Performance Modeling of Subnet Management on Fat Tree InfiniBand Networks using *OpenSM*

ABHINAV VISHNU, AMITH R MAMIDALA, HYUN-WOOK JIN AND D. K. PANDA

Technical Report
OSU-CISRC-1/05-TR05

Performance Modeling of Subnet Management on Fat Tree InfiniBand Networks using *OpenSM* *

Abhinav Vishnu Amith R Mamidala Hyun-Wook Jin Dhabaleswar K. Panda
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{vishnu, mamidala, jinhy, panda}@cse.ohio-state.edu

Abstract

InfiniBand is becoming increasingly popular in the area of cluster computing due to its open standard and high performance. Fat Tree is a primary interconnection topology for building large scale InfiniBand clusters. Instead of using a shared bus approach, InfiniBand employs an arbitrary switched point-to-point topology. In order to manage the subnet, InfiniBand specifies a basic management infrastructure responsible for discovery, configuration and maintaining the active state of the network. In the literature, simulation studies have been done on irregular topologies to characterize the subnet management mechanism. However, there is no study to model subnet management mechanism on regular topologies using actual implementations.

In this paper, we take up the challenge of modeling subnet management mechanism for Fat Tree InfiniBand networks using a popular subnet manager OpenSM. *We present the timings for various subnet management phases namely topology discovery, path computation and path distribution for large scale fat tree InfiniBand subnets and present basic performance evaluation on small scale InfiniBand cluster. We verify our model with the basic set of results obtained, and present the results for the model by varying different parameters on Fat Trees.*

1 Introduction

In the past couple of years, the computational power of commodity PCs has been doubling about every eighteen months. At the same time, commodity network interconnects that provide low latency and high bandwidth are also emerging. This trend makes it very promising to build high performance computing environments by using *Clusters*. It combines the computational power of commodity PCs and the communication performance of high speed network interconnects. Fat Trees [6] are being used as a primary interconnection topology for building large scale clusters. Recently, InfiniBand Architecture [5] has been proposed as the next generation interconnect for I/O and inter-process communication. Due to its open standard and high performance, InfiniBand is becoming increasingly popular for cluster computing.

InfiniBand defines a technology for interconnecting the I/O nodes with the processing nodes, which forms a *System Area Network (SAN)*. The end nodes are connected to each other in a switched point-to-point fashion. The end nodes can have one or more *Channel Adapters* to connect to the network.

InfiniBand specifies a small number of management classes. It specifies a *Subnet Management* mechanism which monitors the state of the subnet and takes appropriate steps to assimilate topology changes in a smooth fashion. It specifies an entity called *Subnet Manager (SM)* which is in charge of discovery, configuration and maintainence of the subnet.

A lot of simulation based studies have been presented in the recent past to evaluate the subnet management mechanism [3] [4]. However, these studies are based on irregular topology networks and lack the management model on regular topologies using actual implementations.

In this paper, we take up the challenge of modeling subnet management mechanism for Fat Tree InfiniBand networks using *OpenSM [1]*. We present the timings for various subnet management phases in InfiniBand [3] such as topology discovery, path computation and path distribution for large scale fat tree InfiniBand networks. We also present the timings on a small scale InfiniBand cluster with different configurations to understand the behaviour of subnet manager. We verify our model with the basic set of results ob-

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, National Science Foundation's grants #CCR-0204429 and #CCR-0311542, and grants from Intel and Mellanox, Inc.

tained, and present the results for the model by varying different parameters on Fat Trees.

The rest of the paper is organized as follows: In Section 2, we provide background information for InfiniBand, Subnet Management and Fat Trees. In Section 3, we discuss various phases in subnet management. In Section 4, we present the timings for various management phases using *OpenSM* for small scale InfiniBand network configurations. In Section 5, we present the subnet management model on Fat Tree Networks. In Section 6, we show the performance results using our management model and validate it by using the timings from section 4. In section 7, we present the related work of this paper. In section 8, we conclude and discuss the future directions.

2 Background

In this section, we provide background information for our work. We first provide a brief introduction of InfiniBand Subnet Management machanism and then discuss about fat tree topologies used in building large scale clusters.

2.1 InfiniBand Subnet Management

The InfiniBand Architecture (IBA) [5] defines a switched network fabric for interconnecting processing nodes and I/O nodes. It provides a communication and management infrastructure for inter-processor communication and I/O. In an InfiniBand network, processing nodes and I/O nodes are connected to the fabric by *Channel Adapters* (CA). *Host Channel Adapter* (HCA) sit on processing nodes.

InfiniBand defines a small number of management classes. The Subnet Management class defines an entity called *Subnet Manager (SM)*, which is in charge of discovering, configuring, activating and managing the subnet. The subnet manager can reside in any subnet device (Switch, Router or Channel Adapter (CA)). By using the subnet management interface (SMI), SM can exchange control packets with the subnet management agents (SMA) present in every subnet device. The SMI is associated to an internal management port in switches, or to a physical port in the rest of the devices. Figure 1 shows the physical model of the subnet management.

The control packets used by subnet management are called subnet management packets (SMPs). Each SMP consists of 256 bytes of data. SMPs use the Unreliable Datagram service, which contain a key to authenticate the sender and use the management virtual lane (VL15).

The SMPs can be classified into *LID routed* and *Direct routed* SMPs. LID routed SMPs are routed by switches by a table lookup of the forwarding table. They are mostly used to check the status of active ports in the subnet. The data packets are routed in the same way as LID routed SMPs. Direct routed SMPs contain the information of the output port to which they need to be fowarded to at each intermediate hop. Direct routed SMPs are used during the subnet discovery before the subnet initialization. LID routed SMPs have lesser overhead than direct routed SMPs, because the direct routed SMPs have to be processed at each intermediate SMI.

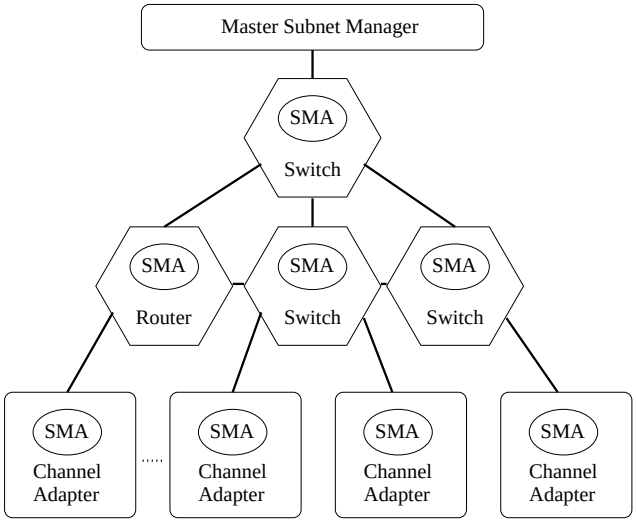

Figure 1. Physical Model of Subnet Management

The SMP header defines the operation to be performed by the SM. These subnet management operations are : *Get*, *Set*, *GetResp*, *Trap* and *TrapRepress*. The *Get* operation is used to get the information about the CA, Switch or a Router port. The *Set* operation is used by the subnet manager to set the attributes of a port at the end of the subnet discovery. The *GetResp* operation is the response to the *Get* SMP of a subnet manager.

The subnet manager performs a sweep to discover the subnet. In addition to sweeping, a switch may optionally inform the SM about the change of the state of a local node by sending a *Trap* notice SMP. In addition, the SMA may periodically repeat the *Trap* message until it receives a notification from the SM to stop the trap by sending a *TrapRepress* message.

The SMI injects SMPs generated by the SM and SMAs into the network. In addition, it validates and delivers incoming SMPs.

SMAs are passive management entities. SMAs process received SMPs, respond to the SM, and configure local components according to the management information received. The received SMPs can contain information about LID assignment of physical ports, the state of the ports, and the number of data operational VLs. Other SMPs are used to update the local forwarding table, the SL to VL mapping table, and the VL arbitration tables. In switches, the SMA

can send traps to the SM to notify the change in state of a local port.

SM is a management entity which manages the subnet. With the help of SMPs the SM is able to discover the subnet topology, configure subnet ports and switches, and receive traps from SMAs. In addition to the Master SM, multiple Subnet Managers may be present in the subnet. However, only one of them can be the master SM. The rest of the Subnet Managers are *Standby SMs*. The standby SMs periodically poll on the master SM and wait for a successful response. In case of failure of the Master SM, one of the Standy SMs becomes the master SM.

2.2 Fat Tree Topology

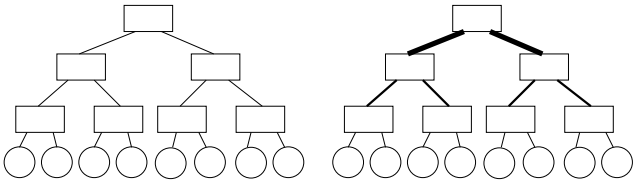

Figure 2. (a)Traditional Binary Tree (b)Binary Fat Tree

Fat tree is a general purpose interconnection topology, which is used for effective utilization of hardware resource devoted to communication. Fat trees differ from the traditional trees in the amount of resource bandwidth available at different levels of the tree. In traditional trees, the link bandwidth is fixed at all levels of the tree. Due to this configuration, there is congestion near the root of the tree.

In a Fat tree, the link bandwidth is higher near the root in comparison to the leaves. For a complete Fat tree, the link bandwidth doubles at every level, starting from the leaves. Figure 2 shows the difference between a traditional fat tree and a binary fat tree. In a Fat tree based interconnection network, leaf nodes represent processors, internal nodes represent switches, and edges correspond to bidirectional links between parents and children.

3 Subnet Management Mechanism

In this section, we present various phases of subnet management in InfiniBand. A more detailed description is present in [3]. The topology discovery phase consists of sending direct routed SMPs to every port in the subnet and processing the responses. The path computation phase comprises of computing valid paths between each pair of end nodes. The path distribution phase consists of configuring forwarding table on switches.

3.1 Subnet Discovery

In order to start the discovery process, the SM sends a *Get* SMP message to its local node using an empty directed path, and waits for the response. Upon receipt of the response, it sends direct routed SMPs with additive depth to facilitate the discovery of other nodes in the subnet and so on. During the discovery process, the SM configures the port attributes, sending *Set* SMPs to the CA and switch ports. A subnet manager may decide to send multiple exploration SMPs to discover the subnet at any point in time [3]. However, *OpenSM [1]* uses one outstanding SMP for exploration, in order to perform controlled breadth first search. The subnet discovery phase can be characterized into multiple phases:

- Setup time for switch ports
- Setup time for CA ports

OpenSM sends a *Get* SMP with *NodeInfo* attribute to get the kind of end node (Switch, CA or Router) and a *Get* SMP with *PortInfo* to get the port attributes. It sends a *Set* SMP to set the LIDs and other port attributes. *OpenSM* provides an option of Trap based subnet discovery. In order to allow a switch to notify the subnet manager about topology change, SM initializes every port on the switch, even though it may not have any active connection with a CA.

3.2 Path Computation

Once the topological information of the subnet is received, the subnet manager needs to compute the paths for data packets to follow in order to reach the destinations. This would constitute the computation of forwarding tables. The InfiniBand specification does not impose any specific routing algorithm for the path computation. When multiple paths are available between the end nodes, *OpenSM* selects the one with the least number of hops. In case of a tie, one of the available paths is chosen randomly. We characterize the path computation time as a function of the number of end nodes in section 5.

3.3 Path Distribution

Once the paths to every device in the subnet have been computed, the forwarding tables at each switch need to be configured for routing of data packets. InfiniBand specification clearly defines the SMPs for updation of switch forwarding tables. However, the update order is not specified. *OpenSM [1]* performs the forwarding table configuration phase once the path computation phase is complete. This policy is optimal, because during the complete subnet discovery, nodes are not ready for communication, unless the whole subnet is in *ACTIVE* state.

4 *OpenSM* Basic Performance

In this section, we present the performance of *OpenSM* on a small scale InfiniBand cluster. We consider a set of configurations: A 1-switch configuration, which consists of an 8-port crossbar switch and 6-switch configuration which consists of six 8-port crossbar switch blocks connected in a fashion as shown in the Figure 3.

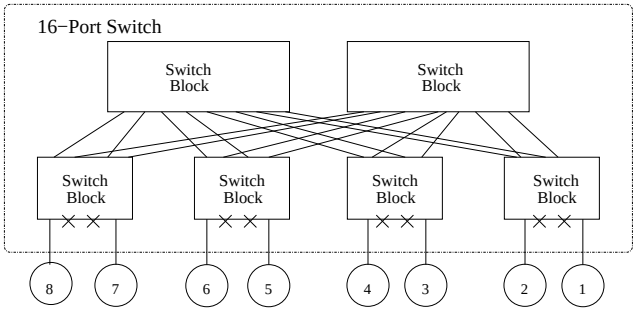

Figure 3. The 6-switch Configuration

For the 6-switch configuration, in order to take the results for *k* nodes, the nodes numbered from 1 to *k* were used. An x represents an unused port in the switch.

4.1 Total Discovery time

Figure 4 shows the total discovery time for multiple switch configurations. The total discovery time increases with the increasing number of nodes. We divide this time in multiple management phases and show the results in the following sections.

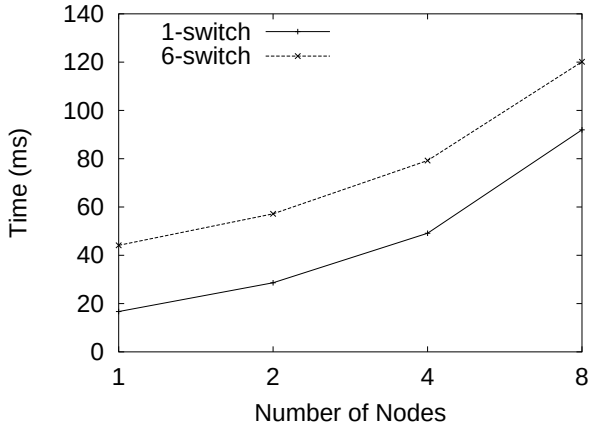

Figure 4. Total Discovery time

We notice that the total discovery time for any of the configurations above does not increase exactly in a linear fashion. This is because, even if a subset of switch ports are connected to end nodes, *OpenSM* needs to setup all the switch ports to facilitate trap based discovery. Hence, as shown in Figure 7, the switch setup time remains constant with increasing the number of end nodes.

4.1.1 Subnet Discovery

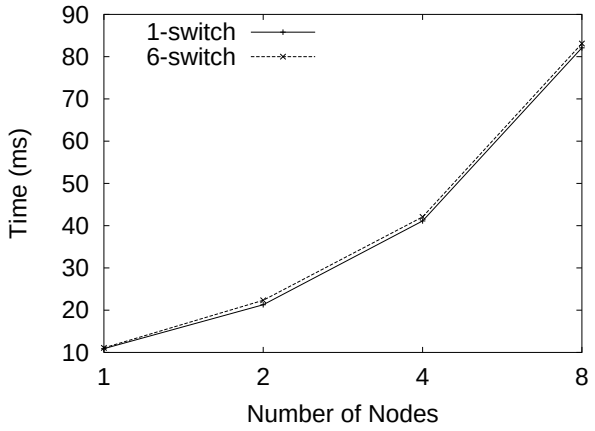

Figure 5. Path Computation Time

The channel adapter setup time remains almost the same for both 1-switch and 6-switch configurations. There is a slight difference for 8 nodes on 1-switch in comparison with the 6-switch configuration as shown in Figure 6. This is due to the difference in number of hops and hence increased SMI processing for the 6-switch configuration. However, SMI processing time is much smaller in comparison to the total setup time and hence we ignore its effect to simplify the performance modeling in section 5.

Once the topology discovery is complete, *OpenSM* needs to configure valid routes between end nodes.

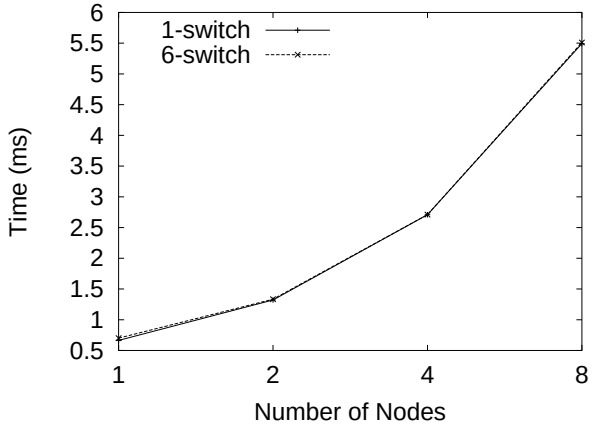

Figure 6. Channel Adapter Setup time

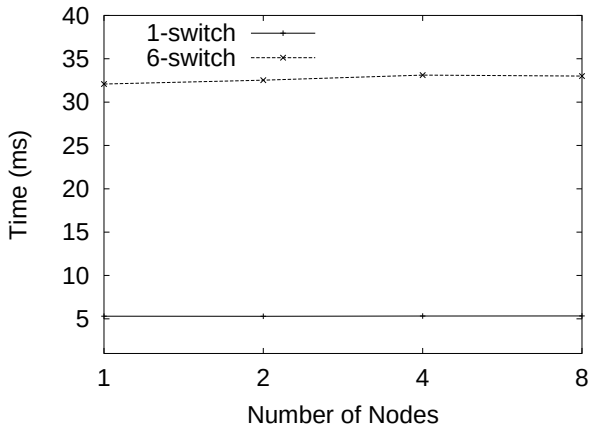

Figure 7. Switch Setup Time

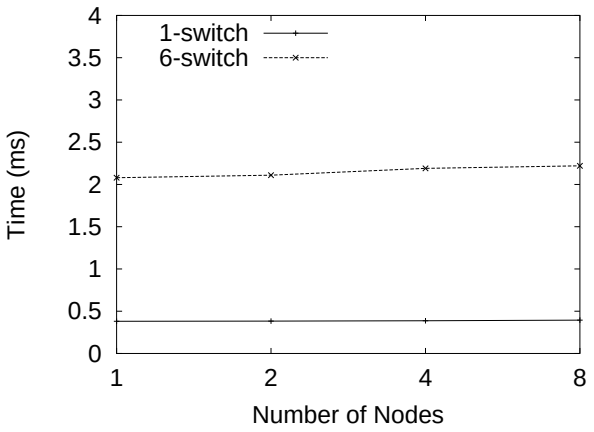

Figure 9. Path Distribution time

4.1.2 Path Computation

We calculate the overhead of path computation in terms of the number of end nodes in the subnet. From Figure 8 we can clearly see, that the path computation overhead increases almost linearly with the number of end nodes. The path distribution phase consists of distributing the forwarding tables to the switches in the subnet. Clearly, it is dependent on the number of switches and number of entries which need to be configured per switch. As shown in Figure 9, for a 6-switch configuration, for the same number of nodes, the path distribution time is higher, because the SM needs to configure the forwarding table on 6 switches instead of 1-switch.

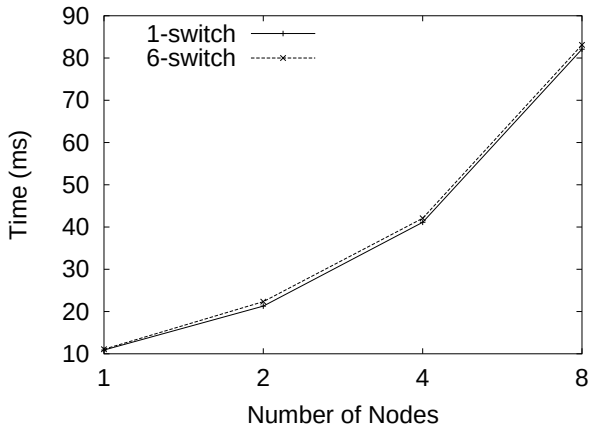

Figure 8. Path Computation Time

4.1.3 Path Distribution

It is interesting to see that the path distribution time remains almost constant with increasing number of nodes. This is because, the per table entry overhead is very small. Each table entry occupies 16-bit for LID and 8-bit for the forwarding port. Hence only one *Set* SMP with *LinearForwardingTable* attribute is required for every $\lceil MTU/3 \rceil$ nodes per switch. In our experimentation platform, the MTU is 2048 bytes and thus only one *Set SMP* is required for 8 nodes.

5 Subnet Management Model on Fat Tree Networks

Routing in a complete Fat tree is simple as there exists a unique shortest path between every pair of processing nodes [7]. However, the requirement of increasing arity of switches makes the physical implementation of switch-based Fat tree infeasible. To solve this problem, some alternatives have been proposed to construct fat trees using constant size elements or use fixed arity switches.

Based on the fixed arity of the communication switches to construct a fat tree network, we use an *m*-port *n*-tree configuration of a fat tree in our analysis and evaluation. Given an *m*-port *n*-tree, it has the following characteristics.

- The tree consists of $2 \times (m/2)^n$ processing nodes and $(2n-1) \times (m/2)^{n-1}$ communication switches
- Each communication switch has *m* communication ports

Before the end nodes can communicate with each other, the SM needs to configure the routes for the intermediate switches and end nodes after the subnet discovery. As mentioned before, to start the discovery process, the SM first sends a *Get(NodeInfo)* to its local node (using an empty directed path) and waits for a response. Each time the SM receives a response *GetResp* SMP, it sends another SMP by increasing the length of the direct route. The SM sends one

SMP per port with *AttributeID* as *NodeInfo* to determine the nature of the port and with *AttributeID* as *PortInfo* to get information about the port. Then it sends *Set* SMPs to configure the port attributes of discovered devices. Thus the total number of *Get* SMPs is the sum of SMPs with *NodeInfo* and *PortInfo AttributeID*.

Let *GetS(m,n)* denote the number of *Get* SMPs generated during topology discovery, let $GetS_{NodeInfo}(m,n)$, $GetS_{PortInfo}(m,n)$ be the *Get* SMPs with *NodeInfo* and *PortInfo* attributes respectively. Thus the number of SMPs generated during the discovery algorithm can be calculated as a function of *m* and *n*:

$$GetS(m,n) = GetS_{Nodeinfo}(m,n) + GetS_{PortInfo}(m,n) \quad (1)$$

As mentioned before, *OpenSM* uses controlled breadth first exploration SMPs. As a result, each port is disocovered only once. Let $NumPorts_{Switch}$ and $NumPorts_{CA}$ be the number of ports with Switches and Channel Adapters respectively. Hence, we have the following:

$$GetS_{Nodeinfo}(m,n) = NumPorts_{Switch} + NumPorts_{CA} \quad (2)$$

Since each communication switch has *m* communication ports, we have:

$$GetS_{Nodeinfo}(m,n) = (2n-1)\times(m/2)^{n-1}\times m + 2\times(m/2)^n \quad (3)$$

Trivially, the total number of *NodeInfo* SMPs is equal to *PortInfo* SMPs. Hence the total number of *Get* SMPs are

$$GetS(m,n) = (4n-2)\times(m/2)^{n-1}\times m + 4\times(m/2)^n \quad (4)$$

Set SMPs are required during multiple phases in the subnet management. During the topology discovery phase, they are used to configure the switch and CA ports with *AttributeID* as *PortInfo*. However, during the path distribution phase, they are used for setting up the forwarding tables with *AttributeID* as *LinearForwardingTable*. More than one *Set* SMP may be required per switch for setting up the linear forwarding table. In particular, each forwarding table entry is 3 bytes (16-bit for LID and 8-bit for forwarding port information). Since SMPs are based on UD transport service, one *Set* SMP is required for every $\lceil MTU/3 \rceil$ end nodes.

Let *SetS(m, n)* denote the number of *Set* SMPs which are generated and $SetS_{PortInfo}$, $SetS_{LFT}$ be the *Set* SMPs with attribute as *PortInfo* and *LinearForwardingTable*, respectively. Hence the total number of *Set SMPs* are

$$SetS(m,n) = SetS_{PortInfo}(m,n) + SetS_{LFT}(m,n) \quad (5)$$

$$SetS_{PortInfo}(m,n) = 2 \times (m/2)^n \quad (6)$$

$$SetS_{LFT}(m,n) = (2n-1) \times (m/2)^{n-1} \times S \quad (7)$$

where S is $(1 + \lceil 2 \times (m/2)^n \times 3/MTU \rceil)$

We now present the timings for various phases in the subnet management. We define t_{TD}, t_{PC} and t_{PD} as the time taken for topology discovery, path computation and path distribution phases, respectively.

The topology discovery time can be calculated in terms of the processing time for *Get* and *Set* SMPs. From the results of section 4, we can conclude that the processing time for both type of SMPs is nearly same. In addition, each of these SMPs needs to be processed at each intermediate SMI. However, intermediate SMI processing time is much smaller than the end SMI processing time, hence we ignore the intermediate SMI processing time to simplify the model.

Hence the topology discovery time can be summarized as:

$$t_{TD}(m,n) = t_{TD}(1,1) \times (GetS(m,n) + SetS(m,n)) \quad (8)$$

From the basic model, we can see that the total path computation time increases almost linearly with the number of nodes. Hence, we calculate the path computation time as:

$$t_{PC}(m,n) = t_{PC}(1,1) \times 2 \times (m/2)^n \quad (9)$$

The path distribution phase is dependent on the number of enteries in the forwarding table and the number of switches in the subnet. However, it does not increase linearly with the number of enteries. This is becuase, each forwarding table entry requires 8-bit for port output, and 16-bit for LID, making it 3-bytes per entry. Each UD packet is of Maximum Transfer Unit (MTU) bytes, hence one MTU is sufficient for sending information for $MTU/3$ - headersize*Set(LinearForwardingTable)* devices, which can be approximated as MTU/3 since the headersize is small. Thus, the path distribution time is:

$$t_{PD}(m,n) = t_{PD}(1,1) \times (2n-1) \times (m/2)^{n-1} \times S \quad (10)$$

where S is $\lceil (2 \times (m/2)^n \times 3)/MTU) \rceil$

6 Performance Evaluation of the Subnet Management Model

In this section, we evaluate the subnet management model presented in section 5. We also verify our results presented in section 4 using our model.

Tables 1 and 2 show the results obtained from the model presented in the previous section alongside with the results obtained in section 4.

For 1-switch configuration we take *m* as 8 and *n* as 1. However, for 6 switch configuration, for a fairer comparison, we remove the overhead of unused ports. Hence, we take *m* as 4 and *n* as 2. In essence, we remove the overheads of the ports which do not have end nodes connected to them.

Table 1. Model Verification Results, *(m, n)* = (8, 1), 1-switch

Phase	Experimental	Model
t_{TD}	10.82	10.6
t_{PC}	82.1	81.3
t_{PD}	.38	.36

Table 2. Model Verification Results, *(m, n)* = (4, 2), 6-switch

Phase	Experimental	Model
t_{TD}	22.4	21.6
t_{PC}	83.11	81.3
t_{PD}	2.32	2.24

We do not show the number of *Get* and *Set* SMPs, because they exactly match the proposed fat tree model. However, the timings are slightly different, because we have approximated our model slightly in terms of the intermediate SMI processing to simplify the timing equations.

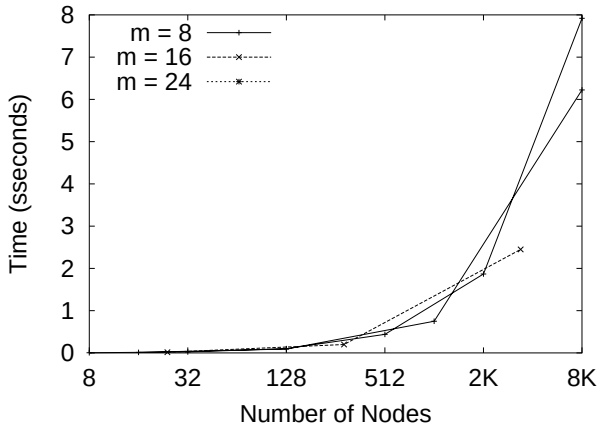

Figure 10. Topology Discovery Time

Figure 10 shows the topology discovery time with different number of ports per switch for the same number of nodes. Notice that some of the points are missing for 8-port switch and 16-port switch configurations, since it is not possible to construct systems in fat tree configuration with these values of *m*. We also show the results for 24-port switch configuration, with system sizes of 24, 288 and 3456 respectively.

The discovery time increases significantly for 8-port switches, because the number of switch ports not connected to the CA increase much more than for 16-port switch for the same number of nodes. The topology discovery time reduces by 22% for 16-port switch, when compared with 8-port switch for 8192 nodes.

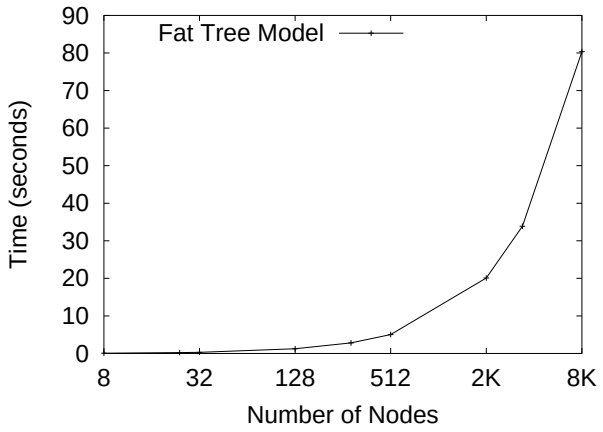

Figure 11. Path Computation Time

Notice that we do not show the Path computation time for different values of *m*, because in our model, the path computation time is only dependent on the number of nodes. Hence, this timing does not change with different topologies.

Figure 12 shows the behaviour of path distribution time. For small number of nodes, one *Set* SMP is sufficient per switch for setting up the forwarding tables. However, as the number of nodes increases, the number of SMPs required also increases.

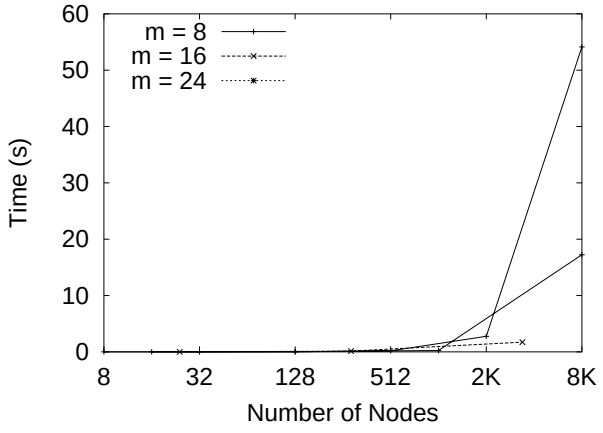

Figure 12. Path Distribution Time

The path distribution time increases much more for 8-port switch when compared with other configurations, since the number of switches are much more for 8-port switch configuration. The path distribution time reduces by 24% for 16-port switch configuration.

7 Related Work

InfiniBand Specification [5] defines a set of management classes, the functions performed by subnet Manager and interactions of various components associated with the Subnet Management. The work in [3] is simulation based study, which characterizes the Subnet Management time into multiple phases and evaluate the performance in terms of number of switches, and the behavior of the subnet manager on addition/removal of a switch. In addition, it also presents the number of discarded packets as a result of topology change. The work in [4] presents an adaptive algorithm for partial discovery. The algorithm performs discovery for only the nodes which are affected by the topology change. This reduces the number of direct routed SMPs which are generated in the subnet, hence reducing the total discovery time.

The work in [9] uses LMC mechanism to provide a fully adaptive routing mechanism for InfiniBand and presents a motivation for using it in parallel MPI applications. The work in [8] presents a strategy to avoid deadlocks in InfiniBand networks by using destination renaming. The work in [2] provides a framework for Quality of Service in InfiniBand Networks, with the use of SL-VL mapping and effcient mechanisms for virtual lane arbitration. However, none of the previous papers model the behavior of OpenSM and study the impact for large systems.

8 Conclusions and Future Work

In this paper, we have presented an evaluation of the popular subnet management implementation, *OpenSM* [1]. We have primarily focussed on the subnet disovery time and various management phases involved within the discovery time. The results were obtained by varying the different configuration parameters, like the number of ports per switch, number of switches, and number of nodes. An analytical model for discovery time and the number of SMPs generated for Fat tree networks has been proposed and validated. Also, within each management phase, we have characterized the timings for various sub-phases to model the discovery time.

In future, we plan to perform evaluations for large scale subnets to verify the results of our model. We also plan to model for different interconnection topologies, in particular 3-D Torus, which is the interconnection topolgy for large scale future InfiniBand networks, with fault tolerance. In addition, we plan to use the information provided by the Subnet Manager to schedule the messages at the MPI layer.

References

[1] Mellanox technologies, opensm release notes. 2004.

[2] F. J. Alfaro, J. L. Sanchez, and J. Duato. Qos in infiniband subnetworks. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):810–823, September 2004.

[3] A. Bermudez, B. Casado, T. M. Pinkston, J. Duato, and F. J. Quiles. Evaluation of a subnet management mechanism for infiniband networks. *International Conference on Parallel Processing*, pages 117–124, October 2003.

[4] A. Bermudez, R. Casado, F. J. Quiles, T. M. Pinkston, and J. Duato. On the infiniband subnet discovery process. *IEEE International Conference on Cluster Computing*, pages 512–517, December 2003.

[5] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2. October 2004.

[6] C. E. Leiserson. Universal networks for hardware-efficient supercomputing. October 1985.

[7] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang. A multiple lid routing scheme for fat-tree-based infiniband networks. *Parallel and Distributed Processing Symposium*, April 2004.

[8] P. Lopez, J. Flich, and J. Duato. Deadlock-free routing in infiniband through destination renaming. *International Conference on Parallel Processing*, pages 427–434, September 2001.

[9] J. C. Martinez, J. Flich, A. Robles, P. Lopez, and J. Duato. Supporting fully adaptive routing in infiniband networks. *International Parallel and Distributed Processing Symposium*, April 2003.