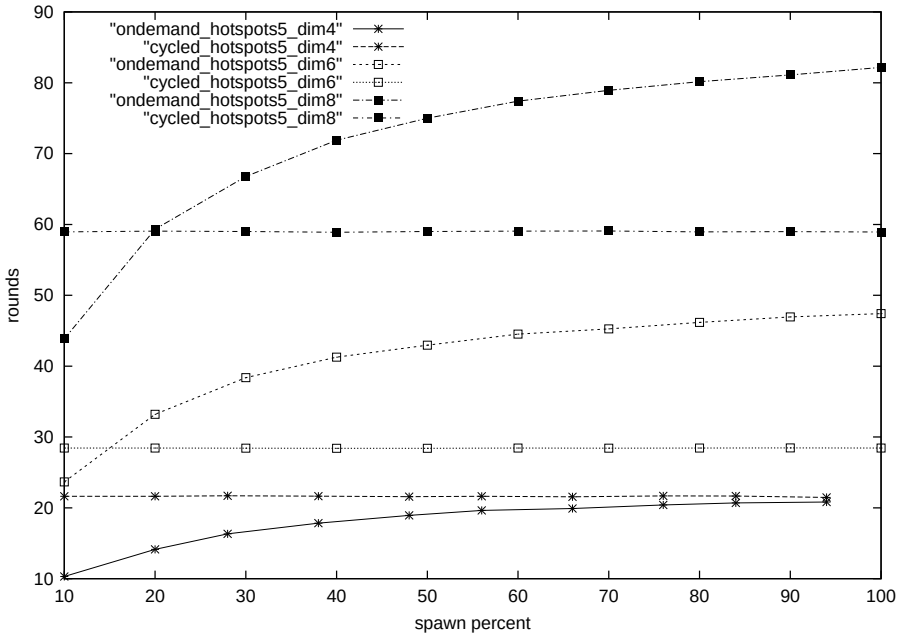

Figure 5.7: comparing on-demand and cycled versions in all dimensions with hotspots of 5%

reach an upper bound on the diameter of the network. For example : reference [1] shows an upper bound of $\sqrt{N}$ to the diameter of the network that contains N nodes. Other works have described several smart algorithms for dynamically finding the shortest path with recalculations that are done on each hop of an advancing message, etc. Our model has special requirements so it is not possible to just take another work and fully implement it in our model. In our work, we have looked for ways to convert some of the ideas behind methods described in other works to our model.

This section is organized as following: First we formalize the definition, assumption and semantics of the multi ring topology in our model, Second we examine static multi ring topologies and investigate what is the best static assignment for the chordal arcs. Finally, we try to add configuration changes and see if it

is profitable in this model.

5.6.1 Model assumptions and semantics

A configuration change request is implemented as a broadcast query on the control network such that each node knows the exact topology as it changes. Therefore, we can assume that each node holds a shortest path routing table to all the other nodes in the system. Each time a configuration change occurs, this table is recalculated to reflect the possibly new routing.

In a ring that contains N nodes, the nodes are numbered sequentially from 0 to $N-1$. In this context, an *even* node means a node which has an even number, and an *odd* node means a node which has an odd number.

5.6.2 Structure

Topology

The topology is based on a regular **uni-directional** ring with each node having one additional chordal ring. The switch is configured such that each node is able to send messages to two other nodes using limited multicast. The first node is the next node in the ring. The second node is the node connected to the chordal arc. Each node can only have two incoming links, the first link from the previous node and the other link from another node connected to it by a chordal arc. An example of a multi ring with fixed assignments can be seen in figure 5.8. In this figure $N = 16$ and for each node i there is a chordal arc to node $i + 4 \, modulo \, N$. The chordal arcs can be used to reduce the number of hops until the destination is reached. In this example the diameter of the network is 4.

Synchronization

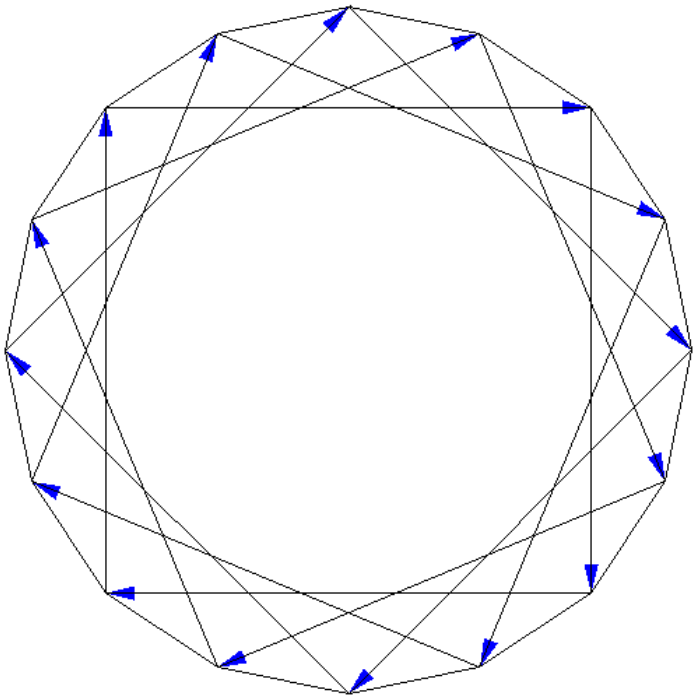

Figure 5.8: A multi ring with a fixed assignment.

In order to insure the property 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 with respect to their number. This means that even nodes can **only** be connected to even nodes and odd nodes can **only** connect to odd nodes.

There are two modes of action. Each mode is valid for a constant period of time called a *round*. Afterwards, the current mode is changed to the other mode. It is possible to synchronize all the nodes to the correct mode in the same method as the one used in the CCC model. In the first mode only even nodes are allowed to send data and in the other mode only odd nodes are allowed to send data. This method insures that there will not be any node which gets data from more then one other node in the same time, at the expense of decreasing the available

bandwidth by a factor of two. It is easy to prove that this method really insures the required property. For example, lets examine an even node with the number of 2i. In the mode where even nodes can send data it can only get messages from the chordal arc, which also has an even number. The previous node, 2i-1 can not send it data in this round, since it has an odd number. In the other mode, where only odd nodes can send data, node 2i can only receive from it's previous node 2i-1 and not from the chordal node since the chordal node has an even number. The same logic works for odd nodes of the form of 2i+1.

Initial configuration

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

- Every node's connection is set by a specific formula in the form of $chordal(i) = j$. An example of such a formula that we have decided to examine is that every node i is connected to node $(i + w)\, modulo\, N$ for some w. This configuration will be called a *fixed* configuration.
- Use a *random* non-fixed assignment in which each node is connected to some other node at random.

Both method must restrict their assignments such that it would comply with the requirement noted above.

Multiple send in the same round

Each link has large bandwidth but it is not unlimited after all. This means we can not send an infinite number of messages in the same round on the same link. Therefore, the number of messages should be limited to a constant number,

called MMIR. The same reasoning for the size of the MMIR in the CCC model (section 5.4.7) applies here.

5.6.3 Routing in Multi Ring

The routing algorithm is straight forward. For a fixed configuration the following method is used: First, use the chordal arcs to jump to the correct segment of the destination node. Since the ring is uni-directional 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. A shortest all-to-all paths table is kept on each node. Each node can independently calculate what is the shortest path to every other node. When performing a send operation the node sends the data on the first hop of this shortest path. Any change in the configuration of the chordal arcs causes an immediate recalculation of the nodes shortest paths tables. This method is used to support algorithms that dynamically set the chordal arcs.

5.7 Adaptive Multi Ring evaluation

5.7.1 The simulation

The parameters for the simulation are the same as those described in the CCC mode (in section 5.5.2):

- The number of nodes (N) in the multi ring. We have investigated the following node numbers: 64 , 384 and 2048. These figures represent small, medium and large systems, and were chosen for an easier comparison with the CCC model which also uses these figures.

- The maximum number of new events that are spawned each round. Similar to the CCC model, this number is called *spawn value*, and the values this number can accept are the same values as in the CCC model. Unless stated otherwise the spawn value used is equal to the number of nodes tested.

- The maximum number of messages that can be sent in one round from one node is limited by a number we will call MMIR. When it is not explicitly stated we will use MMIR=35 as in the CCC model.

- In a fixed configuration w is the arc length of a chordal arc. Unless stated otherwise we have used $w = \sqrt{N}$ due to the result in reference [7] which claims this figure is optimal.

5.7.2 Theoretical analysis

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.

In the average case with a fixed configuration the distance between two random nodes is $\sqrt{N}$ hops (The calculation is simple and not interesting, so it is omitted).

When using a random configuration or a dynamic configuration it is difficult to theoretically estimate the distance between two random nodes. This calculation is difficult both for the average case and for the worst case. We have used the simulation results to measure the average case figure.

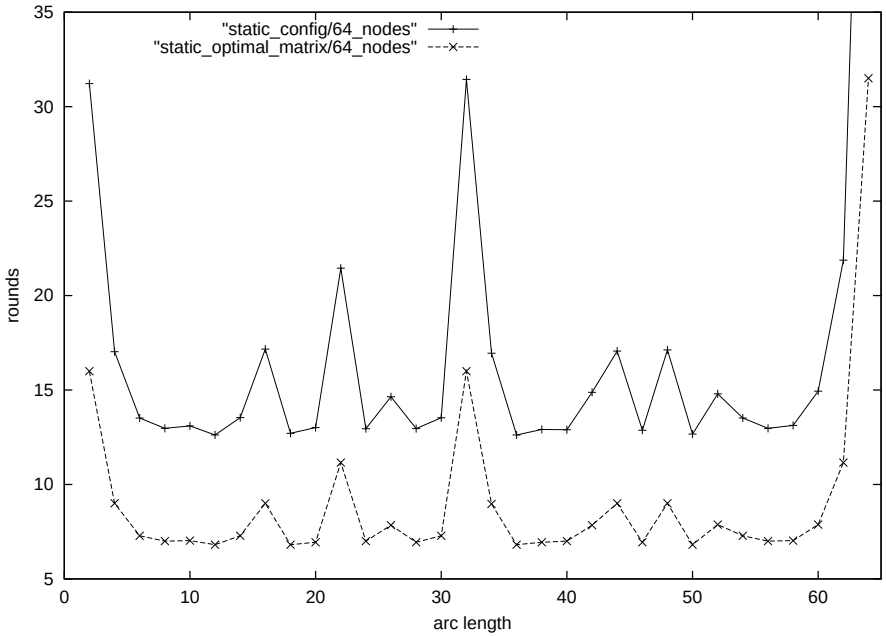

Figure 5.9: Static configuration and optimal static configuration average number of hops

5.7.3 Simulation results

Static fixed configuration

We were interested in finding the optimal figure for w, in a static configuration. We have checked varying numbers of w, from 0 to the number of nodes. Our simulation results for $N = 64$ appear in figure 5.9.

The dotted line represents an optimal average number of hops between any source and destination. This value was calculated in the following manner: First, the arcs size were set to the current value. Second, an all-to-all shortest path algorithm such as Floyd was employed on all the nodes. Then, the average path distance was calculated for all the nodes, assuming that all the paths have an equal weight in the final average . The other line was calculated in a regular simulation with spawn value equal to the number of nodes. Therefore,

the average number of hops should be higher than the optimal values since the optimal values do not include any load on the system. This exactly matches our results. Two other interesting phenomena can be seen: First, there is a remarkable correlation between the two graphs. This correlation means that there is a correlation between the arc length in a static configuration to the performance of the system (under our assumption). Second, the two graphs have a symmetry between the left part and the right part of the graphs. There seems to be a good correlation between the average number of hops for $w = i$ and for $w = N - i$. The correlation can be intuitively explained with the following: let's say we examine $w_0 = \frac{N}{4}$ or $w_1 = \frac{3N}{4}$. In both cases $N - w_j = w_{1-j}$. We try to find the shortest path from node 0 to node $\frac{N}{4}$ and to node $\frac{3N}{4}$. With w_0 it will take 1 hop to reach the first node and 3 hops to reach the second node. With w_1 it would take 3 hops to reach the first node and 1 hop to reach the second node. This means that for each i: $w = i$ and $w = N - i$ are in average equal with respect to the average number of hops between a random source and a random destination.

According to reference [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. Other values that are optimal are $\sqrt{N} * i$ for $i = 1, 3, 5...$. Note that only odd multiples of $\sqrt{N}$ are optimal and even multiples result in values that are clearly not optimal. This phenomena is caused due to the parameters of the system. We have taken the "number of nodes" parameter to have an even square root. When the ring has an even number of nodes then multipels of even numbers repeat the same points after a full circle. On the contrary, odd multipels do not repeat the same points after a full round and thus give a better coverage of the ring. There are also other values for w that are close to optimal but they are usually found around the $\sqrt{N}$ multiples.

Dimension	Optimal fixed	Average fixed	Optimal random	Average random
4	6	12	4	7
6	18	35	7	10
8	42	84	9	14

Table 2: Number of rounds needed to finish a random event for a fixed/random configuration

Static random assignment

The topology presented above is very ordered. For every node i there is a chordal arc to node $(i + w)\, modulo\, N$. We have tried to find other topologies that yield good results. The topology found is that of a random assignment for the chordal arcs. We have found out that a random assignment of the chordal arcs outperforms the fixed assignment of those arcs as it was described above. Table 2 summarizes random configuration results versus fixed configuration results. For each mode (fixed or random) there are two columns: The first column shows the results of the theoretical optimal number of hops as calculated in the previous section. The other column shows the results of a real simulation with spawn values equal to the number of nodes. It is clearly seen that the random configuration yields better results than the fixed configuration in both cases. The source for the superiority of the random version is the low diameter size of the network. We think that in a successful random configuration the diameter of the network gets closer to $log(N)$. Table 2 shows the optimal number of rounds in the random version. This figure is actually the average diameter size and is very close to $log(N)$. In the fixed version the diameter is always $\sqrt{N}$. The gap between the diameter sizes explains the superiority of the random version.

MMIR values

Contrary to the CCC model, the effect of changing the MMIR in the multi ring topology is much smaller. Simulations show that if the MMIR is not limited than the largest number of messages sent in parallel is usually around 10. From

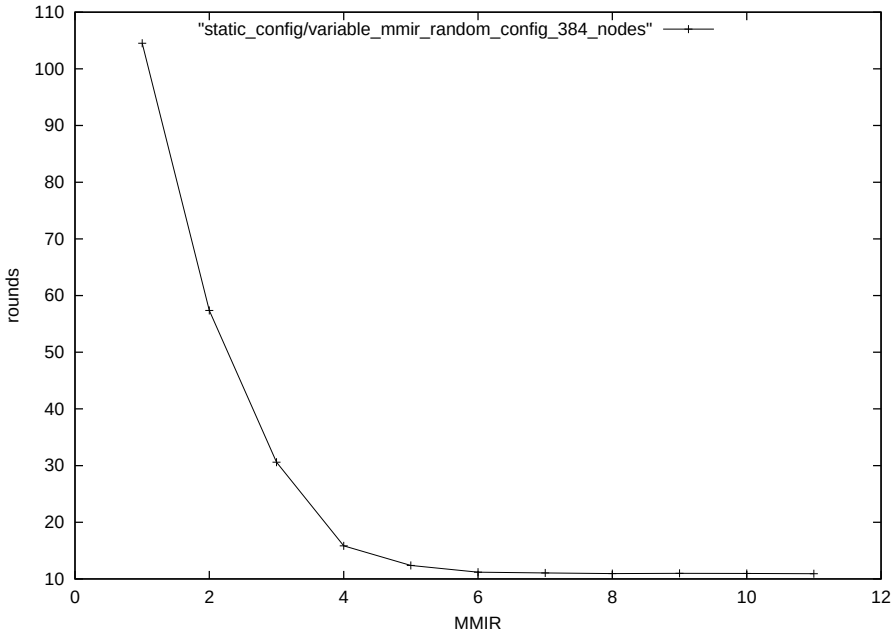

Figure 5.10: Variable values of MMIR with a static random configuration where $N = 384$

the simulations, we have found out that using MMIR=20 insures that we will not suffer almost any slowdown due to blocked messages. For example with $N = 384$, figure 5.10 illustrates the effect of changing the MMIR value on a static random configuration. In spite of this fact we have used MMIR=35 as in the CCC model to ease the comparison between these two models.

Hot Spots affect on performance

Small values of hot spots percentage do not affect the performance of the multi ring. The degradation in performance is linear with respect to the hot spots percentage. The explanation to the linear degradation is that this degradation is caused by messages that are blocked due to exceeding the MMIR value. The results are shown in figure 5.11.

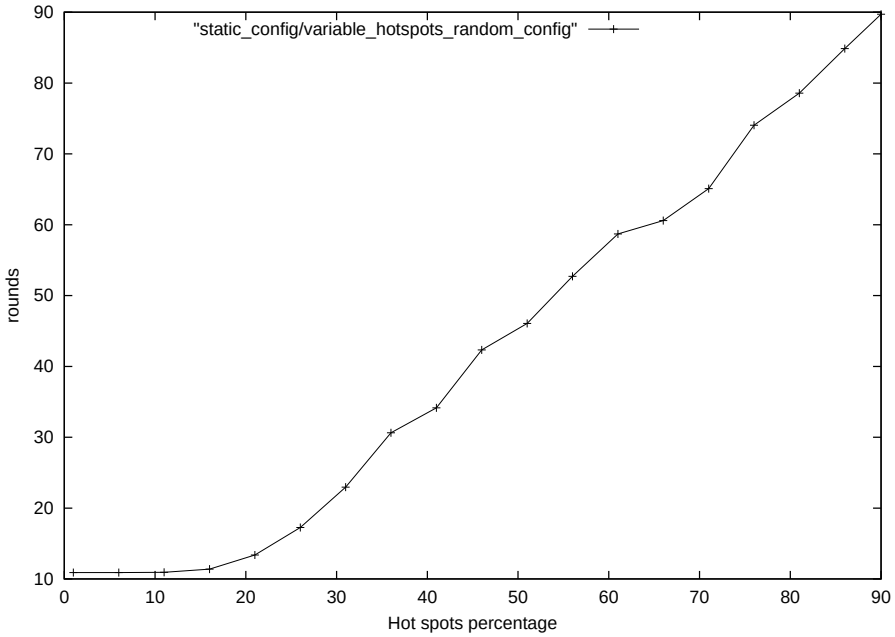

Figure 5.11: Variables values of hot spots with a static random configuration where $N = 384$

5.7.4 Conclusion

The multi ring approach seems to achieve relatively good results. The results of the simulations show that the average number of rounds is proportional to the logarithm of the number of nodes. The algorithm works well when the configuration of nodes is set at random. Using small percentage of hot spots and small values of MMIR affect the overall performance in a mild manner.

5.8 Comparison between CCC and Multi Ring

It is clearly evident that the multi ring model yields better results than the CCC model. A summary of the results is shown at table 3.

The reasons for the advantage of the multi ring model:

Dimension	Average # of round with cycled/on-demand CCC	Average # of rounds with multi ring
4	21 / 10-21	7
6	28 / 24-47	10
8	58 / 45-82	14

Table 3: Average number of rounds needed to finish a random event

1. In the multi ring the limited multicast is done on two nodes and in the CCC model the limited multicast is done on three nodes. The overhead to synchronize three nodes is larger than the overhead to synchronize two nodes.

2. The random topology in the multi 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 in the CCC model:

1. The routing in CCC is straight forward. In the multi ring each node must keep the whole topology and the corresponding routing tables.

2. Our discussion was restricted to the communication pattern of random messages. However, people that write programs for real systems, usually 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 multi ring model, which uses a random assignment.

5.9 Attempt for a profitable configuration changes model

Until now we have dealt with a model which has limited multicast and forwarding. This model has proved itself to be efficient. Our initial intention was to also use configuration changes in this model. This section is dedicated to describing several methods for deciding which configuration to change and when to perform the change. Although these methods seemed to be promising, they were discovered to slow down the system, and did not improve the overall performance.

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 of a gain in 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. Since the distribution is biased we have assumed that there might be better configuration than a random one for certain scenarios.

5.9.1 Requirements

- The algorithm should not make too many configuration changes due to the physical nature of the system.
- Every change should maintain the properties of the multi ring. Local changes are preferred.

5.9.2 Semantics

- $i \rightarrow j$ notes that node i sends a message to node j.
- $i \Rightarrow j$ notes that node i is configured to send messages to node j.
- *strength(i,j)* is defined to be an integer that should reflect the load on the link from i to j. This strength is initialized with a constant value which is some small integer. Each message that is sent from $i \rightarrow j$ increments the $strength(i, j)$ by one. This means that the strength will be raised for all the nodes along the path from the source to the destination. Every time one of the strength values reaches a large constant value (currently 100), all the strength values for all the pairs are divided by two. This division makes sure those values are normalized to some scale, and that previous load will have a decayed effect on the current load. The strength value is used by the algorithms to make decisions about configuration changes.

5.9.3 Algorithms

All the algorithms are based on the same idea but differ in its implementation. Lets assume that in the current configuration a new event is added which includes sending a message $i \rightarrow j$. Lets also assume that the current configuration is $i \Rightarrow l$ and $k \Rightarrow j$.

1. Now, if $strength(k, j) < C$ for some constant C then it means that not too many packets go on $k \rightarrow j$. Then we allow ourself to break this link and make a direct connection $i \Rightarrow j$. A side affect of this process is that in order to keep the properties of the multi ring we must also configure $k \Rightarrow l$. Then, the new message will take only one hop to arrive. If $strength(k, j) >= C$ we do nothing and let the message get

forwarded as usual. Of course this action can only be made accordingly to the restriction we took in section 5.6.2. If the parity of i and j are different we can not perform this optimization. Note that in about half the cases we can perform this optimization.

2. In the current configuration build a shortest path matrix that holds distances from all the nodes to all other nodes using Floyd's algorithm. Compute a weighted average on this matrix, where the weight is the $strength(i, j)$. For the first matrix we then get the average: M_1. Perform the same calculation on the same matrix, but where the configuration is changed such that $i \Rightarrow j$ and $k \Rightarrow l$. The second weighted average yields M_2. Now, if $M_2 < M_1$ then this change lowered the average and is more efficient. In this case commit the change. Otherwise, roll back the whole process. The advantage of this version is that it takes a global approach to evaluating a configuration change and not just a local perspective.

3. In version 2 we changed the configuration to point from the source to the destination when we estimated it was profitable. When the distribution is a hot spots distribution this approach has a big disadvantage. Each time a new node might point to one of the nodes in the the hot spot group. This group can be very small, and in fact in our tests it included only one node. Hence, this change will not improve the overall performance. To cope with this situation, we added the following improvement: instead of trying to configure $i \Rightarrow j$ we will try to configure $i \Rightarrow k$ where from k to j the distance is a small number of hops. The number of hops is chosen at random, but is limited by a constant number. We tried several methods: k is a ring arc before j ; k is a chordal arc before j ; or a hybrid of both methods. Using this improvement, many nodes will try to create a "short cut", chordal arc, to a random node before the hot spot and the chance of

colliding this assignment gets much lower.

4. Generate a new random configuration every random number of rounds. Do the comparison between the weighted average of the old matrix that corresponds to the old configuration and the weighted average of the new matrix that corresponds to the new configuration. In case the new matrix has a lower average value switch to the new configuration.

5.9.4 Results

All the versions having dynamic configuration performed worse than the static configuration. We do not present the exact results since none of the versions presented showed any sign of performing well, even on fractions 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 think we have understood the main reasons that contributed to this failure:

- Any configuration change is harmful in the short term because messages routing is based on the existing routing. Any change to the configuration can cause messages to re-route and to increase the average hop count. We tried to limit the number of configuration changes, but even that didn't help.
- Under our assumptions of hot spot distribution there is nothing the network has to adapt to. At most the dynamic model adapts to a configuration that is very similar to the random configuration it started with. A dynamic configuration is needed where the load changes. We also tried to use moving hot spots to emulate it but it was not successful either.

It seems that under the current model there can not be a better solution than the static model.

5.10 Conclusion

Both network topologies presented in this chapter yield good results in the P2P model. The multi ring model is a better general solution. The CCC model is more suited for exploiting advantages in the way people really write parallel programs.

We have tried to design a model which has all the building blocks of this system: forwarding, multicast and configuration changes. This attempt failed because of two different reasons. The first reason is that in our theoretical model (especially in the distribution model), it is probably impossible to gain any advantage with configuration changes. The second reason is the synchronization overhead. The multicast feature enhances the system but requires the added synchronization. This synchronization can eventually degrade the performance of the system more than the multicast can contribute to it.

6 Algorithm for all to all broadcast operation

6.1 Introduction

An all to all broadcast operation is an operation in which every node receives a message from every other node in the system. This operation can be easily implemented in a network of N nodes using N consecutive broadcasts, where each broadcast is done by a different node. However, this method is not efficient. Many algorithms were developed in order to make this operation effecient. Usually it was done by pipelining or merging different broadcasts. In this chapter we present a simple algorithm for all to all broadcast operation which is exactly suited for our model.

For simplicity sake, we assume that any broadcasted message can be sent in a single transmission unit, without the need to divide it into several packets.

6.2 Algorithms

We start by reminding the reader of two known solutions for the all-to-all broadcast operation. These solutions are briefly described and their running time is analyzed. It is presumed that the reader is familiar with these two algorithms, which are mentioned in order to highlight our way of measuring the algorithm's running time. Finally, we present our suggestion for an optimal algorithm, analyze it and show that it is really optimal.

6.2.1 Hypercube

This algorithm is based on hypercube topology. In stage i, a node sends all the messages it received until that stage on the link that connects it to dimension i. Therefore, in every stage each node sends twice the number of messages it did

in the previous stage. The process takes $O(logN)$ stages to finish. Although only $O(logN)$ stages are needed, the number of messages sent by each node is still $O(N)$. There are also $O(logN)$ configuration changes.

6.2.2 Tree

This algorithm is based on building a full tree with a degree that is equal to the limited multicast level. First, each node propagated its message and those received by it to the root of this tree. Second, the root multicasts all the messages to its direct links, and recursively to all the nodes of this tree. Gathering the information to the root takes $O(logN)$. Scattering the information back to all the nodes also takes $O(logN)$. Hence, the whole process takes $O(logN)$ stages to finish. Similarly to 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 one configuration change for the multicast tree.

6.2.3 Ring

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. The exact formula can be represented by: $2T_d + (N-1)S * T_f$ (the meaning for each symbol is defined in the third chapter).

6.3 Algorithm evaluation

The question that should be asked is how to evaluate the performance of these algorithms in our model. We claim that the best method to measure the performance of these algorithm is by the number of messages that are sent by each node, and also considering the number of configuration changes. The second argument is clear since configuration changes are a costly operation in our model. On the other hand, the first argument should be explained. In the following subsection we explain the intuition behind this argument and in the next subsection we present a more formal proof for it.

6.3.1 Intuition

In most systems it is extremely important to minimize the number of times a node starts to send messages. There is a large overhead for sending just a single message. This overhead is caused by the operating system's network stack. Adding more messages to an existing transmission is very cheap, in terms of time. In our system, it is possible to use message forwarding. With this feature, all messages are forwarded in hardware to their destination without reaching the operating system level. Therefore, there is no overhead for **forwarding** only one message and no advantage for **forwarding** a package that contains many messages.

6.3.2 Formal explanation

Usually global communication algorithms are measured in the number of stages that they require to finish. It is usually implicitly assumed that each stage can finish in a constant time. However, in the hypercube algorithm, there is only one message sent in the first stage, but $\frac{N}{2}$ messages sent in the last stage. These two

stages can not take the same amount of time on our model. Therefore, in this context this assumption is invalid. The only solid figure that can be measured is the number of messages sent by each node. Without any configuration changes (as presented in the ring algorithm), this figure determines the minimal running time of the whole process. Now it is left to show that under this criteria the ring algorithm we presented is really optimal. Consider the following arguments:

1. In an all-to-all broadcast operation each node must by definition receive N (different) messages from other nodes. There can not be any overlapping in the receive operation of every node. Only one message can be received at a time.

2. A node can only send k messages in parallel using limited multicast where k is a constant that is not related to N. In reality k is limited by the optic medium to about 4. Hence, in average, each node must send at least $O(N)$ messages so the condition in the first point will hold. Also here, there can not be any overlapping send operations.

3. In the ring algorithm presented above, each node sends and receives $O(N)$ messages.

Due to these arguments our algorithm is equivalent to the optimal algorithm, in terms of the number of messages sent. In this algorithm there is only one configuration change which is the minimum number of configurations possible. Thus, our algorithm is optimal in the two measures: the number of configuration changes and in the number of messages sent. This means that our algorithm is optimal. Of course, a constant improvement might be possible, but only a constant one. The optimal algorithm must remain at least $O(N)$ (messages).

6.4 Conclusion

There is a basic difference between our evaluation model to the common one. Our model is best evaluated by the number of messages sent by each node in the whole operation and by the number of configuration changes. Therefore, in our model it is best to use a ring topology with the simplest algorithm presented above to perform an all-to-all broadcast operation.

7 Conclusion and future work

7.1 Conclusions

This work is based on a novel optical switching technology. This technology can be the basis for the next generation of networks for parallel computers. In spite of the great advantages of this technology, there are still some open issues that must be deeply examined before this technology can become practical. We have chosen the slow configuration time, which is one of the main issues, as the center of our study. Our model is composed out of three main elements: forwarding, multicast and configuration changes. It was noticed that in general, the successful models included a mixture of using these three elements.

In the first part of the thesis, we have developed models for performing efficient broadcast and have theoretically analyzed them. These models used configuration changes and forwarding. We have found out that these models are best based on a variation of trees. In trees the running time can be logarithmic and the initial configuration can be done in the first phase, which saves some of the configuration changes in the following phases.

Second, we have developed algorithms for a P2P communication model and used simulations to evaluate their performance. In these models we also used the multicast feature. We have worked on two models: cube connected cycles and multi ring. Both of these models used multicast and forwarding without configuration changes. An important conclusions of the study in this section is that in our model (P2P communication) it is unlikely to have all the three building blocks working together successfully. This is why our two models didn't use configuration changes.

In the last part of the thesis we have examined algorithms for an all-to-all broadcast operation. We came to the conclusion that the optimal algorithm for

this operation, under the assumptions of our model, is based on a ring topology. This conclusion is backed up by theoretical arguments.

7.2 Future work

In the theoretical area it would be preferable to work on global communication operations such as: all-to-all, global exchange, etc.

In the practical aspect, the building of the prototype should be finished. Then, it would be possible to conduct tests on it, using the algorithms presented in this work. Once this happens, the algorithms effectiveness could be checked in a real system and not only on simulations. It is also known that good protocols and algorithms usually need some fine tuning in real systems to achieve optimal performance. When the system is built, a session of fine tuning on the algorithms should be conducted.

References

[1] Bruce W. Arden and Hikyu Lee. Analysis of chordal ring network. *IEEE Trans. Computer*, 30(4):291–295, Apr 1981.

[2] Ralph Ballart and Yah-Chau Ching. Sonet: Now it's the standard optical network. *IEEE Communications Magazine*, pages 8–15, March 1989.

[3] Meliksetian D.S and Chen C.Y.R. Optimal routing algorithm and the diameter of the cube-connected cycles. *IEEE Trans. Parallel and Distributed Systems*, 4(10):1172–1178, October 1993.

[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. *SPIE Optical Networks Magazine*, 1(3):1–6, July 2000.

[5] P.E. Green. Optical networking update. *IEEE Journal on Selected Areas in Communications*, 14(5):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. *Journal of Lightwave Technology*, 14(6):1198–1206, June 1996.

[7] B. Mans. On the interval routing of chordal rings. In Zomaya, Hsu, Ibarra, Horiguchi, Nassimi, and Palis, editors, *International Symposium on Parallel Architectures, Algorithms and Networks*, pages 16–21. IEEE, June 1999.

[8] Benny Pesach, Guy Bartal, Eli Refaeli, Aharon J. Agranat, Joel Krupnik, and Dan Sadot. Free-space optical cross-connect switch by use of electroholography. *Applied Optics*, 39(5):746–758, February 2000.

[9] R. Sabella, E. Iannone, and E. Pagano. Optical transport networks employing all-optical wavelength conversion: Limits and features. *IEEE Journal on Selected Areas in Communications*, 14(5):968–978, June 1996.