The University of Kansas

Information and Telecommunication Technology Center

Technical Report

Open Control Architectures Final Report

Chandrasekar Ramachandran, Balasubramanian Ramachandran & Joseph Evans

ITTC-FY2001-TR-18835-01

January 2001

Project Sponsor:
Sprint Corporation

Open Control Architectures – Executive Summary

Value to Sprint

- The study and experiments of the protocols namely VSI and GSMP will give valuable clues on the choice of protocol for deployment based on requirements.
- Experiments and results on VSI can provide insight in to the delays and latencies that will be experienced due to the protocol and hardware.

Lessons Learned

Applicability of Open Control

- Switches or optical cross-connects using open control can be useful for service providers by providing vendor and some media independence.
- Service outages and problems occur due to any upgrade in protocol software in the provider networks. Open control provides virtual partitioning and facilitates redundancy, and helps to minimize such outages.
- Protocols that involve some form of signaling can most benefit from open control.

VSI and GSMP

- Changing partition resources will be necessary due to changes in the network or the addition of new customers. VSI supports dynamic partitioning and makes network management easier.
- Due to the very high speeds in provider backbones, using hardware standby or configuring multiple paths is better than relying on the control protocol to provide controller redundancy.
- GSMP allows for interoperability of boxes of different vendors whereas VSI does not.
- VSI requires a high-end router as controller and is will more likely increase deployment cost. GSMP can be implemented on any platform and is more likely to reduce deployment cost.

MPLS CoS

- The multi-VC approach used by our implementation of MLPS CoS uses ATM services better than the approach that uses the MPLS EXP field, but this may not be better if the number of service classes is larger (e.g., multiple classes based on different drop precedences as in DiffServ AF PHB).
- Experiences with our MPLS CoS implementation as well as Cisco's VSI indicate that reservations for each class are not rigid but relative. This may not be desirable in situations where the requirements for delay and jitter are very strict.

Table of Contents

1 Introduction

The number of businesses moving towards Internet is rapidly increasing every day. This would inevitably lead to numerous applications that try to integrate them to the global network. Such applications not only demand huge capacity but also quality. Timely and correct delivery may become very important. The need of services arises due to the limitation of bandwidth and the provisioned bandwidth is likely to be used up very soon. Also premium services will generate higher revenues for the providers. The current model of Internet is not well equipped to handle increasing demand for traffic with service demands which makes QoS problem very difficult to solve. Even if the capacity of the links and router processing engines increases rapidly, the demand too is likely to keep pace with it. To efficiently support such applications, creation/deletion/management of services is required on-demand. The service may have to be created across various administrative domains. Service management is easier and more efficient when the control and the data planes are separated in the network infrastructure.

The control and management functionality of most of the current networks is tightly coupled to the network hardware of vendors. With Open Control, network devices and resources are controlled and managed by external devices through an open control interface. Open control reduces the association between the control plane and the switching or data plane allowing them to evolve independently. This makes the open control systems easily upgradable and deployable.

The concept of open control is more relevant to ATM switches than IP routers due to the connection oriented-ness of ATM. The control interface of a switch should allow the controller to influence the flow or connections through the switch. In IP routers, the notion of connection or flow is not enforced. The forwarding is based only on the destination address. But in ATM, the notion is required at the call setup before any data is transferred. But off-board control can be more appropriate for MPLS [1] routers or switches that involves some sort of signaling across an Autonomous System (AS) before data transfer. This is true when MPLS is run on almost all link layers like ATM, frame relay or IP.

Generic Switch Management Protocol (GSMP) and Virtual Switch Interface (VSI) are two primary switch control protocols. Although initially, more importance is given to ATM switches, the protocols are likely to support many types of switches.

1.1 Switch Control Interface

The deployment of open control requires a standard interface [2] for the control architecture to communicate with the switch or router and to manipulate the resources to perform its control and management functions. The control interface serves to abstract the ATM switch to allow the control architecture to fully utilize the physical resources of the switch using this abstract model. When the interface to the switch is opened up, anyone with the knowledge of the interface can implement control software.

1.1.1 Generic Switch Management Protocol (GSMP)

The General Switch Management Protocol (GSMP) was designed first by Ipsilon and has been available as open document in the form of IETF RFCs and drafts. The latest version GSMPv2.0 that was released on August 1998 as RFC2297 [3] and the previous versions were intended primarily for ATM switches. Currently IETF GSMP working group is working on GSMPv3 [4] that will run on different kinds of switches and is targeted to run with all kinds of MPLS routers too.

GSMP allows the controller to

- create and release connections across the switch
- add and delete leaves of a multicast connection
- reserve resources for different types of connections
- manage switch ports
- request switch configuration information and switch statistics

A physical switch can be partitioned in to virtual switches called partitions. Such partitions have to be created prior to running GSMP. In GSMPv3, a partitioned switch is controlled through the use

of partition identifier contained in every GSMP message. Hence each partition has a one to one relationship with a single logical controller entity. Every connection is created with a set of specified QoS parameters.

1.1.2 Cisco's Virtual Switch Interface (VSI)

Virtual Switch Interface (VSI) [5] is Cisco's proprietary protocol and is available with Cisco WAN switching software and IOS releases. The latest version VSI 2.2 requires WAN Switching Software release 9.2.20 for BPX as minimum switching software requirement. Currently, VSI functions only with ATM switches. A BPX switch partition can be controlled by MPLS or PNNI control plane. Each MPLS node consists of a Cisco 6400 or 7200 or 7500 router and a BPX ATM switch

The MPLS controller is also called Label Switch Controller (LSC). The latest release of VSI supports three MPLS partitions in an interface. VSI allows the PNNI controller to create VCs with desired ATM Forum QoS parameters. Control of partition for PNNI requires a dedicated box Service Expansion Shelf (SES).

1.2 Outline of Tasks

The Open Control project at KU is divided in to four major tasks

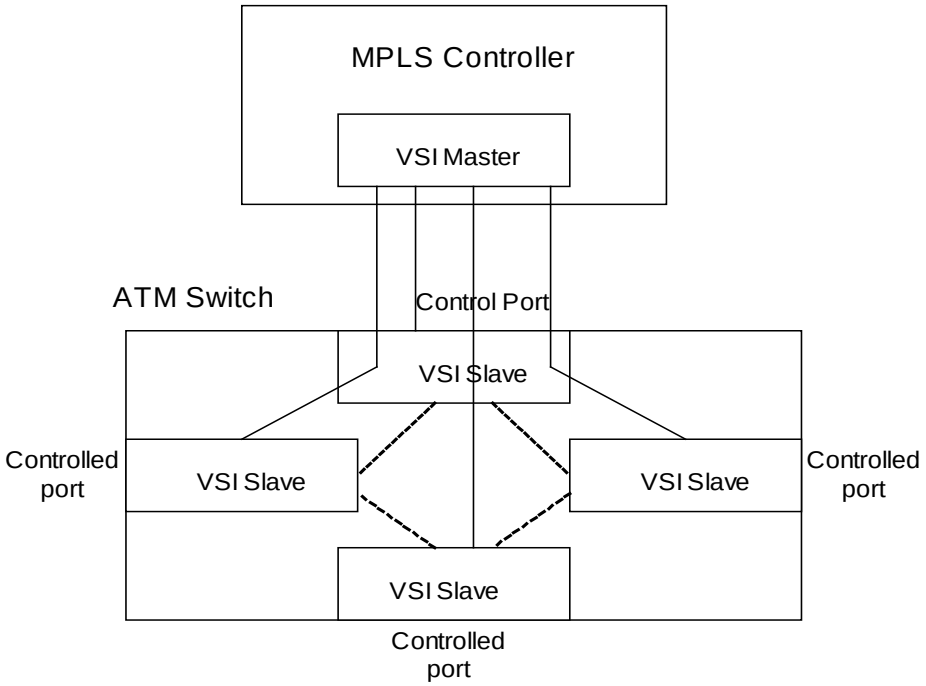
- **Study of VSI as Switch Control Interface** – This task involves configuration and testing of VSI features and study of merits and demerits of VSI as a switch control protocol. As part of this task comparative study is made on the relative advantages and drawbacks of VSI and GSMP. The outcome of the study as part of this task is presented in Sections 2, 3 and 5.
- **Implementation of Open Control Architectures** – this task involves implementation and testing of control architectures primarily label switching architectures. This task is presented in Section 6.
- **Open Control Measurements** – this task involves testing the performance of latest available control implementations from vendors. Measurements are done with VSI 2.2 with WAN Switching Software release 9.2.23 for BPX. The results are presented in Section 4.

- **Contribution to Open Control Community** – this task involves contribution of the experiences of the study conducted during this work to the Open Control community as technical report. The contributions are outlined in Section 7.

2 Study of Cisco's VSI as Switch Control Interface

2.1 VSI Operation Model

VSI is implemented in BPX 8600 using distributed slave model where each BXM card in the BPX switch is a VSI slave and communicates with the controller and other slaves when processing VSI commands. The operation model of VSI is shown in figure 1. The controllers establish link between the master and every slave on the switch and the slaves in turn establish links between each other.



Connection between master and slaves

Connection between slaves

Figure 1 – Logical view of VSI master-slave connections

2.2 Hardware and Software Requirements

BPX controller card type - BCC-4 or BCC-3-64

Controller Type	IOS Software	BPX Image	BXM Firmware	VSI Version
MPLS	12.0(5) T	9.2.10	MEA	VSI 2.2
PNNI/MPLS	12.0(5) T	9.2.23	MEA	VSI 2.2
PNNI and MPLS	12.0(5) T	9.2.30	MEA	VSI 2.2

Table 1: Minimum software/hardware requirements for configuration of VSI MPLS partitions and controllers

2.3 Configuration of VSI Label Switch Controlled partition

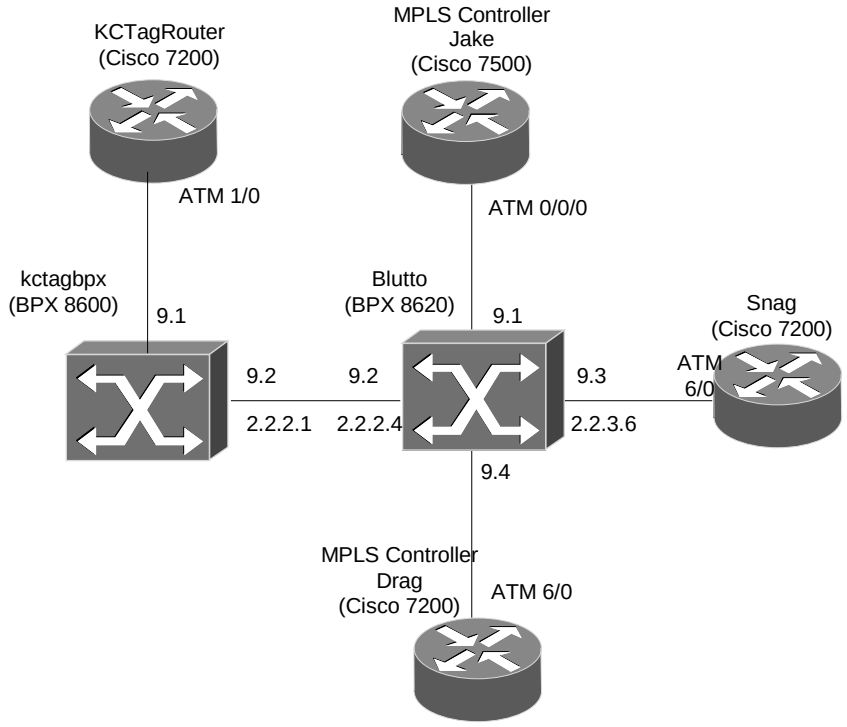


Figure 2 - VSI test set up

Configuration Steps

Configuration of LSC partition involves the following steps:

(i) Enabling the interfaces

The BXM cards or switch interfaces should be in "Standby" or "Active" active state, if not do *'resetcd 9 h'* (slot number is 9) to reset the card. Then the trunks are enabled using *'uptrk <slot.port>'* command.

(ii) Creation of Label Switch partition in *blutto* (BPX)

Partitions are created in BPX switch software using *'cnfrsrc'* command. Initially all the switch resources are used by the Automatic Routing Management or auto-route. Using the *'cnfrsrc'* command resources like bandwidth, label ranges and connections (LCNs) are allocated on each switch port for the LSC partition.

```
cnfrsrc 9.2
PVC LCNs: [0] {accept default value}
max PVC bandwidth[0]: 0
Edit VSI Parms? [N]: y
partition: 1
enabled: e
VSI min LCNs: 0
VSI max LCNs: 7000 {varies with BXM type }
VSI start VPI: 2
VSI end VPI: 3
VSI min b/w: 352207
VSI max b/w: 352207
```

Or

```
cnfrsrc 9.2 0 0 y 1 e 0 7000 2 3 352207 352207
```

(iii) Adding a controller for the partition

The 7200/7500 router is added as LSC using '*addshelf*' command by specifying the physical port of the switch to which it is connected and the partition the controller intends to control. Here the feeder type is set to "vsi" or simply "v". To add PNNI controller the feeder type is set to "X".

```
addshelf 9.4 vsi 2 1
```

Here 9.4 is the port number, the controller id is 2 and partition used is 1. If a partition has two controllers, the controller ID helps the switch software to differentiate between them.

Command '*tag-control-protocol vsi*' enables VSI on the ATM interface of the LSC. Once the router box is attached to the BPX as an LSC for a partition, the switch ports that are included in the partition controlled by the router become the "extended interfaces" or XTagATM interfaces. For each XTagATM interface, a control VC and a range of VPI for label space are automatically configured at the LSC. The configuration at the LSC is shown below.

At the Router/Controller:

```
drag(config)# interface ATM6/0
drag(config-if)# no ip address
drag(config-if)# no ip directed-broadcast
drag(config-if)# no ip route-cache distributed
drag(config-if)# tag-control-protocol vsi id 2
drag(config-if)# no atm ilmi-keepalive
!
drag(config)# interface XTagATM92
drag(config-if)# ip unnumbered Loopback0
drag(config-if)# ip directed-broadcast
drag(config-if)# extended-port ATM6/0 bpx 9.2
drag(config-if)# tag-switching atm control-vc 2 40
drag(config-if)# tag-switching atm vpi 2-3
drag(config-if)# tag-switching ip
!
drag(config)# interface XTagATM93
drag(config-if)# ip unnumbered Loopback1
drag(config-if)# no ip directed-broadcast
drag(config-if)# extended-port ATM6/0 bpx 9.3
drag(config-if)# tag-switching atm control-vc 4 32
drag(config-if)# tag-switching atm vpi 3-4
drag(config-if)# tag-switching ip
```

The management of resources on the VSI slaves requires that each slave in the BPX have a communication control VC to each of the controllers attached to the switch. When a controller is added to the BPX with the '*addshelf*' command, the BCC sets up the set of master-slave connections between the new controller port and each of the active slaves in the switch. The connections are set up using a well-known VPI.VCI. The value of the VPI is 0. The value of the VCI is (40 + (slot - 1)), where slot is the physical slot number of the slave.

In our test setup, once *drag* is configured as LSC using '*tag-control-protocol vsi*' the controller creates a PVC 0/48 through which it creates master-slave control channels (since slot 9 contains the VSI slave entity). The maximum number of BXM slots possible in BPX is 15 out of which two should be BCC. It can be observed from the output of '*show atm vc*' shown below that *drag* has

created 14 channels with BXM slaves in order to establish connection with VSI slaves that could be present in any or all of the remaining slots.

Cross-connects in *blutto*:

```
drag#sh xtagatm cross-connect
Phys Desc    VPI/VCI     Type   X-Phys Desc  X-VPI/VCI   State
0.9.2.0      2/39        ->     0.9.3.0      4/39        UP
0.9.2.0      2/35        ->     0.9.3.0      4/37        UP
0.9.2.0      2/37        ->     0.9.4.0      4/39        UP
0.9.2.0      2/33        ->     0.9.4.0      4/38        UP
0.9.2.0      2/42        <-     0.9.3.0      4/48        UP
0.9.2.0      2/38        <-     0.9.3.0      4/46        UP
0.9.2.0      2/36        <-     0.9.4.0      4/41        UP
0.9.2.0      2/34        <-     0.9.4.0      4/40        UP
0.9.2.0      2/40        <->    0.9.4.0      2/32        UP
0.9.3.0      4/48        ->     0.9.2.0      2/42        UP
0.9.3.0      4/46        ->     0.9.2.0      2/38        UP
0.9.3.0      4/38        ->     0.9.4.0      4/37        UP
0.9.3.0      4/36        ->     0.9.4.0      4/36        UP
0.9.3.0      4/39        <-     0.9.2.0      2/39        UP
0.9.3.0      4/37        <-     0.9.2.0      2/35        UP
0.9.3.0      4/35        <-     0.9.4.0      4/35        UP
0.9.3.0      4/33        <-     0.9.4.0      4/34        UP
0.9.3.0      4/32        <->    0.9.4.0      4/32        UP
0.9.4.0      4/41        ->     0.9.2.0      2/36        UP
0.9.4.0      4/40        ->     0.9.2.0      2/34        UP
0.9.4.0      4/35        ->     0.9.3.0      4/35        UP
0.9.4.0      4/34        ->     0.9.3.0      4/33        UP
0.9.4.0      4/33        <->    0.9.2.0      2/40        UP
0.9.4.0      4/32        <->    0.9.3.0      4/32        UP
0.9.4.0      4/39        <-     0.9.2.0      2/37        UP
0.9.4.0      4/38        <-     0.9.2.0      2/33        UP
0.9.4.0      4/37        <-     0.9.3.0      4/38        UP
```

Master slave channels:

```
drag#sh atm vc
           VCD /                                    Peak  Avg/Min Burst
Interface  Name  VPI VCI  Type Encaps  SC    Kbps   Kbps  Cells  Sts
6/0        1     0   40   PVC  SNAP    UBR   155000                UP
6/0        2     0   41   PVC  SNAP    UBR   155000                UP
6/0        3     0   42   PVC  SNAP    UBR   155000                UP
6/0        4     0   43   PVC  SNAP    UBR   155000                UP
6/0        5     0   44   PVC  SNAP    UBR   155000                UP
6/0        6     0   45   PVC  SNAP    UBR   155000                UP
6/0        7     0   46   PVC  SNAP    UBR   155000                UP
6/0        8     0   47   PVC  SNAP    UBR   155000                UP
6/0        9     0   48   PVC  SNAP    UBR   155000                UP
6/0        10    0   49   PVC  SNAP    UBR   155000                UP
6/0        11    0   50   PVC  SNAP    UBR   155000                UP
6/0        12    0   51   PVC  SNAP    UBR   155000                UP
```

```
6/0       13    0    52    PVC  SNAP     UBR  155000                UP
6/0       14    0    53    PVC  SNAP     UBR  155000                UP
```

(iv) Deleting a controller

A controller can be disassociated from a BPX partition using *'delshelf'* command by specifying the physical port of the switch to which the controller is connected. Disabling the ATM control interface at the controller can also disable the controller.

At *blutto*:

```
delshelf 9.4
```

2.4 Multiple Partitioning

VSI with release 9.2.x supports a total of three partitions with one partition of each type given below:

- Automatic Routing Management
- Label Switch Controlled partition
- PNNI controlled partition

Automatic Routing Management or auto-route is a default partition that is controlled by the BPX switching software itself. Any remaining bandwidth resources after the creation of MPLS or PNNI partition are automatically taken over by the auto-route management. The resources that are partitioned among the different partitions are:

- LCNs
- Bandwidth
- VPI range

Even though the resources are configured and allocated per interface, the pool of resources like LCNs are maintained at switch level. Table 2 gives the limits on port groups and LCN's supported by the various BXM card types.

BXM Card Type	No. Of Port groups	Port Group Size	LCN Limit per Port Group	Average Connections per Port
8-T3/E3	1	8 ports	16K	2048
12-T3/E3	1	12 ports	16K	1365
4-OC3	2	2 ports	8K	4096
8-OC3	2	4 ports	8K	2048
1-OC12	1	1 ports	16K	16384
2-OC12	2	1 ports	8K	8192

Table 2 – The limit on number of port groups, their sizes and limit on number of LCNs per port group

Port groups and LCNs in BPX (*blutto*):

```
Command: dspchuse 9
Channel Management Summary for Slot 9
          mx chans  tot used avail nw used  pvc cnfg  vsi vcs vsi cnf
card 9    :16320    15930    390   1084     2         244     14600
portgrp 1:8160      7774     386   542      0         232     7000
portgrp 2:8160      8156     4     542      2         12      600
```

The maximum number of cards that can be controlled by external controller in the switch is 12. (since two of the 15 slots need to have BCC cards and one need to have an ASM card). Therefore maximum of 12 LCNs in a directly connected slave are required for each controller attached to the BPX. For each slave that is not directly connected, the master-slave control VC consists of two legs one from the VSI master to the backplane through the directly connected slave, and second leg from the backplane to the corresponding VSI slave (figure 1). Hence 1 LCN is required in each of the slaves which are not directly connected to the controller attached to the shelf. These LCNs will be allocated from the auto-route pool. This pool is used by auto-route to allocate LCNs for connections and networking channels. Hence for a given slave the number of VSI management LCNs required from the common pool is $n \times 12 + m$

where

n is the number of controllers attached to this slave,

m is the number of controllers in the switch directly attached to other slaves

The VSI partitioning has the following characteristics and limitations.

- Once partitions are created, dynamic re-allocation of resources between MPLS and PNNI partitions is not allowed. But the resources of the auto-route partition can be redistributed to MPLS or PNNI partition. Partition resources can only be increased.
- No multiple partitions on virtual trunks (section 2.6) are allowed. Virtual trunk is managed by either auto-route or a single VSI partition
- Only one controller can be connected to a BPX port (also called as trunk).

WAN switching software release 9.3 for BPX that is released recently will support a total of three MPLS partitions apart from auto-route. But both MPLS and PNNI partitions cannot coexist as in release 9.2.

2.5 Configuration of VSI master redundancy

VSI supports exclusive redundancy i.e., only one of the controllers can be active in a partition at any time. When a redundant controller is added using '*addshelf*' command, the newly added LSC takes control of the switch partition. The release notes of 9.2.x claimed to support another type of redundancy where multiple LSCs are active in the same partition providing load balancing. But the feature was not found to be present in 9.2.x releases, and in release 9.3 the reference to this type of redundancy is not present.

2.5.1 Configuration commands

The test setup is shown in Figure 2**.** *drag* has already been configured as the controller of an MPLS partition following the steps in section 2.3. The connections in the switch and the OSPF routes in the controller (*drag*) are shown below.

Routes at controller (*drag*)

```
drag#sh ip route
Gateway of last resort is 129.237.127.254 to network 0.0.0.0
     2.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
O       2.2.3.2/32 [110/2] via 2.2.3.2, 00:00:16, XTagATM93
S       2.2.2.0/24 is directly connected, Null0
```

```
O       2.2.2.1/32 [110/2] via 2.2.2.1, 00:00:16, XTagATM92
C       2.2.3.0/24 is directly connected, Loopback1
C       2.2.2.4/32 is directly connected, Loopback0
     3.0.0.0/32 is subnetted, 1 subnets
O       3.1.1.1 [110/2] via 2.2.3.2, 00:00:16, XTagATM93
     8.0.0.0/32 is subnetted, 1 subnets
O       8.1.1.1 [110/2] via 2.2.2.1, 00:00:16, XTagATM92
     129.237.0.0/21 is subnetted, 1 subnets
C       129.237.120.0 is directly connected, Ethernet3/0
S*   0.0.0.0/0 [1/0] via 129.237.127.254
```

Configuration of jake as redundant controller

(i) Add new shelf at port 9.1 to the BPX

A new shelf or a controller box can be added to the BPX using *'addshelf'* command and the controller ID differentiates between the already existing LSC *drag* and the new LSC *jake*. The default value of controller id is 1.

At *blutto*:

```
addshelf 9.1 vsi 1 1
```

After the addition, the XTagATM interfaces of *jake* comes up and *jake* gets the information of the switch regarding existing connections and takes over the control of all switch resources.

(ii) Enabling the router to run VSI on controller port

At redundant controller (*jake*)

```
jake(config)# interface ATM0/0/0
jake(config-if)# no ip address
jake(config-if)# no ip directed-broadcast
jake(config-if)# no ip route-cache distributed
jake(config-if)# tag-control-protocol vsi
jake(config-if)# no atm ilmi-keepalive
!
jake(config)# interface XTagATM92
jake(config-if)# ip unnumbered Loopback0
jake(config-if)# ip directed-broadcast
jake(config-if)# extended-port ATM0/0/0 bpx 9.2
jake(config-if)# tag-switching atm control-vc 2 40
jake(config-if)# tag-switching atm vpi 2-3
jake(config-if)# tag-switching ip
!
jake(config)# interface XTagATM93
jake(config-if)# ip address 192.168.20.2 255.255.255.0
jake(config-if)# no ip directed-broadcast
jake(config-if)# extended-port ATM0/0/0 bpx 9.3
jake(config-if)# tag-switching atm control-vc 4 32
```

```
jake(config-if)# tag-switching atm vpi 3-4
jake(config-if)# tag-switching ip
```

Routes at *jake* after configuration after *addshelf*

```
jake#sh ip route
Gateway of last resort is 129.237.127.254 to network 0.0.0.0
     2.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
O       2.2.3.2/32 [110/2] via 2.2.3.2, 00:00:16, XTagATM93
C       2.2.2.0/24 is directly connected, Loopback0
O       2.2.2.1/32 [110/2] via 2.2.2.1, 00:00:16, XTagATM92
C       2.2.3.0/24 is directly connected, Loopback1
C    192.168.90.0/24 is directly connected, ATM1/0/0
     3.0.0.0/32 is subnetted, 1 subnets
O       3.1.1.1 [110/2] via 2.2.3.2, 00:00:16, XTagATM93
     8.0.0.0/32 is subnetted, 1 subnets
O       8.1.1.1 [110/2] via 2.2.2.1, 00:00:17, XTagATM92
     129.237.0.0/21 is subnetted, 1 subnets
C       129.237.120.0 is directly connected, Ethernet6/1/0
     10.0.0.0/30 is subnetted, 1 subnets
C       10.0.0.0 is directly connected, FastEthernet6/0/0
S*   0.0.0.0/0 [1/0] via 129.237.127.254
```

Cross-connects at *blutto* after *addshelf*:

```
jake#sh xtagatm cross-connect
Phys Desc    VPI/VCI     Type   X-Phys Desc  X-VPI/VCI   State
0.9.1.0      2/43        ->     0.9.2.0      2/36        UP
0.9.1.0      2/42        ->     0.9.2.0      2/34        UP
0.9.1.0      2/35        ->     0.9.3.0      4/35        UP
0.9.1.0      2/34        ->     0.9.3.0      4/33        UP
0.9.1.0      2/33        <->    0.9.3.0      4/32        UP
0.9.1.0      2/32        <->    0.9.2.0      2/40        UP
0.9.1.0      2/41        <-     0.9.2.0      2/37        UP
0.9.1.0      2/40        <-     0.9.2.0      2/33        UP
0.9.1.0      2/39        <-     0.9.3.0      4/40        UP
0.9.1.0      2/38        <-     0.9.3.0      4/38        UP
0.9.1.0      2/37        <-     0.9.3.0      4/36        UP
0.9.2.0      2/39        ->     0.9.3.0      4/39        UP
0.9.2.0      2/35        ->     0.9.3.0      4/37        UP
0.9.2.0      2/37        ->     0.9.1.0      2/41        UP
0.9.2.0      2/33        ->     0.9.1.0      2/40        UP
0.9.2.0      2/38        <-     0.9.3.0      4/48        UP
0.9.2.0      2/36        <-     0.9.1.0      2/43        UP
0.9.2.0      2/34        <-     0.9.1.0      2/42        UP
0.9.2.0      2/40        <->    0.9.1.0      2/32        UP
0.9.3.0      4/48        ->     0.9.2.0      2/38        UP
0.9.3.0      4/40        ->     0.9.1.0      2/39        UP
0.9.3.0      4/38        ->     0.9.1.0      2/38        UP
0.9.3.0      4/36        ->     0.9.1.0      2/37        UP
0.9.3.0      4/39        <-     0.9.2.0      2/39        UP
0.9.3.0      4/37        <-     0.9.2.0      2/35        UP
0.9.3.0      4/35        <-     0.9.1.0      2/35        UP
```

```
0.9.3.0      4/33        <-     0.9.1.0      2/34        UP
0.9.3.0      4/32        <->    0.9.1.0      2/33        UP
```

Master-slave channels at *jake:*

```
jake#sh atm vc
           VCD/                                        Peak  Avg/Min Burst
Interface  Name  VPI VCI Type Encaps  SC    Kbps   Kbps   Cells  Sts
6/0         1     0  40  PVC  SNAP    UBR  155000                 UP
6/0         2     0  41  PVC  SNAP    UBR  155000                 UP
6/0         3     0  42  PVC  SNAP    UBR  155000                 UP
6/0         4     0  43  PVC  SNAP    UBR  155000                 UP
6/0         5     0  44  PVC  SNAP    UBR  155000                 UP
6/0         6     0  45  PVC  SNAP    UBR  155000                 UP
6/0         7     0  46  PVC  SNAP    UBR  155000                 UP
6/0         8     0  47  PVC  SNAP    UBR  155000                 UP
6/0         9     0  48  PVC  SNAP    UBR  155000                 UP
6/0         10    0  49  PVC  SNAP    UBR  155000                 UP
6/0         11    0  50  PVC  SNAP    UBR  155000                 UP
6/0         12    0  51  PVC  SNAP    UBR  155000                 UP
6/0         13    0  52  PVC  SNAP    UBR  155000                 UP
6/0         14    0  53  PVC  SNAP    UBR  155000                 UP
```

Control VCs at *drag*

```
drag#sh xtagatm cross-connect
Phys Desc    VPI/VCI     Type   X-Phys Desc  X-VPI/VCI   State
0.9.2.0      2/40        <->    0.9.4.0      2/32        UP
0.9.3.0      4/32        <->    0.9.4.0      4/32        UP
0.9.4.0      2/32        <->    0.9.2.0      2/40        UP
0.9.4.0      4/32        <->    0.9.3.0      4/32        UP
```

LSC information at *blutto*

```
Command: dspctrlrs
BPX 8620 VSI controller information
Ctrl Id  Part Id   Trunk  Ctrlr Type   Intfc Type   Name
   1        1       9.1   VSI          VSI          VSI
   2        1       9.4   VSI          VSI          VSI
```

2.5.2 Application of exclusive redundancy

This type of redundancy can be used to perform hitless software or hardware upgrades in an active LSC. But if an active LSC crashes or ceases to function, then the other LSC does not take over automatically. The new LSC should be attached to the switch administratively using '*addshelf*'. In the intermediate period, all the control packets (e.g., TDP) destined to the LSC will be discarded.

The neighbor LSRs may also stop sending TDP or LDP messages if the LSC does not send LDP keepalive packets at regular intervals.

2.6 Virtual Trunks

A trunk is nothing but a physical port or interface in the BPX switch. A virtual trunk is a logical trunk defined over a public ATM service. It is used to connect to a public ATM network through VPC. Using an additional level of reference called virtual trunk number denotes a virtual trunk. Within the public ATM cloud, virtual trunk is equivalent to one VPC.

Note: Virtual trunking feature needs to be separately enabled through a request to CISCO TAC.

.

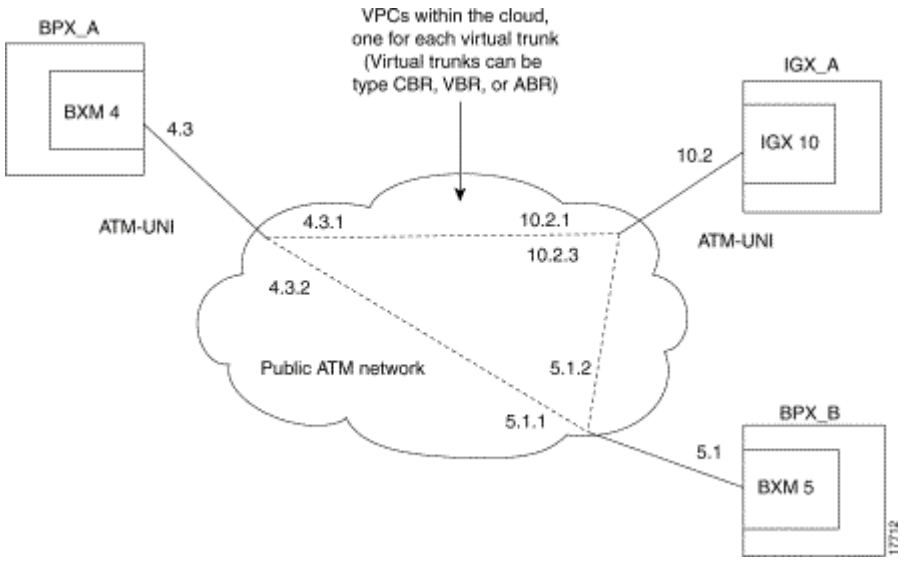


Figure 3 – Virtual Trunks across public ATM network

To configure a virtual trunk numbered 1 in interface 4.3 use

uptrk 4.3.1 to activate the virtual trunk

cnftrk 4.3.1 to set up VPI values and trunk parameters

cnfrsrc 4.3.1 to enable the partition

The BXM card in the BPX has 31 virtual interfaces and each interface has queues and buffers assigned to it. One virtual interface is assigned to each logical (physical or virtual) trunk when the

trunk is enabled. Each virtual trunk uses one of the 31 total virtual interfaces present in the BXM. The allocation of virtual interfaces to logical trunks is shown in figure 4. Sixteen qbins are assigned to each virtual interface. Qbins 0-9 are used by Auto Route and 10-15 are used by VSI. Virtual trunk resources can be allocated to either MPLS/PNNI partition.

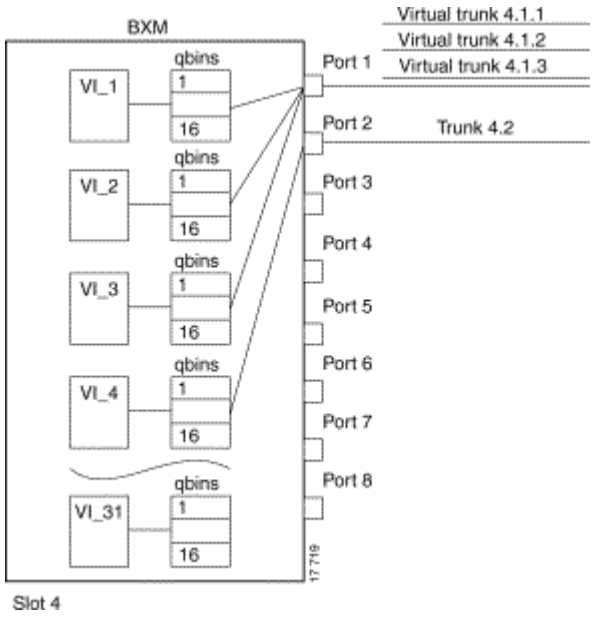


Figure 4 – BXM virtual interfaces and qbins

2.7 Configuration of Service Classes

VSI allows the controller to create connections with desired QoS parameters. Class of Service Templates (CoS Templates) provide a means of mapping the connection control parameters of the switch to controller platform specific extender parameters. LSC understands IP CoS parameters and to create a connection, the CoS parameters need to be translated to BPX ATM QoS parameters.

2.7.1 Class of Service Templates

Each VC that is created is served through one of a number of Class of Service buffers called qbins and the qbins are differentiated by their QoS characteristics. When the port is enabled using '*uptrk*', a set of default service templates consisting of default QoS parameters are loaded from the BCC to

the BXM. The service templates contain two classes of data. One class consists of parameters necessary to establish a connection (i.e., per VC) and includes entries such as UPC actions, various bandwidth- related items and per VC thresholds. The second class of data items includes those that are necessary to configure the associated qbins that provide QoS support. The service class template and qbin values can be modified using *'cnfsct'* and *'cnfqbin'* commands.

The general types of parameters passed from a VSI Master to a Slave include

- a service type identifier,
- QOS parameters (CLR, CTD, CDV),
- Bandwidth parameters (for example PCR, MCR),
- Other ATM Forum Traffic Management 4.0 parameters

When a connection setup request is received from the VSI master in the LSC, the VSI slave uses the service type identifier to index into a CoS Template database containing extended parameter settings for connections matching that index. The slave uses these values to complete the connection setup and program the hardware.

Service Class Template information from *Blutto*

Command: **dspsct**
Service Class Templates

```
Template   Name
   1       MPLS1
   2       ATMF1
   3       ATMF2
```

```
Command: dspsct 1
Service Class    Qbin
    Default       13
    Signaling     10
    Tag0          10
    Tag1          11
    Tag2          12
    Tag3          13
    Tag4          10
    Tag5          11
    Tag6          12
    Tag7          13
    TagAbr        14
```

```
Command: dspsct 1 Tag0
Service Template: MPLS1 (1)
Service Category                Tag0 (200)
Qbin                            10
UPC Enable                      NONE
Scaling Class                   Scaled 1st
CAC Treatment                   LCN
VC Max Threshold                61440            (cells)
VC Dscd Selection               EPD
VC CLP High                     100     (% of Vc MAX Threshold)
VC EPD                          40      (% of Vc MAX Threshold)

Command: dspqbin 9.2 10
Qbin Database 9.2 on BXM qbin 10    (Configured by MPLS1 Template)
                                        (EPD Enabled on this qbin)
Qbin State:               Enabled
Discard Threshold:        105920 cells
EPD Threshold:            95%
High CLP Threshold:       100%
EFCI Threshold:           100%
```

2.7.2 Configuration of MPLS Class of Service

Service providers can use premium services to achieve traffic differentiation for supporting preferred services like VPN. Type of Service (ToS) byte in the IP header is looked up to get the level of preference of the packet. The mapping of ToS byte to CoS type is given in Table – 3.

In MPLS, each LSR requests a label VC from its downstream neighbor for each destination prefix called FEC. To forward traffic based on CoS type, each LSR sets up four label VCs, each VC created with QoS parameters required to get traffic differentiation.

Class of Service Type	TOS byte values
Available	0/4
Standard	1/5
Premium	2/6
Control	3/7

Table 3 – Different IP service types and corresponding TOS byte value

The LSC through VSI creates these label VCs with appropriate CoS parameters. Traffic differentiation is achieved by setting aside bandwidths for each type of traffic. For example, premium traffic can be allocated 40% bandwidth, control 30% and available the remaining 30%. This is the only form of controlling the traffic differentiation from the MPLS controller. The command '*tag-switching atm cos*' is used to allocate desired bandwidth to traffic of specified CoS type. The command below assigns 50% of the bandwidth of the switch interface to premium traffic.

```
jake(config-if)#tag-switching atm cos premium 50
```

The QoST group [6] at KU studied the working and performance of MPLS CoS. We have presented here the test setup (Figure 5) and results for reference.

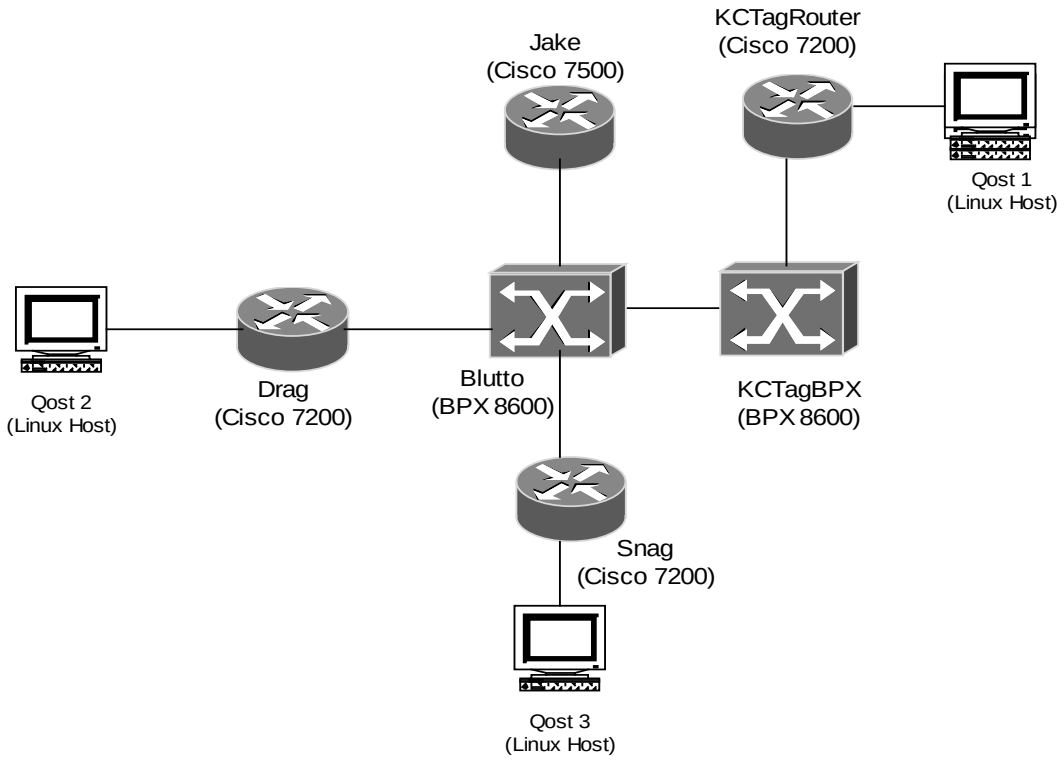


Figure 5 - QoS test setup

2.7.3 Test results

Traffic streams are generated from qost2 and qost3 to qost1 using Netspec [7] tool. Table 4 presents the results of the CoS experiment. Each row shows the percentage bandwidth allocated to specified precedence, the transmitted and received throughputs. The received throughputs are less compared to the transmitted throughput predominantly due to low UDP throughputs at 7200 routers (edges) at OC3 speeds.

The results indicate noticeable service differentiation achieved using MPLS CoS. It can also be noted that the bandwidth allocated is not strict and all the traffic types get some share of bandwidth. For example, test #5 shows that even if all the bandwidth is allocated to premium, the control traffic is given some bandwidth.

Test #	Bandwidth Allocation	Source	Precedence	Transmitted Thruput (Mbps)	Received Thruput (Mbps)
1	100% available	Qost2	0	133.851	22.879
		Qost3	0	133.834	23.284
2	100% available	Qost2	2	133.827	15.446
		Qost3	3	133.833	29.261
3	100% control	Qost2	2	133.841	19.693
		Qost3	3	133.852	27.550
4	100% premium	Qost2	2	133.845	32.254
		Qost3	0	133.837	11.571
5	100% premium	Qost2	2	133.830	32.065
		Qost3	3	133.848	11.744
6	50% available 50% premium	Qost2	2	133.847	22.934
		Qost3	3	133.838	23.583

Table 4 – Results of CoS test results conducted by QoST project

2.8 Protocol or software upgrades

- VSI 1.1 and VSI 2.2 are not interoperable.
- Upgrade of WAN switching software may also require new BXM firmware versions. All the connections in the switch will not be lost. But there will be a temporary disruption of connectivity.

3 Cplane's SSDK

Cplane's Switchlet Service Development Kit (SSDK) [8] versions 1.0 and 2.0 enable an ATM switch to be partitioned into switchlets™ that are nothing but switch partitions. Cplane uses GSMPv1 as the control interface.

3.1 Hardware and Software Requirements

The ssdk2.0 is available for Linux/x86 and Solaris 2.6 environments. The partition manager runs FORE ASX200BX switch with modified ForeThought 5.3 image. Libraries of GSMPv1 client software for the controller are provided with SSDK and the GSMP server running at the slave or switch is integrated in to FORE switch image S_ForeThought_5.3.0.

3.2 Management Interface

The SSDK provides three kinds of interfaces to access the divider or partitioning software in the switch such as CORBA, RPC and SNMP. The partitioning software is accessed through Linux or Solaris boxes to create/delete and to manage partitions. The easiest way to interface with the divider in the switch is through RPC. The switch divider listens on TCP port 10105 and any port can be used at the host side. RPC interface is used as below. The divider can also be directly connected to the switch through an ATM interface.

```
host% rpc_client <-tcp|-atm> -remotehost <remote host> -remoteport
<remote port> -localport <local port>
```

3.3 Partition Management

The various components of the partition management system and the interaction between them are shown in Figure 6.

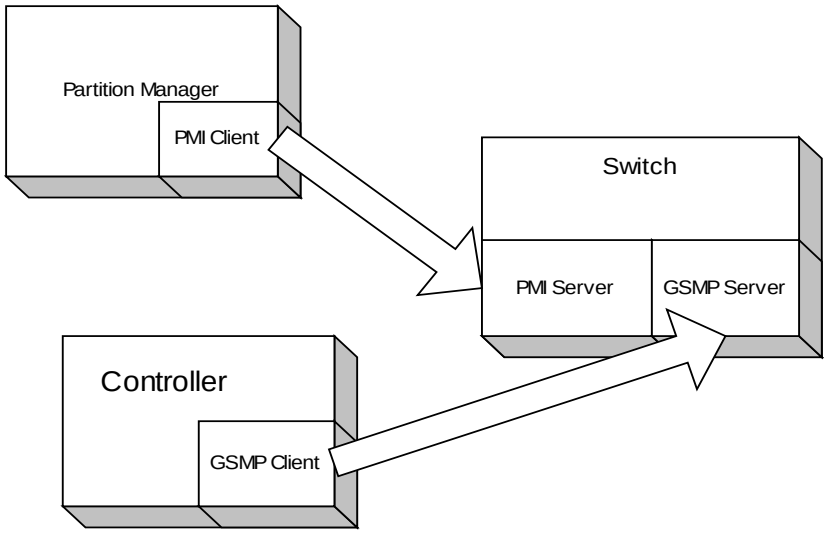


Figure 6 - The switchlet based open controller with onboard divider

3.3.1 Partition Configuration file

For each partition that is to be created, a partition configuration file is required. A sample partition configuration file is given in Appendix. It contains the following details.

- the interface that the controller or control architecture would use (e.g., GSMP, TCP, UDP),
- the port number through which the controller would communicate with the switch,
- the physical ports that the controller will control,
- the resources (label space, bandwidth) allocated for the partition.

3.3.2 Creation and deletion of switch partitions

The partition manager in the FORE switch is accessed through RPC interface as mentioned in section 3.2.

To create a new partition

createSwitchlet <switchlet ID> <switchlet config file>

Switchlet or partition ID is a 32-bit integer that is used to identify the partition and the switchlet config file is a file similar to the one given in Appendix.

To create a switchlet 0 in a switch:

```
dividerMgr> createSwitchlet 0 SwitchletConfig-s0.txt
dividerMgr>  PrintDividerMgrResponse::CreateSwitchletResp  :  msg_id
= 0, retval = MOD_default :DEC_noError
dividerMgr> getSwitchletIds
dividerMgr>  PrintDividerMgrResponse::GetSwitchletIdsResp  :  msg_id
= 3, retval = MOD_default :DEC_noError
Number of switchlets = 1
Switchlet Id 0
```

DEC_noError indicates that the partition has been created successfully. The other messages indicate some form of error like client is not connected to the divider, resource not available for partition creation, the specified port does not exist or syntax error.

To create another switchlet or partition, give a different partition ID and new configuration file. The resources specified in the new configuration file should not include those that are already allocated to partition 1 (e.g., label space).

To delete a partition:

deleteSwitchlet <switchlet ID>

The command returns DEC_noError if successful and returns appropriate error messages if the operation fails.

3.3.3 Addition and shrinking of a partition's resources

3.3.3.1 Addition and removal of ports

To add a switch port to an existing partition:

addPort <switchlet ID> <port resource config file>

A port config file contains the physical port number, the logical port number to be assigned for the port, the label space in the port for the partition and the bandwidth (Appendix). The controller will address the port by its logical port number. The command returns DEC_noError on success

To remove a switch port

removePort <switchlet ID> <logical port number>

3.3.3.2 Addition and removal of label space

To add or remove label space to an existing partition

addLabelSpace <switchlet ID> <logical port> <label space config file>

removeLabelSpace <switchlet ID> <logical port> <label space config file>

3.3.3.3 Addition and removal of bandwidth

To add or remove bandwidth to a partition

addBandwidth <switchlet ID> <logical port> <bandwidth config file>

removeBandwidth <switchlet ID> <logical port> <bandwidth config file>

3.4 Setting Service Categories for connections

SSDK2.0 supports creation of connections based on ATM Forum service categories.

setServiceCategories <switchlet ID> <logical port> <service category mask>

where service category mask is the bitwise OR of the values below.

bit 0 - ATM Forum UBR

bit 1 - ATM Forum CBR

bit 2 - ATM Forum rt VBR

bit 3 - ATM Forum nrt VBR

bit 4 - ATM Forum ABR

To allow connections of type UBR and CBR be created on logical port 1 of partition 1,

```
dividerMgr> setServiceCategories 0 1 3
```

```
dividerMgr>   PrintDividerMgrResponse::SetServiceCategoriesResp   :
msg_id = 5, retval = MOD_default :DEC_noError
```

If any service category is not supported, the switch divider returns error message.

4 VSI Measurements

This section presents the results of measurements conducted using VSI with WAN switching software release 9.2.23 for BPX and Cisco IOS 12.0 (7)T.

4.1 Methodology

All the measurements are taken using the debug timestamps provided by the Cisco 7200/7500 router that acts as LSC. The debug of the router shows all the events taking place along with the timestamps. The router's device driver records the timestamps as soon as the event occurs. Hence the results reflect accurately the behavior or events that take place. Also in an off-board control scenario, even if the resources in the switch are available in the switch partition, it is the view of the controller that is of more importance in the event of putting the resources to use.

4.2 Connection setup and tear-down times

4.2.1 Single VC or Connection setup and teardown times

The process of creating a connection or VC involves

- transfer of VSI protocol message for VC creation from the master to the slave through the control channel specifying the input and output ports in the switch,
- the transfer of information from the VSI software in the BPX to the various slaves (for each interface) to complete the connection with specified QoS,
- notification from the slave to the controller that the connection set up is complete.

The creation of VC involves the traversal of different states

```
DOWN -> ABOUT_TO_CONNECT
ABOUT_TO_CONNECT -> CONNECTING
CONNECTING -> UP.
```

Deletion of a VC or connection involves similar steps.

```
UP -> ABOUT_TO_DISCONNECT
ABOUT_TO_DISCONNECT -> DISCONNECTING
DISCONNECTING -> DOWN
```

Command *'debug vsi packets'* displays all the VSI packets that are exchanged between the VSI master and the slave through the control channel in decoded form including the time these packets arrive or leave the controller. The debug outputs in the LSC are used to measure the setup time with resolution up to milliseconds. The results are shown in Table 5.

VSI Packet debugs

```
drag#debug vsi packets
00:24:46.581: VSI Master: conn 0x90400/4/51->0x90300/4/62:
DOWN -> ABOUT_TO_CONNECT
00:24:46.581: VSI Master: conn 0x90400/4/51->0x90300/4/62:
            ABOUT_TO_CONNECT -> CONNECTING
00:24:46.585: VSI Master: conn 0x90400/4/51->0x90300/4/62:
            CONNECTING -> UP

drag#debug vsi packets
 00:26:11.153: VSI Master: conn 0x90400/4/51->0x90300/4/62:
            UP -> ABOUT_TO_DISCONNECT
00:26:11.153: VSI Master: conn 0x90400/4/51->0x90300/4/62:
            ABOUT_TO_DISCONNECT -> DISCONNECTING
 00:26:11.165: VSI Master: conn 0x90400/4/51->0x90300/4/62:
            DISCONNECTING -> DOWN
```

	Mean (ms)	Std. Deviation (ms)
Setup time	11.407	5.93
Teardown time	16	9.3

Table 5 - Single Connection setup and teardown times

4.2.2 Plot of Connection setup times Vs number of cross connect VCs

The timing of various VSI protocol events and timers play a major role in the setup times of multiple connections. For this test, up to 50 loopback interfaces are created at *kctagrouter* and advertised through OSPF. This triggers the creation of label VC's by TDP. This test can be used to measure the performance of VSI in creation/deletion of various number of cross connect VC's or label VCs.

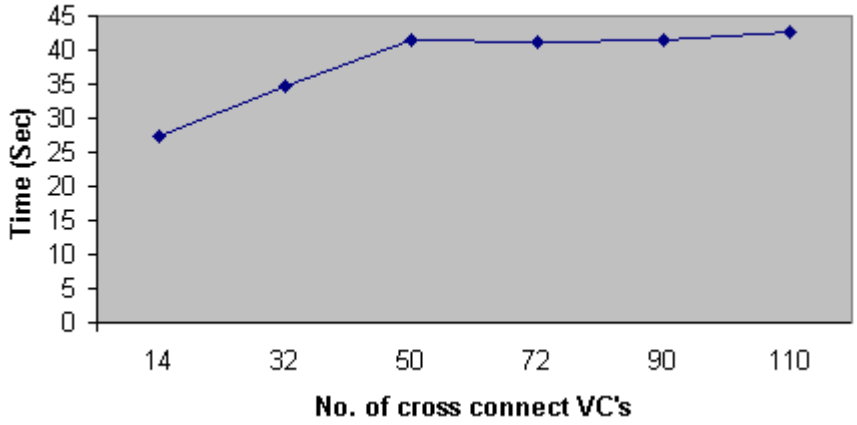


Figure 7 – Setup times for multiple cross-connect VCs

Number of Cross connects	14	32	50	72	90	110
Mean (Sec)	27.374	34.716	41.5	41.168	41.43	42.55
Standard Deviation	1.7	2.06	1.74	0.53	1.16	1.98

Table 6 - Setup Time for multiple cross-connect VCs

The setup time in the result does not include the time taken for the controller to bring the XTagATM interface up (results in section 4.3). The creation of VCs start after the enabling of XTagATM interfaces.

The setup and deletion times for multiple connections are not linearly related (Figure 7). As the number of connections created increases, we do not observe equal increase in the setup times. The

setup time increases as the number of cross connects from 14 to 50 VC's and then remains a constant up to 110 VC's as shown in Figure 7. One possible reason may be that VSI combines many individual connection setup requests into a single message. Expectedly, similar results can be observed for redundant controller also as shown in Figure 8.

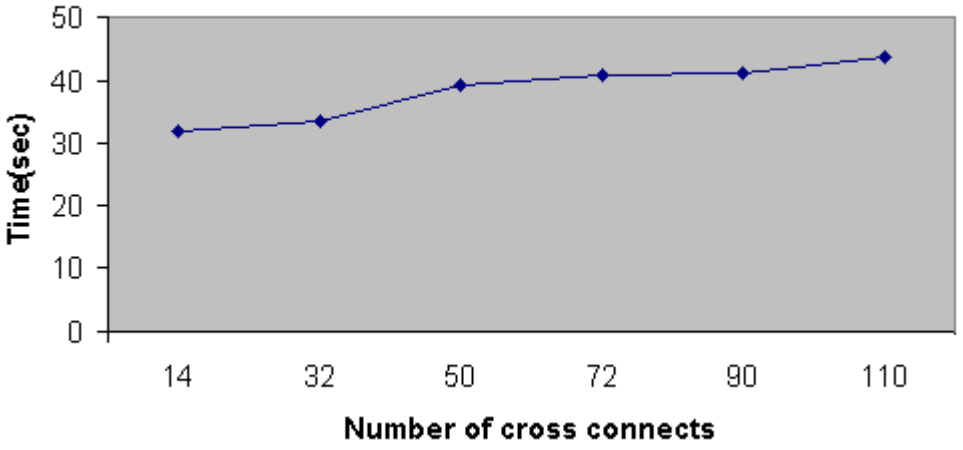


Figure 8 - Setup times for multiple cross-connect VCs (Redundant Controller)

Number of Cross connects	14	32	50	72	90	110
Mean (Sec)	31.99	33.452	39.2067	40.972	41.24	43.688
Standard Deviation	1.308	1.003	2.2	0.54	3.6	1.8

Table 7 - Setup Time for cross connects (Redundant controller)

The deletion time for cross connections (Figure 9) increase significantly from 90 to 110 cross connects but remains a constant up to 130 cross connections. The result should be due to the aggregation of multiple delete requests in a single message

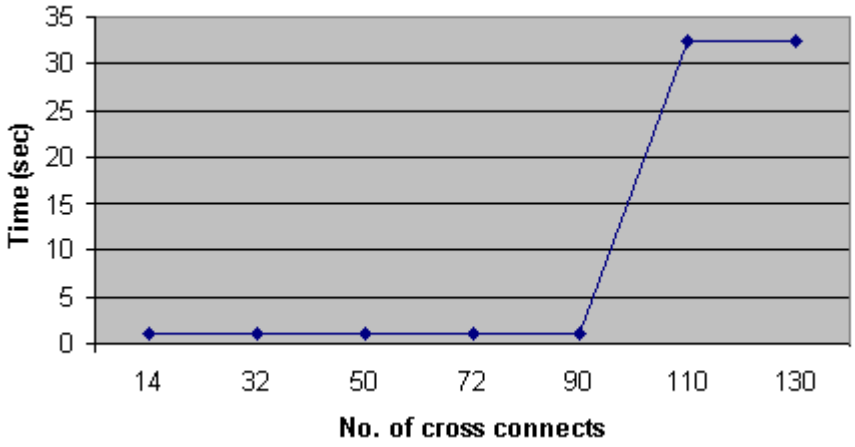


Figure 9 - Deletion times for multiple cross-connects

4.3 Extended interface up/down times

After the addition of controller to the switch, the VSI slave transfers the information of the resources mentioned below to the controller.

- existing connections,
- bandwidth allocated to the partition
- label range on each port
- class of service buffer or qbin information
- to the master (LSC). The controller then views the switch interfaces as its own virtual interfaces called extended or XTagATM interfaces.

After the addition of controller shelf at the BPX using *'addshelf'*, there is time latency before the XTagATM interfaces are enabled at the LSC. This time is measured by observing the router clock at the instant of the executing of *'addshelf'* or *'delshelf'* command and the time the XTagATM interface comes up (through the console log of the controller or router).

```
20:49:28.427 UTC Fri Jul 14 2000 ===> Clock at the time of
execution of 'addshelf'
```

The console log when the interface state changes

```
Jul 14 20:49:40.831: %LINK-3-UPDOWN: Interface XTagATM92, changed
state to up
Jul 14 20:49:40.855: %LINK-3-UPDOWN: Interface XTagATM93, changed
state to up
Jul 14 20:49:41.831: %LINEPROTO-5-UPDOWN: Line protocol on
Interface XTagATM92, changed state to up
Jul 14 20:49:41.855: %LINEPROTO-5-UPDOWN: Line protocol on
Interface XTagATM93, changed state to up

*13:49:36.047 UTC Fri Jul 14 2000 ====> Clock at the time of
execution of 'delshelf'

*Jul 14 13:51:24.827: %LINK-3-UPDOWN: Interface XTagATM92, changed
state to down
*Jul 14 13:51:24.827: %LINK-3-UPDOWN: Interface XTagATM93, changed
state to down
```

	Mean (sec)	Std. Deviation (sec)
Single Controller	12.4928	4.3
Redundant Controller	7.1608	1.294

Table 8 – Time taken to bring up the XTagATM interface at the controller

Master-slave control channel deletion time

After a new LSC is added as redundant controller, the old LSC will have a channel to communicate with the slave software in the switch. Even though the older LSC retains connectivity with the slave, it does not take over the control of the partition in case the active LSC accidentally crashes or ceases to function. And to make the older LSC active again, we need to delete it first using *'delshelf'* and then do *'addshelf'*.

Master-slave control channel deletion time	Mean (sec)	Std. Deviation (sec)
	103.2608	3.62

Table 9 - Time taken for deletion of VSI Master-Slave control channel after *delshelf*

4.4 Connectivity loss during controller change

Exclusive redundancy can be used to provide hitless software/hardware upgrades. This test studies the effect of the switch-over on the cells traversing already created cross-connects and the packets destined for the LSC (or control packets like LDP) is also measured. Both are important considerations for a provider network. Ideally, we would expect almost zero connectivity loss for packets traversing the already created VCs and very minimal loss for packets destined for the LSC extended interface. The set up is the same as in Figure 2.

1. The connectivity loss to the LSC is determined by sending ICMP echoes to the extended interface of the controllers. ICMP echoes are sent from LER-2 to the extended interface 2.2.3.6. Packets go to LSC-1 when it is active and to LSC-2 when it takes over.
2. The loss of the transit packets is found by sending ICMP echoes to the host for which the labels are already established by the LSC. ICMP echoes are sent from LER-2 to LER-1. In this case, through LDP direct VCs will be created between ports 9.2 and 9.3.

Configurations at *snag* (LER)

```
interface ATM6/0
 description to roots
 ip address 192.168.60.1 255.255.255.0
 no ip directed-broadcast
 no ip mroute-cache
 no atm ilmi-keepalive
!
interface ATM6/0.200 tag-switching
 ip unnumbered Loopback0
 no ip directed-broadcast
 tag-switching atm control-vc 4 32
 tag-switching atm vpi 4-5
 tag-switching ip
```

Configurations at *kctagrouter* (LER)

kctagrouter is the controller for *'kctagbpx'* which in turn is connected to *'blutto'*.

```
interface ATM1/0
 no ip address
 no ip directed-broadcast
 tag-control-protocol vsi
 no atm ilmi-keepalive
!
```

```
interface XTagATM92
 ip unnumbered Loopback0
 no ip directed-broadcast
 extended-port ATM1/0 bpx 9.2
 tag-switching atm control-vc 2 40
 tag-switching atm vpi 2-3
 tag-switching ip
!
interface XTagATM93
 no ip address
 no ip directed-broadcast
 extended-port ATM1/0 bpx 9.3
 tag-switching atm control-vc 3 32
 tag-switching atm vpi 3-4
 tag-switching ip
```

During the test, we observed a 36 second connectivity loss with 5 routes for both control and data packets.

Ping output from Snag to the loopback interface of controller (control packet)

```
snag#ping
Protocol [ip]:
Target IP address: 2.2.3.6
Repeat count [5]: 1000
Datagram size [100]: 1000
Timeout in seconds [2]:
Extended commands [n]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 1000000, 1000-byte ICMP Echos to 2.2.3.6, timeout is 2
seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!.................!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 93 percent (228/246), round-trip min/avg/max =
1/1/8 ms
```

Ping output from Snag to kctagrouter (data packet)

```
snag#ping
Protocol [ip]:
Target IP address: 2.2.2.1 ==> snag to kctagrouter (data packet)
Repeat count [5]: 1000
Datagram size [100]: 1000
Timeout in seconds [2]:
Extended commands [n]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 1000, 1000-byte ICMP Echos to 2.2.2.1, timeout is 2
seconds:
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!.....
.....!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!
Success rate is 96 percent (325/340), round-trip min/avg/max = 1/3/4 ms
```

The reason for connectivity loss for control packet can be easily explained. During the switchover period, the extended interface of the existing controller is brought down and that of the new controller is brought up. When the redundant controller LSC-2 is brought up, i.e., the VSI slave at the switch and master in LSC-2 establish connection, the OSPF database at LSC-2 does not contain any routes and the database exchange process with LER-2 has to start again. This happens with TDP process too. Consequently the label VCs are removed from the tag-forwarding table. When the new controller comes up, the OSPF routes are discovered again and new label VCs are obtained.

It can be observed that this connectivity loss time is for 5 routes. If the number of routes is more of the order of a hundred or a thousand, the time taken depends on the time taken for the OSPF routing daemon to discover the routes again from the neighbors. This result shows that traffic is going to be hit no matter how fast the control interface is. This is because of the resynchronization of the routers' OSPF and TDP databases. Hence all the labels are withdrawn by LDP. To start routing packets again, OSPF has to discover the routes again and LDP negotiation should be done. The change in the label VCs in the tag forwarding table at snag before and after switch-over shows that labels are re-negotiated.

Before switch-over

Tag Forwarding Table at *snag*

```
snag#sh tag-switching forwarding-table
Local Outgoing   Prefix        Bytes tag  Outgoing   Next Hop
tag   tag or VC  or Tunnel Id  switched   interface
26    4/42       2.2.2.1/32    0          AT6/0.200  point2point
27    4/36       2.2.2.4/32    0          AT6/0.200  point2point
28    4/38       2.2.3.6/32    0          AT6/0.200  point2point
29    4/44       4.1.1.2/32    0          AT6/0.200  point2point
```

After switch-over

```
snag#sh tag-switching forwarding-table
Local Outgoing     Prefix         Bytes tag  Outgoing   Next Hop
tag   tag or VC    or Tunnel Id   switched   interface
26     4/54        2.2.2.1/32     0          AT6/0.200  point2point
27     4/36        2.2.2.4/32     0          AT6/0.200  point2point
28     4/38        2.2.3.6/32     0          AT6/0.200  point2point
29     4/56        4.1.1.2/32     0          AT6/0.200  point2point
```

5 Comparison of VSI features with that of GSMP

5.1 Resource management

5.1.1 Creation of new partition

In this section, the configuration complexity of Cisco VSI 2.0 in release 9.2.30 and Cplane's GSMP in SSDK 2.0 are compared.

VSI

Resources are set aside on each of the switch port for the new partition to be created. This is achieved through *'cnfrsrc'* command for all the BPX ports. The number of *'cnfrsrc'* is equal to the number of switch interfaces that are to be included in the partition. Partition resources can be added or deleted using the same '*cnfrsrc*' command. LSCs or PNNI controllers can be added using '*addshelf*' command.

Cplane's GSMP

The resources like ports, bandwidth and label ranges are included in the switchlet or partition configuration file (Appendix) and '*createSwitchlet*' command uses this file to create the partition. The SSDK kit contains some default configuration files whose values can to be modified to include resources. The controller can be connected to the ATM switch through any transport like UDP or TCP.

VSI provides a much better and sophisticated command line interface for the creation/deletion of partitions. But with Cplane's SSDK, the controller host need not be directly connected to the switch as in the case of VSI.

5.1.2 Limit on number of partitions

VSI with BPX 9.2.x releases support two controlled partitions (one MPLS and one PNNI) and auto-route partition. But VSI with release 9.3 supports up to three MPLS or PNNI partitions and one Automatic Routing Management (auto-route) partition.

GSMP protocol does not set any limit on the number of partitions. But creation of large number of partitions may not be useful for real-world scenarios. Presence of large number of partitions will increase the difficulty of resource management in provider network. Hence it is desirable to have number of partitions to manageable limit depending on the requirement and try to manage resources effectively. For example, a provider can partition the switches in the network so that one of the partition carries exclusively voice traffic and other partitions carrying data, or one of the partitions carrying high preference (e.g., VPN) traffic and other partitions carrying normal traffic.

5.1.3 Dynamic partitioning

Dynamic partitioning allows the resources of the partitions to be dynamically increased, shrunk or reallocated. For example, frequently the provider need to increase the capacity of their nodes. VSI allows only the increase of partition resources. In many of the cases, increase of partition resources is the most reasonable requirement.

But with GSMP, once a controller is attached to a partition, addition or removal of resources is not possible. This is because GSMP protocol does not include message that the switch partitioning software can use to inform the controller that the resources are added/removed. In this aspect, VSI can be more beneficial than GSMP in that it decreases the resource management complexity.

5.2 Support for different control planes

GSMP protocol supports many control planes like MPLS, Q.2931, PNNI and Frame Relay. This is mostly attributed to the fact that GSMP is an open protocol and has the contributions from larger group of people. VSI supports only MPLS and PNNI controller types.

5.3 Support for Redundancy

Redundancy is one of the best methods of providing fault tolerance in any networking system. In provider or carrier networks carrying customer traffic, redundancy is almost inevitable. There are two kinds of scenarios that need redundancy.

1. We have an LSC controlling a partition of a switch and it needs to be upgraded for some reasons and we cannot afford to drop the control of the partition during this upgrade. Here the disruption is administratively controlled and hence it may not be called redundancy in a strict sense.
2. Providing general standby redundancy that automatically takes control when the primary goes down. Standby redundancy can also be achieved using proprietary hardware backups provided by the switch vendor (this can also be called hardware redundancy unlike protocol or software redundancy).

Case 1 can be useless if the primary goes down unexpectedly and case 2 can be wastage of resources of the standby controller. The optimum solution is to have two controllers in the same partition and each of them assigned some subset of resources to control through administrative configuration. If one of the controllers goes down or needs to be upgraded, the other controller automatically takes over the orphaned section of the partition. This can also be termed as load sharing which provides redundancy as well as efficient use of the controller. This can be effected through redundant hardware or through control interface protocol. We would later argue that the protocol solution is not efficient.

Cisco in the latest release proposes an indirect way of implementing load-sharing type of redundancy. Configure two MPLS partitions controlled by two independent LSCs. The division of the switch is dependent on the network operator. He may or may not choose to divide the partitions equally. The actual and the logical network with equal cost multipath are shown in Figure 10.

However, if one of the controllers fails, that portion is not taken over by any other controller. To support near zero disruption of packets when one of the controllers goes down, one needs to use hardware standby redundancy. Here a backup BPX port card that is connected to the redundant LSC is required. This standby card will synchronize its state with the active card and on any disruption, the standby card takes over. Each controller will store the internal state of the other controller. This solution does not use the control interface (VSI) to maintain redundancy.

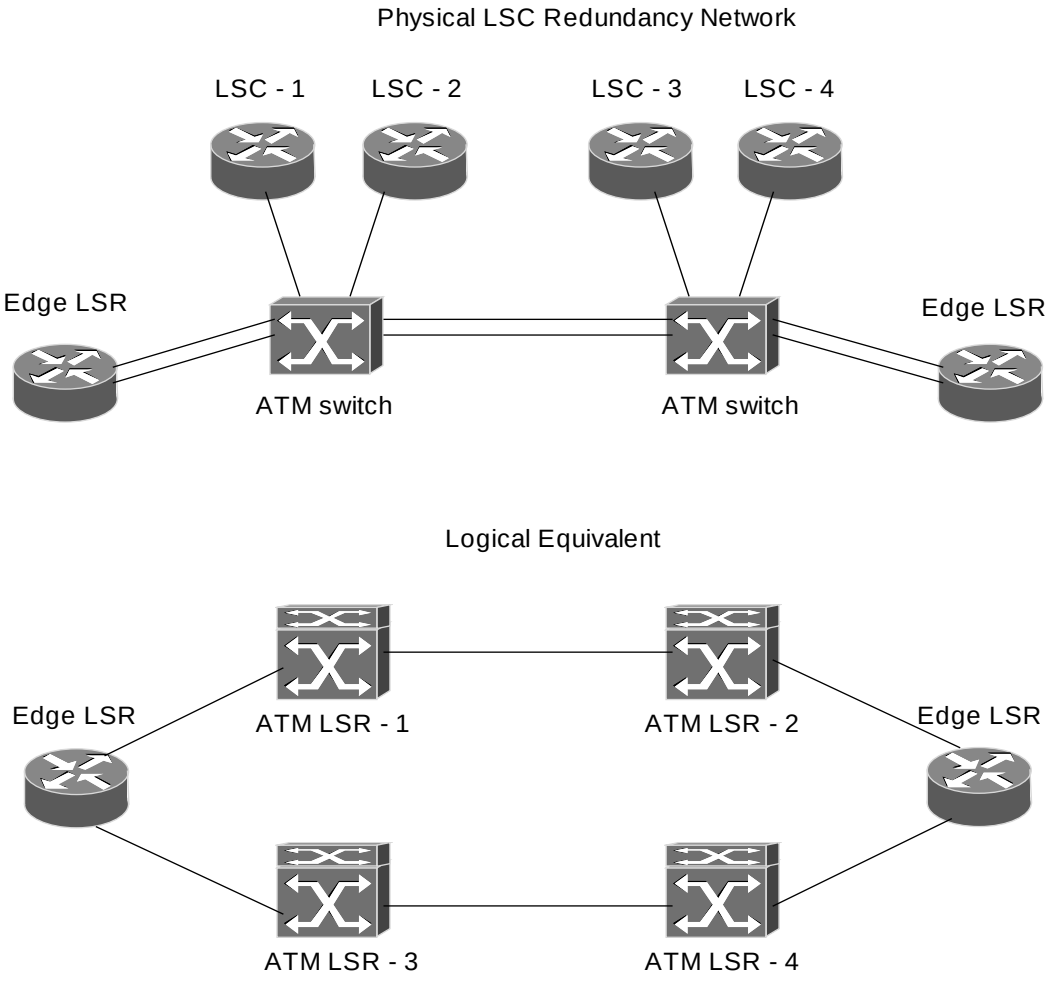


Figure 10 – LSC redundancy using IGP multipath

Multiple switch controllers may jointly control a single switch partition using GSMP. The controllers may control a switch partition either in a primary/standby fashion or as part of multiple

controllers providing load sharing for the same partition. It is the responsibility of the controllers to co-ordinate their interactions with the switch partition. In order to assist the controllers in tracking multiple controller adjacencies to a single switch partition, the Adjacency Update message is used to inform a controller that there are other controllers using the same partition. It should be noted that the GSMP does not include features that allow the switch to co-ordinate cache synchronization information among controllers. The switch partition will service each command it receives in turn as if it were interacting with a single controller. Controller implementations without controller entity synchronization should not use multiple controllers with a single switch partition.

Results in section 4.4 show that packet loss is inevitable if the OSPF and LDP states of the controllers are not synchronized. With large number of routes as present in the provider networks, it is not efficient and feasible to allow for such packet losses. Even if the control interface (GSMP or VSI) supports automatic switch over of controllers, it will not be a deployable solution at the networks where the number of routes is large and reliability is very important. Hence hot standby redundancy using hardware is better. Multipath solution is also an effective solution. But when one of the LSCs goes down, the effective available bandwidth of the traffic is reduced by the capacity of the partition whose LSC goes down. The choice of the type of redundancy depends on the requirements of the network.

5.4 Support for Service Models

As mentioned before, the motivation for going to off-board control is the need to support multiple services. Open control provides the flexibility for the control software to evolve independently that can create new and essential services. The service delivery problem can be handled in two ways. One by making reservations right from the source to the destination or by defining some forwarding classes that determine the level of priority or precedence to be assigned to the packet. IntServ and ATM use the first method and Diff-Serv uses the other method. Some applications need reservations whereas some applications may be satisfied with some increased performance. The optimum way is to make reservations not in each switch or router but in each transit Autonomous System (AS) and use the priority field in the packet to differentiate the packets within a domain.

VSI supports

- IP service types (MPLS partition),
- ATM Forum service categories (PNNI controlled partition).

The controller can use VSI to create connections of specified QoS parameters and type. Connections in a Label switch partition can be created based on IP CoS parameters providing desired bandwidth share for different IP traffic types (section 2.7.2).

The latest version of GSMP protocol (v3) will support various service types.

- ATM Forum Service Categories,
- Integrated Services,
- MPLS CR-LDP,
- Frame Relay,
- Circuit Emulation

Even though work on GSMPv3 is still not complete at IETF, GSMP is expected to support wider range of service models due to its open nature.

5.5 Conclusions

Our study of the features of VSI and GSMP control protocols indicate that:

- VSI does not allow two LSC to be active in a partition; the redundancy supported is exclusive. During the switch over to the redundant LSC, packet loss is inevitable. The time for recovery will depend on the OSPF and LDP processes to discover routes and negotiate labels respectively.
- The two effective methods for providing LSC redundancy will be hot standby redundancy and multipath LSC redundancy. Such solutions are preferable to the solution provided by the control protocol. Since GSMP does not support synchronization of LSC databases, such solutions are good with GSMP too.
- GSMP supports wider range of control planes and service models than VSI.

- GSMP does not support dynamic partitioning whereas VSI allows increasing of partition's resources. In this aspect, VSI gives better partition management capability.
- An open control interface like GSMP provides more avenues for improvement and interoperability among multiple vendors.

6 Implementation of Open Control Architectures

The concept of off-board or open control is more relevant to ATM switches than IP routers due to signaling present in ATM. The control interface of a switch allows the controller to setup and modify the flow or connections through the switch or router. In IP routers, the notion of connection or flow is not enforced. The forwarding is based only on the destination address. In ATM, the notion is required at the call setup before any data is transferred. On the other hand off-board control can be more appropriate for MPLS routers or switches that require signaling across a domain before the transfer of data. This is true when MPLS is run on almost all device layers like ATM, frame relay or DWDM switches. Also considering into account the emergence of IP as a common network protocol, MPLS is more likely to be deployed widely than other control architectures that do not use IP as network layer.

As the number of commercial organizations desire to use Internet more for their business and marketing purposes, the demand for value added services also increases. Initially the providers sought to provide service guarantees using connection-oriented ATM protocols. Approaches such as Classical IP over ATM (CLIP) and Multi-Protocol over ATM (MPOA) achieve IP over ATM operation by overlay model which limits network scalability. Another problem for the providers is the complexity of managing large ATM networks. MPLS enables ATM switches to be fully integrated into IP networks thereby improving scalability and reducing complex QoS translation mechanisms. Standards also exist for MPLS on other link layers like Packet Over SONET (POS), Frame Relay, DWDM and Ethernet.

The following are our efforts as part of this project for implementation of open control architectures.

- to integrate missing functionality to the previous version of the implementation of offboard tag control architecture

- to integrate and test IP Class of Service (CoS) functionality to the KU implementation of MPLS LDP

6.1 Implementation of offboard Tag Switching Architecture

Tag Switching, developed by Cisco systems a few years back, combines the performance and virtual-circuit capabilities of link-level switching with the scalability, flexibility and robustness of network routing. Tag switching, along with other Layer 3 switching schemes have been submitted to the IETF Multi-Protocol Label Switching (MPLS) Working Group. The working group is responsible for standardizing a base technology for using label swapping forwarding paradigm (label switching).

6.1.1 Tag Switching Concepts

The tag is a short, fixed length identifier that is assigned to packets belonging to a certain flow of date. A tag switching network consists of:

- Tag Edge Routers (TERs) that are located at the edge of the Tag Cloud.
- Tag Switch Routers (TSRs) that switch tagged packets based on the tags and may support Layer 3 routing or Layer 2 switching apart from Tag Switching.

Tag switching consists of two components:

- Forwarding Component - Responsible for forwarding packets based on the tag
- Control Component - Responsible for obtaining and maintaining tag using the Tag Distribution Protocol (TDP) or extensions to existing routing protocols.

TERs and TSRs use standard routing protocols to identify routes through the network and fully inter-operate with non-tag switching routers. Tag switches use the tables generated by the standard routing protocols to assign and distribute tag information via the Tag Distribution Protocol (TDP).

6.1.2 The Tag Distribution Protocol (TDP)

This section provides a brief description of the Tag Distribution Protocol (TDP) [11]. TDP runs over a connection-oriented transport layer (primarily TCP) with guaranteed sequential delivery. It provides a means for TSRs to distribute, request, and release tag binding information. To communicate with other TSR the TDP establishes sessions. Sessions between two TSRs and sessions among the same TSRs are all independent. TDP does not require any keepalive notification from the transport, but implements its own keepalive timer.

The different types of Protocol Information Elements (PIEs) in TDP are:

- Type 0x100 TDP PIE OPEN – to start a new session with the neighbor
- Type 0x200 TDP PIE BIND – to provide label binding (by downstream TSR)
  Type 0x300 TDP PIE REQUEST BIND – to request label binding (by upstream TSR)
- Type 0x400 TDP PIE WITHDRAW BIND – the LSR that provided the binding uses this message to withdraw
- Type 0x700 TDP PIE RELEASE BIND – the LSR that requested the binding uses this message to release the binding
- Type 0x500 TDP PIE KEEP ALIVE – keepalive message
- Type 0x600 TDP PIE NOTIFICATION- to notify the neighbor of error in a TDP message

KU offboard tag switching implementation uses the TCP/IP stack in Linux hosts, which we will refer henceforth as Tag Switch Controller (TSC) nodes, for routing purposes. By looking up the Linux routing table at the TSC node, TDP is initiated between a node and it's downstream neighbor and tags are obtained. In this implementation, a tag is the same as an ATM VCI. The tags are distributed and a Tag Information Base (TIB) is maintained at the tag switching nodes, based on the network topology as reflected by the routing tables. Based on these distributed tags, appropriate switched paths are set up across ATM switches that are controlled by these TSC nodes.

We implemented a TDP state machine and each of the TSC nodes maintains TDP state information. This implementation supports Destination-based routing and Downstream-on-demand tag allocation only. The TSCs communicate with the ATM switches via SNMP interface to establish the tag switched paths.

6.1.3 Implementation Environment and Tools

CMU's SNMP library

linux-2.2.9 kernel and atm-0.59 with tag forwarding patch

Implementation of TDP

FORE ASX-200BX switches running Forethought 4.1 or Forethought 5.3

6.1.4 Implementation Structure

Each Linux host is associated with an ATM switch to function as a TSR (TSC + ATM Switch) in our off-board implementation of Tag Switching. The implementation consists of TDP daemon that implements TDP state machine and continuously monitors routing table, opens TDP sessions with neighbors and gets/receives label bindings. The state machine consists of three independent processes.

- **wait_for_tdp_packets** – waits for TDP packets on all interfaces and processes the requests/messages from the TDP neighbors. This process implements a concurrent server and spawns a new process for each new session.
- **wait_for_tags_and_maintain_tib** – waits for messages from different TDP processes on a known port and installs/deletes tags/PVCs in the kernel and the ATM switch accordingly. This process also creates VCs in the ATM switch using SNMP interface.
- **dynamic_route_updater** – monitors Linux routing table and on any route change initiates request bind or withdraw operation. When a new binding is received from the neighbor for the bind request, this process gets available free VC from ATM switch and sends request to the known internal port for creation of VC at the host and the switch.

When a new route is added at the TSC, the daemon detects the change in route and initiates a new TDP session with the next hop. The next hop TSC, on receiving a tag request, queries the ATM switch it is controlling to get a free available VC, and returns the tag binding (destination, tag pair). When the binding is complete, the module that gets the binding sends the tag information on well-known internal port. The other module that listens on the internal port gets the binding information and establishes appropriate switched paths based on the tag and the TIB. The SNMP protocol's

GetNextRequestPDU is used to obtain free VCs on the switch and SetRequest-PDU is used for connection setup/tear-down.

When a route present in the TIB is lost, the TSC sends withdraw or release message to the appropriate neighbor and cleans up the VCs in the kernel and the ATM switch. The Linux kernel especially the CLIP module is modified to incorporate the functionality of forwarding IP datagrams segmented into ATM cells based on destination address lookups in the TIB and if not available, route them based on kernel routing table.

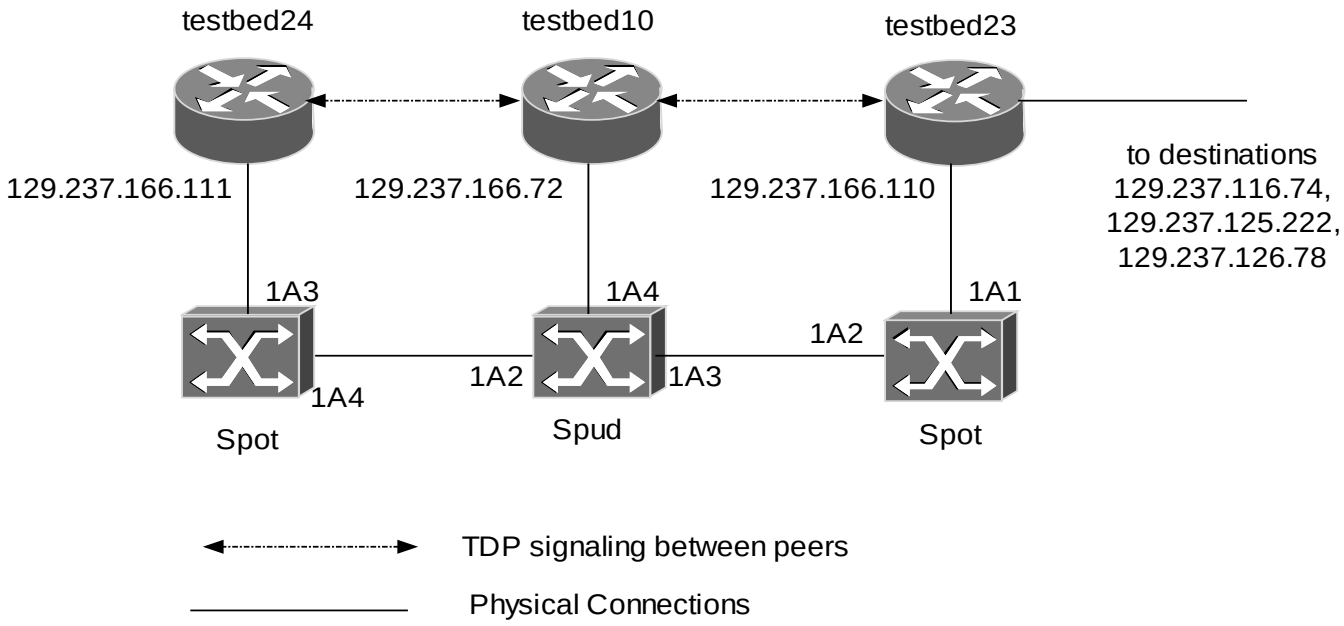


Figure 11 - Tag Switching test setup

The messages TDP PIE WITHDRAW BIND and TDP PIE RELEASE BIND were not implemented in the previous implementation. Also the associated clean up of resources were not implemented. The current work involves the implementation of these messages and the clean up of resources when tag bindings become obsolete.

6.1.5 Test Results

testbed24, *testbed10* and *testbed23* are TSCs connected to the switches spot and spud as shown in Figure 11. *testbed24* is the ingress, *testbed10* is the core and *testbed23* is the egress.

TDP database at the TSRs

```
testbed24% show tdp database
Destination     Upstream   Inbound tag     Downstream      Outbound tag
129.237.116.74  Ingress         -1         129.237.166.72  102
129.237.125.222 Ingress         -1         129.237.166.72  103
129.237.126.78  Ingress         -1         129.237.166.72  104


testbed10% show tdp database
Destination     Upstream         Inbound tag  Downstream    Outbound tag
129.237.116.74  129.237.166.111   102       129.237.166.110    102
129.237.125.222 129.237.166.111   103       129.237.166.110    103
129.237.126.78  129.237.166.111   104       129.237.166.110    104


testbed23% show tdp database
Destination     Upstream         Inbound tag   Downstream   Outbound tag
129.237.116.74  129.237.166.72    102           Egress          -1
129.237.125.222 129.237.166.72    103           Egress          -1
129.237.126.78  129.237.166.72    104           Egress          -1
```

PVCs in the TSCs

```
testbed24 [131] % more /proc/atm/pvc
Itf        VPI   VCI    AAL           RX(PCR,Class)   TX(PCR,Class)
  0        0     102     5             0 UBR                0 UBR
  0        0     103     5             0 UBR                0 UBR
  0        0     104     5             0 UBR                0 UBR
```

No PVCs in the core, since all the traffic are directly switched.

```
testbed10 [131] % more /proc/atm/pvc
Itf  VPI       VCI         AAL     RX(PCR,Class)  TX(PCR,Class)


testbed23 [185] % more /proc/atm/pvc
Itf        VPI   VCI    AAL           RX(PCR,Class)   TX(PCR,Class)
  0        0     102     5             0 UBR                0 UBR
  0        0     103     5             0 UBR                0 UBR
  0        0     104     5             0 UBR                0 UBR
```

PVCs on switch *spot*

```
spot::configuration vcc> show
Input                      Output
Port       VPI        VCI   Port   VPI      VCI   UPC   Protocol   Name
```

```
1A2          0        102   1A1      0      102    0    pvc        N/A
1A2          0        103   1A1      0      103    0    pvc        N/A
1A2          0        104   1A1      0      104    0    pvc        N/A
1A3          0        102   1A4      0      102    0    pvc        N/A
1A3          0        103   1A4      0      103    0    pvc        N/A
1A3          0        104   1A4      0      104    0    pvc        N/A
```

PVCs on switch *spud* (cross-connects at the core)

```
spud::configuration vcc> show
Input                       Output
Port        VPI       VCI   Port   VPI      VCI   UPC   Protocol   Name
1A2          0        102   1A3      0      102    0    pvc        N/A
1A2          0        103   1A3      0      103    0    pvc        N/A
1A2          0        104   1A3      0      104    0    pvc        N/A
```

Route to 129.237.116.74 is deleted at *testbed24* (ingress)

On route deletion at the ingress (*testbed24*), the ingress sends TDP PIE RELEASE BIND message requesting *testbed10* to release the binding. If the request is successful *testbed10* still has the route, it will now be the ingress and creates VC from itself to the switch.

```
testbed24% show tdp database
Destination     Upstream   Inbound tag     Downstream      Outbound tag
129.237.125.222 Ingress         -1         129.237.166.72  103
129.237.126.78  Ingress         -1         129.237.166.72  104


testbed10% show tdp database
Destination     Upstream        Inbound tag  Downstream    Outbound tag
129.237.116.74  Ingress           -1        129.237.166.110    102
129.237.125.222 129.237.166.111   103       129.237.166.110    103
129.237.126.78  129.237.166.111   104       129.237.166.110    104


testbed23% show tdp database
Destination     Upstream        Inbound tag   Downstream   Outbound tag
129.237.116.74  129.237.166.72    102           Egress          -1
129.237.125.222 129.237.166.72    103           Egress          -1
129.237.126.78  129.237.166.72    104           Egress          -1


testbed24 [131] % more /proc/atm/pvc
Itf        VPI  VCI    AAL           RX(PCR,Class)   TX(PCR,Class)
  0        0    103     5             0 UBR               0 UBR
  0        0    104     5             0 UBR               0 UBR


testbed10 [131] % more /proc/atm/pvc
Itf        VPI  VCI    AAL           RX(PCR,Class)   TX(PCR,Class)
```

```
  0        0     102      5              0 UBR                 0 UBR

testbed23 [131] % more /proc/atm/pvc
Itf        VPI   VCI     AAL           RX(PCR,Class)    TX(PCR,Class)
  0        0     102      5             0 UBR                 0 UBR
  0        0     103      5             0 UBR                 0 UBR
  0        0     104      5             0 UBR                 0 UBR
```

The cross-connect VC from 1A2 to 1A3 is deleted and new VC from 1A4 to 1A3 is created.

```
spud::configuration vcc> show
Input                      Output
Port        VPI       VCI   Port   VPI       VCI   UPC   Protocol   Name
1A4          0        102   1A3      0       102   0     pvc        N/A
1A2          0        103   1A3      0       103   0     pvc        N/A
1A2          0        104   1A3      0       104   0     pvc        N/A
```

Route to 129.237.125.222 is deleted at *testbed10* (core)

When *testbed10* (core) loses a route, it sends RELEASE BIND to its downstream (here the egress) and WITHDRAW BIND to the upstream (or ingress). In this test, when core loses route to 129.237.125.222, all the VCs pertaining to this destination are deleted.

```
testbed24% show tdp database
Destination     Upstream   Inbound tag     Downstream      Outbound tag
129.237.126.78  Ingress         -1         129.237.166.72  104

testbed10% show tdp database
Destination     Upstream         Inbound tag  Downstream    Outbound tag
129.237.116.74  Ingress            -1        129.237.166.110    102
129.237.126.78  129.237.166.111    104       129.237.166.110    104

testbed23% show tdp database
Destination     Upstream         Inbound tag   Downstream   Outbound tag
129.237.116.74  129.237.166.72     102           Egress          -1
129.237.126.78  129.237.166.72     104           Egress          -1

testbed24 [131] % more /proc/atm/pvc
Itf        VPI   VCI     AAL           RX(PCR,Class)    TX(PCR,Class)
0          0     104      5             0 UBR                 0 UBR

testbed10 [131] % more /proc/atm/pvc
Itf        VPI   VCI     AAL           RX(PCR,Class)    TX(PCR,Class)
  0        0     102      5             0 UBR                 0 UBR
```

```
testbed23 [131] % more /proc/atm/pvc
Itf        VPI   VCI     AAL           RX(PCR,Class)    TX(PCR,Class)
  0         0     102      5             0 UBR                0 UBR
  0         0     104      5             0 UBR                0 UBR


spud::configuration vcc> sh 1a2 0
Input                      Output
Port         VPI        VCI   Port    VPI       VCI   UPC   Protocol   Name
1A2           0         104   1A3       0       104    0    pvc        N/A
```

Route to 129.237.126.78 is deleted at *testbed23* (egress)

When egress (*testbed23*) loses a route, it sends TDP PIE WITHDRAW BIND message to the core which in turn sends another WITHDRAW BIND for the same destination to the ingress deleting all the resources allocated for the particular destination.

```
testbed24% show tdp database
Destination     Upstream   Inbound tag     Downstream      Outbound tag


testbed10% show tdp database
Destination     Upstream        Inbound tag  Downstream    Outbound tag
129.237.116.74  Ingress           -1        129.237.166.110    102


testbed23% show tdp database
Destination     Upstream        Inbound tag   Downstream   Outbound tag
129.237.116.74  129.237.166.72    102           Egress          -1


testbed24 [131] % more /proc/atm/pvc
Itf   VPI   VCI     AAL      RX(PCR,Class)    TX(PCR,Class)


testbed10 [131] % more /proc/atm/pvc
Itf        VPI   VCI     AAL           RX(PCR,Class)    TX(PCR,Class)
  0         0     102      5             0 UBR                0 UBR


testbed23 [131] % more /proc/atm/pvc
Itf        VPI   VCI     AAL           RX(PCR,Class)    TX(PCR,Class)
  0         0     102      5             0 UBR                0 UBR
```

Since *testbed10* and *testbed23* have the route to 129.237.116.74, each still has a binding for that destination.

6.2 Implementation of MPLS CoS

Multi protocol Label Switching (MPLS) is a high performance forwarding method. The reachable destinations are classified into various Forwarding Equivalence Classes (FECs). Packets belonging to an FEC are treated in the same manner. For example, a destination prefix present in the routing table can be an FEC and all packets going to that prefix are forwarded in the same way. The packets are forwarded using labels. A label is a short fixed sized integer (16 bits) that has only local significance and summarizes the information contained in the packet header. The label distribution is achieved through a different protocol like Label Distribution Protocol (LDP) or enhanced routing protocol. Since the label is of fixed size, the routing table search time can be greatly reduced.

Since LDP [12] is designed for label signaling, it provides more flexibility for MPLS signaling than other enhanced routing protocols. LDP is the set of procedures and messages by which Label Switched Routers (LSRs) establish Label Switched Paths (LSPs) through a network by mapping network-layer routing information directly to data-link layer switched paths. When to request a label or advertise a label mapping to a peer is largely a local decision made by an LSR. In general, the LSR requests a label mapping from a neighboring LSR when it needs one, and advertises a label mapping to a neighboring LSR when it wishes the neighbor to use a label. This implementation of LDP supports down stream on demand label distribution.

Internet services are better sold if they provide certain guarantees rather than indeterministic increase in performance. But attempts to implement reservations like RSVP are not feasible due to lack of scalability at the core of the network. MPLS stores states from edge-to-edge of an AS and hence the number of stored states will be proportional to the number of nodes in the MPLS domain. By proper architecture of MPLS domains, the LSP states can be kept to manageable level. This will let the providers to support reservations for preferred traffic.

6.2.1 MPLS CoS over ATM

MPLS uses the switching ability if present of the link layer on which it is run. Since majority of the devices in the core at present are ATM switches, proper translation of IP or MPLS CoS to ATM QoS parameters is essential for service differentiation. IP CoS can have a maximum of 8 different service classes and the translation mechanism would translate each service class to:

- an ATM service category (CBR, VBR or UBR),
- cell drop parameters (EPD and PPD)

This implementation recognizes 4 IP service classes 0/4, 1/5, 2/6, 3/7 and uses different drop settings and bandwidth allocation to achieve service differentiation.

MPLS working group of the IETF suggested two approaches for translation:

- The most significant three bits of the IP ToS octet are copied to EXP field of MPLS shim header and appropriate treatment is given based on the EXP field of the label. In this model, RED and per-VC CBFQ provide service differentiation. The advantages of the QoS support provided by ATM is not properly utilized.
- In the second approach, LDP is used to signal N labels per precedence per IP source-destination pair. This model provides more flexibility in resource allocation and utilizes the ATM QoS more effectively. There is a possible drop at the edge and the core of the network in this case. This gives rise to congestion management at every hop which is an added advantage. The implementation uses this approach.

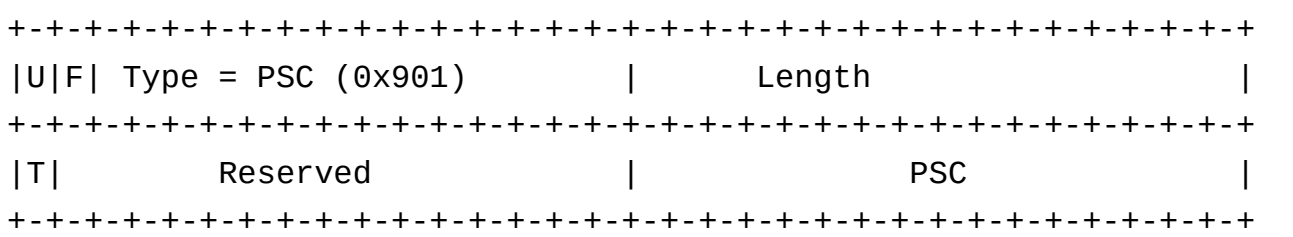
6.2.2 Implementation Approach

The service differentiation is achieved at

- Edge using per CoS CBFQ and appropriate filters
- Core using per CoS CBFQ and EPD on ATM switch

UBR CLP 0/1 with appropriate Q size and CLP thresholds are used to achieve traffic differentiation among four classes. CLP set for precedences 0-3 and not set for precedences 4-7.

The DiffServ TLV [13] is used to signal precedence/codepoint information along with the label request and label release messages. The DiffServ TLV for an L-LSP is given below.



T: LSP Type. This is set to 1 for an L-LSP

Reserved: 15 bits - This field is reserved. It must be set to zero on transmission and must be ignored on receipt.

PSC: 16 bits - The PSC indicates a PHB Scheduling Class to be supported by the LSP.

Using the DiffServ TLV, the LDP peers exchange labels and associate label values with appropriate precedences

6.2.3 Implementation Environment and Tools

- CMU's SNMP library.
- TC tool for Linux and iproute2 [14].
- Linux 2.3.99 pre2 kernel and atm-0.71.
- Implementation of LDP.
- GNU's zebra [15] routing software (version 0.85) – contains OSPF and BGP routing processes

FORE switch MIB variables

- chanrInputPort, chanrInputVPI, chanrInputVCI, chanrOutputPort, chanrOutputVPI, chanrOutputVCI, chanUpcContract of channelGroup
- upcContractKey, upcContractPCR01, upcContractPCR0, upcContractCDVT, upcContractAal5Epd, upcContractDoGCRAPolicing, upcContractIsAAL5, upcContractDoPacketDiscard, upcContractDoPPPolicing, upcContractEstimatedUbrBandwidth of the UPC Configuration group.

- pshmemQsizeforCBR, pshmemClpThreshforCBR of portShmemGroup

6.2.4 Implementation Structure

Linux PCs support the TCP/IP suite and can act as routers if they have multiple interfaces and the kernel is configured to support IP forwarding. These Linux boxes act as Label Switch Controllers (LSCs). Each Linux node is associated with an ATM switch to function as a LSR (LSC + ATM Switch) in the offboard implementation of MPLS. SNMP was used for communication between Linux and the Fore switch.

The LDP daemon implementation that takes care of signaling, requesting and binding labels starts with four different independent processes each having a different function.

send-hello-packets-periodically - sends LDP HELLO packets to a multicast address on all the physical interfaces. This is used to discover LDP peers in the network.

process-hello-packets - processes LDP HELLO packets that arrive on an the multicast address. HELLO packets are processed to obtain the capabilities of the peer and if the session can be established with the neighbor.

wait-process - starts a new LDP session upon reception of a LDP OPEN message. The process does the initial LDP state transition and then lets the LIB process take care of processing requests.

LIB process - takes care of processing request for labels and binding them. The main function of the LIB process is to build a Label Forwarding Information Base LFIB. The LIB process also does the following:

- implements a concurrent server that takes care of processing requests from the clients (sessions),
- creates session processes to manage exchange between the two MPLS peers
- takes care of label allocation and cross-connecting using SNMP

The zebra daemon manages the routing tables on Linux and interacts with OSPF and BGP daemons to obtain topology information of the network. It also interacts with the LIB process to get/set certain network and QoS specific parameters.

The Multi-VC module of the LIB process takes care of establishing four LSPs per source destination pair and assigning appropriate Queue size, CLP threshold etc to each VC. It interacts with the Fore switch to get free VCs, establish cross connects and set UPC contracts using SNMP. It also sends the LIB entries to the kernel through the kernel module. The module also adds the DiffServ TLV to the label request and label mapping messages.

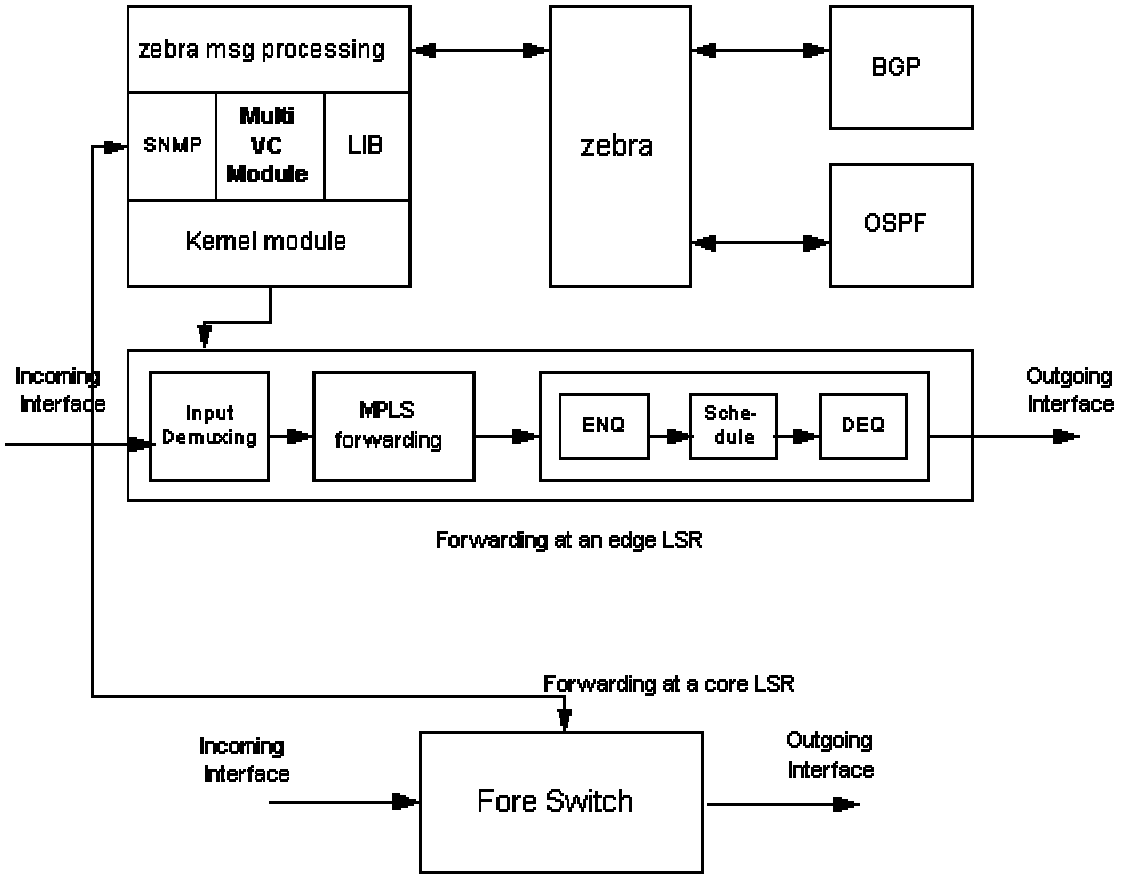


Figure 12 – Architecture of MPLS CoS Implementation

Zebra is modified to let CoS parameters be configured on Zebra's command line interface. The zebra msg processing module takes care of requests from zebra and processes them and sends LIB entries and session information to zebra. This module is responsible for indicating a CoS configuration change to the multi-VC module. The multi-VC module sets up appropriate queues

and filters at the edge and IPC parameters at the core (ATM switch). The *iproute2* and *tc-tools* are used to setup queues and filters.

When an LER receives a packet to be forwarded, it enqueues the packet in a Class Based Queue. The scheduler after dequeuing the packet looks up in the kernel Label Information Base (LIB) to find the proper outgoing VC and precedence. The dequeuing based on precedence bits provides service differentiation at the edges. The ATM switches at the core merely switch packets from one VC to another based on the cross-connects that were created using SNMP during LDP signaling. When egress LER receives the packet, it does normal IP forwarding.

6.2.5 CoS Implementation

Different types of CoS mechanisms are required at the core and at the edge. As mentioned earlier, the edge LSR has to implement per-CoS CBFQ and the core LSR has to implement per-CoS CBFQ and per-CoS EPD. The mechanisms used to implement per-CoS CBFQ at the edge is different from that used at the core.

6.2.5.1 At the Edge

The TC tool for Linux is used to implement queues and filters on Linux. This implementation uses the Class Based queues that come with the TC tool. Four classes are created with user configurable bandwidths. Eight filters are then setup at the ingress to filter packets that belong to any one of the eight precedences onto the appropriate queue. These information relating to these queues and filters are then sent to the kernel using *netlink* socket. In the kernel, these *netlink* messages are received and Qdiscs are setup according to the specifications in the Netlink messages. The way in which Qdiscs and classes are setup on a particular interface are as follows

1. Create a parent Qdisc with bandwidth equal to the total aggregate bandwidth of the four classes.
2. Create four classes with user configurable bandwidths and assign priorities of 0, 1, 2 and 3 to the four classes. The class with the higher priority is chosen in preference to a class with a lower priority when scheduling.

3. Associate the four classes with the parent Qdisc.
4. Create 8 filters for the parent Qdisc to match the ToS octet and associate a flow ID with each filter. Use the flow ID in the classes to match filters.

6.2.5.2 At the Core

At the core, SNMP is used to configure the switches to provide appropriate CoS. Fore switches have Usage Parameter Control (UPC) MIB variables using which one can assign bandwidth to a particular Class. An UPC contract defines the parameters that traffic belonging to the contract should confine to. UPC contracts can be used for policing, rate limiting, queueing, scheduling etc.
The way this implementation uses UPC contracts is as follows. Three user configurable UPC contracts are created on each port of ATM switch. Parameters like PCR, MBS, SCR, PCR for 0 CLP, EPD, CDVT and estimated UBR bandwidth can be set using UPC contracts.

Each VC of classes 1, 2 and 3 is created using an UPC contract. Hence all the VCs belonging to the same class will have properties associated that particular UPC contract. No UPC contract is defined for class 0. This helps in achieving per-CoS treatment at a LSR.

6.2.6 Test Results and Observations

The test scenario that was used for testing the implementation is shown in Figure 13. The MPLS cloud consists of three LSRs, two LERs and one core. Two static routes were also configured to enable connectivity from qost3 to *wintermute* and vice-versa through the MPLS cloud.

Traceroute results from *qost3* to *wintermute* before LDP was run is as shown below.

```
qost3 [3] # traceroute wintermute
traceroute to wintermute.ittc.ukans.edu (129.237.126.161), 30 hops
max, 38 byte
packets
 1  qost4 (129.237.126.36)  0.186 ms  0.126 ms  0.119 ms
 2  192.168.125.2 (192.168.125.2)  0.371 ms  0.368 ms  0.354 ms
 3  192.168.126.2 (192.168.126.2)  0.593 ms  0.591 ms  0.583 ms
 4  wintermute (129.237.126.161)  0.698 ms  0.660 ms  0.648 ms
```

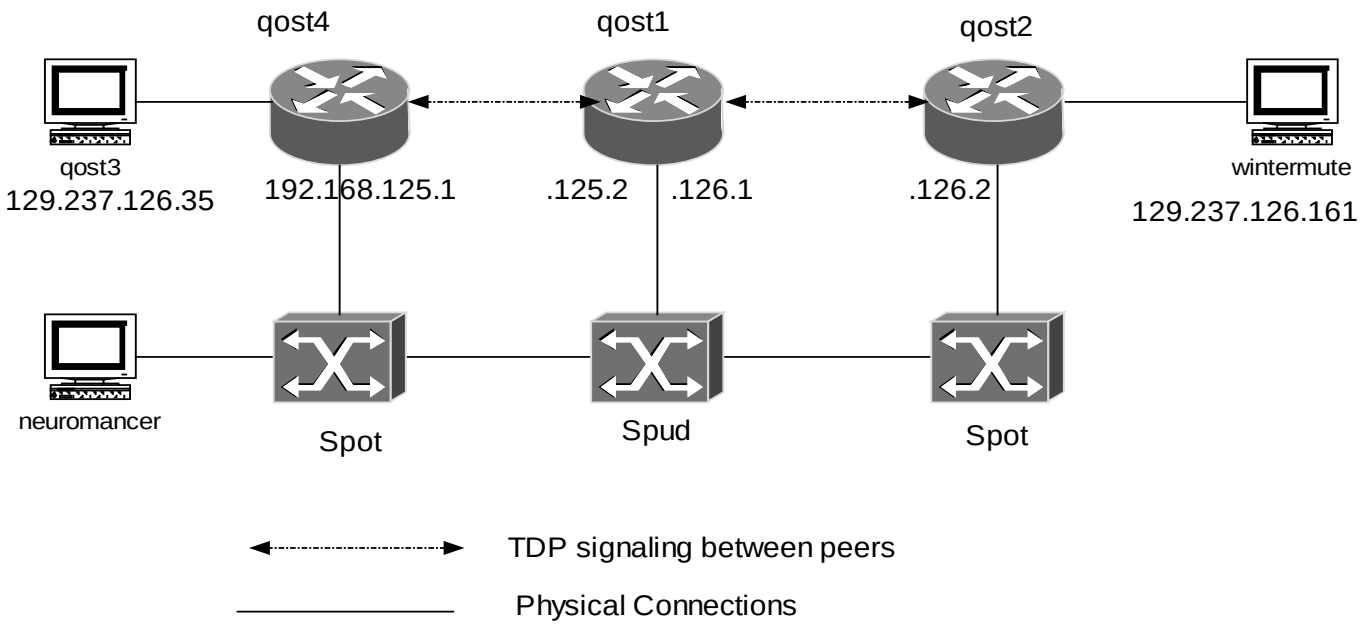


Figure 13 - MPLS CoS test setup

LDP was run on all the three LSRs. The LFIB, session information and traceroute results after LDP was run is as given below. It can be seen that traceroute sees the MPLS cloud as a single hop.

```
qost3 [4] # traceroute wintermute
traceroute to wintermute.ittc.ukans.edu (129.237.126.161), 30 hops
max, 38 byte
packets
 1  qost4 (129.237.126.36)  0.314 ms  0.135 ms  0.125 ms
 2  192.168.126.2 (192.168.126.2)  0.670 ms  0.610 ms  0.581 ms
 3  wintermute (129.237.126.161)  0.824 ms  0.659 ms  0.656 ms
```

6.2.6.1 Test #1

Two traffic streams were generated from *qost3* to *wintermute* using Netspec. The first stream was given a precedence of 1 and the second stream was given a precedence of 2. The allocated

bandwidth and throughputs are shown in Table 10. The total bandwidth is approximately 133 Mbps.

Precedence	Allocated Bandwidth (Mbps)	Transmitted thruput (Mbps)	Received thruput (Mbps)
1 (class 1)	60	36.280	35.696
2 (class 2)	60	36.215	35.547

Table 10 – Two streams with equal bandwidth allocation

It can be seen that allocating equal bandwidth to the two classes (class 1 and class 2) results in both the flows getting almost the same received throughput.

6.2.6.2 Test #2

Here, class 1 was allocated a bandwidth of 20 Mbps and class 2 was allocated 60 Mbps and the same test was conducted. The result is in Table 11.

Precedence	Allocated Bandwidth (Mbps)	Transmitted thruput (Mbps)	Received thruput (Mbps)
1 (class 1)	20	36.273	4.178
2 (class 2)	60	36.197	36.176

Table 11 – Two streams with unequal bandwidth allocation

A result of similar test with 20 Mbps allocated to class 2 flow is given in Table 12.

Precedence	Allocated Bandwidth (Mbps)	Transmitted thruput (Mbps)	Received thruput (Mbps)
1 (class 1)	60	36.278	36.198
2 (class 2)	20	36.235	4.418

Table 12 - Two streams with unequal bandwidth allocation

Observation

The results in Table 11 and 12 show that even though the total volume of the two streams is not high, the stream that is allocated lesser bandwidth suffered heavy drops. Since at the core packets are switched, the drop is due to the queuing done at the ingress.

6.2.6.3 Test #3

In this test, 4 stream bursts were sent between *qost3* and *wintermute* and the link between *qost4* and *qost2* though the Fore switch was congested by background traffic of 10 Mbps and highest precedence (precedence = 7). The transmitted and received throughputs of the traffic streams are shown in Table 13. Also the throughputs of the background traffic are shown.

Precedence	Allocated Bandwidth (Mbps)	Transmitted thruput (Mbps)	Received thruput (Mbps)
6 (class 2)	40	20.005	19.914
2 (class 2)		20.006	19.905
5 (class 1)	60	20.009	19.819
1 (class 1)		20.012	0.398

Table 13 – Four streams with one stream CLP set

Background traffic

```
neuromancer [227] # netspec neuro-winter | grep Thru

   Thruput transmitted    :          10.001 Mbps
  Thruput received        :           9.960 Mbps
```

It can be seen that class 1 traffic whose CLP was set has a low throughput compared to the other flows. Other flows with CLP not set did not suffer drops. Also, the highest precedence background traffic did not suffer any drop.

6.2.6.4 Test #4

In this test, a background stream of 40 Mbps was used to fill the pipe between *qost4* and *qost2* though the Fore switches. Two full blast streams were then sent to compete for bandwidth between *qost4* and *qost2*. The results are shown in Table 14. The precedence of the background traffic is 6.

Precedence	Allocated Bandwidth (Mbps)	Transmitted thruput (Mbps)	Received thruput (Mbps)
1 (class 1)	40	36.255	5.490
2 (class 2)	80	36.235	28.116

Table 14 – Two streams allocated unequal bandwidths with background traffic

```
neuromancer [283] # netspec neuro-winter | grep Thru
  Thruput transmitted           :          39.975 Mbps
  Thruput received              :          39.969 Mbps
```

It can be seen that the background traffic suffered no drops in the ATM core. The class 1 flow suffered drops even though it was allocated a considerable bandwidth and the class 2 flow did not suffer heavy drop. This test clearly shows that relative bandwidth was the criteria used to do per CoS treatment.

6.3 Conclusions

As part of this project, missing functionality to the previous version of the implementation of offboard tag control architecture are implemented and IP Class of Service (CoS) functionality is integrated and tested to the KU implementation of MPLS LDP. The MPLS CoS implementation forms the basis for providing end-to-end QoS service in a QoS aware network spanning IP and MPLS domains. Future work will be the integration of a Bandwidth Broker to negotiate/maintain SLAs and set/reserve resources and mapping from DiffServ to MPLS CoS.

7 Contributions to Open Control Community

As part of this task, the observations and experiences gained in this work are presented to Open Control community as a technical report [16]. The contributions also include the elucidation of the relative merits and demerits of VSI and GSMP as switch control protocols.

Some salient observations are:

- Resource management will be made simpler if the partition resources are allowed to be increased or be readjusted. In most of the situations, reducing partition resources may affect the connections that are already present in the partition. The control interface should provide messages for the controller to increase or readjust the switch resources. VSI allows increase of partition resources but with GSMP, the partitioning is static and has to be done before running GSMP adjacency.
- Hardware or multipath redundancy is better for deployment than redundancy support provided by the control protocol. This is because packet loss is inevitable in the control protocol method since the internal states (OSPF, LDP databases) of the controllers cannot be kept synchronized completely without hardware support.
- GSMP supports wider range of control planes and provides for better interoperability among multitude of controller and switch vendors.

8 Conclusions

Open control architectures facilitate the evolution and rapid deployment of new, innovative and interesting control architectures that use the switching resources more efficiently. Flexible service creation is easier with network devices that run with open control. Our effort is to study, configure and evaluate the features of two popular label switch control interfaces namely VSI and GSMP and implement Tag switching and MPLS control architectures that make use of open control.

Recently we observe dominance of different vendors in different levels of the network. Some vendors dominate primarily in high-speed switches based on optical solutions and some in edges. This will lead the providers to look for interoperable solutions to make better use of latest

developments. Work on GSMPv3 is ongoing and IETF has not come up with final standard yet. GSMPv3 supports wide range of link layers and control planes and finalization of GSMPv3 standard will most likely enable many vendors to market their implementations. Future work will be to study the usefulness of these control protocols with high-speed optical switches.

9 References

[1] E. Rosen et. al., Multiprotocol Label Switching Architecture, draft-ietf-mpls-arch-07.txt, July 2000

[2] Jim McEachern, Service Control Interface Requirements Multiservice Switching Forum (MSF) Contribution, 21 June 1999

[3] P. Newman et. al., Ipsilon's General Switch Management Protocol Specification Version 2.0 (GSMPv2.0), RFC 2297, March 1998

[4] A. Doria et. al., General Switch Management Protocol V3, draft-ietf-gsmp-06.txt, August 2000

[5] Cisco CCO Documents, Release Notes and Configuration Guides

[6] QoS Translation (QoST) Project, Information and Telecommunication Technology Center, University of Kansas

[7] R. Jonkman, NetSpec: A Network Performance and Evaluation Tool, University of Kansas

[8] Cplane's Switchlet Software Development Kit (SSDK) 2.0, Instructions Manual

[9] K. Sundell and A. Doria, GSMP WG response to MSF SCI Requirements, draft-ietf-gsmp-msf-response-00.txt, 20 August 1999

[10] K. Sundell and A. Doria, General Switch Management Protocol Applicability, draft-ietf-gsmp-applicability-01.txt, July 2000

[11] P. Doolan et. al., Tag Distribution Protocol, draft-doolan-tdp-spec-01.txt, May 1997

[12] L. Andersson et. al., LDP Specification, August 2000

[13] Le Faucheur et. al., MPLS Support of Diff-Serv, draft-ietf-mpls-diff-ext-07.txt, August 2000

[14] Linux 2.4 Advanced Networking HOWTO, http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html

[15] Zebra Routing Software version 0.85, www.zebra.org

[16] B. Ramachandran, C. Ramachandran and Dr. J.B.Evans, Experiences with VSI and GSMP as offboard switch control protocol, ITTC Technical Report.

Appendix A

Switchlet Configuration file:

```
switchletConfig {
        switchletPaConfig {
                paType = gsmpSwitchletPa;
                paTransport {
                        transportType = UDP;
                        localPort = 30001;
                }
        }
        switchletResourceConfig {
                portConfig {
                        physicalPort = 0;
                        logicalPort = 0;
                        portBandwidthConfig {
                                inBw = 500;
                                outBw = 500;
                        }
                        portLabelSpaceConfig {
                                inLabelSpaceConfig {
                                        numVcRanges = 1;
                                        vcRange {
                                                minVpi = 1;
                                                maxVpi = 1;
                                                minVci = 400;
                                                maxVci = 450;
                                        }
                                }
                                outLabelSpaceConfig {
                                        numVcRanges = 1;
                                        vcRange {
                                                minVpi = 1;
                                                maxVpi = 1;
                                                minVci = 400;
                                                maxVci = 450;
                                        }
                                }
                        }
                }
                portConfig {
                        physicalPort = 1;
                        logicalPort = 1;
                        portBandwidthConfig {
                                inBw = 500;
                                outBw = 500;
                        }
                        portLabelSpaceConfig {
                                inLabelSpaceConfig {
                                        numVcRanges = 1;
                                        vcRange {
```

```
                                        minVpi = 1;
                                        maxVpi = 1;
                                        minVci = 400;
                                        maxVci = 450;
                                }
                        }
                        outLabelSpaceConfig {
                                numVcRanges = 1;
                                vcRange {
                                        minVpi = 1;
                                        maxVpi = 1;
                                        minVci = 400;
                                        maxVci = 450;
                                }
                        }
                }
        }
}
```