Fibre Channel Switch Modeling at Fibre Channel-2 Level for Large Fabric Storage Area Network Simulations using OMNeT++

Suresh Muknahallipatna, Timothy J. Brothers, Joseph Miles, and Howard Johnson

Abstract—**Typically, in the current enterprise data centers dedicated fabrics or networks are implemented to meet their LAN, Inter-Processor communication and storage traffic requirements. The storage traffic requirements of a group of servers are met through multiple storage area networks based on fibre channel, which has become the standard connection type. Typically, this fibre channel storage area networks are small (maximum of 32 switches/directors in a single fabric) and do not experience any scaling, stability and other performance issues.**

The advent of I/O consolidation in enterprise data centers for multiple traffic types to converge on to a single fabric or network (typically Ethernet platform) to reduce hardware, energy and management costs has also the potential to allow implementation of large storage area networks based on the fibre channel standards. Large storage area networks are being planned with more than two hundred switches/directors in a single fabric or network in addition to servers and storages connected to the fabric on Ethernet platforms. Even though these large storage area networks are envisioned to operate on Ethernet platform, they still have to satisfy the stringent operating and performance requirement set forth by the fibre channel standards. The two important issues of concern with large storage area networks are scaling and stability. The scaling and stability issues are dependent on the interactions and performance capabilities of various fabric servers located on each switch/director in the fabric in order to provide fabric services. In order to determine the extent of scaling and stability issues of a large fabric first the detailed models of the switch/director addressing the operations of the individual fabric servers are required. Next, the interactions of the switches/directors using the detailed models are to be simulated to study the scaling and stability issues.

In this paper, the detailed modeling of the fibre channel switch and the fabric servers using the OMNeT++ discrete event simulator is presented first. Detailed models are developed addressing the behavior of the switch at the level-2 of the fibre channel protocol since this layer addresses the requirements and operations of various mandatory fabric services like fabric build, directory, login, nameserver, management, etc. Next, using the OMNET++ discrete event simulator large fabrics are simulated. The results from the simulation are compared against the test bed traffic and the accuracy is demonstrated. Also, results and analysis of multiple simulations with increasing fabric size are presented.

DOI: 10.5176_2010-2283_2.1.138

I. INTRODUCTION

Current enterprise data center architecture use dedicated networks or fabrics to meet the inter-process communication (IPC), LAN and storage traffic needs. The LAN traffic needs are commonly met with Ethernet where as the IPC traffic that requires low latency and high bandwidth are met with high performance interconnects such as InfiniBand. The storage traffic need of zero losses of packets is satisfied by storage area network (SAN) based on fibre channel (FC) interconnects. However, the use of dedicated fabric for each class of traffic in a single data center requires separate components like adapters, cables, switches, management software, etc., for each type of fabric. This architecture of dedicated fabrics increases the hardware, energy and management costs and limits the size of the dedicated fabrics. In current data centers, the size limitation is overcome by deploying multiple dedicated fabrics to serve groups of servers, storages, etc.

The development of PCI-Express and 10 gigabit Ethernet has recently led to emphasis on I/O consolidation by converging multiple traffic classes on to a single fabric. The current industry approach of converging to a single fabric is through Ethernet based convergence. The FC based storage data frames are encapsulated in Ethernet packets known as FC over Ethernet (FCoE) and retain all the management features and characteristics of FC based storage area network. In theory this should allow large customers having massive investments in FC based SAN infrastructure and management to seamlessly transit to this new technology. The convergence to a single fabric has the potential to reduce hardware, energy and management costs and allow deployment of large fabrics at the enterprise data centers.

Typically, in the current enterprise data centers a number of small to medium sized FC based SANs are deployed to connect groups of servers to storages. A typical SAN is illustrated in Fig. 1.

Fig. 1 depicts the most commonly used SAN topology, namely the switched fabric SAN. In this topology, the servers are connected to the storage through a fabric comprised of switches, directors, gateways and cables. In Fig. 1, it can be seen that the client workstations access the server through the traditional local area network, whereas the servers access the storage through the SAN. The servers can access the storage through alternative paths, and the clients can access the data on the storage through multiple servers [10].

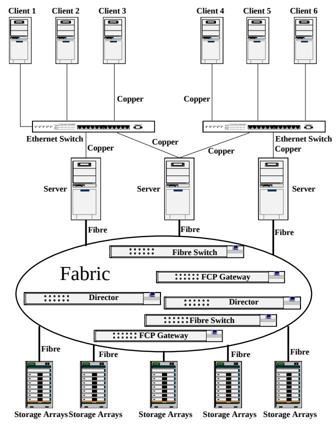


Fig. 1. A Typical Storage Area Network.

The size of a SAN is defined by the number of switches/directors in the fabric. In small or medium sized fabrics the numbers of switch range from a few to a maximum of 32. The scaling, stability and performance of a SAN are dependent on the interactions and performance capabilities of various fabric servers providing mandatory fabric services located on each switch in the fabric. Majority of the fabric services plays an important role in meeting the stringent timing requirements during the building and later maintaining the stability of the fabric [8][11].

With the focus shifting towards convergence in enterprise data centers this small or medium SANs are being combined to fewer or in some cases single large SANs with the number of switches in the fabric reaching a maximum of 256. This large size of the fabric makes the SAN prone to scalability, stability and performance issues. The amount of traffic generated during the building and management of the fabric can pose a serious impediment to the stability of the network. In the event of network disruption, the network can experience a significant delay in reaching a steady state (from a transient rebuild operation) leading to non-availability of data. Obviously, the growth of the fabric cannot be allowed to have a significant detrimental impact on the availability of the data in the SAN. Therefore, it is necessary to ascertain first whether a large fabric is stable and scalable. In case of small or medium sized fabrics it is possible to determine the stability and scalability issues by performing tests on an actual fabric or by using the hardware simulator. However, with large fabrics it would be far too expensive and impractical to implement a large fabric simply for testing purposes.

In this paper, we propose a discrete event simulator for the FC based switched fabric SAN using OMNeT++, an open source tool. This paper is continuation of the preliminary work in paper [1] previously presented at the local computer network conference in 2007.

The initial focus of the simulator is to develop models to simulate the FC-2 level of the FC protocol (FCP) dealing with fabric building and manageability under the large switched fabric conditions. Using the simulator, the performance of switched fabric of various sizes and topology will be analyzed to determine the switched fabric scaling and stability issues.

The organization of the rest of the paper is as follows: Section II provides a discussion of related work on discrete event simulation of SANs. Section III provides a brief summary of OMNeT++. Section IV provides a discussion of FC switched fabrics, single and distributed server models. In section V the modeling of the distributed fabric controller server is discussed in detail. Section VI presents the results and analysis. Finally, Section VII concludes by providing some insights to lessons learned during the model development using OMNeT++ and future direction.

II. RELATED WORK

Several studies have examined discrete event simulation of SANs. Reference [14] discusses a discrete event simulator for simulating a SAN developed using the CSIM18 simulation engine. The CSIM18 engine provides a library of routines for use by C programmers to implement discrete event simulation models of complex systems. The simulator [14] is capable of simulating a single FC switch and simulates I/O traffic between servers and storages. Preliminary network latency performance results with real world I/O traffic have been presented [14]. Reference [15] extends the capability of [14] further, to simulate multiple FC switches in a multi-stage topology. The simulator simulates class-3 service I/O traffic, which is connectionless thereby not strictly implementing the FC standards. Reference [2] discusses a discrete event simulator "SimLab" based on C++ with message passing interface libraries. SimLab is capable of simulating real-time delivery of data and verification of distributed algorithms in a SAN environment. Wang et al. presented SANSim [21] an event driven simulator exclusively for FC arbitrated loop (FC-AL) based SAN. The SANSim can simulate I/O traffic and also can simulate a rudimentary FC switch. Molero et al. using the CSIM18 based simulator [15][17] presented the effect on I/O traffic due to different switch architectures and link failures. Rueda et al. presented [18] a discrete event simulator based on OPNET to model the synchronous write operations during I/O traffic.

In all the above previous work, the I/O traffic is simulated with the assumption that the network has reached a steady state and there is no traffic due to network building and management. Many of the above simulators are limited to simulate a fabric with single or few FC switches.

III. SIMULATION FRAMEWORK

After comparing discrete event simulators like OPNET, ns-2, and OMNeT++ with respect to ease of use, flexibility, modular architecture, and open source code base, OMNeT++ was chosen. OMNeT++ is a C++ based object-oriented discrete event simulation package developed at the Technical University of Budapest [20]. The focus area of OMNeT++ is the simulation of computer networks and other distributed systems. It allows design of modular simulation models, which can be combined to develop complex modules. The modules can be composed of any granular hierarchy and simple modules (lowest hierarchy level) can be combined to form compound modules of varying levels of hierarchy. The modules communicate by passing messages through connections between modules or directly to the destination module. The model is constructed by defining its structure (modules and their interconnections) by using the network description (NED) language of OMNeT++. Individual model components are compiled and linked with the simulation library, along with one of the user interface libraries to form an executable program. References [12][13] discuss in detail the features of OMNeT++ and the simulation process.

IV. FC SWITCHED FABRIC

FC switched fabric consists of FC switches and nodes, which can be either servers with FC host bus adaptors (HBA) or storages. The ports on HBAs are referred to as node ports (N_Ports), and they connect to fabric ports (F_Ports) on fabric switches to join the fabric. As the number of nodes grows to exceed the number of F_Ports available on a single fabric switch, another switch is added to the fabric through a pair of expansion ports (E_Ports). The resulting connection between switches is called an inter-switch link (ISL). Each FC switch in a fabric is identified by a manufacturer assigned 64-bit unique number known as the world-wide name (WWN).

Since establishing a fully functioning ISL is the first and an important operation in constructing a stable fabric [8][11], the modeling of FC switch with respect to establishing the ISLs and forming the fabric is considered in this work.

A. Fibre Channel Switch and Fabric Servers

A high level architecture of a FC switch is shown in Fig. 2. FC switch architecture consists of identical I/O blocks corresponding to the number of physical ports on a switch. The switch consists of a single cross bar and a controller. The controller implements the fibre channel services (FCS) which provide addressability of the fabric, principal switch selection, building routing tables, node login, name server etc., as required by fibre channel switch fabric [8], fabric channel link services [7], and fibre channel generic services [6] standards.

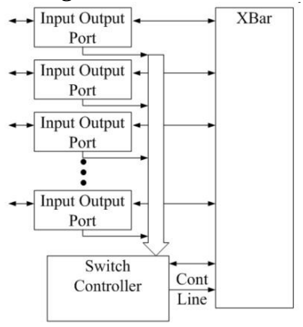


Fig. 2. A High Level Architecture of FC Switch.

The FCS from an architecture standpoint is a collection of server functions located at different addresses. Since, the addresses are defined by the standard; they are referred to as well-known addresses (WKA). A high level architecture of the switch controller shown in Fig. 3, implements the functionality of various mandatory and non-mandatory fabric services as servers at specific WKA.

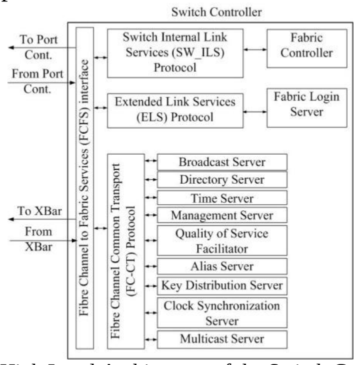

Switch Controller
To Port Cont.
From Port Cont.
To XBar
From XBar
Fibre Channel to Fabric Services (FCFS) interface
Switch Internal Link Services (SW_ILS) Protocol
Fabric Controller
Extended Link Services (ELS) Protocol
Fabric Login Server
Fibre Channel Common Transport (FC-CT) Protocol
Broadcast Server
Directory Server
Time Server
Management Server
Quality of Service Facilitator
Alias Server
Key Distribution Server
Clock Synchronization Server
Multicast Server

Fig. 3. High Level Architecture of the Switch Controller.

The fabric controller (WKA: 0xFFFFFD) and fabric login (WKA: 0xFFFFFE) are mandatory servers that use switch internal link services (SW_ILS) and extended link services (ELS) protocols respectively to communicate with other switches and nodes in the fabric. The fabric controller server is responsible for establishing the addressability of the fabric, principal switch selection, building routing tables and other tasks associated with fabric building and manageability.

The fabric login server is responsible for assigning an address to a node attempting to join the fabric (known as fabric login) and inform fabric capabilities to the joining node. The protocols used by both these servers correspond to FC-2 level. Even though the directory, time etc., are non-mandatory servers, a majority of FC switch manufacturers implement the servers in the fibre switched architecture. The non-mandatory servers use fibre channel common transport (FC-CT) protocol corresponding to FC-2 level to communicate with other switches and nodes. From the above discussion, it can be seen that the fabric controller server provides the crucial functionality for building and managing a multi-switch fabric. The functionality of all other servers is dependent on the functionality of fabric controller server. In a fabric, only a single switch or an external server could implement these fabric service servers and provide functionality to the entire fabric or incorporate instances of the fabric service servers on every switch leading to distributed servers [6]. The approach of implementing a single instance of the fabric service servers is not popular due to single point failure, failure of the connecting link, and path bottleneck during significant traffic. Hence, manufactures supporting switched fabric invariably support the distributed server architecture.

B. Distributed fabric Controller Server Model

When multiple FC switches with integrated server functions are connected together in some topology to form a switching fabric, these integrated servers must coordinate their operations to ensure consistent information. A typical switching fabric [11] with the distributed fabric controller server (DFCS) as the only integrated server on each switch is shown in Fig. 4.

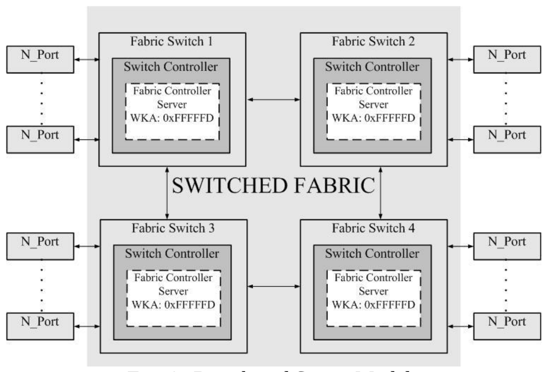


Fig. 4. Distributed Server Model.

Each switch in the fabric maintains the information of all nodes connected locally and the servers service any local requests. When a request cannot be serviced entirely by the local server, requests can be made to similar servers on other switches in the fabric. Hence, the term distributed server. For the DFCS model to work, the communication links between identical servers have to be established first. In Fig. 4, it can be seen that every instance of the fabric controller server has the same WKA and hence each instance must have a way to send requests to other server instances on a specific switch. In order to identify each instance, another address known as domain identifier address of the format 0xFFFCXX where XX will be unique to each switch is used along with the individual server WKA. In most switch implementations, the domain identifier address is an alias to the particular instance of the fabric controller server. The XX portion of the domain identifier address for each switch is allotted by the fabric controller server located on the principal switch. Each instance of the fabric controller server participates in selecting the principal switch, assigning the domain identifier address and other fabric construction operations. Hence, in this paper the modeling of the distributed fabric controller server using OMNeT++, and simulation of large fabric build and management process is presented.

V. DISTRIBUTED FABRIC CONTROLLER SERVER MODEL USING OMNeT++

The fabric controller server builds the switched fabric by establishing the communication links between individual FC switches. In Fig. 5, a fabric controller server process flow chart is presented.

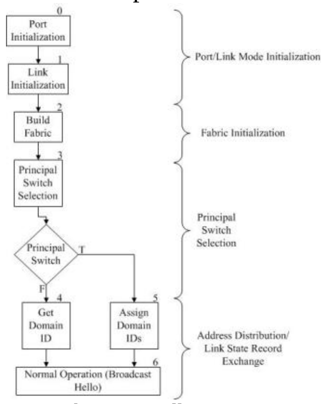

0
Port Initialization
1
Link Initialization
2
Build Fabric
3
Principal Switch Selection
Principal Switch
T
F
4
Get Domain ID
5
Assign Domain IDs
6
Normal Operation (Broadcast Hello)
Port/Link Mode Initialization
Fabric Initialization
Principal Switch Selection
Address Distribution/ Link State Record Exchange

Fig. 5. Fabric Controller Server Process.

In Fig. 5, it can be seen that the fabric controller process consists of four stages [8][19][21] listed below:

- Port/Link Mode Initialization
- Fabric Initialization
- Principal Switch Selection
- Address Distribution/Link State Record Exchange

The progressive flow through the above four stages is to enable communication between any two adjacent fabric controller servers by first establishing the two connecting physical ports on each switch as E_Ports. Next, establish a functioning inter switch link between these two E_Ports. Each stage consists of at a minimum one state machine and numerous states to perform the transitions.

A Port/Link Mode Initialization

The port/link mode initialization consists of two state machines corresponding to port initialization and link initialization shown in Fig. 6. Every physical port on a switch on power reset enters the port initialization state.

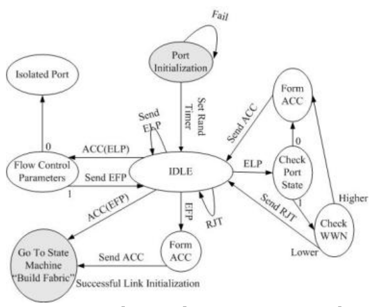


Fig. 6. Port/Link Initialization State Machines.

On entering this state, the port performers the following operations:

1. Determine its port classification namely F_Port, E_Port or generic port (G_Port) capable of operating as an E_Port or F_Port.

2. If it is a F_Port then travel to a state in the state machine [5] of the fabric login server.

3. If it is an E_Port then start a small random timer and travel to IDLE state starting the link initialization state machine as shown in Fig. 6. The small random timer is used to replicate the different instances each switch in the fabric will enter the IDLE state.

4. If the port determines its classification to be of the arbitrated loop class [4] then the port initialization fails, and it never travels to the next state in Fig. 6.

In the link initialization state machine, the switch stays in the IDLE state waiting for an exchange link parameter (ELP) frame from its neighboring fabric controller server or for the small random timer to expire and then perform the following state transitions:

1. If the timer expires before an ELP is received, the fabric controller server will transmit its own ELP and continue to stay in the IDLE state. ELP frame is exchanged between two E_Ports to exchange parameters governing communications between the two ports. The parameters exchanged are the type of class service, low level ISL flow control mode (Vendor unique or R_RDY), and buffer credits [7].

2. The neighboring fabric controller servers on receiving an ELP will travel to the "Check Port State". In this state, the server checks whether it has already sent an ELP or not.

3. Transit to the "Form ACC" state if the fabric controller server has not yet transmitted an ELP. In the "Form ACC" state a switch accept (SW_ACC) frame (identical to ELP with the parameters reflecting its own capabilities) is transmitted with the destination address set to the source address of the received ELP frame.

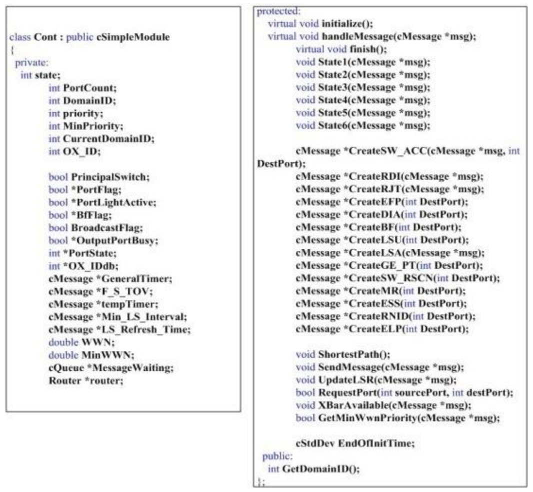
4. If the fabric controller server has already transmitted an ELP then check for its WWN identifier being higher than the WWN received through the ELP frame. If, it's WWN is higher, then travel to "Form ACC" state else transmit a switch reject (SW_RJT) frame indicating the received ELP frame is redundant.

5. On receiving a SW_ACC frame for the transmitted ELP, the fabric controller server travels to "Flow Control Parameters" state. In this state, the flow control parameters received through the SW_ACC frame and its own are compared and if there are any incompatibilities, the involved ports enter the isolated state and cannot participate in any further ISL communications.

6. If the flow parameters are compatible then transmit an exchange fabric parameter frame and travel to IDLE state. The EFP frame is sent in this state machine to reset the fabric parameters like address allocation, principal switch etc.

7. Next, on receiving a SW_ACC frame (response to the previously transmitted EFP) or transmit a SW_ACC frame (response to received EFP frame), the fabric controller server travels to the next state machine "Build Fabric".

The above state transition indicates a proper establishment of an ISL between two E_Ports, and these two E_Ports and the ISL can participate in a fabric building process. The object oriented implementation of the proposed state machine in shown in Fig. 7.



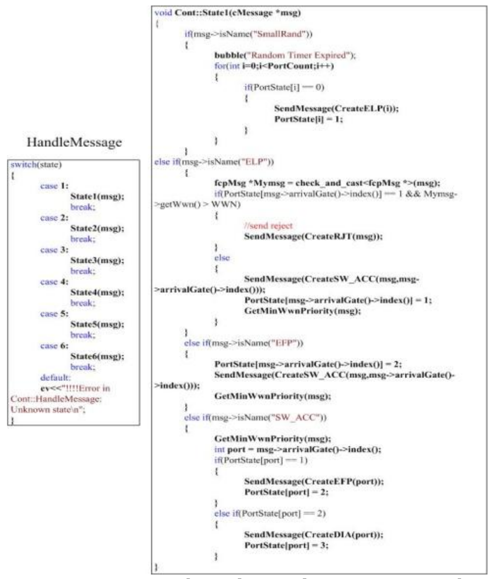
```
class Cont : public cSimpleModule
{
 private:
  int state;
        int PortCount;
        int DomainID;
        int priority;
        int MinPriority;
        int CurrentDomainID;
        int OX_ID;

        bool PrincipalSwitch;
        bool *PortFlag;
        bool *PortLightActive;
        bool *BfFlag;
        bool BroadcastFlag;
        bool *OutputPortBusy;
        int *PortState;
        int *OX_IDdb;
        cMessage *GeneralTimer;
        cMessage *F_S_TOV;
        cMessage *tempTimer;
        cMessage *Min_LS_Interval;
        cMessage *LS_Refresh_Time;
        double WWN;
        double MinWWN;
        cQueue *MessageWaiting;
        Router *router;
```

```
protected:
  virtual void initialize();
  virtual void handleMessage(cMessage *msg);
        virtual void finish();
        void State1(cMessage *msg);
        void State2(cMessage *msg);
        void State3(cMessage *msg);
        void State4(cMessage *msg);
        void State5(cMessage *msg);
        void State6(cMessage *msg);

        cMessage *CreateSW_ACC(cMessage *msg, int
DestPort);
        cMessage *CreateRDI(cMessage *msg);
        cMessage *CreateRJT(cMessage *msg);
        cMessage *CreateEFP(int DestPort);
        cMessage *CreateDIA(int DestPort);
        cMessage *CreateBF(int DestPort);
        cMessage *CreateLSU(int DestPort);
        cMessage *CreateLSA(cMessage *msg);
        cMessage *CreateGE_PT(int DestPort);
        cMessage *CreateSW_RSCN(int DestPort);
        cMessage *CreateMR(int DestPort);
        cMessage *CreateESS(int DestPort);
        cMessage *CreateRNID(int DestPort);
        cMessage *CreateELP(int DestPort);

        void ShortestPath();
        void SendMessage(cMessage *msg);
        void UpdateLSR(cMessage *msg);
        bool RequestPort(int sourcePort, int destPort);
        void XBarAvailable(cMessage *msg);
        bool GetMinWwnPriority(cMessage *msg);

        cStdDev EndOfInitTime;
 public:
  int GetDomainID();
};
```

Fig. 7. OMNeT++ Class Definition of Fabric Controller Server.

In Fig. 7, a C++ class derived from cSimpleModule (OMNeT++ library) abstracting the behavior of the fabric controller is shown. The class consists for data members describing the error timers, WWNs, domain identifier etc, and member functions implementing the various states. The handleMessage function depending on a particular event will execute the corresponding state machine function causing the state transitions. The initialize function implements the port state machine. Some of the code snippets in the HandleMessage and the individual state machine function are shown in Fig. 8. Using the C++ switch structure, the HandleMessage function calls the corresponding state machine function.

In Fig. 8, it can be seen that any message with respect to Port/Link Mode initialization triggers executing the State1 function. Based on the received message an appropriate state and its transitions are performed using simple C++ if else structures. Similar code development architecture has been used in implementing the other proposed state machines [9][13].


HandleMessage

```
switch(state)
{
        case 1:
                State1(msg);
                break;
        case 2:
                State2(msg);
                break;
        case 3:
                State3(msg);
                break;
        case 4:
                State4(msg);
                break;
        case 5:
                State5(msg);
                break;
        case 6:
                State6(msg);
                break;
        default:
        ev<<"!!!!Error in
Cont::HandleMessage:
Unknown state\n";
}
```

```
void Cont::State1(cMessage *msg)
{
        if(msg->isName("SmallRand"))
        {
                bubble("Random Timer Expired");
                for(int i=0;i<PortCount;i++)
                {
                        if(PortState[i] == 0)
                        {
                                SendMessage(CreateELP(i));
                                PortState[i] = 1;
                        }
                }
        }
else if(msg->isName("ELP"))
        {
                fcpMsg *Mymsg = check_and_cast<fcpMsg *>(msg);
                if(PortState[msg->arrivalGate()->index()] == 1 && Mymsg-
>getWwn() > WWN)
                {
                        //send reject
                        SendMessage(CreateRJT(msg));
                }
                else
                {
                        SendMessage(CreateSW_ACC(msg,msg-
>arrivalGate()->index()));
                        PortState[msg->arrivalGate()->index()] = 1;
                        GetMinWwnPriority(msg);
                }
        }
        else if(msg->isName("EFP"))
        {
                PortState[msg->arrivalGate()->index()] = 2;
                SendMessage(CreateSW_ACC(msg,msg->arrivalGate()-
>index()));
                GetMinWwnPriority(msg);
        }
        else if(msg->isName("SW_ACC"))
        {
                GetMinWwnPriority(msg);
                int port = msg->arrivalGate()->index();
                if(PortState[port] == 1)
                {
                        SendMessage(CreateEFP(port));
                        PortState[port] = 2;
                }
                else if(PortState[port] == 2)
                {
                        SendMessage(CreateDIA(port));
                        PortState[port] = 3;
                }
}
```

Fig. 8. OMNeT++ Port/Link Mode Initialization State Machine Implementation.

B. Fabric Initialization

In Fig. 9, the proposed build fabric state machine is shown. The goal of this state machine is to ensure that the build fabric (BF) frame is broadcasted across the entire fabric and there by initiate a non-disruptive principal switch selection and fabric configuration. This state machine uses two special timers known as fabric stability time out value (F_S_TOV) and build fabric time out value (B_F_TOV).

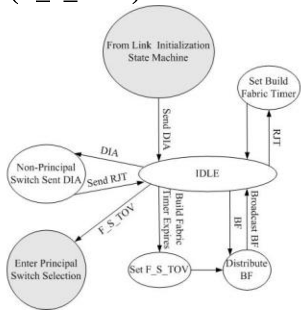

From Link Initialization State Machine
Set Build Fabric Timer
Send DIA
RJT
DIA
Non-Principal Switch Sent DIA
Send RJT
IDLE
F_S_TOV
Build Fabric Timer Expires
BF
Broadcast BF
Enter Principal Switch Selection
Set F_S_TOV
Distribute BF

Fig. 9. Build Fabric State Machine.

The F_S_TOV is used during flooding operations and the timer specifies how long a switch must wait after flooding event before it considers the flood to have reached all switches in the fabric and the fabric to have stabilized. The default value of F_S_TOV is 5 seconds [8]. The B_F_TOV is used to ensure that a fabric reconfiguration does not occur due to an error. A fabric controller enters the "build fabric" state will perform the following transitions:

1. Every fabric controller transmits a domain identifier assigned (DIA) frame assuming itself as the principal switch.
2. Every other fabric controller receiving the DIA rejects by sending a SW_RJT frame and then starts the B_F_TOV (3 ms) and enters IDLE state. After the B_F_TOV expires, every fabric controller sets F_S_TOV and broadcasts BF frame on every established ISL. Received BFs are forwarded on every ISL other than the one on which the frame was received. Every BF received is acknowledged with a switch accept frame. This operation continues until the F_S_TOV expires and then the fabric controller travels to the principal switch selection state machine indicating that every fabric controller is ready for non-disruptive principal switch selection and fabric configuration.

C. A Principal Switch Selection

This state machine is used to elect a switch in the fabric as a principal switch which would then be capable of issuing domain identifiers to all other switches known as non-principal switches. The proposed state machine is shown in Fig. 10 and the transitions performed are discussed below:

1. A timer corresponding to twice the F_S_TOV is set by every fabric controller and then exchange fabric parameter frame is broadcasted. The EFP frame in this state machine consists of a single byte principal switch priority value and the principal switch WWN. The priority value indicates whether it was previously the principal switch (0x02) or mandatory principal switch due to highest priority value (0x01) or cannot be a principal switch (0xFF).

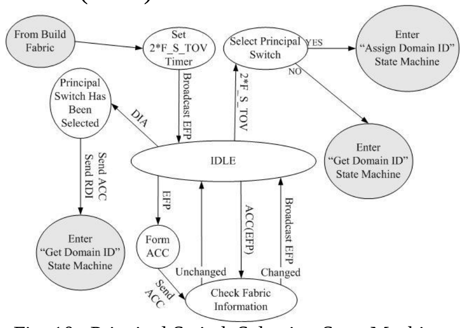


Fig. 10. Principal Switch Selection State Machine.

2. On receiving an EFP or a SW_ACC for the transmitted EFP, the received fabric information is compared with its retained principal switch priority and principal priority value. If the received values are lower than the retained values, the fabric controller uses the lower value and broadcasts a new EFP. If, the fabric is being built from power on, each switch is under the assumption that it is the principal. This transition continues until twice the F_S_TOV expires or the fabric controller receives a DIA frame.
3. On timer expiration, if the lowest WWN and priority value is same as the switch WWN and priority value, then the switch assumes the role of the principal switch and travels to “Assign Domain IDs” state machine. If, the switch WWN and priority value is not the lowest, then it travels to “Get Domain ID” state machine.
4. On receiving a DIA frame before the timer expires, the switch concludes another switch has assumed the role of the principal switch, and it travels to “Get Domain ID” state machine.

D. Address Distribution/Link State Record Exchange

The address distribution and link state record (LSR) exchange process consists of two state machines the “Assign Domain IDs” and “Get Domain ID”. The “Assign Domain IDs” state machine with its transitions shown in Fig. 11 is executed by the fabric controller only on the elected principal switch. The states and the transitions are discussed below:

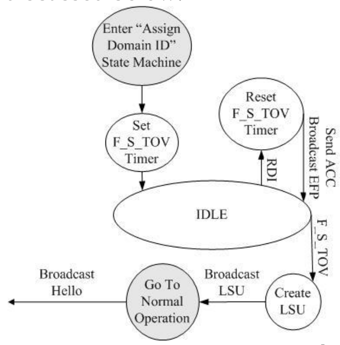


Fig. 11. Assign Domain IDs State Machine.

1. The fabric controller sets the F_S_TOV timer and waits for a request domain identifier (RDI) frame from all other switches in the fabric which are classified as non-principal switches. The RDI frame contains the requesting switch WWN and desired domain identifier.
2. On receiving a RDI frame, the principal switch fabric controller transmits a SW_ACC frame with requesting switch WWN and granted domain identifier. Next, the F_S_TOV is reset and an EFP frame with the current domain identifier list is broadcasted. The EFP frame is broadcasted to inform all non-principal switches of the presence of switches with assigned or valid domain identifiers in the fabric.
3. If RDI frame is not received within a F_S_TOV, the principal switch assumes that all non principal switches have been assigned with domain identifiers, and then it broadcasts a link state update (LSU) frame and travel to normal operation state. The LSU frame contains a link state record (LSR) comprising of link descriptors describing the connectivity of all inter-switch links (directly connected fabric switches), associated with one specific switch. This LSU creates the initial topology database. The LSU frames are not acknowledged during the initial fabric configuration.

The “Get Domain ID” state machine and its transitions shown in Fig. 12 are executed by the fabric controllers on all non-principal switches to obtain domain identifiers from the principal switch and create the initial topology database.

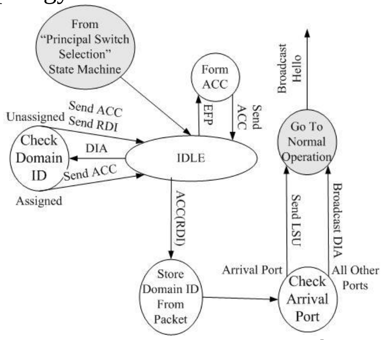


Fig. 12. Get Domain ID State Machine.

1. A fabric controller on receiving DIA frame checks, whether it has already been assigned a domain identifier from the current principal switch. If the domain identifier has not been assigned, the fabric controller transmits a RDI frame and also a SW_ACC frame.
2. On receiving the SW_ACC frame for the RDI frame, the new granted domain identifier is stored and travels to “check arrival port” state.
3. In the “check arrival port” state, it identifies the physical E_Port on which SW_ACC frame for the RDI was received. A LSU is now sent on the identified physical E_Port, forward the DIA frame (from the principal switch) on all other E_Ports, and travel to normal operation state.

The fabric controllers on completing the four stages shown in Fig. 5 reach the stable operation state in which the fabric controllers broadcast at regular intervals “Hello” frames” on all ISLs. Initially, the Hello frames are sent without domain identifiers of its neighbors and the switch would be capable of performing only “one-way” communication. Next, on receiving Hello frames from neighbors, the switch learns the domain identifiers of its neighbors and their output ports used to send the Hello frames. On all subsequent Hello frame transmissions, the switch will include neighbor’s neighbor's domain identifier, and establish “two-way” communication between neighboring switches. At this stage, the switches can start initial topology synchronization and the process of determining optimal routes. A non-reception of the periodic hello frame on an ISL would indicate that the neighboring switch is no longer operational.

E. Models of Ancillary Components

In addition to the DFCS modeling, the Xbar (Crossbar), IO Port Controller, Buffer Credit Mechanism, Name Server, Routing and FC Packet were also modeled to simulate the fabric. Reference [3 provides a detailed discussion of modeling these ancillary and necessary components.

VI. SIMULATION VALIDATION AND RESULTS

The individual finite state machines discussed in section 5 are implemented first as simple module objects derived from cSimpleModule class (OMNeT++ library) similar to that shown in Fig. 7. Next, the simple module objects are initialized and connected in a particular fashion using the OMNeT++ NED based descriptor shown in Fig. 13 to instantiate the switch object shown in Fig. 14.

```
 import "FIFO", "PortCont", "XBar", "Cont", "Router", "NameServer ", "trace";
module FabricSwitch
    parameters :
        switchNumber : numeric const ;
    gates : in : in [ ]; out : out [ ];
    submodules :
        buffer : FIFO [ s i z e o f ( in ) ] ;
            display : "i=block/queue;p=134 ,414 , row;q=queue";
        pc : PortCont [ s i z e o f ( in ) ] ;
            display : "i=device/cpu_s;p=134 ,214 , row;q=queue";
        xbar : XBar ;
            gatesizes : in [ s i z e o f ( in ) ], out [ s i z e o f ( in ) ];
          display : "i=block/ switch_l ;p=366 ,54 , row;q=queue" ;
      cont : Cont ;
          gatesizes : PCin [ s i z e o f ( in ) ], PCout [ s i z e o f ( in ) ];
          display : "i=device/cpu;p=230 ,54 , row;q=queue" ;
      router : Router ;
          gatesizes : in [ s i z e o f ( in ) + 1 ];
          display : "p=134 ,110;b=40 ,24" ;
      nameServer : NameServer ;
          display : "p=138 ,54;b=40 ,24" ;
      tracer : Tracer [ s i z e o f ( in ) ] ;
          display : "p=73 ,302;b=40 ,24" ;
   connections :
      xbar . Contout —> cont . XBarin ;
      cont . XBarout —> xbar . Contin ;
      cont . XBarCont —> xbar . ContLine ;
      cont . RouterOut —> router . in [ s i z e o f ( in ) ] ;
      nameServer . out —> cont . NameServerIn ;
      for i = 0 . . s i z e o f ( in ) −1 do
          in [ i] —> tracer [ i ] . in ;
          tracer [ i ] . inputOut —> buffer [ i ] . in ;
          buffer [ i ] . out —> pc [ i ] . in ;
           pc [ i ] . FIFOout —> buffer [ i ] . PCin ;
          pc [ i ] . XBarout —> xbar . in [ i ] ;
          xbar . out [ i] —> pc [ i ] . XBarin ;
          cont . PCout [ i] —> pc [ i ] . Contin ;
          pc [ i ] . Contout —> cont . PCin [ i ] ;
          pc [ i ] . out —> tracer [ i ] . outInput ;
          tracer [ i ] . out —> out [ i ] ;
          pc [ i ] . RouterOut —> router . in [ i ] ;
      endfor ;
   display : "b=904 ,492" ;
endmodule
```

Fig.13. NED based Switch Descriptor.

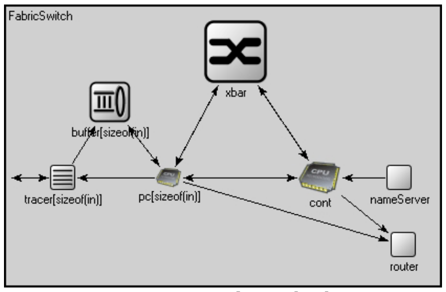
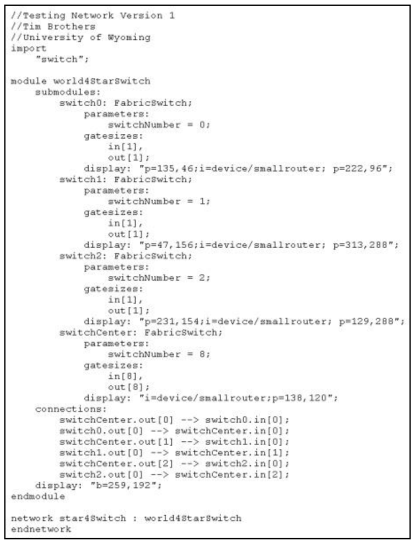


Fig. 14. Instantiated Switch Object.

In Fig. 13, it can be seen first the individual modules are imported and then the parameters of the individual modules like the number of ports, the buffer size and type (FIFO) are initialized. Next, the desired connections between the individual modules to simulate the FC switch are described. The simulated FC switch titled “FabricSwitch” in Fig. 14 consists of the simple module object titled “cont” that encapsulates the state machines of the fabric controller server. Also in Fig. 13 the implementation of the other essential components of the FC switch namely “nameServer”, “Router”, “Xbar”, etc., as simple modules are shown. Next, the above simulated FC switches are connected in a desired topology by another NED descriptor to simulate the FC fabric and in turn the operation of the distributed fabric controller server.



```
//Testing Network Version 1
//Tim Brothers
//University of Wyoming
import
    "switch";

module world4StarSwitch
    submodules:
        switch0: FabricSwitch;
            parameters:
                switchNumber = 0;
            gatesizes:
                in[1],
                out[1];
            display: "p=135,46;i=device/smallrouter; p=222,96";
        switch1: FabricSwitch;
            parameters:
                switchNumber = 1;
            gatesizes:
                in[1],
                out[1];
            display: "p=47,156;i=device/smallrouter; p=313,288";
        switch2: FabricSwitch;
            parameters:
                switchNumber = 2;
            gatesizes:
                in[1],
                out[1];
            display: "p=231,154;i=device/smallrouter; p=129,288";
        switchCenter: FabricSwitch;
            parameters:
                switchNumber = 8;
            gatesizes:
                in[8],
                out[8];
            display: "i=device/smallrouter;p=138,120";
    connections:
        switchCenter.out[0] --> switch0.in[0];
        switch0.out[0] --> switchCenter.in[0];
        switchCenter.out[1] --> switch1.in[0];
        switch1.out[0] --> switchCenter.in[1];
        switchCenter.out[2] --> switch2.in[0];
        switch2.out[0] --> switchCenter.in[2];
    display: "b=259,192";
endmodule

network star4Switch : world4StarSwitch
endnetwork
```

Fig. 15. NED Descriptor for FC Fabric with Four Switches in Star Topology.

In Fig. 15, the NED descriptor for simulating a FC fabric consisting of four FC switches in the Star topology is shown. In this descriptor, the generic switch model provided with OMNeT++ is first imported and then the each instance of the generic switch is set to the simulated FC switch. Next, the desired interconnection between the simulated FC switches is described. The OMNeT++ simulation runtime will read this descriptor and simulate the fabric shown in Fig. 16.

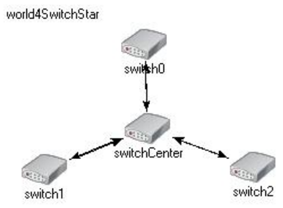


Fig. 16. FC Fabric with Four FC Switches in Star Topology.

A. Simulation Validation

The validation of the simulations consists of comparing the fabric initialization time determined through simulations of a small FC fabric with the fabric initialization time of the actual FC fabric. Fig. 17 shows a test bed of six Brocade Communications Systems Fibre Channel switches connected in a star configuration along with a fabric analyzer to collect the traffic during fabric build.

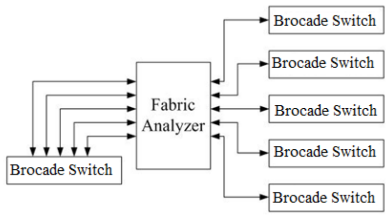


Fig. 17. Hardware Test Bed.

The trace of the traffic during the fabric initialization of a two switch fabric from the test bed and simulation are shown in Fig. 18. It can be seen in Fig. 18 the communication traffic of the test bed and the simulation are similar with minor differences. In case of the testbed, the second EFP sent by switch 0 is rejected. The reason code of the RJT frame indicated that the switch 1 was busy performing state transitions of other fabric service servers. Since the simulator has only the fabric controller server state machines it does not reject the frame.The other anomaly is the multiple hello and LSU frames sent in case of the test bed.

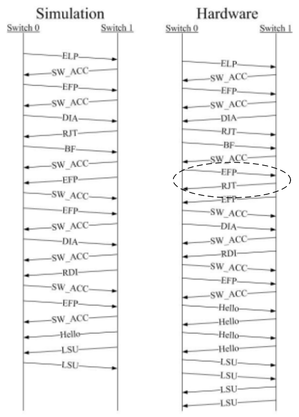


Fig. 18. Comparison of Fabric Initialization Traffic between Simulation and Test bed.

In the simulations, the periodic time interval of sending hello frames has been made large in comparison to that in the testbed in order to reduce the number of hello frames to be transmitted. This is a valid option as described in the FCP standards [8].The additional LSU frames in the testbed are sent to maintain the topology database which is not implemented in the simulation.

Next the timing information of the Fibre Channel switch for each state transition by the fabric controller server (frame reception and processing) with the link data rate of 1 Gbps was introduced into the finite state machine models of the fabric controller server through another NED descriptor, which is read by the OMNeT++ simulation runtime during simulation. The fabric initialization simulations were performed with increasing number of switches and trace of the traffic was collected from the simulation and test bed. From the trace of the traffic, the fabric initialization time defined as the time required by the distributed fabric controller servers to construct a stable fabric by traversing the four stages of the fabric controller server process shown previously in Fig. 5 were computed. In actual fabric, the fabric initialization occurs if there is the loss of principal switch or fabric merge occurs. During fabric intitialization the data cannot be transmitted (domain IDs are reassigned), and hence it is crucial to have a measure of the intitialization time. The average intitialization time for fabric simulations with two to six switches was obtained by repeating the simulations five times and are shown in Fig. 19.

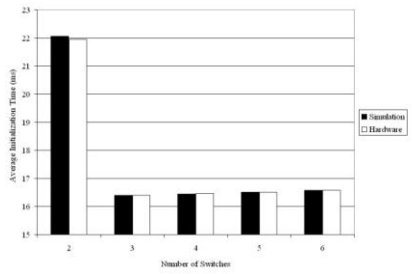


Fig. 19. Comparison of Average Initialization Time for Fabric with Two to Six Switches.

In Fig. 19, it can be seen that the average initialization time to build the fabric from both simulations and the test bed are almost the same. The increase in the initialization time with increasing switches from simulations indicates it is negligible, and it correlates with measurements from the test bed.

It can be seen in Fig. 19, the average initialization time of the fabric with two switches is higher in comparison to that with three or more switches. The reason for this behavior is the DIA rejection scheme. During fabric build, every switch assumes the role of the principal switch and issues a DIA command. Any switch on receiving a DIA command will reject the

DIA, start the build fabric time out (B_F_TOV) timer (default: 3 ms) and enters IDLE state as shown in Fig. 9. At the end of B_F_TOV, the switch is supposed to exit the IDLE state and issue a BF command. The B_F_TOV timer is introduced to avoid fabric rebuilds due to errors. However, if the switch receives a second DIA before the B_F_TOV timer expires; it will exit the IDLE state prematurely and issue a BF command. In case of two switch scenario, the possibility of the switch receiving a second DIA does not exist where as in case of three or more switches in star topology the switches do receive more than one DIA before the B_F_TOV timer expires leading to lower initialization time.

B. Large Fabric Simulations

First, a series of simulation was performed to determine fabric initialization time of fabrics consisting of 2 to 31 switches in a single fabric connected in star and full mesh topologies. Before the advent of IO consolidation the maximum number of switches allowed in a single fabric was 31. The star topology is a cost efficient topology where all the switches are connected to a central switch. The downside of this is that it creates a single point of failure. The full mesh topology creates a connection between all the switches. This is a much more expensive implementation, but there are multiple paths in case of a link or switch failure. In Fig. 20, the average initialization time for the star and full mesh topology with the number of switches increasing from 2 to 31 are shown.

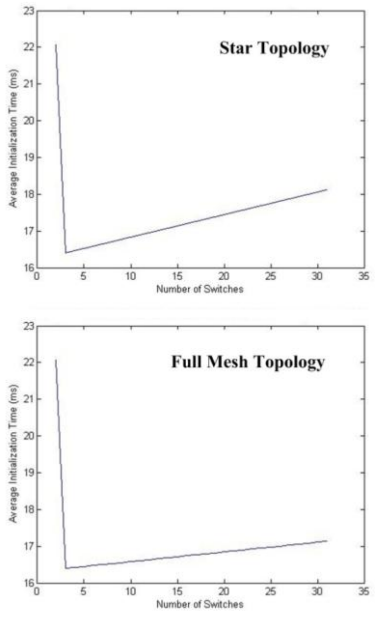


Fig. 20. Average Initialization Time for Star and Full Mesh Topologies with 2 to 31 switches.

In Fig. 20, the average initialization time increases linearly with increasing number of switches. The linear increase is due to the corresponding increase in the number of communications per switch to construct the fabric. Also, in Fig. 20, it can be noticed that the average initialization time for the same number of switches in star topology is slightly higher in comparison to that with the full mesh topology. This is due to the entire communication traffic of the fabric passing through the single switch at the hub of the star topology causing a bottleneck and introducing additional delay. For both the topologies, the average initialization time is well within the bounds of the fabric stability time out value (default F_S_TOV: 5 secs). Next, simulations were repeated for both the topologies with the current maximum number of switches allowed in a single fabric by the industry which is 239. The average initialization time for the star and full mesh topologies with a maximum of 239 switches in a single fabric is shown in Fig. 21.

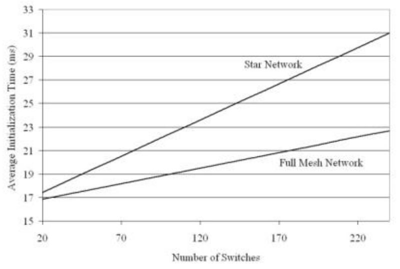


Fig. 21. Large Fabric Average Initialization Time.

The average initialization time even with the maximum number of switches in a single fabric as shown in Fig. 21 is well within the bounds of the default F_S_TOV. As expected, the fabric with star topology experiences a higher initialization time.

The fabric initialization time can also be considered as the time duration in which there cannot be any actual data transmission between servers and storages. Based on Fig. 21, a fabric with 239 switches connected in star topology undergoing a fabric rebuild will not allow data transfer between servers and storages for 31 ms at a minimum which corresponds to 3.9 MB of lost data at a data rate of 1 Gbps or the servers should have the capability to buffer this data for the duration.

In an actual fabric, data transfer will occur only after the topology of connected servers and storages is disseminated across the entire fabric by exchanging information among the nameServers located on each switch. Each switch after fabric initialization will send out a Get Entry by Port Type (GE_PT) query to one hop neighboring switches. A switch on receiving a GE_PT command will send information of any locally connected servers/storages to the originator of the GE_PT command and at the same time forward the query to its one hop neighboring switches and they will also send information of their locally connected server/storages back to the originator. The originator on receiving responses to its GE_PT query will update its nameServer database. By this mechanism after fabric initialization every switch has the topology of the entire fabric. Therefore, a finite amount of time defined as the topology initialization delay is required for the topology to be disseminated across the entire fabric. This delay will increase with the number of switches and nodes on each switch in a fabric.

Next, a full mesh fabric with 2048 nodes distributed across the fabric switches was simulated and the topology initialization delay was collected. The topology initialization delay on two fabrics with eight and sixteen switches and same number of nodes are shown in Fig. 22.

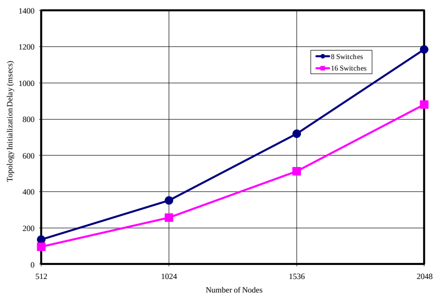


Fig. 22. Topology Initialization delay with Increasing Number of Nodes and Switches.

In Fig. 22, it can be seen that the topology initialization delay increases roughly quadratic and reaches a value of 1200 and 900 msecs for 2048 nodes distributed among 8 (256 nodes/switch) and 16 (128 nodes/switch) switches respectively. The fabric with 16 switches experiences a smaller topology initialization delay since each switch has to respond with information of only 128 nodes in comparison to 256 nodes with 8 switches.

The topology initialization delay is significantly large in comparison to the fabric initialization time, and it increases with the number of nodes and switches in the fabric. If this fabric undergoes fabric rebuild it will not allow any data transfers between servers and storages for 1200 msecs, which corresponds to 150 MB of lost data at a data rate of 1 Gbps.

From the above simulations, we can see that even though a large fabric with 239 switches is stable however due to the significantly larger topology initialization delay the servers have to be equipped with a large application buffer to avoid loss of significant application data. From Fig. 22, it can be seen that the topology initialization delay will reach a value larger than F_S_TOV with increasing number of nodes and switches leading to fabric instability. Also, from Fig. 22, it can be seen that distributing the same number of nodes across a larger fabric (number of switches) will reduce the topology initialization delay since each switch has to process information of fewer nodes.

VII. CONCLUSIONS

In this paper, we have demonstrated the SAN simulator simulating the FC-2 level traffic between distributed fabric controller servers during the fabric build and topology initialization operations. The various finite state machines of the distributed fabric controller has been presented along with their implementations machines using OMNeT++ has been presented. The performance of the simulator has been presented by comparing simulation traffic against the traffic of an actual SAN fabric. The simulator has been demonstrated to simulate fabrics with 239 switches at the FC-2 level. We have identified by simulations that the fabric build/rebuild process is stable even with the maximum number of switches in a single fabric and the fabric initialization time is in the order of 30 msecs. We have also identified that the topology initialization delay increases significantly with increasing size of the fabric and number of nodes/switch causing a significant data loss during fabric rebuild. Using the simulator, the distribution of nodes given a fabric size can be determined in order to reduce the topology initialization delay. Since, the simulator has been implemented using object-oriented principles with C++ and open source discrete event simulator OMNeT++, the simulator is highly flexible, tailored for incorporating additional components with ease.

The next stage of development will address the remaining fabric services servers along with actual storage data traffic.

REFERENCES

[1] Timothy Brothers, Suresh Muknahallipatna, Jerry Hamann, and Howard Johnson, "Fibre Channel Switch Modeling at Fibre Channel-2 Level for Large Fabric Storage Area Network Simulations using OMNeT++: Preliminary Results," 32nd IEEE Conference on Local Computer Networks (LCN), Oct. 15-18, 2007, Dublin, Ireland, pp. 191- 199.

[2] Berenbrink P, Brinkmann A, Scheideler C. "SIMLAB – A Simulation Environment for Storage Area Networks", Proceedings of the 9th IEEE Euromicro Workshop on Parallel and Distributed Processing, pp. 227 – 234, Feb. 7-9, 2001.

[3] Brothers TJ. "A Discrete Event Simulator Model of a Fibre Channel Switch at the Fibre Channel 2 Level", PhD dissertation, Dept. of ECE, University of Wyoming, May. 2007.

[4] "FC-AL-2: Fibre Channel 2nd Generation Arbitrated Loop", International Committee for Information Technology Standardization (INCITS), Oct. 23, 2001, Rev. 7.0.

[5] "FC-DA-2: Fibre Channel Device Attach", International Committee for Information Technology Standardization (INCITS): NPIV Acquisition Procedure, October 2008, Rev. 1.03.

[6] "FC-GS-6: Fibre Channel Generic Services", International Committee for Information Technology Standardization (INCITS), March 2009, Rev. 9.3.

[7] "FC-LS-2: Fibre Channel Link Services", International Committee for Information Technology Standardization (INCITS), May 2009, Rev. 2.11.

[8] "FC-SW-5: Fibre Channel Switch Fabric", International Committee for Information Technology Standardization (INCITS), June 2009, Rev. 8.5.
[9] Janson M, Karlsson M. "WOK – A Simulation Model for DFS and Link Adaptation in IEEE 802.11A WLAN", Masters Thesis, Linkoping University, Jan. 2004.
[10] Kembel RW." Fibre channel: A Comprehensive Introduction, 1st Edition", Northwest Learning Associates Inc.; 2002a.
[11] Kembel RW. "Fibre Channel Switched Fabric, 1st Edition", Northwest Learning Associates Inc.; 2002b.
[12] Kogekar A, A. Gokhale A. "Performance Evaluation of the Reactor Pattern Using the OMNeT++ Simulator", Proceedings of the 44th Annual ACM Southeast Regional Conference, pp. 708 – 713, Melbourne, FL, 2006.
[13] Lai J, Wu E, Varga A, Sekercioglue YA, Egan GK. "A Simulation Suite for Accurate Modeling of Ipv6 Protocols", Proceedings of the 2nd International OMNeT++ Workshop, Berlin, Germany, Jan. 2002.
[14] Molero X, Silla F, Santonja V, Duato J. "Modeling and Simulation of Storage Area Networks", Proceedings of the 8th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 307 – 314, Aug. 29 – Sept. 01, 2000a.
[15] Molero X, Silla F, Santonja V, Duato J. "A Tool for the Design and Evaluation of Fibre Channel Storage Area Networks", Proceedings of the 34th IEEE Annual Simulation Symposium, pp. 133 – 140, April 22-26, 2001a.
[16] Molero X, Silla F, Santonja V, Duato J. "On the Switch Architecture for Fibre Channel Storage Area Networks", Proceedings of the 8th International Conference on Parallel and Distributed Systems, ICPADS 2001, pp. 484 – 491, June. 26-29, 2001b.
[17] Molero X, Silla F, Santonja V, Duato J. "On the Effect of Link Failures in Fibre Channel Storage Area Networks", Proceedings of the International Symposium on Parallel Architectures, Algorthims and Networks, I-SPAN 2000, pp. 102 – 111, Dec. 7-9, 2000b.
[18] Rueda A, Pawlak M. "Performance Modeling on Synchronous Write Operations of Storage Wide Area Network (SWAN)", Proceedings of the OPNETWORK 2003, pp. 1 – 6, Aug. 2003.
[19] Shu J, Li B, Zheng W. "Design and Implementation of a SAN System Based on the Fiber Channel Protocol", IEEE Trans. on Computer,Vol. 54, No. 4, pp. 439 – 448, April 2006.
[20] Varga A. "OMNeT++ object-oriented discrete event simulation system", http://www.hit.bme.hu/phd/vargaa/omnetpp.htm, 1996.
[21] Wang CY, Zhou F, Zhu YL, Chong CT, Hou B, Xi WY. "Simulation of Fibre Channel Storage Area Networks Using SANSim", Proceeding of the 11th IEEE International Conference on Networks, pp. 349 – 354, Sept. 28 – Oct. 01, 2003.
[22] Watson RW. "High Performance Storage System Scalability: Architecure, Implementation and Experience", Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST' 05), April 11 – 14, 2005.

Biographies

Suresh Muknahallipatna received his B.E. degree in Electrical Engineering and Master's of Engineering from the University of Bangalore, India, in 1988 and 1991, respectively. He completed his Ph.D. degree at the University of Wyoming in 1995, with an emphasis on Neural Networks. He is a currently Associate Professor in the Dept. of ECE at the University of Wyoming. His current areas of expertise are performance analysis, modeling and simulations of storage area networks and mobile ad-hoc networks.

Timothy Brothers received his BS degree in Computer Engineering in 2002 and Ph.D. degree in 2007 both at the University of Wyoming with research on Storage Area Network limitations. Areas of research include modeling/simulation, robotics, and teaching strategies.

Joseph Miles received his MS degree in Electrical Engineering from the University of Wyoming, in 2006. He is currently pursuing Ph.D. in the area of Mobile Adhoc Networks with special emphasis on localization techniques.

Howard Johnson is currently a Technical Director at Brocade Communications Systems, Inc., and is responsible for furthering Brocade's lead in FICON technology. His expertise encompasses Brocade's ESCON and FICON products and includes an extensive relationship with IBM's zSeries I/O development team.