

Design and Implementation of Shortest Path Bridging For Network Simulator 3

by

Yoonsoon Chang, B.Eng

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of
Master of Applied Science in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

April 2013

© Copyright 2013, Yoonsoon Chang



Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-94657-2

Our file Notre référence

ISBN: 978-0-494-94657-2

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

ABSTRACT

Ethernet has evolved to support various network topologies while maintaining its backward compatibility and simplicity. Virtualization of the provider's Ethernet network enables support for fine-grained services for different users. Spanning Tree Protocol, STP, meets these properties but, still could benefit from improvements on utilization and convergence time. Shortest Path Bridging (SPB, IEEE 802.1aq) is developed to overcome these shortcomings of STP. This thesis presents the design of SPB simulator for NS-3 (open source network simulator) and the results of simulations. The modified version of Floyd-Warshall algorithm is used to compute routes. Multicast and unicast communications are simulated in SPBM (SPB Mac) mode to show the simulator's capability. The results prove that the communication maintains the crucial property of SPB; congruency between multicast and unicast, and symmetry between forward and backward paths. The traffic route selected among candidate paths with the same cost is in accordance with the SPB standard.

ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to Professor Samuel Ajila for the continuous support of my research.

I also thank to Alex Craig for his help in proofreading my work.

TABLE OF CONTENTS

ABSTRACT.....	III
ACKNOWLEDGEMENTS	IV
TABLE OF CONTENTS	V
LIST OF TABLES	VII
LIST OF ALGORITHMS.....	VII
LIST OF APPENDICES	VII
LIST OF FIGURES	VIII
LIST OF ACRONYMS	X
CHAPTER 1: INTRODUCTION.....	1
1.1 Thesis motivation	2
1.2 Contributions of the thesis.....	2
1.3 Thesis Outline.....	2
CHAPTER 2: LITERATURE REVIEW	4
2.1 Problems of legacy Ethernet protocols.....	4
2.2 Resolving the problems of legacy Ethernet.....	6
2.3 Virtualization and Fine-grained Services of Ethernet	7
2.4 Network Simulator NS-2 and NS-3	9
2.5 Network topology of NS-3	11
CHAPTER 3: ARCHITECTURE OF THE SIMULATION.....	13
3.1 Architecture of SPB node	13
3.2 Actions of a SPB node.....	15
CHAPTER 4: SHORTEST PATH BRIDGING CONTROL PLANE	17
4.1 IS-IS for Control Plane protocol.....	17
4.2 Extension of IS-IS for SPB.....	18

4.3	Control Plane Software Architecture.....	22
4.4	Implementation of IS-IS extension for SPB	24
4.5	Link State Database (LSDB) for SPB	30
4.6	Routing Algorithm Based on LSDB.....	31
4.7	Algorithms for Computing Routes and Populating Forwarding Database	32
CHAPTER 5: SHORTEST PATH BRIDGING DATA PLANE		38
5.1	SPBM Unicast Data Path.....	40
5.2	SPBM Multicast Data Path.....	40
5.3	SPBM Protocol Handler	42
CHAPTER 6: SIMULATION SETUP AND RESULTS.....		44
6.1	Simulator Helper class.....	44
6.2	Simulation outputs.....	45
6.3	Setting Up Simulation	47
6.4	Simulation Results.....	50
CHAPTER 7: CONCLUSION		54
APPENDIX A: 16 PRIORITY SETS OF HEXADECIMAL NUMBERS		58
BIBLIOGRAPHY		59

LIST OF TABLES

Table 1. NS-3 Models	12
Table 2. ECT-Algorithm Masking Value (Hex)	38

LIST OF ALGORITHMS

Algorithm 1. Modified Floyd-Warshall All-pairs Shortest Path	33
Algorithm 2. Initialize Matrices	34
Algorithm 3. Populating SPB Nodes	37

LIST OF APPENDICES

Appendix A: 16 Priority sets of hexadecimal numbers.....	55
--	----

LIST OF FIGURES

Figure 1.a STP Problems	6
Figure 1.b Possible SPB routes	7
Figure 2. En/Decapsulation at UNI port	9
Figure 3. Nodes with Internet Stack and CSMA Channel in NS-3	12
Figure 4. Nodes with SPB Stack and SPB Channel	13
Figure 5. SPB instance sub-TLV [17].....	20
Figure 6. Proposed Control Plane Architecture of IS-IS extensions for SPB.....	22
Figure 7. SpbInterface Class Diagram	24
Figure 8. SpbHelloExtension Class Diagram	25
Figure 9. Acquiring Neighbor's Bridge ID.....	27
Figure 10. SpbInstanceSub and VIDTuple Class Diagram	28
Figure 11. SpbLinkMetrics Class Diagram	29
Figure 12. SpbmServiceIdentifier Class Diagram	29
Figure 13. SpbLinkStateDataBase Class Diagram	30
Figure 14. SPB Routing Architecture	31
Figure 15. Shortest Path Tree Rooted at Node 0 when ECT-Algorithm 0 is used.	39
Figure 16. Multicast B-DA Bit Format.....	41
Figure 17. Receive and Send	43
Figure 18. SpbHelper Class Diagram	44
Figure 19. Experimental Topology	47
Figure 20. Topology in Script.....	48
Figure 21. Grouping in Script	49

Figure 22. Initial and Final Matrix When ECT is 1	51
Figure 23. FDBs when ECT is 1	52
Figure 24. Multicast When ECT 1	53
Figure 25. Multicast When ECT 2	53

LIST OF ACRONYMS

B-DA	Backbone Destination Address
BEB	Backbone Edge Bridge
B-SA	Backbone Source Address
B-VID	Backbone VLAN ID
ECT-Algorithm	Equal Cost Tie-breaking Algorithm
IIH	IS-IS Hello protocol
I-SID	I-component Service Instance Identifier
IS-IS	Intermediate System to Intermediate System
LAN	Local Area Network
LSDB	Link State Database
NIC	Network Interface Card
OAM	Operations, Administration, and Maintenance
PDU	Protocol Data Unit
SPB	Shortest Path Bridging
SPBM	SPB Mac-in-Mac
SPBV	SPB Q-in-Q
STP	Spanning Tree Protocol
TLV	Type Length Value
UNI	User Network Interface

CHAPTER 1: INTRODUCTION

Since Ethernet was released to the market of local area network (LANs), it has been successfully and widely deployed. The main strengths of Ethernet are simplicity and backward compatibility. The Plug-and-play feature of Ethernet simplifies network configuration. This feature is desirable especially in cases where user locations are dynamically decided. Backward compatibility makes Ethernet switches more attractive compared to other competing LAN technologies [1].

In cloud data centers it is desirable to minimize network configuration while supporting large numbers of switches. Virtual machine instances in a data center can be moved any time and located any place. A cloud data center consists of large numbers of switches and servers. Cost efficiency of switches plays major role in constructing a cloud data center. As seen above, Ethernet fulfills these requirements sufficiently [2].

Since networks have to evolved to demand more functionality, Ethernet has evolved to support virtualization. First, Virtual LAN (VLAN) emerged to support Ethernet virtualization. Its frame contains VLAN tags [3]. Second, Ethernet supports service identification for individual users. The traffic for each user is assigned the unique number in the provider network. This identifier called I-SID [4] enables a provider to offer a fine-grained service to individual users. Finally, Ethernet has evolved to support more sophisticated routing algorithms. Ethernet at its conception only supported broadcast communication through shared medium. Spanning Tree Protocol (STP) [5] introduced a new control scheme to support redundant paths in Ethernet domain. However, STP still does not fully utilize all links in a network

because of loop formation. Shortest Path Bridging (SPB) [6] addresses the shortcomings of STP through a sophisticated control scheme and routing algorithm.

1.1 Thesis motivation

Though Shortest Path Bridging (SPB) is emerging, there is currently no open source SPB network simulator. The absence of an open source SPB simulator is a major obstacle for many research institutions. First, Ethernet bridges running SPB protocol are still not affordable. Second, constructing real life experimental network for research prototyping requires major efforts relative to simulation. Thus, this thesis proposes a SPB simulator on the open source, discrete event network simulator NS-3. NS-3 is the successor of the previously popular open source simulator NS-2.

1.2 Contributions of the thesis

In order to develop the SPB simulator running on NS-3, research on SPB protocol and NS-3 has been conducted. First, the open source software module for IPv4/OSPF network was analyzed so as to design the software architecture for the SPB simulator. Second, the IEEE and IETF SPB standard documents were reviewed. Third, the SPB simulator was built on NS-3. Finally, experiments for validating the simulator were conducted. Details of each step are described throughout this report.

1.3 Thesis Outline

The rest of this report is organized as follows. Chapter 2 reviews the literature referenced by this report. This chapter explains history of Ethernet and LAN

technology evolution. It describes why Shortest Path Bridging (SPB) has emerged. The open source simulator NS-3 is also examined. Chapter 3 presents the software architecture of SPB simulator. Chapter 4 describes the control plane of SPB and implementation details about the control plane of the simulator. Chapter 5 explains the data plane of SPB and implementation details. Chapter 6 demonstrates and examines the method of conducting a SPB simulation using the developed simulator. This chapter presents the analysis of the results of simulations. Chapter 7 draws conclusions, and examines possible area for future work with the simulator.

CHAPTER 2: LITERATURE REVIEW

Shortest Path Bridging (SPB) has been developed to overcome the limitations of legacy Ethernet protocols while allowing the application of advanced networking concepts on the Ethernet. As Ethernet expands its application from LAN to provider networks, SPB also is designed to operate in large scale networks such as providers' networks, back-haul networks, and metro Ethernet. This chapter reviews literature related to SPB. First, this chapter explains the back ground of why SPB developed. The problems and limitations of legacy Ethernet protocols are reviewed. Legacy Ethernet protocols are the major Ethernet protocols currently deployed in real world networks. This chapter does not explain about problems of older Ethernet protocol developed in 90's. Second, this chapter examines the advanced network techniques adopted by SPB. Finally, NS-3, an open source network simulator, is introduced.

2.1 Problems of legacy Ethernet protocols

Shortest Path Bridging (SPB) addresses the problems of legacy Ethernet network. There are problems in both control plane and data plane. In control plane, minimizing convergence time after topology changes is a major challenge. In data plane, inefficiency of link utilization is the main problem.

Switches in an Ethernet network share the same physical medium. The routing system of an Ethernet network was still based on the flood-and-learning mechanism. Massive amounts of frames flooding in the network may cause broadcast storms, effectively melting down the network. To prevent broadcast storms while permitting such a flood-and-learning mechanism, the Spanning Tree Protocol (STP) was

designed [5]. Redundant paths in an STP network are disabled to suppress loop formation. The STP locates each end point on a spanning tree hence is a single point of failure. This simple connectivity is not on the optimal path between two end points. To guarantee the loop free status at all times, any topology change shuts down all connectivity on the spanning tree until the new tree has converged [7]. Though this shutdown period is no longer than a few seconds, it is still not acceptable.

The main problem of STP's data plane is inefficient use of links in a network. First, to avoid loop formation in STP region, some of the links in a network must be disconnected. Another problem is that traffic within an STP region does not follow the optimal path frequently. Since an STP region is a replica of shared medium, it does not provide a sophisticated routing mechanism. Figure 1.a demonstrates the problems of STP data plane. First, the link between switch B and D is disabled to prevent loop formation. If the link is enabled, there is the loop $A \Rightarrow B \Rightarrow D \Rightarrow A$. Second, traffic between switch C and E, does not follow the optimal path $C \Leftrightarrow B \Leftrightarrow D \Leftrightarrow E$.

Shortest Path Bridging (SPB) resolves the main problems of STP's data plane. In an SPB region, all links can be used and none need be blocked for loop prevention [1]. All switches in a SPB region share a global topology view. Based on the global topology view, each switch configures itself to carry traffic on the optimal path.

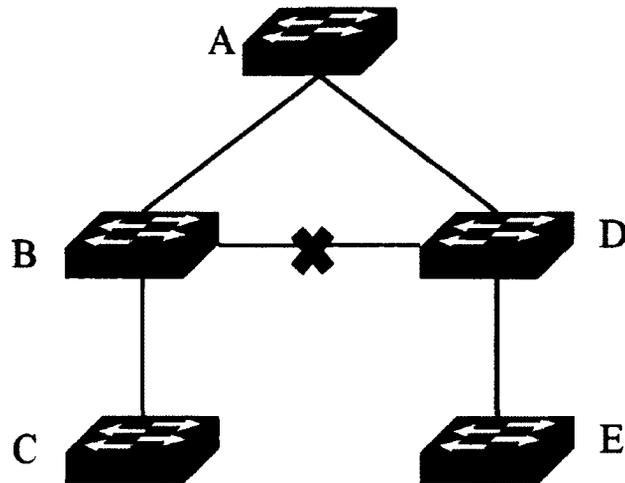


Figure 1.a STP Problems

2.2 Resolving the problems of legacy Ethernet

As seen above, Spanning Tree Protocol (STP) does not construct optimal topologies and its convergence time is unacceptable. Shortest Path Bridging (SPB) emerges to overcome these shortcomings of STP [8]. SPB, as its name suggests, offers optimal paths between each end point. The switches in the same SPB region share the same topology information. In contrast to STP, SPB does not require topology information exchanges through the flood-and-learn mechanism.

Switches in the Shortest Path Bridging (SPB) domain build Link State Databases (LSDB) in which global topology information is stored. A LSDB is periodically updated through Intermediate Systems to Intermediate Systems (IS-IS) protocol [9]. A topology change also triggers LSDB updating. Since a SPB enabled switch responds to the event of topology changes without a flood-and-learn process, its convergence time is faster than Spanning Tree Protocol (STP).

All switches in a SPB region share the same topological information at a LSDB in each one. Unlike STP, all links in a SPB network can be utilized. Thus, it is possible to compute the optimal path in the given topology [7] [10].

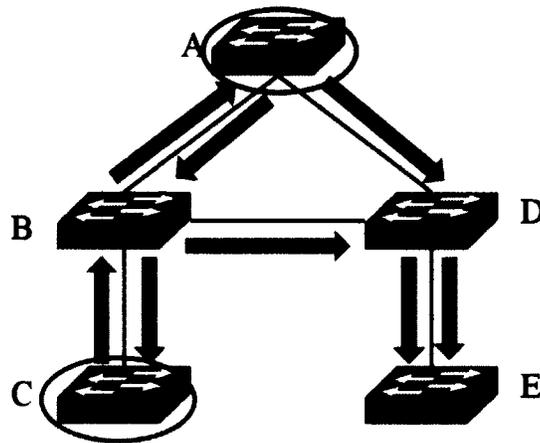


Figure 1.b Possible SPB routes

Figure 1.b Possible SPB routes demonstrates possible traffic routes when SPB is applied to the same topology of Figure 1.a. Since every node in the SPB network computes a spanning tree rooted at it and the traffic sourced at the node always follows the spanning tree, SPB network does not need to disconnect one of the links to prevent loop formation. Traffic sourced at node C in Figure 1.b Possible SPB routes follows red arrows while traffic from node A follows blue arrows. There is no unused link in the topology. Therefore, SPB has higher link utilization factor than STP.

2.3 Virtualization and Fine-grained Services of Ethernet

Virtualization of an Ethernet network is dividing the network as multiple logical domains without regard to physical topology of the network. This approach has numerous benefits. First, it reduces a size of a broadcast domain. Second, it enhances the security of the network by blocking unknown traffic. Third, it makes it possible to direct traffics through pre-assigned routes. Virtualized Ethernet LANs are called Virtual LANs (VLANs) [3]. VLAN inherits all these benefits from the virtualization.

In cloud network, where virtual machine instances are relocated dynamically, minimizing the overhead from managing virtual machines locations is necessary. VLAN can add, remove and relocate virtual machines using software configuration. Therefore, VLAN techniques can be used to minimize overhead. Shortest Path Bridging (SPB) adopts the VLAN concept so it has the same ability as VLAN. Each frame in SPB networks has the slot for VLAN identifier. This slot is called Backbone VLAN ID (B-VID).

SPB identifies a service instance. A service instance is a group of Backbone Edge Bridges (BEBs) that support a given customer's VLANs [11]. Every service instance is assigned I-component Service Instance Identifier (I-SID). I-SID came from Provider Backbone Bridge (PBB) [4], IEEE 802.1ah, frame header as well as B-VID. Physical or logical User Network Interface (UNI) ports are assigned I-SID. UNI ports are at BEBs and they are facing customer's networks. BEBs are located at the border between a SPB network and a customer network as shown in Figure 2. For example, a customer frame with a VLAN identifier reaches to a UNI port at a BEB. The frame passes through the UNI port and arrives at I-component which encapsulates the frame with an I-SID matched with the customer VLAN identifier as well as Backbone Source Address (B-SA) and Backbone Destination Address (B-DA). This encapsulated frame travels across a SPB network and reaches another BEB with the B-DA. Finally, it is decapsulated at the I-component linked to a UNI port.

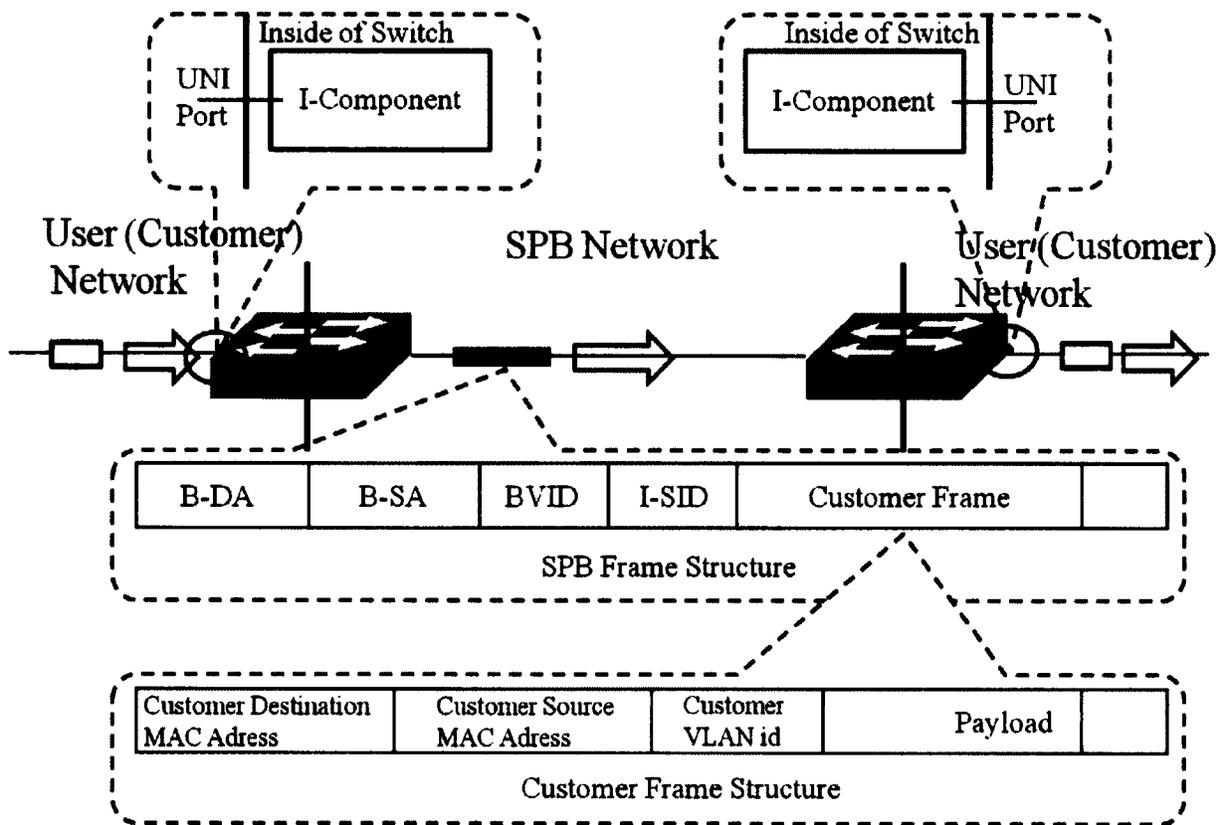


Figure 2. En/Decapsulation at UNI port

2.4 Network Simulator NS-2 and NS-3

NS-3[12] is a discrete-event network simulator. It is an open source project and successor of the previously most popular open source network simulator NS-2[13]. NS-3 consists of core modules and additional libraries of simulation models. Core modules are event scheduler, tracing system, attribute system, core network models (such as node, device, channel and application). Additional libraries come from NS-3 community. These libraries are used to support new protocols or hardware models.

With NS-3, researchers can develop experimental protocols and simulate them on arbitrary simulation topologies. Simulation results can be collected as text files. NS-3 supports a file format used in many networking tools like TcpDump and

Wireshark¹. Since its source code is open, researchers can develop a tracing system generating specific file format used to draw diagram, graph and animations. Its design concept and goals are referenced in [14][15].

NS-2 consists of two different programming language domains. A script, which is in charge of building simulation topology and configuring simulation parameter in NS-2, is written in TCL/OTCL languages [16]. The core of NS-2, conducting simulation and generating results, is written in C++. These script and core domains communicate with each other by binding their class and variables. Therefore, depth of NS-2 call stack is usually very deep. The depth of call stacks and two different language domains make debugging difficult. When NS-2 was released, building an executable file took a long time. Simulation scripts change frequently so NS-2 separated script domain from the core. Therefore, a change in a script does not require building whole binary file.

NS-3 also supports python and C++ binding but it can run script written in C++. In contrast to when NS-2 was released, current computing hardware is much faster than the past. So NS-3 does not need to follow NS-2 architecture any more. In NS-3, a script is main function of executable file and core parts are linkable libraries to the executable file. NS-3 supports both static and dynamic linking.

¹ TcpDump and WireShark : packet analyzers. It allows the user to intercept and display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. <http://en.wikipedia.org/wiki/Tcpdump> .

2.5 Network topology of NS-3

In the real world, a computer network consists of hosts and links. NS-3 offers software simulation models of hosts and links. The model of a host in NS-3 is called node and its corresponding C++ class is “Node”. A link is represented as the “Channel” class in NS-3 simulation.

Since a host in the real world consists of several components, and likewise a node in NS-3 has several components. A host, which can be a server or a router, has software parts and hardware parts. A network interface card (NIC) is an interface between a host and a link. A frame or packet comes in or goes out of NIC is managed by software protocol handlers. These protocol handlers also manage data from upper layers and user applications.

In NS-3, a node is abstraction of a host in the real world, a just like a real world node, a NS-3 node also has the models of NIC, protocol handlers and application code. The NIC model of NS-3 is “NetDevice” class. NS-3 offers IPv4 and IPv6 protocol stacks. These stacks interact with TCP/UDP socket layers. A user application model of NS-3 is “Application” class.

Figure 4 demonstrates how NS-3 models are combined. The nodes in figure 4 have OnOffApplication and Sink classes as an application model. Their protocol stacks consist of UDP/IP and ARP classes with respect to transport and network layers models. CsmaNetDevice and CsmaChannel classes are selected as their link and physical layer models respectively.

OSI Layers	Corresponding NS-3 models
Applications	UDP echo, OnOffApplication, sink
Transport Layer	UDP, TCP
Network Layer	IPv4/IPv6 Global(OSPF), Static Routing..
Link Layer	PointToPoint, CSMA, 802.11
Physical Layer	Wired channel (CSMA, Point to Point) , Wireless channels (802.11a, LTE..)

Table 1. NS-3 Models

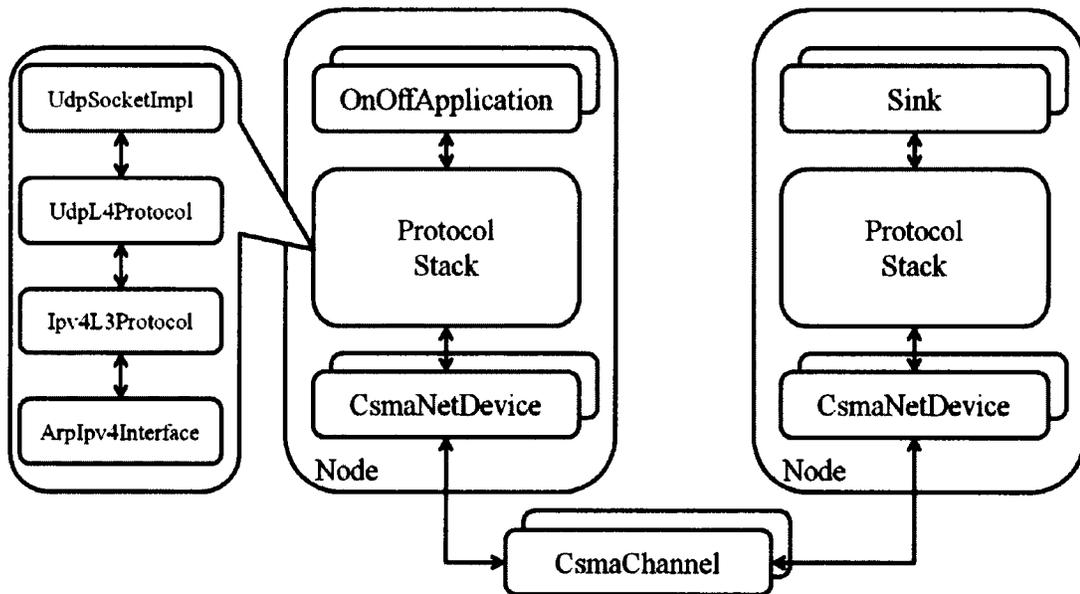


Figure 3. Nodes with Internet Stack and CSMA Channel in NS-3

CHAPTER 3: ARCHITECTURE OF THE SIMULATION

3.1 Architecture of SPB node

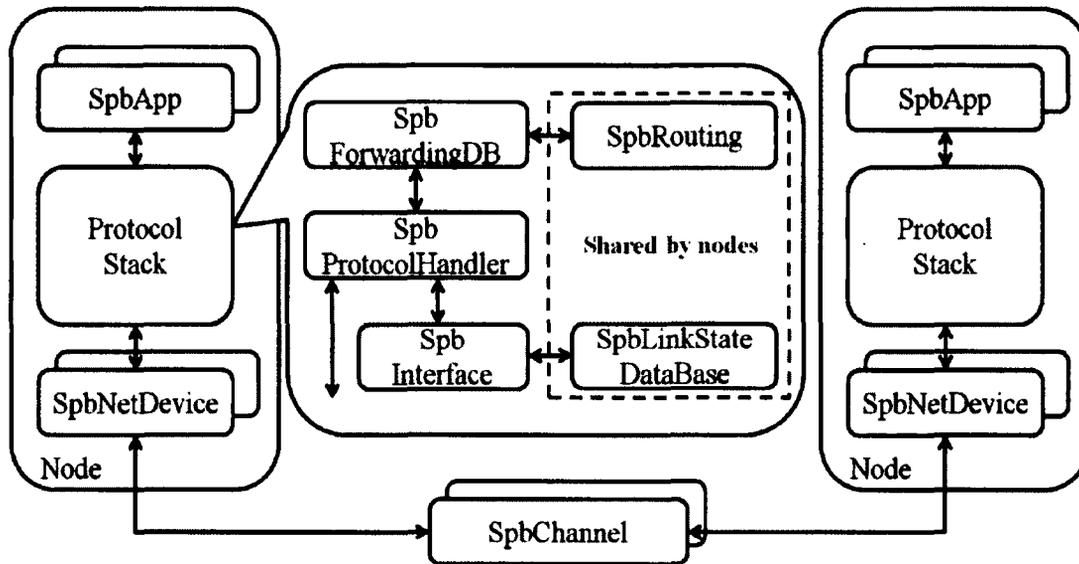


Figure 4. Nodes with SPB Stack and SPB Channel

Figure 4 presents our architecture of SPB nodes. An SPB node models the bridge capable of running the SPB protocol. Channel is the model of a link. SpbRouting and SpbLinkStateDataBase are singleton objects shared by all SPB nodes. In the real world, an SPB switch may have its own SPB link state database. A link state database in a SPB node is synchronized with other databases installed in other nodes unless there is a miss-synchronization. Therefore, in most cases link state databases in the same SPB domain are identical. Current simulation design does not consider miss-synchronization because the objective of our design is to offer a base SPB model for future extension. In contrast to a normal Internet node, SPB node's protocol handler communicates with SpbNetDevice directly since that SPB is a layer 2 protocol. The following is the description of each module.

- **SpbLinkStateDataBase:** This is the hash table where all references to each SpbInterface are saved. Its key is a nodal B-MAC address which identifies a node. Therefore, a query with a nodal B-MAC returns the SpbInterface on the node. This is a singleton object which has only one instance in the simulator. It is a global database shared by every node in the simulator.

In a real world situation, a SPB enabled switch has its own Link State Database (LSDB) and the LSDB is synchronized with other LSDBs installed in other switches.

- **SpbInterface:** This is the interface to the control plane module for a node. It offers APIs for accessing Intermediate System to Intermediate System (IS-IS) sub-TLVs for SPB. Through this module, we can set or get sub-TLV's values and manage the sub-TLVs. For example, we can assign an I-SID on a node in order to register the node in the specific group represented by the I-SID.

In a real world situation each SPB switch (represented by a node in simulation) constructs IS-IS sub-TLVs and exchanges the digests² of them with neighboring switches. After exchanging the digests, a node in a SPB network can build LSDB.

- **SpbApp:** The model of a packet source and sink. It generates a packet and forwards it to SpbProtocolHandler to send a packet to other nodes. It also consumes the packets whose destination is matched by the node on which it is installed.

² IS-IS exchanges a summary of information like node identifiers and link costs

- **SpbProtocolHandler:** This module extracts the SPB headers of received frames from SpbNetDevice. It queries the forwarding (filtering) database (FDB) of the node to select the proper output ports.
- **SpbNetDevice:** The software/hardware model of the network driver of SPB and the input/output interface of a frame to/from outside of the node. This is the same as output port of a bridge. It is inherited from NS-3 base class NetDevice. It encapsulates the packet received from layer above the current layer. When it receives a frame from the SpbChannel, it forwards the frame to SpbProtocolHandler.
- **SpbChannel:** The model of a SPB link. It has two references of SpbNetDevice as endpoints of the link. The weight of the link is user configured.
- **SpbForwardingDB:** The model of a forwarding database of the SPB enabled by the bridge. When SpbProtocol-Handeller and SpbRouting objects query SpbForwardingDB, SpbForwardingdDB returns the reference of the table entry or the output port referencing the SpbNetDevice object.
- **SpbRouting:** This module is in charge of computing the path and populating the forwarding databases on nodes according to the result of the computation.

3.2 Actions of a SPB node

The following is the brief description of how a node handles the received frame:

1. When a node receives a frame from outside, the frame arrives at SpbNetDevice. A frame is a pointer value referencing the NS3::Packet object.

2. SpbNetDevice calls the callback function registered at the node.
3. The node looks for the correct handler for the packet type.
4. If it is a SPB packet, the node calls SpbProtocolHandler.
5. SpbProtocolHandler extracts SPB header from the received packet.
6. SpbProtocolHandler queries SpbForwardingDB to know the output port number. Every SpbNetDevice has corresponding output port number. If the output port number is '0', SpbProtocolHandler moves the packet to SpbApp. Otherwise, move the packet to corresponding SpbNetDevice

When a node send a frame:

1. SpbApp generates a packet and forwards it to SpbProtocolHandler with I-SID and traffic type. There are two traffic types, unicast and multicast.
2. SpbProtocolHandler queries SpbForwardingDB and SpbInterface. First, SpbProtocolHandler queries SpbInterface to get BVID corresponding to I-SID received from SpbApp. Second, it queries SpbForwardingDB with BVID and Nodal MAC address of the node.
3. SpbProtocolHandler checks the outport referencing one of the SpbNetDevice installed in the node. SpbProtocolHandler adds SPB header to the packet and moves it to SpbNetDevice.
4. SpbNetDevice corresponding to the output port sends the frame to connected SpbChannel.

CHAPTER 4: SHORTEST PATH BRIDGING CONTROL PLANE

In general, the term control plane³ refers to a part of the network architecture that collects the information of a network topology, performs the routing calculations required to direct traffic, and manages networks. The information collected from the control plane is used to build a Forwarding Database (FDB). Since Shortest Path Bridging (SPB) has evolved from the latest Ethernet technology the SPB control plane not only deals with the traditional LAN topology information, but also the service and virtual LAN identifiers. The important thing to consider is that an SPB switch encapsulates a customer frame into its frame when it receives the customer frame, and thus the SPB control plane is isolated from a customer network.

4.1 IS-IS for Control Plane protocol

The Intermediate System to Intermediate System (IS-IS) link state protocol is the base protocol used to control Shortest Path Bridging (SPB) network [6]. The basic mechanism of link state protocol is that every node, either routers or switches, have the same view of the network topology. The Hello protocol of IS-IS is used to learn about adjacent nodes and maintain adjacencies between neighboring nodes. A flooding protocol (link state update) is used to exchange information such as link metrics, path identifiers and group memberships. This information about nodes, links, paths and memberships is globally unique in a single Shortest Path Bridging (SPB) network domain. A network topology discovered by IS-IS is represented as a graph consisting of nodes and connecting links.

³ In routing, the control plane is the part of the router architecture that is concerned with drawing the network map and building routing table. http://en.wikipedia.org/wiki/Routing_control_plane

IS-IS is also an interior gateway protocol which is used within a single administrative network. IS-IS provides the distributed database for each SPB node in the same network. The information about nodes, links, path and memberships are exchanged between nodes in the same SPB network, and this data is used to populate the database at each node. Based on the database, each node in the network computes the forwarding path to other nodes independently. Based on the forwarding decision, each SPB nodes produces forwarding tables (FDB) by using its database.

IS-IS can work in OSI network layer to discover a network topology. Another link state and interior gate protocol Open Shortest Path First (OSPF) carries information over IP protocol. On the other hand, IS-IS does not depend on certain network address format (such as IP addresses). Since SPB is a layer 2 routing protocol for an interior network, thus IS-IS fulfils the requirements of SPB [17].

4.2 Extension of IS-IS for SPB

Shortest Path Bridging (SPB), IEEE 802.1aq, extends the normal Intermediate System to Intermediate System (IS-IS) Protocol Data Unit (PDU) to carry SPB network information. This information includes node identifiers, link metrics, and adjacencies represented by SPB network terms. SPB can run in parallel with other network layer protocols. Therefore, SPB requires Network Layer Protocol Identifier (NLPID) and the NLPID value 0xC1 assigned to SPB. A node advertising NLPID value 0xC1 in IS-IS Hello (IIH) protocol can be a member of a SPB network.

IS-IS extension for SPB is augmented with a small number of Type Length Value (TLV) and sub-TLVs. SPB supports two different modes of operation. In

SPBM (SPB MAC) mode, a group of service is identified by I-SIDs (Ethernet Services Instance Identifier). In SPBV (SPB VID) mode, a group of services is identified by MAC addresses. Our scope in this thesis is limited to SPBM mode.

Control Protocol of SPB and extensions to IS-IS consists of four major parts. The First part is Hello (IIH) protocol extensions. IS-IS Hello (IIH) Protocol is used to detect neighbor nodes that are capable of running SPB protocol. IIH is also used to exchange and digest information. The digest information holds two bridge identifiers (bridge priority || bridge sysID) and link metrics of neighboring nodes sharing the same link. The associations between Virtual LAN IDs (VID) and equal cost tie-breaking algorithms are also included in a digest. The digest information is used to validate the commonality of Link State Databases (LSDB) in the same SPB domain. Since IIH is a digest of topology information, other IS-IS extensions for SPB carry the full information. Thus the SPB simulator uses IIH only for detecting neighboring nodes.

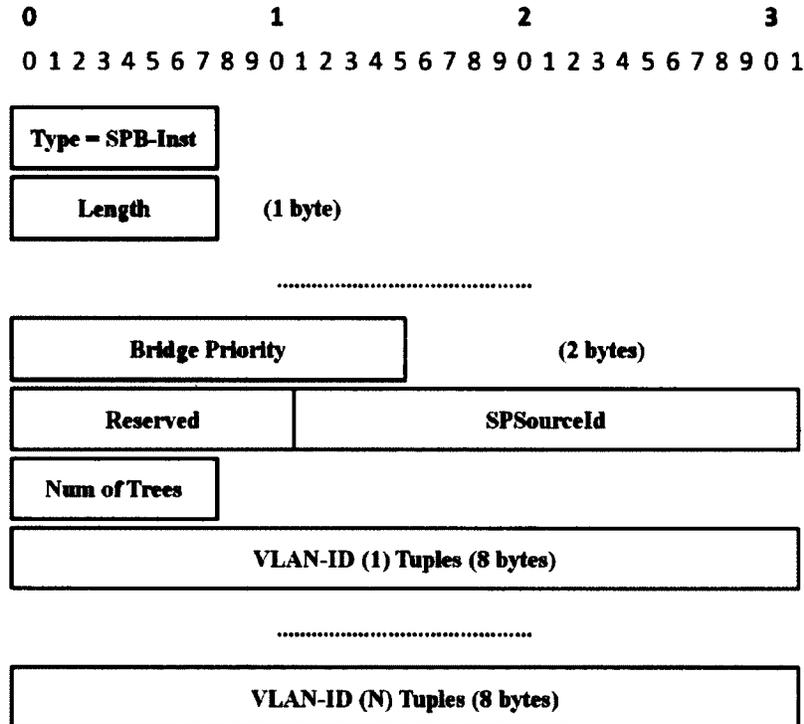


Figure 5. SPB instance sub-TLV [17]

The Second part is SPB Instance (SPB-Inst) sub-TLV which carries nodal information with the following characteristics:

- **SPSourceID:** 20-bit value of nodal nick name. It forms multicast destination address (DA).
- **Bridge Priority:** This 16-bit value combined with 4 bytes System ID forms the Bridge Identifier.
- **Number of Trees:** It is set to the number of VLAN-ID tuples.
- **VLAN-ID tuples:** It consists of 4 bytes ECT-Algorithm⁴, 12-bit Base VID, 12-bit SPVID and U, M, A flag bits. When an SPB bridge advertised an ECT-

⁴ Appendix A lists up all ECT-Algorithms. Chapter 4.7 explains how ECT-Algorithms affect on routing decisions.

Algorithm with Base VID in the same tuple, the ECT-Algorithm is applied on the given Base VID. Base VID is used for SPBM mode and SPVID is used for SPBV mode.

The third part is the adjacency information extensions. Its corresponding sub-TLV is SPB link Metric sub-TLV. This sub-TLV consists of:

- **SPB-Link Metric:** 24-bit unsigned number. Administrative weight of a link. Smaller number indicates lower weight. When two neighbor nodes advertise different SPB-Link Metric value on the same link, the maximum value is considered as the right value of the given link.
- **Port Identifier:** 16-bit. The standard IEEE port identifier

The fourth part is service information extensions and it is carried in SPBM Service Identifier and Unicast Address (SPBM- SI) sub-TLV. It consists of:

- **B-MAC address:** A unicast address of the node. Each SPB node has the unique nodal Backbone MAC address. This may address a port. When multiple B-MACs are used, this TLV must be repeated per B-MAC. The SPB simulator so far supports only one nodal B-MAC.
- **Base VID:** This Base VID also appeared as VLAN-ID tuples in SPB-Inst sub-TLV. Thus, B-MAC address, Base VID, and ECT-Algorithm are all associated.
- **I-SIDs:** 24-bit group identifier. This I-SID set is assigned the Base VID in the same SPBM-SI sub-TLV. If two different nodes advertise the same I-SID, intermediate nodes between the two nodes will configure themselves to carry

traffic generated from the two different nodes. The intermediate nodes will create Filtering Database (FDB) for unicast and multicast addresses.

4.3 Control Plane Software Architecture

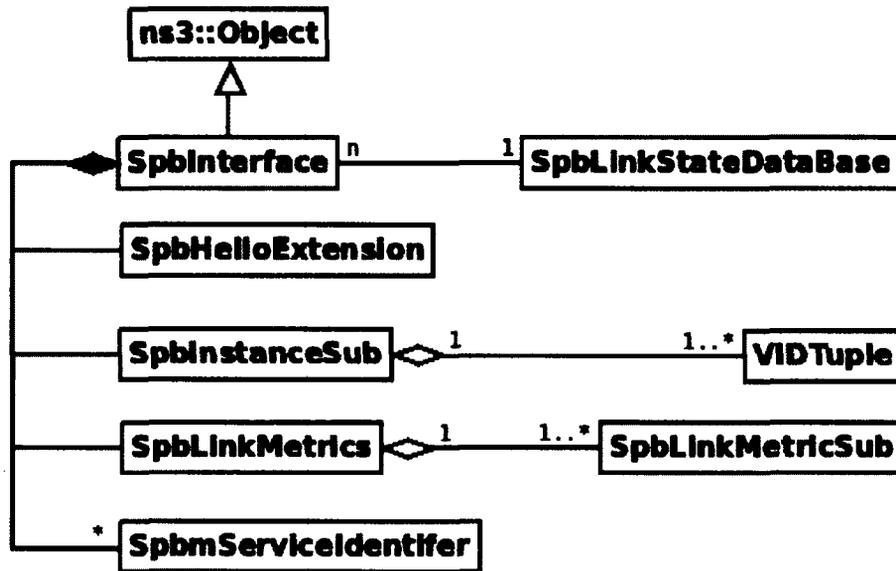


Figure 6. Proposed Control Plane Architecture of IS-IS extensions for SPB

Figure 6 shows the class diagram of sub-TLVs and link state database (LSDB). SpbInterface inherits “ns3::Object.” An “ns3::Object” instance can be aggregated to a Node instance. SpbLinkStateDataBase is a hash table with keys corresponding to node MAC addresses and values containing SpbInterface pointers referencing the node with the MAC address. Since a node’s MAC address is a unique value identifying a node, we can use it as the key value of SpbLinkStateDataBase. Here is the brief description of each class:

- **SpbHelloExtension:** Corresponding to Intermediate System to Intermediate System (IS-IS) Hello protocol extensions (IIH). IIH consists of three different sub-TLVs. However, in our simulations, we simplify IIH since we assume that

link state database is always synchronized correctly. This class is used for detecting adjacent nodes.

- **SpbInstanceSub:** corresponds to SPB Instance sub-TLV. It can include multiple VIDTuple classes.
- **SpbLinkMetrics:** corresponds to SPB Link Metric sub-TLV. It can include multiple SpbLinkMetricSub.
- **SpbmServiceIdentifier:** corresponds to SPBM Service Identifier and Unicast Address sub-TLV
- **VIDTuple:** It consists of flag bits and a set of ECT-Algorithm, Base VID and SPVID. SPVID is used only for SPBV (SPB Q-in-Q) mode. A SPB node may be assigned multiple B-VIDs so the SpbInstanceSub class can include multiple VIDTuple classes.
- **SpbLinkMetricSub:** Since a SPB node is attached at least one link, the SpbLinkMetrics class needs multiple SpbLinkMetricSub in order to represent each link.

4.4 Implementation of IS-IS extension for SPB

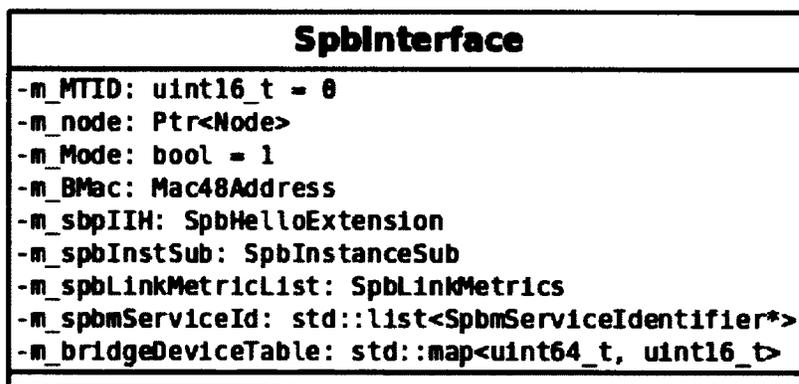


Figure 7. SpbInterface Class Diagram

Figure 7 presents the member variables of SpbInterface class. SpbInterface class is the interface between IS-IS sub-TLVs for SPB and outside classes. Outside classes set and get a variables of IS-IS sub-TLVs.

SpbInterface offers API's to manage sub-TLVs. SpbInterface plays a crucial role in configuring experimental topologies in simulation. It creates ECT-Algorithm:BVID and BVID:I-SID pairs. Routing decisions are based on these pairs. SpbInterface also offers major APIs below.

- **AddSpbSet (uint8:ect, uint16:BVID, uint32:ISID) :** Takes input 'ect' as index of ECT-Algorithm. Sets up SPBM Service Identifier sub-TLV and SPB Instance sub-TLV. Adds (ECT:BVID) pair and (BVID:ISID) pair into control plane
- **BuildAdjacency (void) :** Accesses neighboring node by channel. Collects neighboring node's bridge identifier through neighboring node's SpbInterface.

- **GetECT (int16:BVID)** : Gets ECT-Algorithm index⁵ assigned on the input BVID
- **GetVID (int32:ISID)** : Returns BVID value of which input ISID makes pair.
- **GetECT (int32:ISID)** : Gets ECT-Algorithm index assigned on the input ISID.
- **GetPortIdTo (uint64:BridgeID)** : Returns output port number to neighboring node with input BridgeID. BridgeID is concatenation of the bridge priority and the bridge system id.

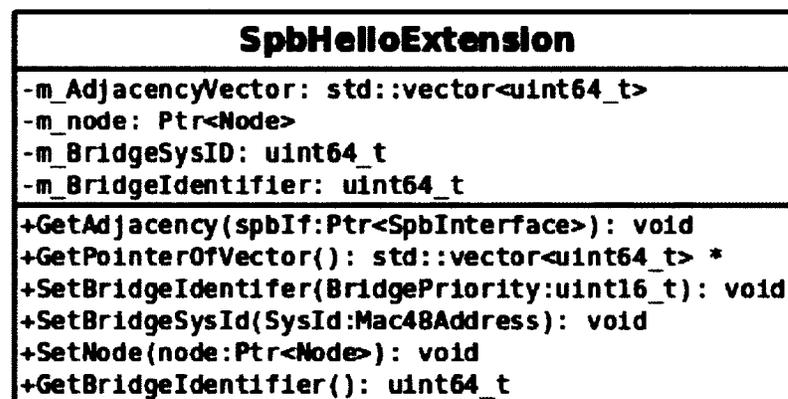


Figure 8. SpbHelloExtension Class Diagram

Figure 8 is the class diagram of SpbHelloExtension. For convenience, each node with SPB stack is assigned the system identifier, the numerical form of MAC 48 address. For example, if an SPB node has MAC address of hexadecimal numbers 00:00:00:00:00:01; its system identifier is 1.

⁵ The value of the left most byte of ECT-Algorithm is the index. ECT-Algorithms are stored as array structure in the simulator. The ECT-Algorithm index is the same as the array index.

Bridge priority is 1 byte unsigned integer value. A network administrator can assign the bridge priority to each SPB node. This helps the network administrator direct traffics.

This class sets the bridge identifier of a node. Bridge identifier, 64 bits unsigned integer type, is a concatenation of a bridge priority and a bridge system identifier (Bridge priority: $16 \ll 48$ | system identifier: 48). For instance, if an SPB node with 0xff bridge priority and 00:00:00:00:00:01 MAC address, the node has 0xff000000000001 as a bridge identifier.

- **GetAdjacency (spbIf:Ptr<SpbInterface>)** : collects the BridgeIDs of neighboring nodes. This is called by BuildAdjacency member function of SpbInterface.
- **GetPointerOfVector(void)** : returns the pointer of vector object which stores BridgeIDs of neighboring nodes

A class Node stores all the references of NetDevice class. SpbNetDevice is the child class thus a SpbNetDevice object can be also stored in a Node object. Each NetDevice object is assigned unique index number to identify it among all the NetDevice attached to the node. Class NetDevice also has the reference to the Node which stores the NetDevice reference. Therefore, Node and NetDevice objects can reference each other.

Class NetDevice has the reference to the Channel object to which it attached. SpbChannel is a child class of Channel. Therefore, each SpbNetDevice object has the reference to the SpbChannel object. A SpbChannel object has two references to the SpbNetDevices at each end.

The cross referencing model explained above makes it possible to have access to a neighboring node object. As Node object aggregates a SpbInterface object, acquiring any SPB link state information saved in SpbInterface object is possible.

Figure 9 demonstrates the flow chart for SpbHelloExtension::GetAdjacency member function.

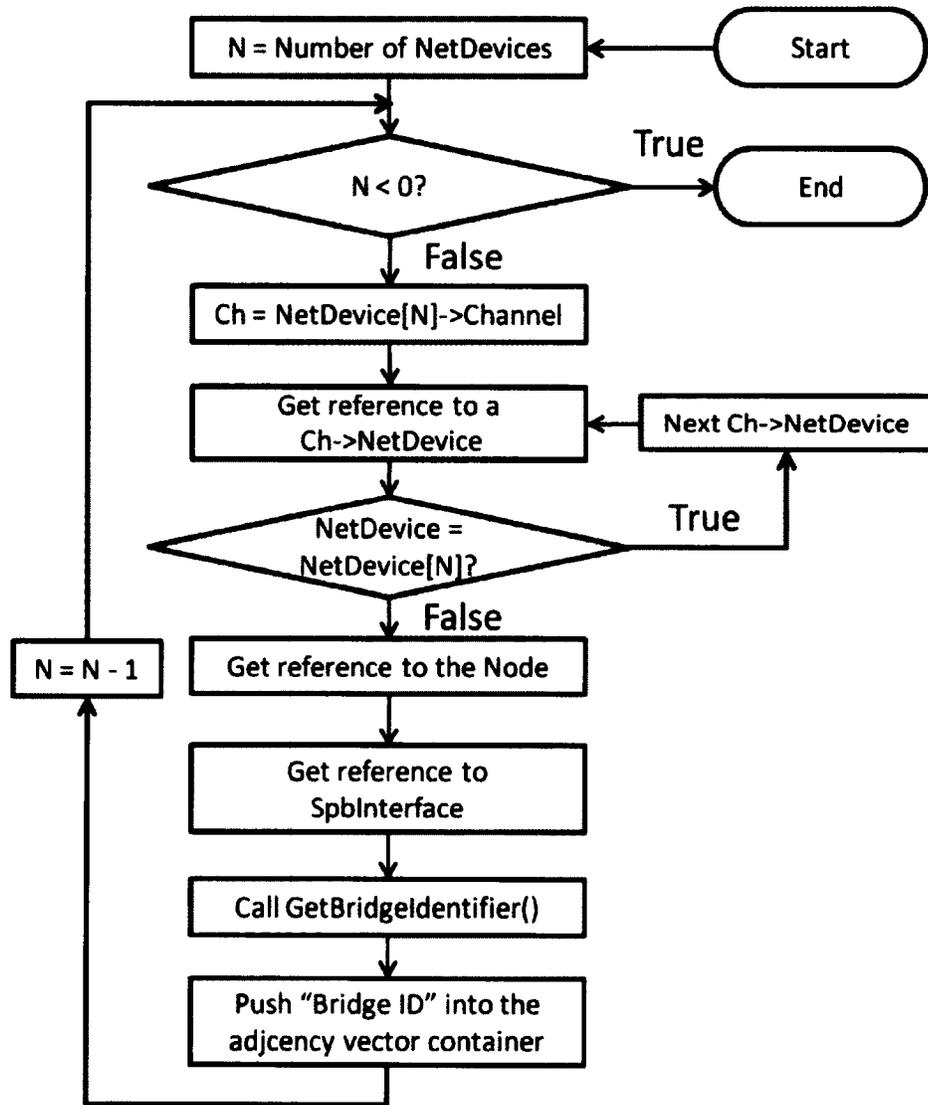


Figure 9. Acquiring Neighbor's Bridge ID

Figure 10, Figure 11 and Figure 12 demonstrate class diagrams designed for sub-TLVs of IS-IS extension for SPBM. Through SpbInstanceSub, SpbLinkMetrics and SpbmServiceIdentifier classes, we can set and get the sub-TLV values.

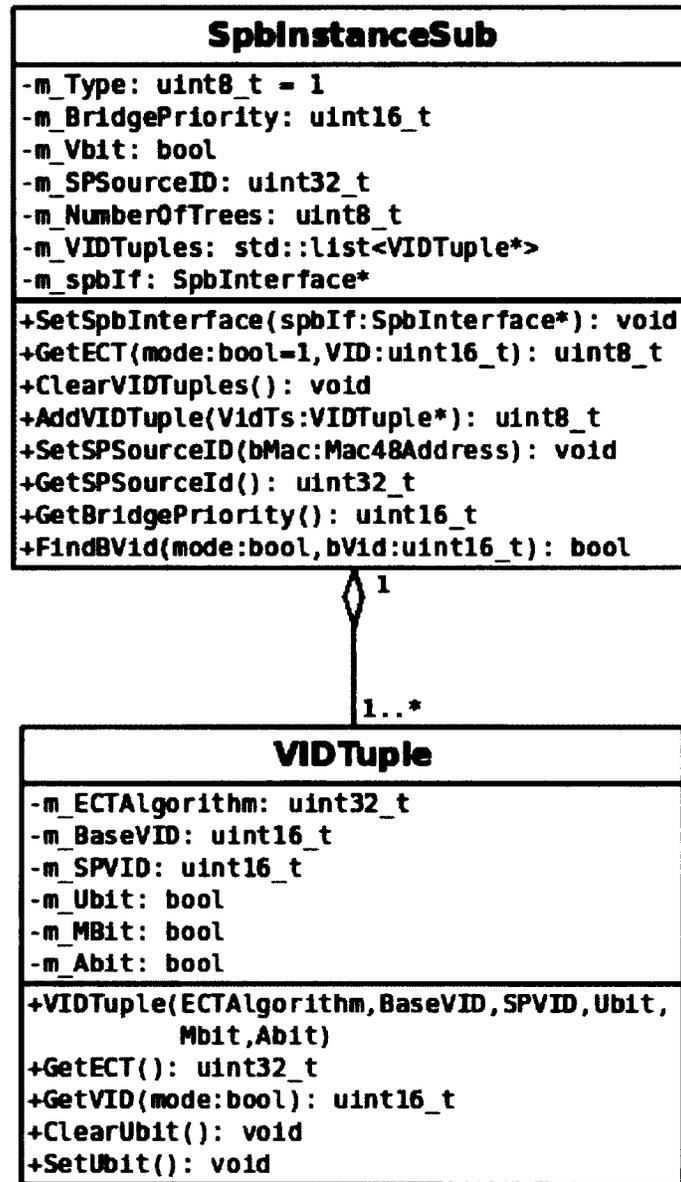


Figure 10. SpbInstanceSub and VIDTuple Class Diagram

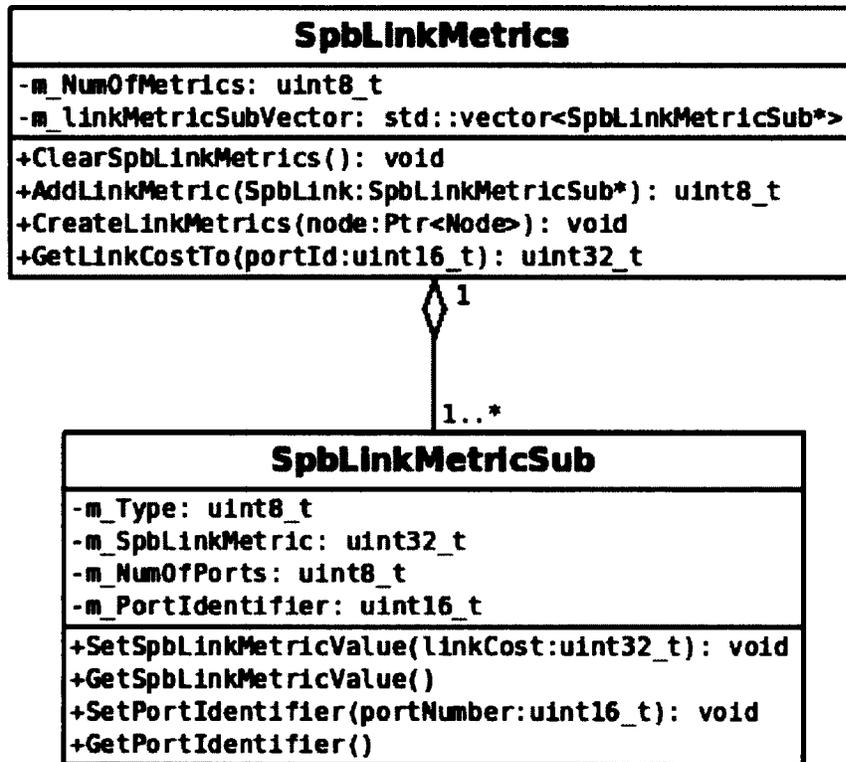


Figure 11. SpbLinkMetrics Class Diagram

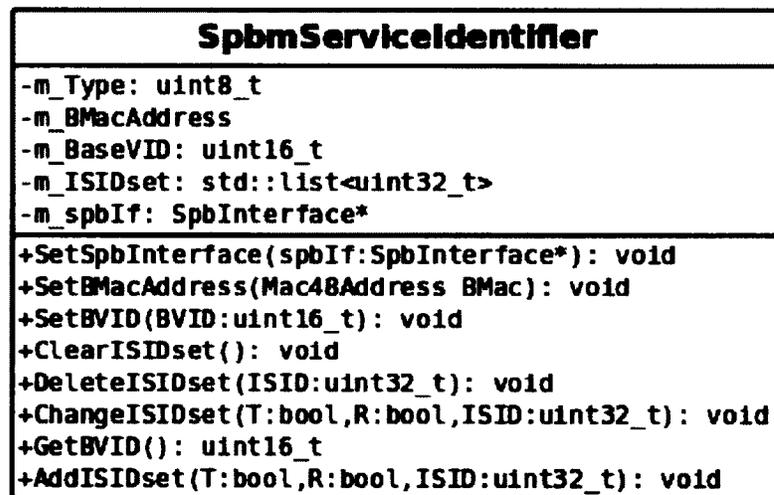


Figure 12. SpbmServiceIdentifier Class Diagram

4.5 Link State Database (LSDB) for SPB

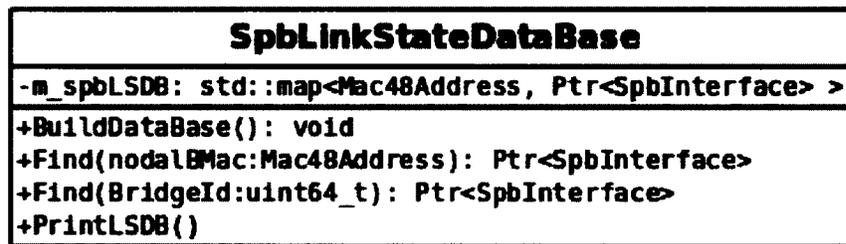


Figure 13. SpbLinkStateDataBase Class Diagram

The SpbLinkStateDataBase class provides all link state information of every SPB nodes in a simulation. Figure 13 presents the class diagram of SpbLinkStateDataBase. Pairs of a nodal MAC address and a reference of SpbInterface object are stored in m_spbLSDB structure. Each member function is described below.

- **BuildDataBase(void)** : iterates all SPB nodes in a simulation and collects the nodal MAC addresses and references to SpbInterface objects. Collected (MAC:*SpbInterface) pairs are stored into m_spbLSDB structure.
- **Find (Mac48Address nodalBMac)** : returns the reference to the SpbInterface object aggregated on the node with the nodalBMac.
- **Find (uint64_t bridgeId)**: As described in the preceding chapter, the bridge identifier is the numerical form of the nodal MAC address. This member function accepts 64 bits bridgeId as an input and converts it to the MAC address. Finally, it calls the Find(Mac48Address nodalBMac) function above with the converted MAC address.
- **PrintLSDB(void)** : prints out rows of (MAC address : BridgeIdentifier) pairs. The BridgeIdentifier is returned from SpbInterface::GetBridgeIdentifier() calls. Thus printed rows may be used to verify the integrity of the database.

4.6 Routing Algorithm Based on LSDB

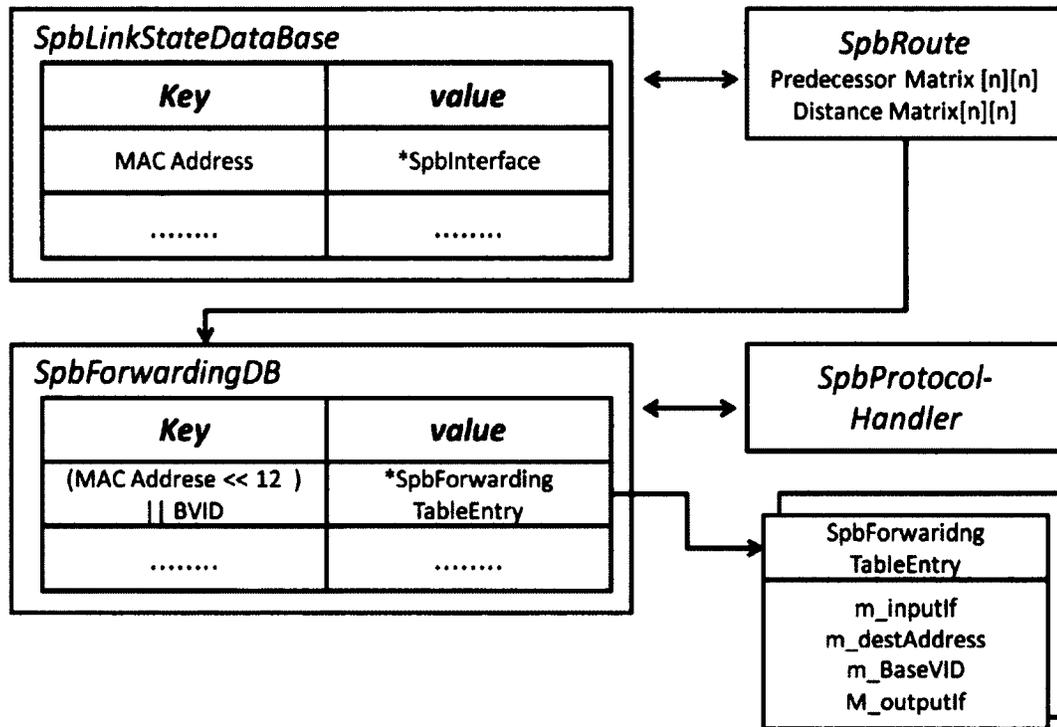


Figure 14. SPB Routing Architecture

Figure 14 demonstrates the routing architecture of the SPB simulator. Below is the brief description of each class.

- **SpbRoute**: computes the path and populates filtering database of all nodes.
- **SpbForwardingDB**: corresponds to a filtering database of a SPB bridge. It has the hash table of which key value is a concatenation of MAC address and BVID. Once a frame arrives at a node, SpbProtocolHandler extracts destination MAC address and BaseVID from the frame. Next, SpbProtocolHandler combines them to make the key. After that,

SpbProtocolHandler queries SpbForwardingDB with the key and get the reference of SpbForwardingTableEntry.

- **SpbForwardingTableEntry**: corresponds to each entry of a filtering database. An entry of a filtering database consists of input port, destination MAC address, Base VID, and output ports.
- **SpbProtocolHandler**: queries the SpbForwardingDB to get the output port when a frame is received from SpbApp or SpbNetDevice.

4.7 Algorithms for Computing Routes and Populating Forwarding Database

In a meshed Ethernet network such as a data center, there may exist multiple equal cost paths. Managing equal cost paths is the key to maximizing network capacity. In our case, we modified Floyd-Warshall all-pairs shortest paths algorithm [18] to manage equal cost paths. Time complexity of Floyd-Warshall algorithm is $O(N^3)$, where N is the number of nodes. In a meshed network, where m and n are number of edges and nodes respectively, the number of edges is larger than the number of nodes. A single-source shortest path algorithm considers the number of edges (e.g. Bellman-Ford and Dijkstra's algorithms). Therefore, the complexity of a single-source algorithm does not scale in a meshed network. For example, if we run Bellman-Ford algorithm to compute routes among all nodes, its complexity equals to $N * O(NM) = O(N^2M)$. All-pairs shortest path algorithm offers better performance in a meshed network.

To compute paths, we use two n by n matrices. The matrix $\text{distance}[n][n]$ contains the distances between nodes in the simulation. For example, if $\text{distance}[i][j]$

is 10, the weight of the path from i to j is equal to 10. Another matrix predecessor[n][n] contains the predecessor to node j on a shortest path from i to j. In other words, predecessor is the intermediate node. A predecessor matrix value is a 64-bit Bridge Identifier which is the concatenation of the bridge priority and the bridge system id. A bridge system id is numerical form of a MAC address of a node. The algorithm is provided below:

```

1: Initialize distance[n][n]
2: Initialize predecessor[n][n]
3: for k = 0 to n do
4:   for i = 0 to n do
5:     for j = 0 to n do
6:       if distance[i][j] > distance[i][k] + distance[k][j] then
7:         distance[i][j] = distance[i][k] + distance[k][j]
8:       else if distance[i][j] = distance[i][k] + distance[k][j] then
9:         Path1 = BuildPath from i to j
10:        Path2 = BuildPath from i to j passing through k
11:        if Path2 has higher priority then
12:          predecessor[i][j] = predecessor[k][j]
13:        end if
14:      else
15:        Do nothing
16:      end if
17:    end for
18:  end for
19: end for

```

Algorithm 1. Modified Floyd-Warshall All-pairs Shortest Path

Lines number 1 and 2 initialize the matrices with the value obtained from the procedure shown at Figure 9. Rows and Columns of two matrices represent source nodes and destination nodes respectively. After the procedure demonstrated in Figure 9, each node can identify its neighboring nodes and corresponding link costs to reach them. Bridge identifiers of neighboring nodes and the link cost are saved in SpbHelloExtension class and SpbLinkMetricList class respectively. Therefore, this information can be accessed through SpbInterface class. SpbLinkStateDataBase is the hash table of which stores pairs of nodal MAC address and the reference to SpbInterface object installed in the node with corresponding node MAC address. Thus, the reference to the SpbInterface object can be retrieved from the SpbLinkStateDataBase. Algorithm 2 initializes the two matrices.

```

//spbLSDB is the object of the SpbLinkStateDataBase class
rows, cols = spbLSDB.size()
create distance[rows][cols]
create predecessor[rows][cols]
//retrieves the first reference of SpbInterface object
for source = spbLSDB.begin() to source != spbLSDB.end() do
    i = (0xffff & source.GetBridgeIdentifier ()) - 1
    distance[i][i] = 0 //distance to the node itself
    adjI = source.GetAdjacencyVector();
    for j = 0 to adjI.size() - 1 do
        destination = adjI[j] //Bridge ID of neighboring node
        distance[i][j] = source.GetLinkCostTo (destination)
        predecessor[i][j] = (0xffff & destination) - 1
    end for
end for

```

Algorithm 2. Initialize Matrices

Lines number 8 to 13 in Algorithm 1 are added to modify Floyd-Warshall's all-pairs source algorithm in order to apply Equal Cost Tie-breaking mechanism of SPB because the original Floyd-Warshall's algorithm is not able to rank the equal cost paths.

Lines number 9 and 10 both create path from i to j. A Path is a list of 64-bit Bridge Identifiers. The difference between lines 9 and 10 is that line 10 creates the path passing through k. Line number 10 joins two paths, one is from i to k and another one is from k to j. If path i to k is on a shortest path and path k to j is, then path, i to j, is also on a shortest path. Computing a shortest path has optimal substructure. Thus, we guarantee that path i to j passing through k also on a shortest path [19]. Line 11 compares two paths using ECT-Algorithm. There are 16 different ECT-Algorithm. The least significant 1 byte of an ECT-Algorithm is used to XOR on each path.

The set of 16 different masking values can create the 16 different sets of hexadecimal numbers of which each hex decimal number has the unique priority compared to other numbers. These 16 different hexadecimal number sets are presented in appendix A. A path is a set of bridge identifiers and each bridge identifier in the set consists of octets. When the prioritized hexadecimal numbers are applied on a path, the path can be ranked among the other equal cost paths. This ranking mechanism allows the path selection to be managed without additional information [6].

Shortest Path Bridging (SPB) has the tie-breaking mechanism to prioritize the equal cost path. Each node advertises the costs of the attached links. These costs are

presented in SPB Link Metric sub-TLV. The sum of the link costs on the path is equal to the cost of the path. If equal cost paths exist between two end points, the path with smaller hop counts has the priority. If there are more than two paths with the same link cost and hop counts, the default tie-breaking mechanism picks up the path traversing the intermediate node with the lower Bridge Identifier [17]. Mesh network such as a data center may have multiple paths with the same link cost and the hop counts. This SPB tie-breaking mechanism guarantees diversity.

Line 11 compares two equal cost paths. A path with lower hop counts has the higher priority. Since a path is a list of bridge identifiers, the number of elements in the list is equal to the hop counts including the source and the destination. Thus, the smaller size of the list, the path, has the higher priority. If two paths have the same hop counts then, ECT-Algorithm value is XORed with the paths. The path with the smaller result has the higher priority. For example, let's assume that path1 has sequence of bridge identifiers 0,1,4 and 5, path2 has sequence of 0, 2, 4 and 5. If path1 and path2 are XORed with 0xFF, path2 produces smaller number. In that case, path2 has higher priority hence it would be selected.

Congruency between unicast and multicast, and symmetry between backward and forward paths makes populating a filtering database simple. We used the algorithm below to populate the node on a path. This algorithm takes a path, a SPB service identifier, and I-SID, as input:

```
1: n = total number of nodes in path
2: for i = 0 to n - 1 do
3:   for j = i + 1 to n do
4:     ForwardPath = Build Path from node[i] to [j]
5:     Install Unicast Entry for ForwardPath
6:     Install Multicast Entry for ForwardPath
7:     BackwardPath = Build Path from node[j] to [i]
8:     Install Unicast Entry for BackwardPath
9:     Install Multicast Entry for BackwardPath
10:   end for
11: end for
```

Algorithm 3. Populating SPB Nodes

CHAPTER 5: SHORTEST PATH BRIDGING DATA PLANE

One of the important characteristics of SPB data plane is congruency. First, forward and backward traffic must follow the same route. Second, unicast and multicast traffic between the nodes must also follow the same route. Third, a route from a source node to a destination node is deterministic. Symmetry between forward/backward paths and congruency between multicast/unicast paths enable SPB runs with legacy Ethernet Operation, Administration, and Maintenance (OAM) protocol.

Supporting multiple equal cost paths for VLAN is another important characteristic of SPB data plane. SPB supports up to 16 different equal cost paths. A path between two nodes is represented as a sequence of 8 byte Bridge Identifiers. 2 bytes of Bridge Identifier is Bridge Priority advertized in SPB instance sub-TLV. Thus, each path has a unique identifier and this identifier is XORed with 1 byte ECT-Algorithm value [6]. Table 2 shows the value.

By XORing on path identifier, SPB can order equal cost paths. Routing in the SPB network is deterministic. A traffic source node selects the set of the intermediate nodes through which the traffic must pass. An intermediate node cannot change the route of the traffic. Since each ECT-Algorithm value represents only one end-to-end path, SPB can utilize up to 16 different equal cost paths.

Index	0 , 1	2	3	4	5	6	7	8
Value	00	FF	88	77	44	33	CC	BB
Index	9	10	11	12	13	14	15	16
Value	22	11	66	55	AA	99	DD	EE

Table 2. ECT-Algorithm Masking Value (Hex)

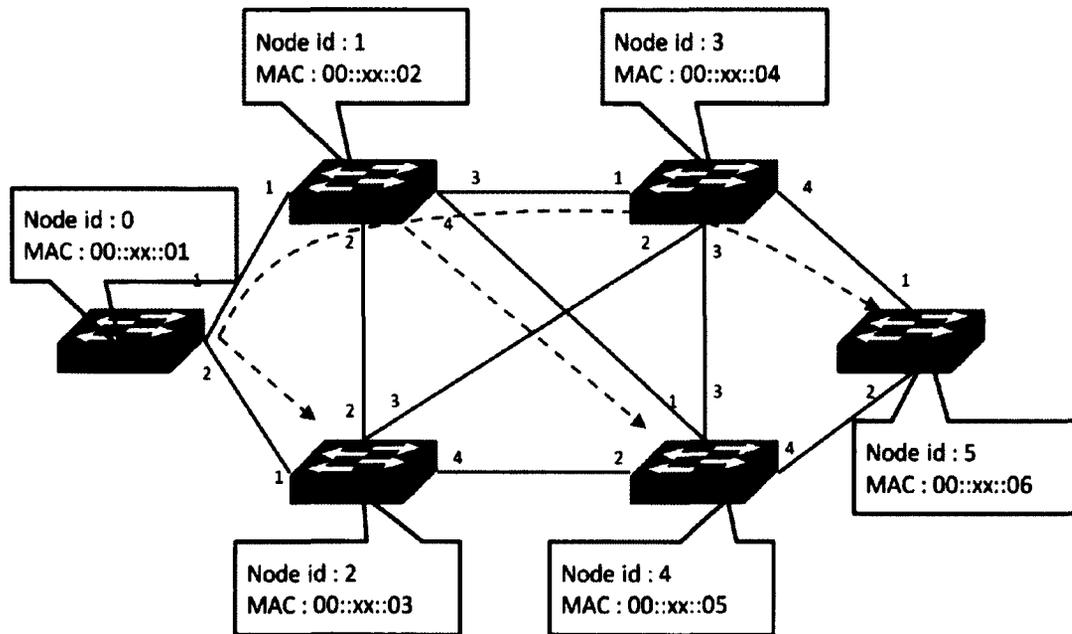


Figure 15. Shortest Path Tree Rooted at Node 0 when ECT-Algorithm 0 is used.

For example, Figure 15 demonstrates routes from node 0 to others. Every link in the above network has the same weight. Route decision is based on node id when there are multiple equal cost paths like from node 0 to node 5. A path crossing the smaller node id has higher priority. In the above topology, all traffic coming out from node 0 follows the paths:

- From node 0 to 1 : node 0 → node 1
- From node 0 to 2 : node 0 → node 2
- From node 0 to 3 : node 0 → node 1 → node 3
- From node 0 to 4 : node 0 → node 1 → node 4
- From node 0 to 5 : node 0 → node 1 → node 3 → node5

The minimal weighted path always has the highest priority. If there were two equal weight paths, the path with smaller hop counts would be selected. After that, if paths with the same hop count still exist, Equal Cost Tie-breaking Algorithm (ECT-

Algorithm) is used to select the path. For example, there are two possible shortest paths from node 0 to node 4. One is the path, node 0 → node 1 → node 4, and another one passing node 2, node 0 → node 2 → node 4. ECT-Algorithm 0 selects the former.

5.1 SPBM Unicast Data Path

SPBM unicast frames consist of a Backbone Source Address (B-SA), a Backbone Destination Address (B-DA), a Backbone VLAN ID (B-VID), and an I-Component Service Instance ID (I-SID). B-SA is backbone MAC address of an ingress SPB bridge while B-DA is backbone MAC address of an egress SPB bridge. I-SIDs are assigned at User Network Interface (UNI) ports and associated with B-VID. When a customer frame reaches an ingress SPB bridge, the frame is encapsulated with B-SA, B-DA, B-VID and I-SID. Therefore, routing decisions in SPB backbone networks are completely independent from customer networks.

An encapsulated frame reaches tandem nodes, which are not border bridges, and its {B-DA, B-VID} field is inspected to direct the frame to right output ports. A Forwarding Database (FDB) stores unicast entries mapping a B-DA and a B-VID. If there is not a unicast entry matched with a pair of B-DA and B-VID, the frame is discarded. Then SPB simulator stops the simulation to notify that an incorrect configuration was discovered in a simulation script.

5.2 SPBM Multicast Data Path

SPBM supports two different types of multicast Head-end Replication and Tandem Replication. Head-end Replication is a series of unicast traffic flows to different

the intermediate nodes install the B-DA in their FDB. These nodes form the spanning tree of the multicast.

5.3 SPBM Protocol Handler

The main role of SPBM (SPB Mac-in-Mac) protocol handler is the routing decision when a SPB node sends and receives a SPBM frame. If a SPB application attempts to send traffic to other nodes, it must consult with SPBM protocol handler first. The application passes a packet with I-SID and the type of traffic (which is either multicast or unicast). Then, the SPBM protocol handler queries the Forwarding Database (FDB) with the I-SID in order to retrieve the right output ports. In NS-3, output and input ports are modeled as the NetDevice class. A Node class saves the index number referencing each NetDevice objects in it. Therefore, when FDB returns the list of index numbers referencing NetDevice objects, the SPBM protocol handler accesses the SpbNetDevice class objects inherited from the NetDevice class. Once the SPBM protocol handler accesses the SpbNetDevice objects, it attaches SPB frame header on the packet and passes it to the SpbNetDevice objects.

Routing decisions for received SPB frames also depends on the SPBM protocol handler. The first object of the received SPB frame is a SpbNetDevice object which is the model of input port in real world networks. As the Send function selects the proper SpbNetDevice to send the packet out, Receive function is given the references to proper SpbNetDevice objects. If the given output port index is equal to 0, this means local delivery, SpbProtocolHandler passes the frame to the SpbApplication object in order to free the memory held by the frame.

A Node class in NS-3 registers callback functions for the protocols it can handle. In case of SPB protocol, the callback function is SpbProtocolHandler::Receive. This callback function is registered at a node when SPB protocol stack is installed in the node. Thus when SpbNetDevice receives a SPB frame, Receive function of SpbProtocolHandler class is called. The receive function also queries FDB like Send function. Figure 17 demonstrates how a SPB frame is received and sent.

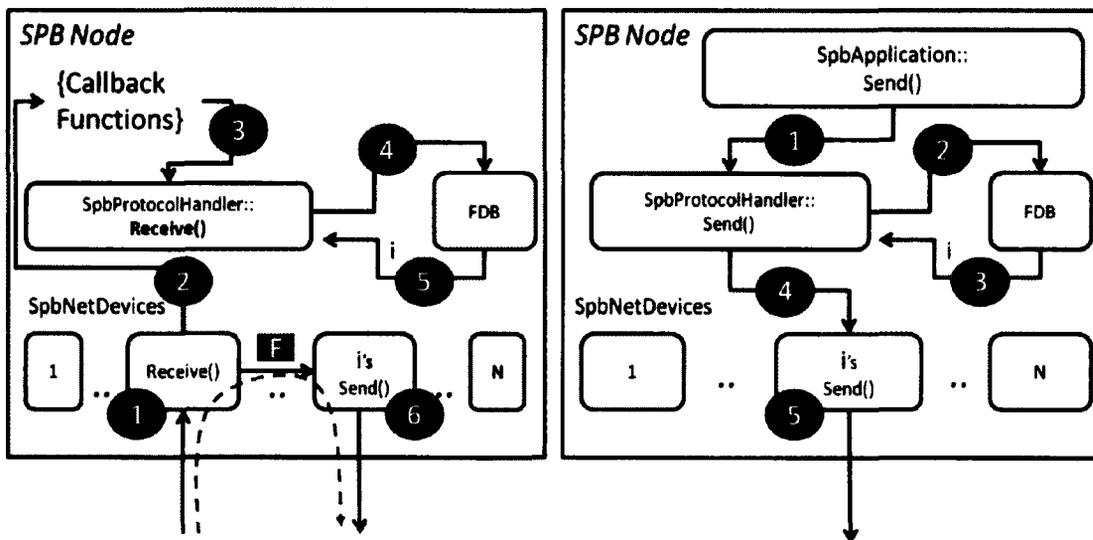


Figure 17. Receive and Send

CHAPTER 6: SIMULATION SETUP AND RESULTS

6.1 Simulator Helper class

A helper class is designed to simplify the simulation script for the SPB simulator.

Figure 18 shows public member functions of SPB helper class.

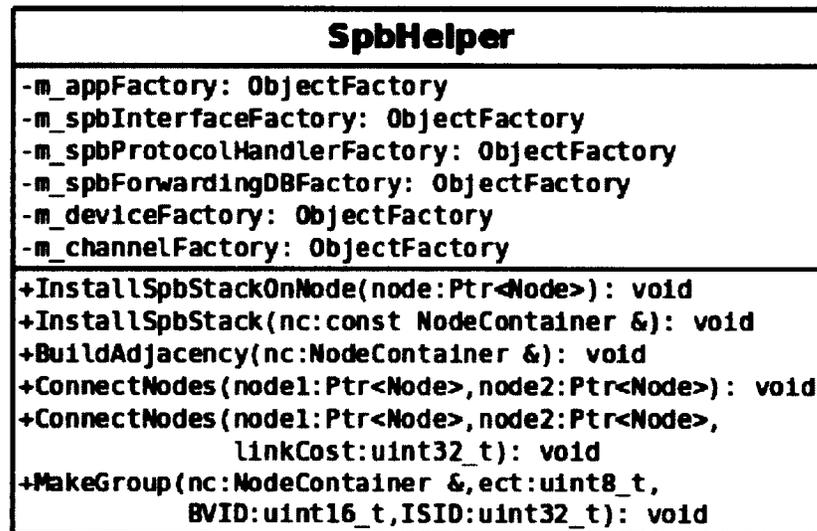


Figure 18. SpbHelper Class Diagram

NS-3 supports the factory design pattern. SpbHelper class uses the design pattern to simplify creation of SPB objects. SpbHelper member functions install the SPB devices and the SPB protocol stack on a node object and connect them to form a SPB network. Each member function is described below.

- **InstallSpbStack** : Installs SPB protocol stack. SPB protocol stack consists of 5 objects. These are SpbApp, SpbInterface, SpbForwardingDB, SpbNetDevice and SpbProtocolHandler.
- **BuildAdjacency** : Calls SpbInterface::BuildAdjacency(). A node detects its neighbors by this method.
- **ConnectNodes** : Sets up a link between two adjacent nodes.

- **MakeGroup** : Groups nodes to communicate each others. Parameter ISID consists of three parts. The most significant bit is T bits. The second most significant bit is R bits. The other bits are ISID. The objective of T, R bits is described in the standard [6].

6.2 Simulation outputs

NS-3 includes a logging system to print out messages from a simulation. The logging system supports multiple log levels. The highest level of logging may print out all messages defined in a simulation. There are 7 different levels. The logging systems can be turned on and off in respect to specific classes. So a user may print out messages only for some classes. Typical usage looks like the following:

```
NS_LOG (LOG_DEBUG, "usage " << m_variable << " ")
```

LOG_DEBUG is one of the levels. The below function example enables logging in the SpbApp class with the highest level:

```
LogComponentEnable ("SpbApp", LOG_LEVEL_ALL);
```

A simulation result can be animated, however, only point-to-point channel type can be animated. NS-3 generates XML file output for animation. The XML file is loaded using NetAnim⁷ program bundled with NS-3 packages.

Though the SPB simulator has its own channel model, it can produce XML output for animation. Modifying one line of codes in NetAnim interface classes make SPB animation possible. SpbChannel class calls point-to-point channel class's trace source defined in NetAnim interface. Therefore every time a frame is received or sent

⁷ NetAnim is the application for animation in a NS-3 package [25].

by SpbChannel, point-to-point channel's animation trace source is called. Trace source is a function triggered by some events. The point-to-point channel's trace source writes each event to the XML file for animation.

Example lines of XML file for animation are provided below.

The line below defines X and Y planes of a topology.

```
<topology minX = "-37.1" minY = "-37.1" maxX = "1015.9" maxY = "1015.9">
```

The following line demonstrates a node with id 0 whose location in a canvas is (100, 500) and RGB color code (255, 0, 0)

```
<node id = "0" locX="100" locY="500" r="255" g="0" b="0"/>
```

Packet's arrival and departure is described below.

- fromId : departure node
- toId : arrival node
- fbTx, lbTx : the first and last bit transmitted time
- fbRx, lbRx : the first and last bit received time.

```
<packet fromId="0" fbTx="1" lbTx="1.1"><rx toId="2" fbRx="2.05" lbRx="2.06"/>
```

6.3 Setting Up Simulation

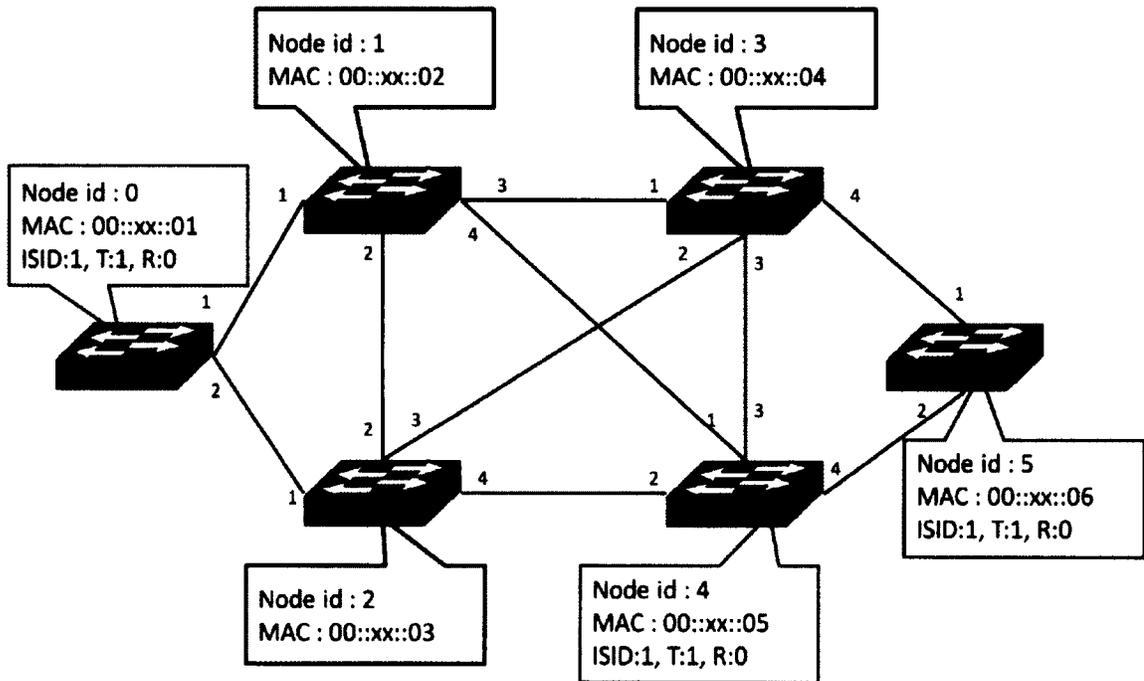


Figure 19. Experimental Topology

SPB simulator has been implemented on NS-3.15 running on Ubuntu Linux 12.04 system. Figure 19 is the experimental network topology. Each node has Id and MAC address. Id is a reference number to identify each node in the simulation. In this experiment, node 0, node 4 and node 5 are grouped with the same ISID 1. T bit and R bit are set to 1 and 0 respectively. Every link in the simulation has the same weight 1. Italic numbers near the links are port numbers.

```

41  /* create nodes */
42  NodeContainer nodes;
43  nodes.Create (6);
44
45  SpbHelper spbHelper;
46
47  spbHelper.InstallSpbStack (nodes);
48
49  #if 1 // Congruency Test
50  MobilityHelper mobility;
51  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
52  mobility.Install (nodes);
53
54  Ptr<MobilityModel> mm = nodes.Get(0)->GetObject<MobilityModel> ();
55  mm->SetPosition( Vector (100, 500, 0));
56
57  mm = nodes.Get (1)->GetObject<MobilityModel> ();
58  mm->SetPosition (Vector (300, 300, 0));
59
60  mm = nodes.Get (2)->GetObject<MobilityModel> ();
61  mm->SetPosition (Vector (300, 700, 0));
62
63  mm = nodes.Get (3)->GetObject<MobilityModel> ();
64  mm->SetPosition (Vector (500, 300, 0));
65
66  mm = nodes.Get (4)->GetObject<MobilityModel> ();
67  mm->SetPosition (Vector (500, 700, 0));
68
69  mm = nodes.Get (5)->GetObject<MobilityModel> ();
70  mm->SetPosition (Vector (700, 500, 0));
71
72  spbHelper.ConnectNodes (nodes.Get (0), nodes.Get (1));
73  spbHelper.ConnectNodes (nodes.Get (0), nodes.Get (2));
74  spbHelper.ConnectNodes (nodes.Get (1), nodes.Get (2));
75  spbHelper.ConnectNodes (nodes.Get (1), nodes.Get (4), 1);
76  spbHelper.ConnectNodes (nodes.Get (1), nodes.Get (3));
77  spbHelper.ConnectNodes (nodes.Get (2), nodes.Get (4));
78  spbHelper.ConnectNodes (nodes.Get (2), nodes.Get (3));
79  spbHelper.ConnectNodes (nodes.Get (4), nodes.Get (3));
80  spbHelper.ConnectNodes (nodes.Get (4), nodes.Get (5));

```

Figure 20. Topology in Script

Figure 20 is the part of the simulation script building the experimental topology of Figure 19.

1. Line 43 creates 6 nodes.
2. Line 47 installs the SPB protocol stack on all nodes

3. From line 50 to 70, all nodes aggregate constant position mobility model. This mobility model fixes the position of a node. SetPosition function takes X and Y coordination of each node.
4. From line 72 to 80, links between nodes are created.

```

117 spbHelper.BuildAdjacency (nodes);
118
119 // build Link State Data Base
120 SpbLinkStateDataBase* spbLSD = new SpbLinkStateDataBase ();
121 spbLSD->BuildDataBase();
122
123 // Connect LSD and Routing Protocol
124 SpbRouting* spbDataPlane = new SpbRouting ();
125 spbDataPlane->SetLSD (spbLSD);
126
127 uint8_t ect = 2;
128 uint32_t isid = 1;
129 uint8_t T = 1;
130 uint8_t R = 0;
131 uint32_t ISID = (T << 31) | (R << 30) | isid;
132 uint16_t bVid = 103;
133
134 NodeContainer group;
135 group.Add (nodes.Get (0));
136 group.Add (nodes.Get (4));
137 group.Add (nodes.Get (5));
138
139 spbHelper.MakeGroup (group, ect, bVid, ISID);
140
141 // Floyd-Warshall with ISID
142 spbDataPlane->CalculateRoute (ISID);
143
144 spbDataPlane->PrintPB ();
145
146 Ptr<SnbApp> srcApp = DynamicCast<SnbApp> (nodes.Get (0)->GetApplication (0));
147 srcApp->SetStartTime (Seconds (1.0));
148 srcApp->SetStopTime (Seconds (2.0));
149 srcApp->EnableSending (true);
150
151 #if 1
152 srcApp->SetTrafficType (true); // True : multicast, False : Unicast
153 srcApp->SetISID (isid);

```

Figure 21. Grouping in Script

Figure 21 shows the part of script for grouping nodes 0,4, and 5 with the same ISID. Each node in simulation has nodal B-MAC address of which node id plus one. Say, a node with id 0 has B-MAC address 00:00:00:00:00:01. Each node aggregated SPB stack already.

5. Line 117 calls SPB helper API BuildAdjacency(nodes). This call fills up the control plane data of each node.
6. Line 120, 121 build Link State Database (LSDB).
7. Line 124, 125 build SpbRouting object and connect the object to the LSDB. SpbRouting object computes the route with the LSDB.
8. Line 127 to 132 set up ECT-Algorithm index, I-SID and BVID.
9. Node id 0, 4 and 5 are grouped together from line 134 to 137.
10. Line 139 assigns the ECT-Algorithm index, I-SID and BVID to the group of node 0, 4 and 5.
11. Line 142 computes the route for the group.
12. Line 144 prints out Forwarding Database (FDB) in all nodes.

From line 146 to 153, SpbApplication in node id 0 sends multicast traffic to its group member sharing the same I-SID from time 1 second to 2 second.

6.4 Simulation Results

Our experiment is designed to validate congruency of Shortest Path Bridging (SPB) routing and capacity to select the highest priority path among the equal cost paths. A simulation generates an XML output file. The XML output file is read through the NetAnim program that produces a graphical result. Since NS-3 only supports graphical result for point to point (P2P) type channel, we use the same interface for P2P. Unicast and Multicast Paths between group members must be identical. From Figure 19, nodes 0, 4 and 5 are in the same group. This group uses Equal Cost Algorithm (ECT). Each Equal Cost Algorithm has a 1byte masking value. The

masking value is XORed on a path identifier which is the System IDs. When ECT index is 1, then the corresponding mask value is also 0, priorities between nodes follow the order of node Id: $0 > 1 > 2 > 3 > 4 > 5$. Because we set up the same bridge priorities to all nodes and 4 byte System ID of a node is numerical form of MAC address. Thus, the lower the system ID value, the higher the node priority. Under these constraints, unicast path between nodes 0 to 4 follows nodes 0, 1, 4 in that order. Another unicast path between 0 and 5 is 0, 1, 3, 5 in order. When multicast is congruent with unicast path, traffic from node 0 must follow node 1 and propagate through port number 3 and 4. Graphical result is shown in Figure 24.

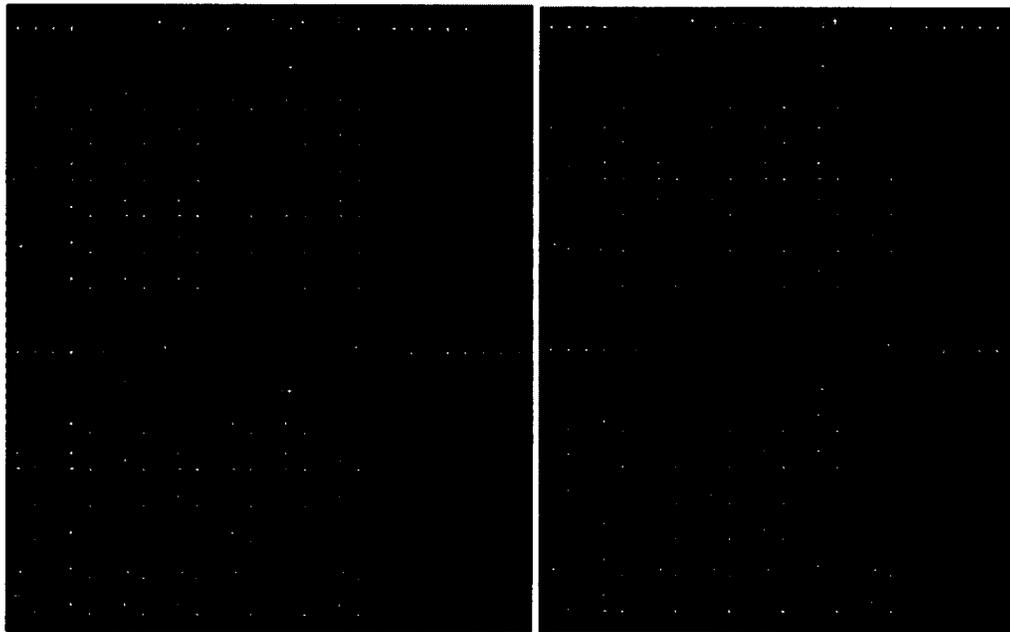


Figure 22. Initial and Final Matrix When ECT is 1

Before running all-pairs source shortest algorithm, distance matrix and predecessor matrix are shown in Figure 22 left-hand side. Each row indicates starting node while each column indicates ending node. For example, distance from node 5 to 0 is X because the shortest path algorithm has not yet computed the distance from

node 5 to 0. After running the algorithm the result is shown at right-hand side of Figure 22. Distance from node 5 to 0 is 3 and the predecessor node from node 5 to 0 is node 4. After matrix computation FDBs on all nodes are installed.

Figure 23 shows filtering database (FDB) of all nodes in the simulation. The capital U and M means unicast and multicast respectively. In and Out columns present port numbers. 'in' column shows input ports and 'out' column shows output ports. Port number 0 indicates the node itself. A Unicast entry of FDB does not consider input ports. Node 0, 4 and 5 has three multicast entries. Each multicast entry holds destination B-MAC address of which consists 20-bit SPSourceID and I-SID assigned to them. SPSourceID comes from B-MAC address of node 0, 4 and 5.

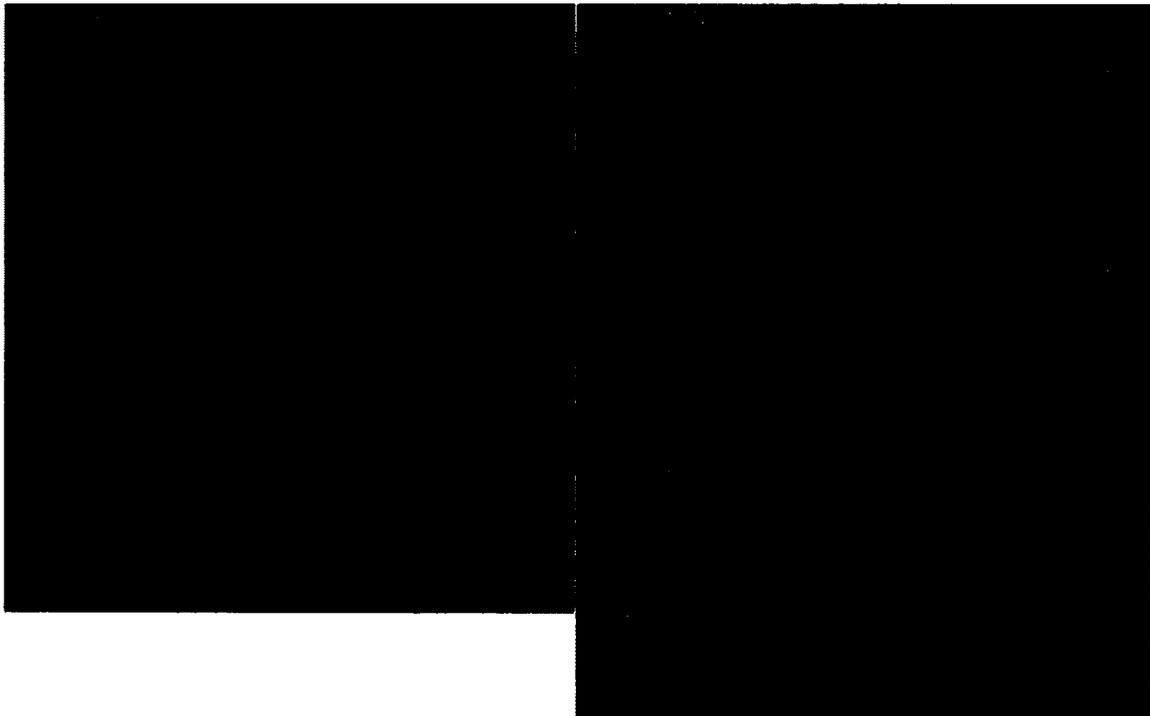


Figure 23. FDBs when ECT is 1

When ECT index is 2, its corresponding mask value is 0xFF [6]. The masking value 0xFF toggles bits in System IDs therefore, priorities between nodes follow the order Node Id: 5 > 4 > 3 > 2 > 1 > 0. Under these constraints, unicast path between nodes 0 to 4 follows nodes 0, 2, 4 in order. Another unicast path between 0 and 5 is 0, 2, 4, 5 in order. When multicast is congruent with unicast path, traffic from node 0 must follow node 2 and 4. Result is shown at Figure 25.

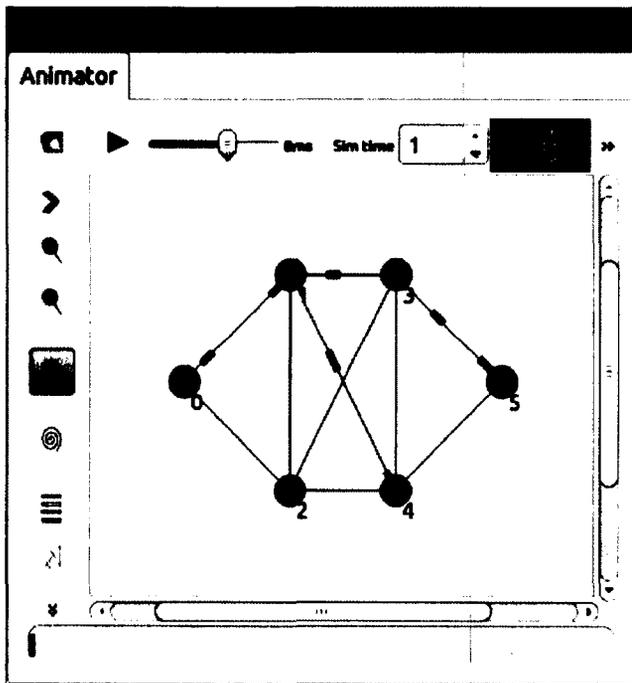


Figure 24. Multicast When ECT 1

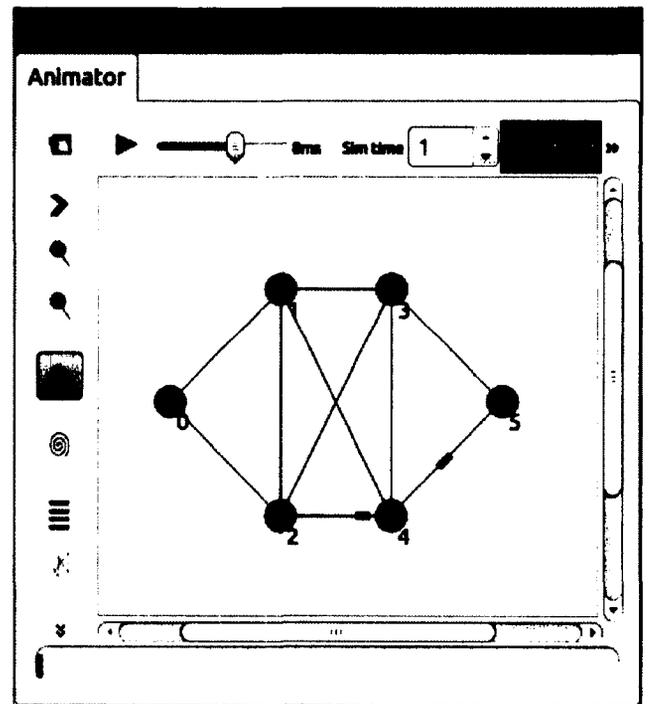


Figure 25. Multicast When ECT 2

CHAPTER 7: CONCLUSION

The cloud data center network requires certain properties. The first property is virtualization. The infrastructure of the cloud data center is shared with multiple customers and each customer requires different levels of services. Without network virtualization, it is hard to offer different classes of services. Individual customer traffic has to be identified in the cloud data center for delivering customized services. The second property is acceptable convergence time. The cloud data center network has to quickly respond to network events. A cloud data center runs almost all type of application. Some of the application may need real time stream of data. In order to meet real time constraints, the convergence time after network changes should be minimized. The third property is higher utilization. Network traffic in a cloud data center should take diverse paths to achieve higher utilization. Fourth property is a minimum configuration. Traffic in a cloud data center changes dynamically. Human intervention to the dynamicity may not be practical. Finally, cost of deploying network should be economical.

Ethernet meets all the requirements described above. It was developed to minimize a network configuration. Its plug-and-play feature can migrate hosts to different locations without configuration. Virtual machine instance in cloud data center may require frequent migrations. The plug-and-play mechanism would reduce the efforts following virtual machine migrations.

Ethernet has evolved to support virtualization and IEEE has developed standard for virtual LAN 802.1Q. Though Ethernet offered simple configuration and virtualization, it was still not suitable for the cloud data center. First, its convergence

time was unacceptable. Second, its link utilization was not higher enough. Spanning Tree Protocol (STP) has been widely used for Ethernet networks but, it still has the two problems. Shortest Path Bridging (SPB) has been developed to overcome those two shortcomings.

Convergence time after network changes depends on how fast information delivered to nodes participating in the same network. STP broadcasts information to build up new topology when network changes occur. Broadcasting takes certain amount of time which is not acceptable for real time applications. While STP uses broadcasting, SPB uses the link state database holding information. Nodes in a network retrieve information from link state databases immediately after network change thus SPB offers acceptable convergence time for real time applications.

Suboptimal link utilization is another problem for applying STP to the cloud data center. In contrast to STP, SPB utilizes links higher in a data center. First, SPB does not require disconnecting one of the links in order to prevent loop formation. Second, traffic in SPB network always takes the optimal path in terms of link cost.

While customer networks are isolated from provider networks, network providers apply different services or policies on customer traffic. This situation occurs at the cloud data center also. IEEE 802.1aq encapsulates customer traffic and it assigns different service identifiers on the customer traffic. This gives network providers the ability to enable different levels of a service agreement for each customer.

SPB employs all the features that the cloud data center network requires.

As SPB emerges as the one of the network protocol for the cloud data center network, we need a simulator for SPB network. Since NS-3 has been widely adapted to research community, we designed and implemented the SPB simulator on NS-3.

This thesis demonstrates the design and implementation of the SPB simulator. The modified version of Floyd-Warshall all-pairs shortest paths algorithm is used to compute routes in a simulation. This algorithm is designed to utilize a meshed network efficiently, which is particularly relevant to data center networks of which may have multiple equal cost paths.

The SPB simulator we proposed here is still in the early stage. Simulating User Network Interface (UNI) ports would give more realistic results. In real world networks, SPB is used for access or backbone networks. Traffic from a customer network reaches to a UNI port at an ingress SPB bridge. The I-Component then maps a customer VLAN ID to an SPB I-SID. The Ethernet network model in the latest NS-3, version 3.15, does not support VLAN which means that the customer Ethernet frame does not have VLAN ID. Due to this limitation, the node model in the SPB simulator adopts an application layer in order to generate the traffic with I-SID. Therefore, it is hard to simulate the interaction between customer and SPB networks.

A SPB network can be operated with SPBM and SPBV bridges together. However, SPBV (SPB VID) mode is not supported in the SPB simulator. The SPBV data plane is different to the SPBM data plane [17]. Simulating the SPBV mode would extend the capability of the simulator.

We strongly believe that the SPB simulator would help the research community of SPB. Future research areas where the SPB simulator can be used are described in [20].

APPENDIX A: 16 PRIORITY SETS OF HEXADECIMAL NUMBERS

TABLE A.1 BRIDGE PRIORITY MASKING [6]

ECI- ALGORITHM	Algorithm MASK	Effect of using MASK on priority nibble
00-80-C2-01	0x00	(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
00-80-C2-02	0xFF	(15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0)
00-80-C2-03	0x88	(8,9,10,11,12,13,14,15,0,1,2,3,4,5,6,7)
00-80-C2-04	0x77	(7,6,5,4,3,2,1,0,15,14,13,12,11,10,9,8)
00-80-C2-05	0x44	(4,5,6,7,0,1,2,3,12,13,14,15,8,9,10,11)
00-80-C2-06	0x33	(3,2,1,0,7,6,5,4,11,10,9,8,15,14,13,12)
00-80-C2-07	0xCC	(12,13,14,15,8,9,10,11,4,5,6,7,0,1,2,3)
00-80-C2-08	0xBB	(11,10,9,8,15,14,13,12,3,2,1,0,7,6,5,4)
00-80-C2-09	0x22	(2,3,0,1,6,7,4,5,10,11,8,9,14,15,12,13)
00-80-C2-0A	0x11	(1,0,3,2,5,4,7,6,9,8,11,10,13,12,15,14)
00-80-C2-0B	0x66	(6,7,4,5,2,3,0,1,14,15,12,13,10,11,8,9)
00-80-C2-0C	0x55	(5,4,7,6,1,0,3,2,13,12,15,14,9,8,11,10)
00-80-C2-0D	0xAA	(10,11,8,9,14,15,12,13,2,3,0,1,6,7,4,5)
00-80-C2-0E	0x99	(9,8,11,10,13,12,15,14,1,0,3,2,5,4,7,6)
00-80-C2-0F	0xDD	(13,12,15,14,9,8,11,10,5,4,7,6,1,0,3,2)
00-80-C2-10	0xEE	(14,15,12,13,10,11,8,9,6,7,4,5,2,3,0,1)

BIBLIOGRAPHY

- [1] D. Allan, and N. Bragg. "IEEE 802.1aq in a Nutshell: Antecedents and Technology, Exploiting Multiple Paths in SPB," in *802.1aq Shortest Path Bridging: Design and Evolution The Architect's Perspective*, New York, IEEE Press, 2012, p. 29.
- [2] D. Allan, and N. Bragg. "The Data Center and General Enterprise Application," in *802.1aq Shortest Path Bridging Design and Evolution*, IEEE Press, 2012, pp. 152-155.
- [3] P. Draft Standard. *802.1 Q/D10, IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks*, Copyright by the Institute of Electrical and Electronics Engineers, Inc, 1997.
- [4] IEEE Std 802.1ah. *IEEE Standard for Local and metropolitan area networks -- Virtual Bridged Local Area Networks Amendment 7: Provider Backbone Bridges*, June 2008.
- [5] IEEE Standard. *802.1D IEEE Standard for Local and Metropolitan Area Networks. Media Access Control (MAC) Bridges*, June 2004.
- [6] P. Draft Standard. *IEEE 802.1aq/D4.6 Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks-Amendment XX: Shortest Path Bridging*, February 2012.
- [7] D. Allan, and N. Bragg. "802.1aq Shortest Path Bridging Design and Evolution," IEEE Press, 2012, p. 3.
- [8] D. Allan, and N. Bragg. "Why the SPB Control Plane Looks as it does," in

- 802.1aq shortest Path Bridging: Design and Evolution The Architect's Perspective*, New York, IEEE Press, 2012, pp. 110-115.
- [9] ISO/IEC, *Information technology-Telecommunications and information exchange between systems - Intermediate System to Intermediate System intra-domain routing information exchange protocol(ISO 8473)*, ISO/IEC, 2002.
- [10] P. Ashwood-Smith. "Shortest Path Bridging IEEE 802.1aq Tutorial and Demo," in *NANOG 50*, 2010.
- [11] D. Allan, and N. Bragg. "Why SPB Looks as it does, History," in *802.1aq shortest Path Bridging: Design and Evolution The Architect's Perspective*, New York, IEEE Press, 2012, pp. 52-60.
- [12] "ns-3," 2013. [Online]. Available: <http://www.nsnam.org>.
- [13] "Nsnam," 5 november 2011. [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/Main_Page.
- [14] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and JB. Kopena. "Network simulations with the ns-3 simulator," in *SIGCOMM demonstration*, 2008.
- [15] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley "NS-3 project goals," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, New York, NY: ACM, 2006.
- [16] H.Ekram, and I. Teerawat, *Introduction to network simulator NS2*, Springer, 2011.
- [17] D. Fedyk, P.Ashwood-Smith, D.Allan, N. Bragg, and P. Unbehagen. *IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging*, 2012.

- [18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction To Algorithms, Third Edition*, Cambridge: MIT press, 2009.
- [19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. "25 All Pairs Shortest Paths," in *Introduction to algorithms, Third Edition*, MIT Press, 2009, p. 686.
- [20] D. Allan, and N. Bragg. *802.1aq shortest Path Bridging: Design and Evolution The Architect's Perspective*, New York: IEEE Press, 2012.
- [21] J. Farkas, and Z. Arato "Performance Analysis of Shortest Path Bridging Control Protocols," in *IEEE GLOBECOM*, 2009.
- [22] D. Allan, J. Farkas, and S. Mansfield. "Intelligent Load Balancing for Shortest Path Bridging," *Communications Magazine, IEEE*, vol. 50, pp. 163-167, july 2012.
- [23] D. Allan, P. Ashwood-Smith, N. Bragg, and D. Fedyk. "Provider Link State Bridging," *Communications Magazine, IEEE*, vol. 46, pp. 110-117, september 2008.
- [24] D. Allan, P. Ashwood-Smith, N. Bragg, J. Farkas, D. Fedyk, M. Ouellete, M. Seaman, and P. Unbehagen. "Shortest Path Bridging: Efficient Control of Larger Ethernet Networks," *Communications Magazine, IEEE*, vol. 48, pp. 128-135, october 2010.
- [25] "NetAnim-Nsnam," 5 February 2013. [Online]. Available: www.nsnam.org/wiki/index.php/NetAnim. [Accessed 18 February 2013].
- [26] D. Allan, and N. Bragg. "Why the SPB control plane looks as it does, Convergence Behaviors," in *802.1aq shortest Path Bridging: Design and*

Evolution The Architect's Perspective, New York, IEEE Press, 2012, pp. 82-87.

[27] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and A. V. Vasilakos. "Survey on Routing in Data Centers:," *Network, IEEE*, vol. 25, pp. 6-10, july-august 2011.

[28] IEEE Std 802.1Q, *IEEE Standard for Local and Metropolitan area networks-Virtual Bridged Local Area Network*, December 2005.