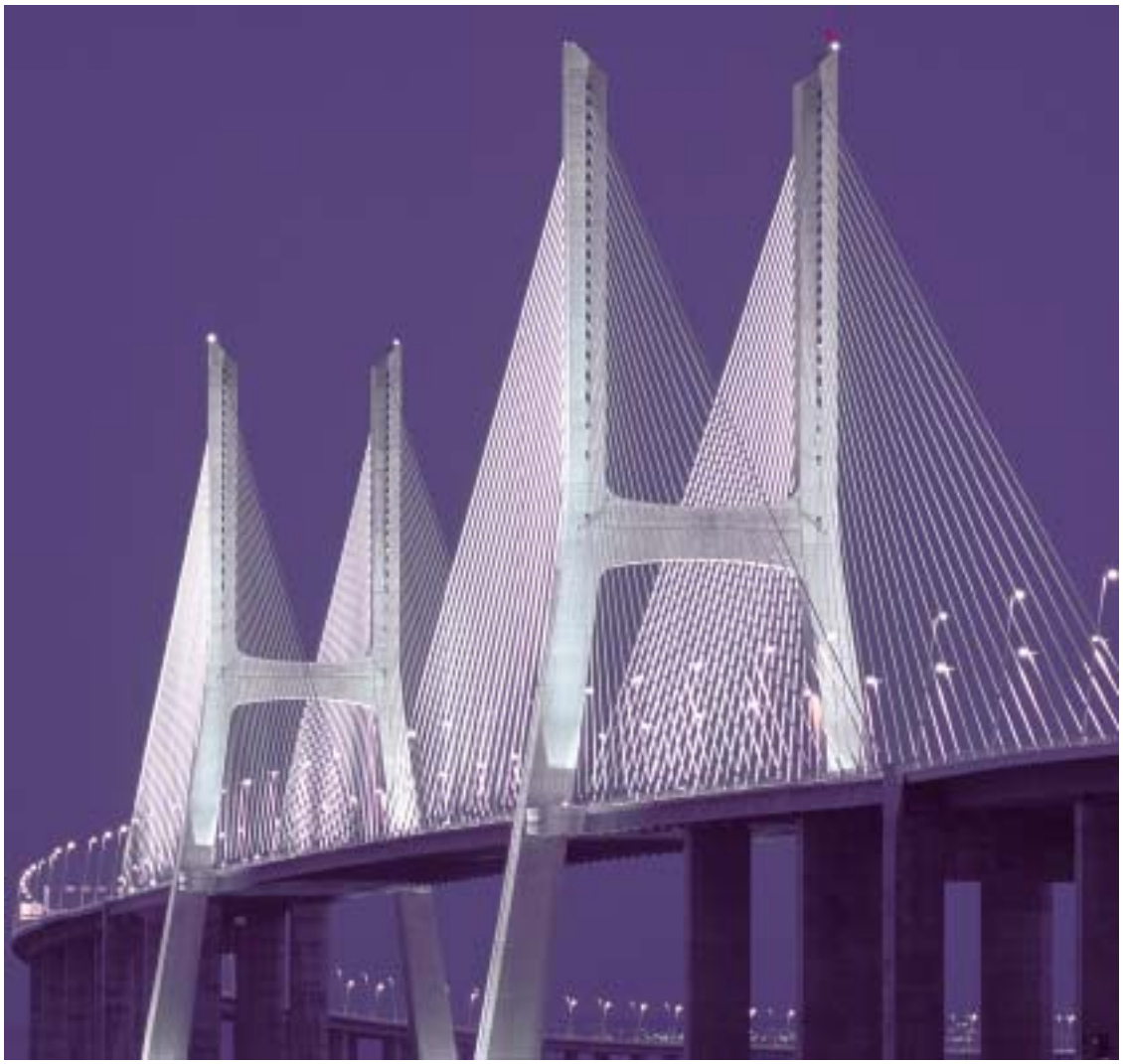


ZX5000 Series User Guide

Switching Solution

OpenArchitect 3

Document No. DC0147-02
November 9, 2004

Copyright Notice

ZNYX Networks
48421 Milmont Drive
Fremont, CA 94538 USA

Telephone: (510) 249-0800 or (800) 724-0911
Fax: (510) 656-2460
Email: sales@znyx.com
support@znyx.com
Web site: www.znyx.com

ZX5000 User Guide

Document # DC0147-02
November 9, 2004

© 2003-2004 ZNYX Networks. All rights reserved worldwide. Information in this document is subject to change without prior notice. ZNYX Networks, RAIN, RAINlink, OpenArchitect, CarrierClass and HotSwap are trademarks or registered trademarks of ZNYX Networks in the United States and/or other countries. All other marks, trademarks or service marks are the property of their respective owners.

ZNYX Networks may have patents, pending patent applications, trademarks, copyrights or other intellectual property rights covered in the subject matter of this document. By furnishing this document, ZNYX Networks does not license nor waive its license to those intellectual property rights except as expressly provided in a written license agreement from ZNYX Networks. Information in this document is subject to change without prior notice.

Table of Contents

About the ZX5000 Manual i
- Document Conventions i
- Technical Support i

Chapter 1 Overview of the ZX5000 Switch 1
- High Performance Embedded Switching 1
- Advanced TCA® Compliant 1
- OpenArchitect Switch Management 1
- Extensible Customization of Routing Policies 2
- Powerful CarrierClass Features 2
- OpenArchitect ZX5000 Models 3
- OpenArchitect Switch Environment 4
- OpenArchitect Software Structure 4
- Software Development Kit 6

Chapter 2 Installation 7
- Preparing for the Install 7
- Installing the ZX5000 Card 7
- Connecting the Cables 8
  - Connecting to the Console Port 8
- Out of Band Ports (OOB Ports) 8
  - Setting the Line Speed of the Out of Band Ports 8
- LED Reference 10

Chapter 3 Switch Configuration 12
- Connecting to the Switch Console 12
- OpenArchitect Configuration Procedure 12
  - Example Configuration Scripts 12
  - Overview of OpenArchitect VLAN Interfaces 13
- Layer 2 Switch Configuration 16
  - Spanning Tree 18
- Layer 3 Switch Configuration 21
  - Layer 3 Switch Using Multiple VLANs 24
- Layer 3 Routing Protocols with GateD 25
  - Using the Provided S55gatedRip1 Script 25
  - To Modify the GateD Scripts: 27
- Class of Service (COS) 28
  - Egress Queues 28
  - Ingress Classification 28
  - Marking and Re-marking 29
  - Traffic Management 29

Zcos Examples 30
FIFO Scheduling 30
Strict Priority 31
WRR Scheduling 32
zfilterd 33
tc and zqosd 38
FIFO Queues (pfifo and bfifo disciplines) 39
Random Early Detection (red discipline) 40
Priority & Weighted Round Robin Scheduling (prio discipline with wrr) 40
HTB Queuing Discipline 42
The U32 Filter 43
Translation Rules 43
Combining Queueing Disciplines 45
COPS: Common Open Policy Service 46
Protocol Architecture 47
OpenArchitect PEP 47
Using *pepd* 48
Chapter 4 High Availability Networking 49
RAINlink 49
Surviving Partner 50
VRRP 50
ZLMD 52
Switch Replacement and Reconfiguration 52
Configuring Surviving Partner 53
zspconfig 54
Example HA Chassis Configuration 57
Central Authority 61
Chapter 5 Switch Administration 63
Setting the Root Password 63
Adding Additional Users 63
Setting up a Default Route 63
Name Service Resolution 64
DHCP Client Configuration 64
DHCP Server Configuration 64
Network Time Protocol (NTP) Client Configuration 64
Network File System (NFS) Client Configuration 65
NFS Server Configuration 66
Connecting to the Switch Using FTP 66
FTPD Server Configuration 66
Connecting to the Switch Using TFTP 67
TFTPD Server Configuration 67

SNMP Agent 67
Supported MIBS 67
Supported Traps 68
SNMP and OpenArchitect Interface Definitions 69
SNMP Configuration 70
SNMP Applications 71
Port Mirroring 71
Link and LED Control 72
Link Event Monitoring 72
Web Interface (thttpd) 72
Starting thttpd 72
Restricting Access 73
Web Interface 73
Customizing the Web Interface 74
Chapter 6 Switch Maintenance 75
Overview of the OpenArchitect switch boot process 75
Saving Changes 77
Modifying Files and Updating the Switch 77
Recovering from a System Failure 77
System Boots with a Console Cable 77
Booting with the –i option 78
System Hangs During Boot 78
Booting the Duplicate Flash Image 79
Upgrading the OpenArchitect Image 79
Upgrading or Adding Files 80
Excluding Saving Files to Flash 80
Upgrading the Switch Driver 80
Using Apt-Get 81
Appendix A OpenArchitect Command Manual Pages 82
vrrpconfig 83
vrrpd 85
zal 88
zbootcfg 91
zconfig 93
zcos 100
zfilterd 103
zflash 104
zgmrpd 105
zgr 107
zgvrpd 110
zl2d 112

zl3d 114
zlc 116
zlmd 118
zlogrotate 120
zmirror 121
zmnt 124
zqosd 126
zrc 128
zreg 129
zrld 131
zspconfig 132
zstats 139
zsync 140
ztmd 142
brctl(8) 143
Appendix B PENDING Legal Notices and Certification 146
Canada Compliance 146
CE Conformity 146
Warranty 146
Appendix C Console Port Pinout 148
Appendix D Pinout of Front Panel Ethernet Ports 149
10/100 Management Port Pinout 149
1000Base T Port Pinout 150
Appendix E Rear Panel Pinouts 151
Table of pins for Zone 1 Connector 151
Table of pins for J23/P23 152
Table of pins for J24/P24 152
Appendix F IPMI 153
I^2C Bus Devices 153
Interrupt Pins 154
Appendix G ZNYX Private MIBs 155
ZNYX-H.MIB 155
ZX5000.MIB 161
Appendix H Networking References 164
Appendix I Glossary 165
Index 169

About the ZX5000 Manual

This manual includes everything you need to begin using the ZNYX Networks ZX5000 series switch with OpenArchitect software.

Document Conventions

Text in italics in this manual is used to reference other chapters or figures, or to refer to applications and commands (e.g., *Figure 1-1*, *zconfig*).

Programming syntax is displayed in courier font. Example output is also displayed in courier font.

Technical Support

If you need further assistance after reading this user manual, ZNYX Networks has a professional technical support team available to answer your questions. Contact us at:

Telephone:	(510) 249-0800 or (800) 724-0911
Fax:	(510) 656-2460
Email:	openarchitect@znyx.com or support@znyx.com

Chapter 1 Overview of the ZX5000 Switch

The ZX5000 with OpenArchitect™ is an extensible managed switch for embedded high-availability Ethernet in CarrierClass™ telecommunications systems. The switch provides Layer 2 switching and Layer 3 IP routing at line-rate, plus Layer 4-7 packet classification and filtering for the base fabric of an Advanced TCA™ standard chassis. The integrated OpenArchitect software environment provides a broad range of "out-of-the-box" protocols to establish extensible routing, control and management policies. As a requirement of Advanced TCA, the switch incorporates the PICMG 3.0 Intelligent Platform Management Interface (IPMI) standard for Field Replaceable Unit FRU) management by the Shelf Management Controller.

High Performance Embedded Switching

The ZX5000 with OpenArchitect combines the performance of a silicon-based switching fabric with flexibility of software-managed routing policies. The switching fabric component integrates 16 10/100/1000 ethernet ports to provide the following functions: a base fabric to base fabric port, 12 node slot ports routed to the ATCA base fabric backplane connectors and 3 front panel 10/100/1000 copper up links on the front panel. There are also two Out of Band (OOB) 10/100 ethernet ports available for management via the network. The switching fabric maintains the forwarding table on silicon, providing the capability to switch and route at full line rate performance on every port.

Advanced TCA® Compliant

The Advanced TCA® standard developed by the PCI Industrial Computer Manufacturer Group's subcommittee 3.0(PICMG 3.0) defines an embedded Ethernet environment for high availability chassis. This environment includes two switch fabric slots that create a dual star Ethernet network to the 12 base node slots. Placing the ZX5000 in a fabric slot provides embedded Ethernet services to each node card of the chassis. A standard configuration is one ZX5000 placed in each of the two fabric slots in a chassis for creation of a redundant, high availability system.

.

OpenArchitect Switch Management

The OpenArchitect software component – open source Linux, IP protocol stack, control applications and the OA Engine – runs on an embedded Motorola MCP8255 PowerPC microprocessor. OpenArchitect provides extensive managed IP routing protocols and other open standards for switch management. Examples include network services; Virtual Redundant Router Protocol; Routing Information Protocol; Open Shortest Path First; Border Gateway Protocol; IP Multicast, Quality of Service and Class of Service; access control lists; and management via Simple Network Management Protocol MIBs, Common Open Policy Services and web.

Extensible Customization of Routing Policies

The OpenArchitect software environment enables rapid porting of other UNIX/Linux-based protocols, including open source RFCs. It also enables the development of application-specific protocol configuration scripts. This extensibility and access to the open source networking software provides OEMs and integrators low cost, rapid development environment for creating new value-added services.

Powerful CarrierClass Features

The ZX5000 has High Availability hardware features for advanced telecommunication applications. The switch implements the PICMG 3.0 Full Hotswap support. This feature provides field replaceable capabilities where a switch can fail and be replaced without impacting the operational performance of a chassis.

Also supported is the new PICMG 3.0 Intelligent Platform Management Interface (IPMI) standard for message-based interfaces that monitor the physical health characteristics of the ZX5000. The switch provides operational status information to an IPMI management application. End customers benefit with advanced notice of potential problems.

The ZX5000 also implements the Media Dependent Interface called Auto MDI-X. Auto MDI-X allows connections to any device, switches, hubs, or systems using a regular straight-through or crossover Cat 5 cable. The RJ45 port will auto detect and switch MDI/MDI-X modes. This IEEE standard makes cabling – especially between switches – faster and less error prone.

OpenArchitect ZX5000 Models

Model	Function	Features
ZX5000 Switch	Linux-based Switch	Front panel Advanced TCA® base switch with 3-10/100/100/1000BaseT ports equipped with OpenArchitect software

You will find ZX5000 Series products are simple to install and configure. Included in this manual are the instructions to complete the following two basic steps before you can begin using the OpenArchitect switch:

Step 1 Installing the ZX5000.

Step 2 Configuring the switch to your networking application

OpenArchitect Switch Environment

The key elements of the OpenArchitect environment include an embedded Linux operating system, OpenArchitect-specific applications and libraries, plus, an innovative switch hardware design.

OpenArchitect hardware is in many ways similar to typical switch architectures. The primary difference in OpenArchitect is that the PCI bus that interfaces with the embedded processor to the switch fabric is at a higher performance level than a typical switch (*see Figure 1-1*). The use of PCI creates a pipe of significant bandwidth between the processor and the switch fabric.

The embedded processor, running Linux and the OpenArchitect processes, controls the flow of all traffic by maintaining the switch forwarding table. Essentially, this table is a set of instructions for the switch fabric to follow when directing the switch traffic. Due to the traffic forwarding decisions being made by dedicated logic, the packets proceed at line rate.

OpenArchitect Software Structure

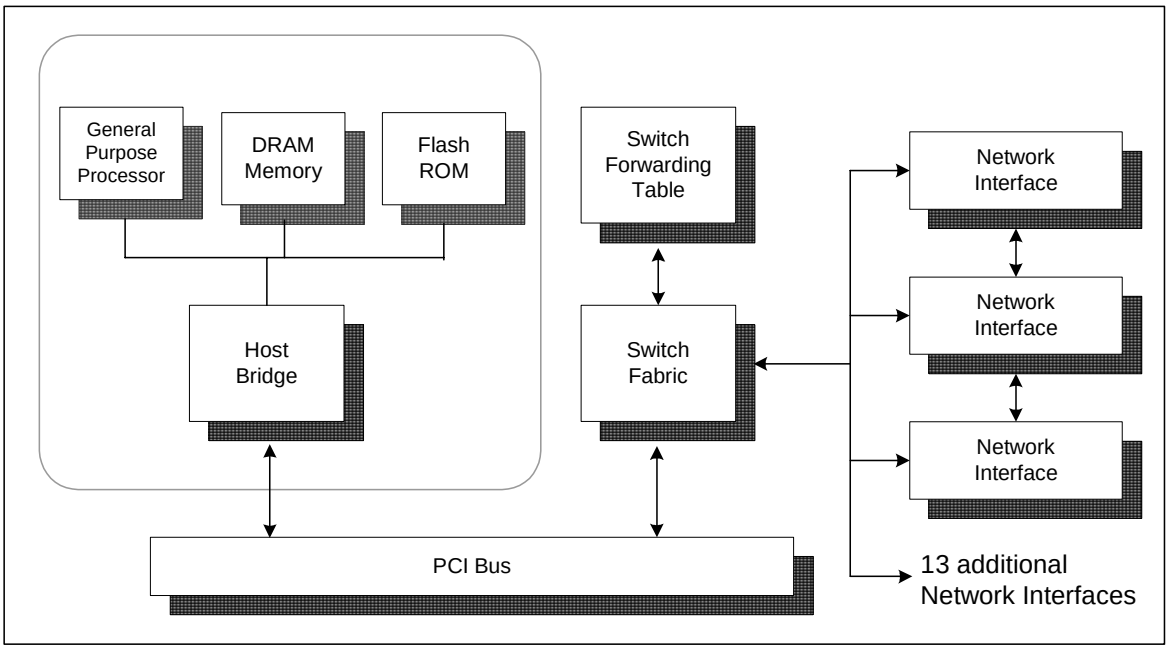


Figure 1-1 OpenArchitect Switch

OpenArchitect is based on an embedded Linux operating system and includes a number of ZNYX Networks-supplied modules. The key element is the Linux routing table, which is an object in a network-enabled Linux implementation.

The purpose of the routing table is to tell the packet forwarding software where to forward the data packets. In “standard” Linux, the packet-forwarding algorithm is operated in software. Normally, the routing tables are maintained by operator configuration and the various routing protocols that run in the application environment of Linux.

OpenArchitect uses an innovative new approach for forwarding packets. It provides embedded software daemons that ensure that the Linux routing tables are replicated (or shadowed) in a forwarding table (*see Figure 1-2*). The OpenArchitect silicon directly monitors the forwarding table and the switch fabric completes most of the “real-time” work in switching network packets. The switch fabric consults its own forwarding table for each incoming packet; and either filters or forwards the packet to any egress port, the embedded CPU, or to any combination.

The OpenArchitect environment includes additional features. For example, installing the OpenArchitect switch gives you immediate implementation of Linux routing protocols. Also, you have complete support of routing table updates and a standardized method for configuration. Finally, you can quickly integrate bug fixes, protocol enhancements and additional protocol implementations from the Linux community. You can also integrate OpenArchitect into other Linux applications including VPN software, voice over IP protocols, Quality of Service, and HTML configuration.

RAIN Management API (RMAPI) is a generic interface for passing control data and is implemented across the majority of ZNYX Networks products. The OpenArchitect libraries are implemented completely above RMAPI. The libraries provide a front-end to RMAPI to simplify application writing. Currently two libraries are implemented, a general library called *zlxlib*, and a table library called *ztlib*. As the OpenArchitect application requirements grow, the existing libraries will be expanded and additional libraries will be created.

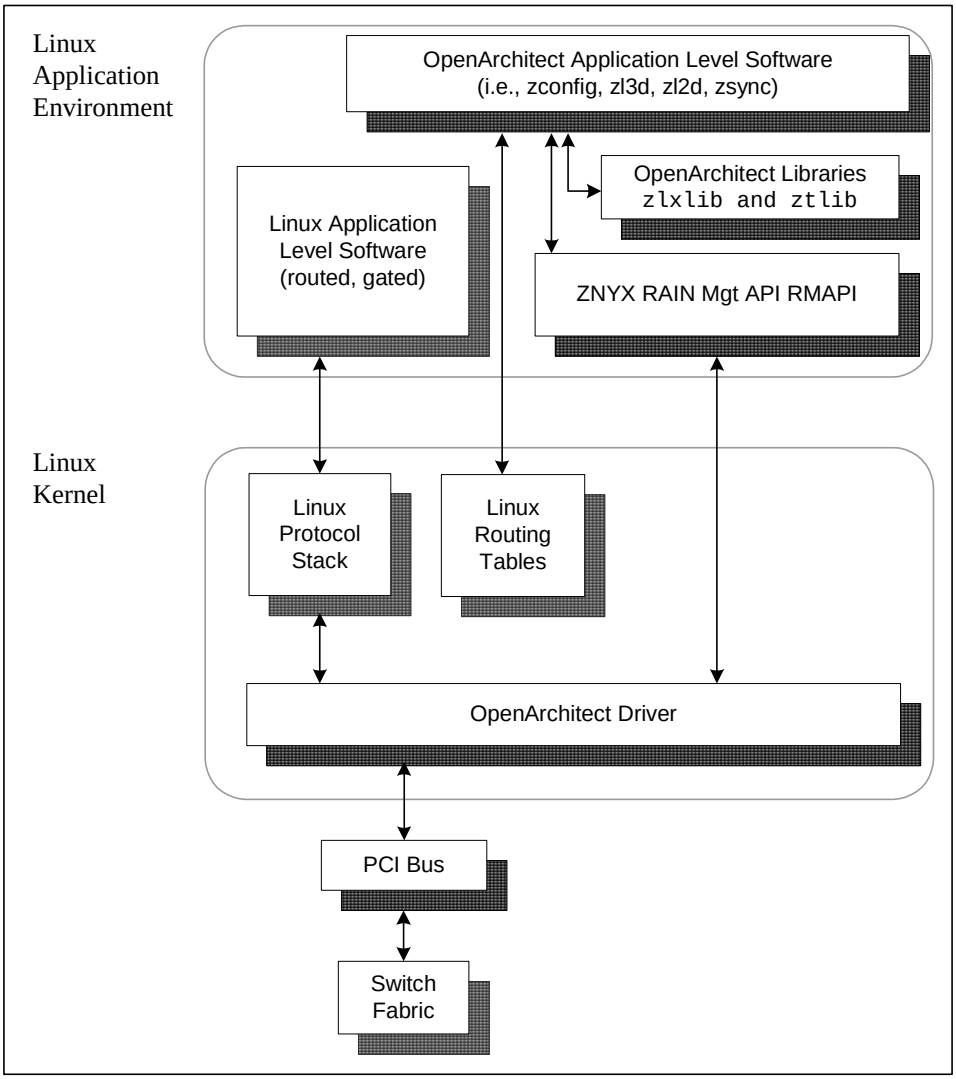


Figure 1-2 OpenArchitect Software Structure

OpenArchitect applications are used to program and configure ZNYX Networks switches. These applications are implemented above the libraries and RMAPI.

Software Development Kit

ZNYX Networks offers an additional Linux-based software development kit for engineering teams to build their own OpenArchitect binaries using the standard Linux development environment. Thus, you can customize to include any desired set of Linux programs, including proprietary value-added software. Contact sales at ZNYX Networks for further information.

Chapter 2 Installation

The Advanced TCA® standard developed by the PCI Industrial Computer Manufacturer's Group subcommittee 3.0 (PICMG 3.0) defines an embedded base Ethernet switch environment for high availability chassis. This environment includes two switch fabric slots that create a dual star Ethernet network to the twelve node slots. Placing the ZX5000 in a fabric slot provides embedded Ethernet services to each node card across the base fabric of the chassis. A standard configuration is to place a ZX5000 in each fabric slot creating a redundant, high availability system.

Preparing for the Install

Refer to your specific system documentation for additional installation reference.

- **You do not need to power off your system**. The ZX5000 is PICMG 3.0 hot swap compliant, however we do recommend powering off the system when inserting units marked as "EARLY ACCESS".

- **Discharge any static electricity** from your body by touching the metal chassis, or by using an anti-static wrist strap. If you do not have a ground strap, maintain physical contact with the case to maintain the same electrical potential with the system.

Installing the ZX5000 Card

1. Select an available fabric slot on your Advanced TCA® system and remove the slot cover from the front panel.

2. Carefully insert the ZX5000 into the slot. Be sure you push the card straight into the slot. The card and the slot must be properly aligned and the card fully seated. A card is seated properly when it touches the system case up to the extraction levers.

Push the extraction levers towards one another to finish seating the switch and secure the retention screws.

Connecting the Cables

Your switch setup may require some or all of the following three types of cables:

10/100/1000 Port Cabling

Use Category 5e cabling for all 10/100/1000 connections. Be sure that your cable length is within the minimum and maximum length restrictions for the Ethernet, otherwise you could experience signal or data loss. All egress ports on the ZX5000 are auto-MDI sensing and will automatically determine whether or not an MDI (straight-through) or MDI-X (crossover) cable is attached.

Console Port Cabling

The switch console can be accessed via an RJ45 service port located on the front panel of the ZX5000. It may be used to recover from a system failure. Use a ZNYX Networks console cable to access the switch software through the console port. Contact your ZNYX sales representative for details or use the pin out specified in Appendix C to build your own.

Connecting to the Console Port

To attach the console cable to the OpenArchitect switch:

1. Plug the RJ45 end of the console cable into the RJ45 Console Port on the front panel
2. Connect the 9-pin header on the cable to a standard COM port (9600, n, 8, 1). Reinsert the switch into the system and power up.
3. Use a terminal emulation program to access the switch console.

Out of Band Ports (OOB Ports)

Two 10/100 ethernet Out of Band Ports (OOB ports) are included for network access to manage the switch. One OOB port is available via the RJ-45 connection on the front panel labeled port 18. The other OOB port is connected to the Shelf Management Controller (ShMC). The 10/100 port located on the front panel is identified as eth0 and OOB port routed to the ShMC is identified as eth1 by OpenArchitect. Use *ifconfig* to bring up and configure the OOB ports.

Setting the Line Speed of the Out of Band Ports

To force the line speed of either OOB port, use the media option of *ifconfig*. For example, to force 100 Mbits/sec, full duplex for the port on the front panel type,

```
ifconfig eth0 media 100fd
```

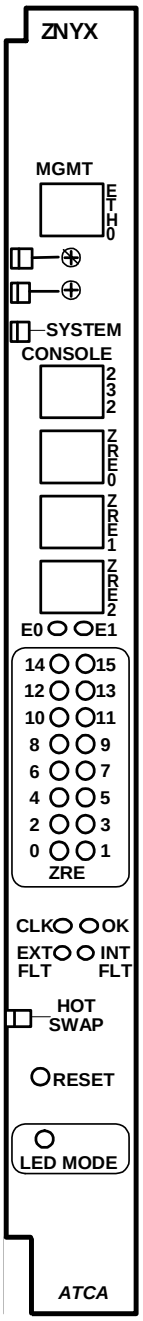
The following table shows the available media parameters for the OOB line speed settings.

Line Speed	Media Parameter
Autonegotiate	auto
100 Full Duplex	100fd
100 Half Duplex	100hd
10 Full Duplex	10fd
10 Half Duplex	10hd

By default both OOB ports autonegotiate.

LED Reference

The following table describes the front panel LEDs and switches



LED	Function	
HEALTHY ⊕	Off	Not Powered and 'No Attention' status
	Amber	Powered or not powered and 'Attention' status. 'Attention' status is indicated by any fault condition of the CLOCK, OK, or INT FLT LEDs
	Green	Powered and 'Not Attention' status
HOT SWAP	Blue	After the extraction levers are released (pulled apart), ON indicates the switch unit is ready to remove. During “hot” insertion (system is active), Hot Swap LED is ON and turns OFF when the switch is correctly seated. OFF during power
Out of Service ⊛	Off Amber	To be implemented per ATCA specification
INT FLT	Amber	ON indicates a failure of the internal tests. OFF during power up
EXT FLT	Amber	ON indicates a failure external to the adapter (e.g., inability to establish a link on a configured port or another connectivity problem external to the adapter).
OK	Green	ON indicates the initialization has completed successfully, and at least one interface is configured up. OFF during power up.
CLK	Green	FLASHING once the driver has successfully loaded. OFF during power up.
LED MODE Switch		Change the Port LEDs display function from “Link /Activity” mode (default) to “Link /Speed” mode.
RESET Switch		Pressing RESET for at least three seconds restarts the switch.

Port LEDs 0-15 (10/100/1000) LED 15 is the fabric link	Off	Link/Activity Mode (default) No link Link/Speed Mode (LINK/SPEED Switch depressed) OFF indicates 10 Mbps connection
	Green	Link/Activity Mode (default) ON indicates link BLINKING indicates traffic Link/Speed Mode (LINK/SPEED Switch depressed) ON indicates 100 Mbps connection
E1 – eth1 (10/100 OOB)		
E0 – eth0 (10/100 OOB on front panel)	Amber	Link/Activity Mode (default) Port is not forwarding packets; disabled by management or blocked by Spanning Tree Protocol Link/Speed Mode (LINK/SPEED Switch depressed) ON indicates 1000 Mbps connection

Chapter 3 Switch Configuration

At this point, the OpenArchitect switch should be installed and powered up for the first time. This chapter helps you connect to the switch and configure the switch for your networking environment by presenting command line examples as well as a discussion of the ZNYX supplied example configuration scripts.

Connecting to the Switch Console

You can connect to the switch console using a *telnet* connection or with a console cable. Use the procedure below for a *telnet* connection. See Chapter 2, Installation, for instructions to connecting to the console port.

1. Connect an Ethernet cable to the host and the switch.
2. Configure a host on the 10.90.90.0 network.
3. The OpenArchitect switch is pre-configured with address 10.90.90.90. Telnet to 10.90.90.90.

```
telnet 10.90.90.90
```

After you are connected, enter the login name `root`. No password is required.

```
ZX5000 login: root
sh-2.04#
```

OpenArchitect Configuration Procedure

Layer 2 and Layer 3 switch configurations can be accomplished with a few simple commands. Once you've configured your switch, the commands should be placed into a start up configuration script. Like most Linux systems, the OpenArchitect switch boot process runs initialization commands and scripts in `/etc/init.d/`. In particular, OpenArchitect runs `/etc/init.d/rcS` which in turn executes all scripts located in `/etc/rcZ.d` starting with an uppercase "S" in alphabetical order. Any configuration scripts you create should be named in the standard Linux/Unix manner, starting with an uppercase "S" and numbered in the sequence you would like them executed. The final step once the switch has been properly configured is to use the *zsync* command to save all files into flash for reloading.

Example Configuration Scripts

ZNYX supplies example scripts that can be used as templates. Use one of the scripts located in the switch `/etc/rcZ.d/examples` directory to help you configure the switch. The default configuration for the switch is located in the script file `/etc/rcZ.d/S50layer2`.

The following scripts are included. Each is examined in more detail later in the appropriate section describing common Layer 2 and Layer 3 configurations.

`S50Layer2`	Script which sets up a basic Layer 2 switch. All 16 10/100/1000 ports are set up on one IP network (VLAN).
`S50Layer2sp`	Script which sets up a basic Layer 2 switch. All 16 10/100/1000 ports are set up on one IP network (VLAN), and turns on bridge support for Spanning Tree.
`S50Layer3`	Script which sets up a basic Layer 3 switch. Each of the 16 10/100/1000 ports are set up on individual IP networks (VLANs). Layer 3 switching is enabled.
`S50multivlan`	Script which sets up multiple untagged VLANs. The first VLAN includes the zre3-zre9 10/100/1000 ports, the next contains the last zre10-zre1510/100/1000 ports, and one for zre0, one for zre1 and one for zre2, front panel ports,. Layer 3 switching is enabled.
`S55gatedRip1`	Script which is used with a Layer 3 switch and calls the GateD daemon to enable RIP 1 routing protocol.
`S55gatedRip2`	Script which is used with a Layer 3 switch and calls the GateD daemon to enable RIP 2 routing protocol.
`S55gatedOspf`	Script which is used with a Layer 3 switch and calls the GateD daemon to enable OSPF routing protocol.

Overview of OpenArchitect VLAN Interfaces

When you initially boot up the switch, one virtual host port is automatically created by OpenArchitect to enable interaction between the software and hardware. This initial host port, called ZNYX Host Port (*zhp*), is a network interface that provides communication between all 16 ports. Therefore, linking to any port on the switch enables you to connect with OpenArchitect.

A *zhp* device is associated with one Virtual Local Area Network (VLAN). A virtual local area network (VLAN) is a logical mapping of workstations and network devices on some basis other than geographic location (e.g., by department, type of user, or primary application). The primary purpose of a VLAN is to isolate traffic and enable communication to flow more efficiently within groups of mutual interest. VLANs reduce the time it takes to implement workstation and network moves, adds and changes. The switch is used to bridge from one VLAN to another. Figure 3-1 Multiple VLANs is an example of a hypothetical company that occupies two buildings to illustrate multiple VLANs in a possible VLAN network structure.

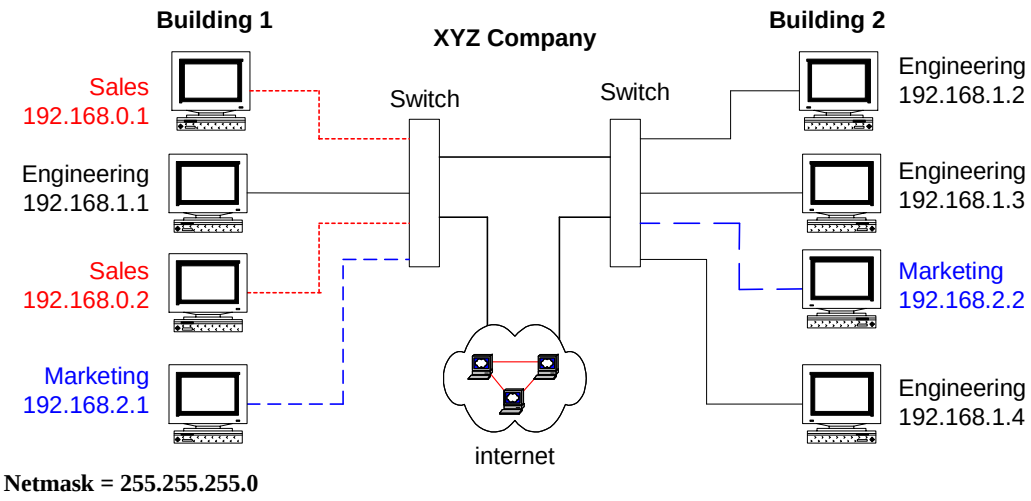


Figure 3-1 Multiple VLANs

In the diagram above, three VLAN's are utilized and divided as follows:

VLAN	Group	IPNetwork	Interface
100	Sales	192.168.0.0	zhp0
200	Engineering	192.168.1.0	zhp1
300	Marketing	192.168.2.0	zhp2

In the example, the sales employees are all located in Building 1. They are combined into VLAN 100 (zhp0), on the 192.168.0.0 IP network, because they interact among customers and each other on a daily basis. The engineering and marketing groups have employees working in both Buildings 1 and 2. Each of these groups is also separated into different VLANs (200 and 300, respectively) and in different subnets.

Outside communication into the company can also be placed on a separate VLAN to maintain control of outside traffic (e.g., security reasons).

Tagging and Untagging VLANs

The OpenArchitect switch is capable of switching tagged and untagged data packets. Tagged packets are labeled in the data header for switching to a specific VLAN by a switch port. Untagged packets enable all untagged packets for a specific VLAN to go through a port.

In the Figure 3-1 example, all data packets in VLAN 200 and 300 would be tagged so that correspondence between the groups in these VLANs would be routed to the correct workstations.

Switch Port Interfaces

For each switch port, OpenArchitect creates a separate interface with its own MAC address called a ZNYX raw Ethernet (*zre*). After the initial power up, 16 *zre* interfaces are created, one for each inband port. You cannot directly access or modify the *zre* interfaces.

During the initial power up of the switch, the default configuration creates a Layer 2 switch. The Layer 2 configuration places all of the *zre* interfaces in the same *zhp* interface. The number after *zre* represents the corresponding switch port number (i.e., *zre1* represents port 1 on the switch).

Layer 2 Switch Configuration

The steps to build a Layer 2 switch involve creating a group of switch ports in a VLAN (or Layer 2 switching domain) and bringing that interface up. *zconfig* creates the VLAN group of switch ports as well as a network interface. Use *ifconfig(1M*) on the network interface to bring up the VLAN group.

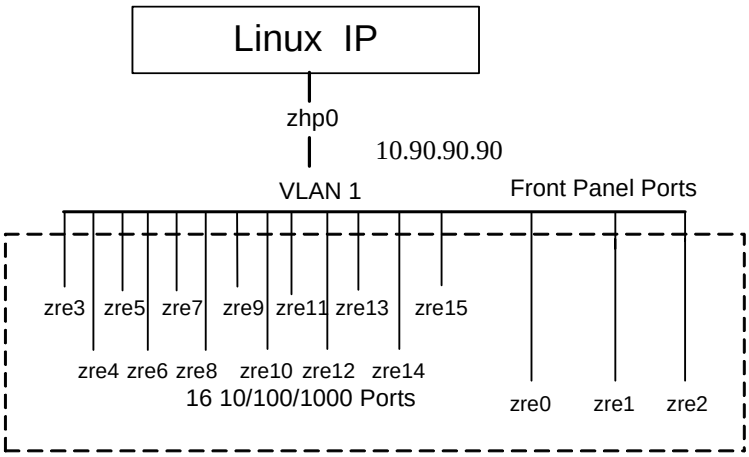


Figure 3-2 Layer 2 Switch

During the initial power up, a startup script called `/etc/rcZ.d/S50layer2` is executed at boot time creating a single untagged VLAN (IP interface labeled as *`zhp0`*) which includes all ethernet and gigabit ports as one Layer2 switch *(see Figure 3-2)*. The interface to the host is then assigned the IP address of 10.90.90.90 to allow access to the switch. The `S50layer2` script does the following:

- Uses *zconfig* to create and configure a single, untagged VLAN that contains all 16 switch ports.

```
/usr/sbin/zconfig zhp0: vlan1=zre0..15
/usr/sbin/zconfig zre0..15=untag1
```

- Uses *ifconfig(1M*) to assign the IP address 10.90.90.90 to the interface.

```
/usr/sbin/ifconfig zhp0 10.90.90.90 up
```

If you wished to create another VLAN that only contained the three front panel ports, first use *zconfig* from the command to build the VLAN and create a network interface for the host.

```
zconfig zhp1: vlan2=zre0..2
```

Then, bring up the interface with *ifconfig(1M)*:

```
ifconfig zhp1 192.168.1.1 up
```

Note that ports *zre0, zre1* and *zre2* are members of both vlan1 and vlan2, and that they are tagged for vlan2. A port cannot be untagged for more than one VLAN. You can view the configured VLANs with *zconfig.*

```
zconfig -a
```

Using the Provided S50layer2 Script

The `S50layer2` script can be used used as an example or edited to customize your Layer2 setup. For example, to reconfigure the IP address on your Layer 2 switch**,**

- Open the `S50Layer2` file in the Linux *vi* editor.
- Change the IP address value listed under the Linux *ifconfig(1M)* command line.
- Save your changes by running OpenArchitect *zsync*.

```
zsync
```

- Reboot the switch.

Spanning Tree

The Spanning Tree Protocol (STP) configures a simply connected active topology from the arbitrarily connected components of a Bridged Local Area Network. STP participants use a simple dialog carried in packets called Bridge Protocol Data Units (BPDUs) for finding the shortest path between two networks and for eliminating loops from the topology. If nodes attached to ports fail or are added or deleted, the topology dynamically changes to accommodate the new configuration. If your network topology is such that there is no real redundancy or chance for loops, you do not need to turn on Spanning Tree.

zl2d is used to create Linux bridges consisting of the name of the previously created *zhp* device or devices preceded with a "b" (e.g. if you are creating a Bridge device from *zhp0*, the resulting device would be *bzhp0*). *zl2d* then starts a background task that monitors the port information of the Linux bridge at a specified interval and updates the Spanning Tree state fields in the hardware when necessary.

brctl(8) is used for configuring certain STP parameters. For an explanation of these parameters, see the IEEE 802.1d specification, or reference the *brctl(8)* man page in Appendix A. The following demonstrates a simple example of setting up a Layer 2 switch and starting STP.

To Enable Spanning Tree:

Create a VLAN containing the ports that will be a part of the Linux bridge running Spanning Tree. This example will use ports 1-4 (untagged):

```
zconfig zhp0: vlan1=zre1..4
zconfig zre1..4=untag1
```

Create a bridge device from the zhp device,

```
zl2d start zhp0
```

A Bridge device named *bzhp0* should now exist consisting of ports *zre1* through *zre4* with Spanning Tree enabled. To view the bridge device, use the *brctl* command,

```
brctl show
brctl showbr bzhp0
```

Port Path Cost

Each port has an associated cost that contributes to the total cost of the path to the Root Bridge when the port is the root port. The smaller the cost, the better the path. The ZX5000 uses the following IEEE 802.1D recommendations based on the connection speed of your port:

Link Speed	Recommended Value	Recommended Range
10 Mb/s	100	50-600
100 Mb/s	19	10-60
1 Gb/s	4	3-10

To change the port path, use the *brctl setpathcost* option. For example, to set the port priority to a value consistent with a gigabit interface,

```
brctl setpathcost bzhp0 zre1 4
```

Fast Forward Port

Fast Forward Port alters the STP algorithm to expedite Spanning Tree recovery by bypassing the Listening and Learning states of STP, and transitioning directly to Forwarding. This can reduce STP recovery time from potentially 50 seconds down to less than 1 second. Fast Forward Port is implemented on a per port basis. Fast Forward port should ONLY be enabled on ports connected to end-stations. Do not enable Fast Forward port on ports that can result in the creation of loops, such as network devices with multiple connections.

To enable Fast Forward Port, setup a bridge device with Spanning Tree enabled, then use *brctl(8)* to enable Fast Forward Port for the specific port on a specific bridge,

```
brctl ffport bzhp0 zre1 1
```

Fast Forward Bridge

Fast Forward Bridge improves on the STP algorithm providing quicker convergence when STP picks a new Root Port. If Fast Forward Bridge is enabled for the bridge, whenever STP picks a blocked port as the new Root Port, that port bypasses the Listening and Learning states and transition immediately to Forwarding. By avoiding the Listening and Learning states and transitioning directly to Forwarding on the new Root port, the time for STP convergence is reduced from potentially 50 seconds down to less that 1 second.

To enable Fast Forward Bridge, setup a bridge device with Spanning Tree enabled, then use *brctl* to enable Fast Forward Bridge for that bridge device,

```
brctl ffbridge bzhp0 1
```

Using the Provided S50layer2sp Script

The `S50layer2sp` script can be used as an example or edited to customize your Layer2 setup, including Spanning Tree. For example, to reconfigure the IP address on your Layer 2 switch**,**

- Copy `/etc/rcZ.d/examples/S50layer2sp` to `/etc/rcZ.d/S50layer2sp`.

```
cp /etc/rcZ.d/examples/S50layer2sp /etc/rcZ.d
```

- Remove the existing `S50layer2` script (a copy exists in the examples directory):

```
rm /etc/rcZ.d/S50layer2
```

- Open the `S50Layer2sp` file in the Linux *vi* editor, and make the changes necessary for your configuration. Save your changes by running *zsync*.

```
zsync
```

- Reboot the switch.

Layer 3 Switch Configuration

The previous section outlines the Layer 2 switch configuration that is automatically configured when you initially bring up the OpenArchitect switch. In order to communicate between Layer2 interfaces, you must properly setup routing.

The steps to build a Layer 2 switch involve creating a group of switch ports in a VLAN (or Layer 2 switching domain) and bringing that interface up. *zconfig* creates the VLAN group of switch ports as well as a network interface. Use *ifconfig(1M*) on the network interface to bring up the VLAN group with Layer 2 switching. Layer3 routing information is then used to route between the Layer2 network devices.

Take a simple example of two VLANs configured on the switch, each with four ports. First teardown any existing configuration,

```
zconfig -t
```

Use *zconfig* to create two new VLANs, each with four ports, and untag them,

```
zconfig zhp0: vlan1=zre1..4
zconfig zre1..4=untag1
zconfig zhp1: vlan2=zre5..8
zconfig zre5..8=untag2
```

Now, use *ifconfig* to assign each *zhp* interface an IP address,

```
ifconfig zhp0 10.0.0.1
ifconfig zhp1 11.0.0.1
```

At this point, the Linux host has enough information to route between the networks of the directly attached interfaces, 10.0.0.0 via *zhp0*, and 11.0.0.0 via *zhp1*.

The next step is to enable the ZNYX *zl3d* daemon to move that routing information from the host to the ZX5000 switching tables in silicon. Once enabled, *zl3d* will monitor the Linux routing tables for changes in configuration and update the switch silicon tables. Start *zl3d* to update the switch tables:

```
zl3d zhp0 zhp1
```

The ZX5000 switch is now configured as a Layer3 switch that can route between two Layer2 devices in silicon.

Using the Provided S50layer3 Script

To modify the configuration to a Layer 3 switch, remove the `S50layer2` file from the `/etc/rcZ.d` directory, and replace it with the example script file, `S50layer3`.

In the `S50layer3` file, each 10/100/1000 port is assigned its own Virtual Local Area Network (VLAN) interface (port interfaces are labeled as *zhpN*, where `N` is an integer). Each VLAN is associated with an individual *zhp* interface. Remember, *zre* and *zhp* interfaces can begin with a zero value but a VLAN cannot. Each *zre* interface is assigned a separate IP address in the example script. (see).

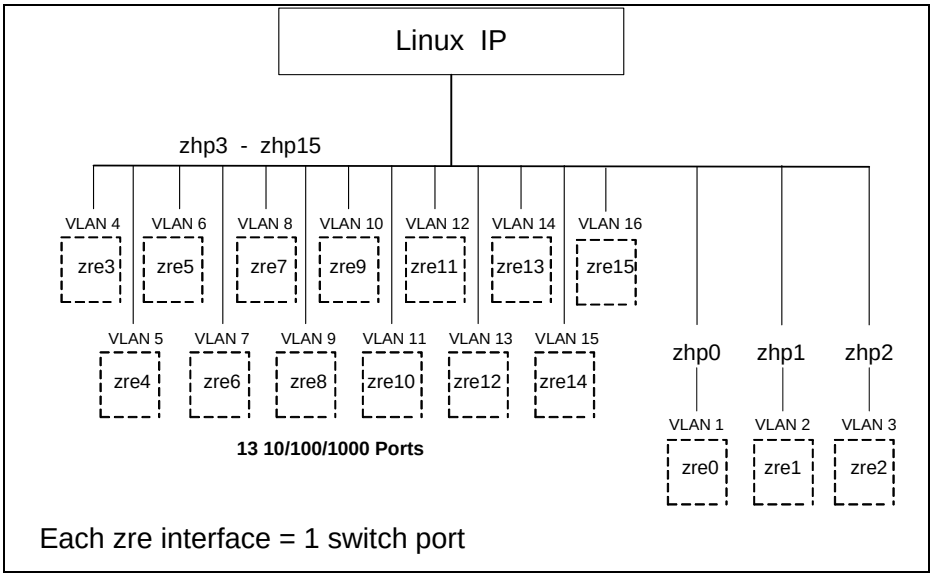


Figure 3-2 Layer 3

The S50layer3 script executes the following commands:

- Runs *zconfig* command to create 16 untagged VLANs (one for each switch port).

```
/usr/sbin/zconfig zhp0..15: vlan1..16=zre0+
/usr/sbin/zconfig zre0..15=untag1+
```

NOTE : Double periods (..) after vlan1 and untag1 are used to indicate a range of values. The plus (+) sign after zre0 is a wildcard character that means auto-incremented and causes each *zhp* interface to hold only one *zre* (i.e., zhp0 has zre0 on vlan1, zhp1 has zre1 on vlan2).

- Runs the Linux *ifconfig(1M)* command for each interface to assign default IP addresses (10.0.0.42-10.0.15.42), sets the netmask and brings up the interfaces.

```
ifconfig zhp0 10.0.00.42 netmask 255.255.255.0 up
ifconfig zhp1 10.0.01.42 netmask 255.255.255.0 up
ifconfig zhp2 10.0.02.42 netmask 255.255.255.0 up
.
.
.
ifconfig zhp13 10.0.13.42 netmask 255.255.255.0 up
ifconfig zhp14 10.0.14.42 netmask 255.255.255.0 up
```

```
ifconfig zhp15 10.0.15.42 netmask 255.255.255.0 up
```

- Runs the OpenArchitect *zl3d*. The *zl3d* application monitors the Linux routing tables and updates the switch routing tables for each interface configured above.

  ```
  /usr/sbin/zl3d zhp0..15
  ```

  zl3d initially creates and adds each zhp interface (VLAN) to the switch routing tables. The `zhp0..zhp15` is shorthand for the list of interfaces (zhp0, zhp1, …, zhp15) to monitor with *zl3d*.

To Modify the Layer 3 Script

- Modify the example script you copied into the `/etc/rcZ.d` directory. Adjust and assign the number of IP addresses as applicable. In the example below, the IP address is changed for the interface in the *ifconfig* command line of the script.

  From:

  ```
  ifconfig zhp0 10.0.0.42 netmask 255.255.255.0 broadcast 10.0.0.255 up
  ```

  To:

  ```
  ifconfig zhp0 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
     up
  ```

- Adjust the number of `zhp` interfaces, that are added to the routing tables, depending on the number of VLANs you are adding for your network. Include any other details, as applicable.

- Run the OpenArchitect *zsync* command to save your changes.

  ```
  zsync
  ```

- Reboot the switch.

- After rebooting, your switch works from your customized Layer 3 configuration.

An example script is also provided for setting up multiple VLANs each with multiple ports.

Using the Provided S50multivlan Script

The Layer 3 switch example file, S50multivlan, is included to help you configure multiple VLANs to a Layer 3 switch. A VLAN can include one or more switch ports. In the `S50multivlan` file, three VLANs are created:

- VLAN 1, zhp0: port 0 and ports 3 through 8
- VLAN 2, zhp1: port 1 and ports 9 through 14
- VLAN 3, zhp2: port 2 and port 15

Each VLAN interface is labeled `zhpN` in the file, where `N` is a value from 0-2. Each interface is untagged and assigned its own IP address.

The S50multivlan script executes the following commands:

- Runs *zconfig* to create and start three VLANs. Switch ports 0 and 3 - 8 are placed in the first VLAN, ports 1 and 9-14 in the second VLAN, ports 2 and 15 are placed in the third VLAN.

```
/usr/sbin/zconfig zhp0: vlan1=zre0,zre3..8
/usr/sbin/zconfig zre0,zre3..8=untag1
/usr/sbin/zconfig zhp1: vlan2=zre1,zre9..14
/usr/sbin/zconfig zre1,zre9..14=untag2
/usr/sbin/zconfig zhp2: vlan3=zre2,zre15
/usr/sbin/zconfig zre2,zre15=untag3
```

NOTE Double periods (..) after vlan1 and untag1 are used to indicate a range of values

- Runs the Linux *ifconfig* command for each interface to assign default IP addresses (10.0.0.42-10.0.2.42), assigns the netmask and brings them up.

```
ifconfig zhp0 10.0.0.42 netmask 255.255.255.0 broadcast 10.0.0.255 up
ifconfig zhp1 10.0.1.42 netmask 255.255.255.0 broadcast 10.0.1.255 up
ifconfig zhp2 10.0.2.42 netmask 255.255.255.0 broadcast 10.0.2.255 up
```

- Runs the OpenArchitect *zl3d* command. The *zl3d* application monitors the Linux routing tables and updates the switch routing tables for each interface configured above.

```
/usr/sbin/zl3d zhp0 zhp1 zhp2
```

To Modify the Layer 3 Multivlan Script

- Modify the example script you copied into the `/etc/rcZ.d` directory. Adjust and assign the number of IP addresses as applicable. In the example below, the IP address is changed for the interface in the *ifconfig* command line of the script.

```
ifconfig zhp0 10.0.0.42 netmask 255.255.255.0 up
ifconfig zhp0 192.168.1.1 netmask 255.255.255.0 up
```

- Adjust the number of *zhp* interfaces depending on the number of VLANs you are adding for your network.

- Include any other details, as applicable.

- Run the OpenArchitect *zsync* command to save your changes.

```
zsync
```

- Reboot the switch.

- After rebooting, your switch works from your customized Layer 3 configuration with multiple VLANs per port.

Layer 3 Routing Protocols with GateD

An advanced networking configuration may require using the GateD software platform for deployment of Routing Information Protocols (RIP 1 or RIP 2) and Open Shortest Path First (OSPF) protocols. Once you've configured your Layer2 and Layer3 devices, start *gated*.

Using the Provided S55gatedRip1 Script

To use GateD protocol with the switch, you need to copy two files into the same directory as your Layer 3 configuration file. From the `/etc/rcZ.d/examples` folder, copy the example script file and its corresponding GateD configuration file (e.g., `S55gatedRip1` and `gated.conf.rip1`).

The example startup script executes the following commands (S55gatedRip1 is used as an example):

- Starts GateD with Rip1 (2 separate files are required):

```
/usr/sbin/gated –f /etc/rcZ.d/gated.conf.rip1

/usr/sbin/gated
   finds and starts the GateD application.

-f gets the configuration information from the specified
   file.
```

```
/etc/rcZ.d/gated.conf.rip1
```

Example gated configuration file for RIP1

The GateD conf file specifies the following configuration commands:

- Implements the passive function so GateD is prevented from rerouting information to a different interface if insufficient information is received.

```
interface 10.0.0.42 passive
interface 10.0.1.42 passive
interface 10.0.2.42 passive
.
.
.
interface 10.0.13.42 passive
interface 10.0.14.42 passive
interface 10.0.15.42 passive
```

- Defines the netmask used in the interface.

```
define 10.0.0.42 netmask 255.255.255.0;
define 10.0.1.42 netmask 255.255.255.0;
define 10.0.2.42 netmask 255.255.255.0;
.
.
.
define 10.0.13.42 netmask 255.255.255.0;
define 10.0.14.42 netmask 255.255.255.0;
define 10.0.15.42 netmask 255.255.255.0;
```

- Sets the RIP1 protocol to open.

```
};
rip1 yes{
```

- Shuts off sending and receiving packets from all interfaces.

```
interface all noripin noripout
```

- Opens sending and receiving packets for selected interfaces.

```
interface 10.0.0.42 ripin ripout version 1;
interface 10.0.1.42 ripin ripout version 1;
interface 10.0.2.42 ripin ripout version 1;
.
.
.
interface 10.0.13.42 ripin ripout version 1;
interface 10.0.14.42 ripin ripout version 1;
```

```
interface 10.0.15.42 ripin ripout version 1;
```

- Imports routes learned through the RIP protocol.

```
import proto rip {
        all;
};
```

- Exports all directly connected routes and routes learned from the RIP protocol.

```
export proto rip {
       proto direct }
        all;
};
proto rip {
      all;
};
```

To Modify the GateD Scripts:

Copy two GateD files, the OpenArchitect "S" file and its corresponding `conf` file, into the `rcZ.d` directory (i.e., `S55gatedRip1` and `gated.conf.rip1`). Notice the files are placed in the same directory as the Layer 3 configuration file.

For RIP1:

```
cp /etc/rcZ.d/examples/S55gatedRip1 /etc/rcZ.d
cp /etc/rcZ.d/examples/gated.conf.rip1 /etc/rcZ.d
```

Or for RIP2:

```
cp /etc/rcZ.d/examples/S55gatedRip2 /etc/rcZ.d
cp /etc/rcZ.d/examples/gated.conf.rip2 /etc/rcZ.d
```

Or for OSPF:

```
cp /etc/rcZ.d/examples/S55gatedOspf /etc/rcZ.d
cp /etc/rcZ.d/examples/gated.conf.ospf /etc/rcZ.d
```

- Open and make configuration changes to the listed `conf` file to coincide with the current Layer 3 configuration (i.e., adjust IP addresses and number of interfaces available). See GateD documentation if you have questions regarding the `conf` file.

- Run the OpenArchitect zsync command to save your changes. Be sure your changes are correct.

```
zsync
```

- Reboot the switch. After rebooting, your switch operates as a Layer 3 switch with GateD routing.

Class of Service (COS)

This following section provides information on using the ZNYX Networks OpenArchitect switch to provide Class of Service (COS) support. The switching fabric architecture defines the scope of the COS parameters. Some apply to an individual port and others apply to the whole switch. It is important for the user to understand the scope of the parameters to ensure that the expected behavior occurs.

Egress Queues

The ZX5000 provides 8 COS queues per egress port, and for packets destined to the CPU from the switching fabric. By default, a freshly booted OpenArchitect switch uses a single queue per egress port (and the CPU). Service for packets in the queues is performed on a strict priority basis. That is, all packets in higher priority queues will be transmitted before any packets in a lower priority queue. Queue 7 has the highest priority. It is possible, however, to more fairly share the bandwidth of the egress port among packets destined for all queues by controlling the rate at which packets are added to these queues.

Ingress Classification

Incoming packets are mapped to queues based on their queue priority. The priority can be set in a number of different places. The default behavior of the ZX5000 is to disable priorities. All packets will go to queue 0. Enabling a given priority is a two step process. Each priority source (except for the classifier) has mapping registers which specify the mapping from the incoming priority source codes to the eight queues. Once the mapping is defined, the precedence of each priority source must be defined, to resolve conflicts when more than one priority source is present. Separate precedence registers are used for switched (L2) and routed (L3) packets. The OpenArchitect system includes a utility, *zcos* to set up these registers. The priority sources are:

For tagged packets, the incoming 802.1p user priority

For untagged packets, the default source VLAN priority for a port

For IP packets, the TOS, the full 6 bit DiffServ field, which includes DTR bits and IP precedence.

Priority defined by a matched Classifier rule

Destination MAC address priority from the L2 table, for addresses looked up in that table (*Not Supported by OA*)

For Unicast IP packets that are routed, the L3 table contains a priority (*Not Currently Supported by OA*)

For IP Multicast packets that are routed, the IP multicast table contains a priority (*Not Currently Supported by OA*)

Classifier rules can be set up using the Linux *iptables* or tc utilities and the Znyx *zfilterd, zqosd, and ztmd* facilities. This allows the queue selection to be based on the type of packet, source or destination IP address, TCP or UDP ports or other information from the packet header. The default OpenArchitect switch behavior has all COS values mapping to a single queue on each of the egress ports.

Marking and Re-marking

The OpenArchitect switch can mark or remark packets using the TOS field or 802.1p tag. For the 802.1p tag, a second set of mapping registers are provided to determine the value inserted in the header of tagged packets on egress. This is also controlled through the *zcos* utility. The TOS field, including the 6-bit DiffServ Code Point, can be set as a result of a classifier match, which can be controlled using the Linux *iptables* utility.

Traffic Management

The ZX5000 OpenArchitect switch has powerful facilities for controlling the queue length and distribution of bandwidth at the egress ports. A number of queueing desciplines are provided. The simplest of these use the COS queues alone to provide priority service based on the queue to which they are assigned, with no bandwidth limits. Beyond this approach, the Linux *tc* utility allows setting up queuing disciplines which use multiple traffic classes for each port. These traffic classes can be allocated guaranteed bandwidth, share remaining bandwidth with other traffic classes and be limited in bandwidth used. The classifier assigns packets to traffic classes as well as COS queues. The primary queuing disciplines supported are:

First In First Out (FIFO) – When only one queue is configured per port, packets are serviced in the order in which they arrive. The maximum length of the queue can be set. This is the default for the OpenArchitect switch.

Strict Priority – This is the default priority when more than one COS queue is used. The highest priority queue, which is also the highest numbered one, is always serviced. As long as there are packets in the highest priority queue, the lower priority queues are not serviced. The danger is that higher priority traffic could block lower priority traffic.

Weighted Round Robin (WRR) – This algorithm is similar to Strict Priority scheduling, but it provides fairness by sharing the bandwidth among the queues. It uses multiple traffic classes to manage the bandwidth used by each queue. Each queue is assigned a *weight*, which is used to determine the rate at which packets are added to the queue. Note that with WRR, the effective priorities of the queues are dependent on the weights allocated. A higher priority queue with a smaller weight will get less wire-time than a lower priority queue configured with a larger weight. Weighted Round Robin is set up using the Linux *tc* utility.

HTB – This algorithm uses multiple traffic classes, with guaranteed bandwidth assigned to each traffic class, and any remaining bandwidth shared among traffic classes according to a weighted priority scheme. Limits on the maximum bandwidth used by a traffic class can also be set. HTB is set up using the Linux *tc* utility.

Znyx provides a command line utility (*zcos*) to configure basic COS parameters. The *zfilterd* and *iptables* utilities provide for more flexible mapping of packets to queues.

Zcos Examples

On the OpenArchitect switch, *zcos* allows for configuring and displaying the priority to COS queue mappings. See the *zcos* man page in Appendix A for usage. Below are some sample configurations. Port *zre3* is used as the example, but one could use a port list instead.

FIFO Scheduling

This is the default setup on boot-up. The only parameter to configure is the maximum memory used by a port. The unit is 64 bytes.

```
zcos -s 256 zre3
```

Finally, map all 802.1p priorities to queue 0 and enable only 802.1p priorities for queue selection.

```
zcos -m  0,0,0,0,0,0,0,0 userQ –o user
```

To validate settings, query the parameters already configured for port zre3

```
zcos -q qinfo zre3
```

The following output should be observed:

```
Port 3: max egress queue size is 256 cells
```

Now display the entire map:

```
zcos -q qmap
```

The output should be:

```
     User Priority in to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
Default VLAN Priority to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
 MAC Address Priority to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
  L3 Address Priority to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
 IP MC group Priority to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
Queue Priority to Processor queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
Processor queue for exception packet map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0
TOS to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0 8->0 9->0
TOS to queue map:  10->0 11->0 12->0 13->0 14->0 15->0 16->0 17->0 18->0 19->0
TOS to queue map:  20->0 21->0 22->0 23->0 24->0 25->0 26->0 27->0 28->0 29->0
TOS to queue map:  30->0 31->0 32->0 33->0 34->0 35->0 36->0 37->0 38->0 39->0
TOS to queue map:  40->0 41->0 42->0 43->0 44->0 45->0 46->0 47->0 48->0 49->0
TOS to queue map:  50->0 51->0 52->0 53->0 54->0 55->0 56->0 57->0 58->0 59->0
TOS to queue map:  60->0 61->0 62->0 63->0
Queue Priority Order for Switched Packets: user
Queue Priority Order for Routed Packets: user
```

Strict Priority

Strict priority introduces COS differentiation. The configuration from the FIFO example above is assumed to be already on the switch. This builds on it.

Four queues are used with a priority mapping placing pairs of priorities into different queues as follows:

Priority	Maps to:
0 and 1	Queue 0
2 and 3	Queue 1
4 and 5	Queue 2
6 and 7	Queue 3

A higher queue number corresponds to a higher priority. The *zcos* commands to create this scenario are:

```
zcos -m 0,0,1,1,2,2,3,3  userQ
```

With strict priority scheduling, no low priority packets should be transmitted if a higher priority packet is currently queued. There is a danger of a high priority queue starving a low priority queue. Querying the settings:

```
zcos -q qmap
```

The following output should be displayed:

```
     User Priority in to queue map:  0->0 1->0 2->1 3->1 4->2 5->2 6->3 7->3
Default VLAN Priority to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
 MAC Address Priority to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
  L3 Address Priority to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
 IP MC group Priority to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
Queue Priority to Processor queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0
Processor queue for exception packet map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0
TOS to queue map:  0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0 8->0 9->0
TOS to queue map:  10->0 11->0 12->0 13->0 14->0 15->0 16->0 17->0 18->0 19->0
TOS to queue map:  20->0 21->0 22->0 23->0 24->0 25->0 26->0 27->0 28->0 29->0
TOS to queue map:  30->0 31->0 32->0 33->0 34->0 35->0 36->0 37->0 38->0 39->0
TOS to queue map:  40->0 41->0 42->0 43->0 44->0 45->0 46->0 47->0 48->0 49->0
TOS to queue map:  50->0 51->0 52->0 53->0 54->0 55->0 56->0 57->0 58->0 59->0
TOS to queue map:  60->0 61->0 62->0 63->0
Queue Priority Order for Switched Packets: user
Queue Priority Order for Routed Packets: user
```

WRR Scheduling

Strict priority scheduling is the simplest means of providing differential traffic treatment. The OpenArchitect switch also supports weighted round robin, which allows a user to assign weights to each queue. The weights are used to distribute the bandwidth among the queues. WRR scheduling is the first example which uses the Linux traffic control program (*tc*) to configure the switch.

zfilterd

zfilterd is a daemon that uses the Packet Classifier of the IXE5416 Switch for filtering based on the Linux rules setup by the tool, *iptables*.

zfilterd intercepts filtering rules entered by the user, checks them for validity and then passes them on to *ztmd* for entry in the switch. See the *zfilterd* man page in Appendix A for details on all options.

ztmd

ztmd is traffic management daemon which accepts messages from traffic filtering and quality of service applications and sets up the hardware.

Running zfilterd

Before starting *zfilterd*, *ztmd* must be running. Your can start both from within a script, or directly from the command line.. For example,

```
ztmd
zfilterd
```

iptables rules can be entered at any time. If your *iptables* filtering rules set is extensive, you may want to move your set of *iptables* commands to a start up script to run upon initialization. This could be accomplished by creating a standalone "S" script and placing that script into `/etc/rcZ.d`.

Restrictions on Implementation

Several restrictions exist on the rules that can be implemented on the FFP hardware. These include:

Actions
: DROP the packet. ACCEPT the packet.

Field values
: If specified as ranges, they must be on power of two boundaries.

Negation
: Can only be used for icmp code, tcp port or flag, or udp port fields.

Fields supported are: Source IP address, destination IP address, IP protocol, TCP or UDP source port or destination port, ICMP type, and TCP flags bits (such as SYN).

The input port and output port may also be specified as either *zre*<*n*>, where <n> is one of the 16 physical ports, as zrm, to select the CPU port, as *zrl*<*n*> , where the trunk used must be previously defined, or as *zhp*<*n*>, where the *zhp* interface used must be previously defined using *zconfig*.

Conflict Resolution

There are differences from the expected behavior of implementing *iptables* in a host: Although the rules are taken from the FORWARD and INPUT chains, they are applied to all packets, including those destined for the local CPU. The order of application of the rules is not necessarily the order in which they appear in the chains. Rules which are more restrictive will be applied first. Once a matching rule is applied, further rules will be ignored. For example, the rules,

```
iptables -A FORWARD -i zhp3 -j DROP
iptables -A FORWARD -i zhp3 -o zhp1 -p tcp --dport smtp -j ACCEPT
```

result in SMTP packets received on any port in *zhp3* to be sent for any port in *zhp1*; all other packets from *zhp3* would be dropped. The order of the two rules in the FORWARD chain does not matter.

On the other hand, in the following sequence of rules, the position of the rule that drops SYN packets is important. Since the set of fields it examines is not a subset of the fields examined by the ACCEPT rules, and visa versa, the ordering rule given above does not apply. In this case, the order it is applied will be the same as its position in the FORWARD chain, and all packets which are TCP SYN packets from *zhp5* for *zhp3* will be DROPPED, even if they also match one of the ACCEPT rules.

```
iptables -A FORWARD -i zhp5 -o zhp3 -j DROP
iptables -A FORWARD -i zhp5 -o zhp3 -p tcp --syn -j DROP
iptables -A FORWARD -i zhp5 -o zhp3 -p tcp --sport smtp -j ACCEPT
iptables -A FORWARD -i zhp5 -o zhp3 -p udp --sport domain -j ACCEPT
iptables -A FORWARD -i zhp5 -o zhp3 -p tcp --sport domain -j ACCEPT
iptables -A FORWARD -i zhp5 -o zhp3 -p tcp --sport www -j ACCEPT
iptables -A FORWARD -i zhp5 -o zhp3 -p tcp --sport 23 -j ACCEPT
```

iptables and filtering

iptables is a firewall management user-space utility used in conjunction with the Linux 2.4 kernels. i*ptables* takes advantage of the netfilter 2.4 kernel code.

ZNYX Networks extends *iptables* with a few more targets to support the hardware filtering functionality available in the Intel IXE5416 silicon on the ZX5000. Generally, all of the *iptables* functionality is usable with a few minor extensions.

A more detailed source on *iptables* can be found at:

http://www.netfilter.org/

Almost all the content described here is derived from there.

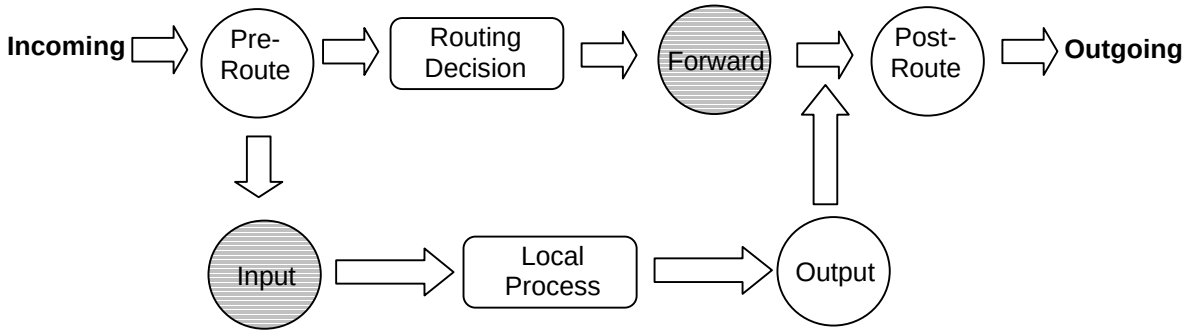
There are also many tutorials and *iptables* manipulation tools, both graphical and command line. This is expressive of the OpenArchitect concept. A good place to start is:

http://freshmeat.net/search/?q=iptables

Introduction

Firewall rules are stored in tables. These tables are sometimes also known as *firewall chains* or just *chains*. Tables normally store rules for what are known as *hooks*, which can be looked as packet-path junctions. There are five defined hooks: PRE-ROUTE, POST-ROUTE, INPUT, OUTPUT and FORWARDING. The example below illustrates the default chains on boot up.

By default, INPUT, FORWARD and OUTPUT chains are installed on boot up. Additional rules can be installed for the other chains. Additionally, one can write software extensions to add more chains.



When a packet reaches a circle in the diagram, that chain is examined to decide the fate of the packet. Two basic fates of a packet are defined as DROP and ACCEPT. If the chain says to DROP the packet, it is killed there; however, if the chain says to ACCEPT the packet, it continues traversing the diagram, ultimately terminating at an application or getting forwarded out of the box. There are additional ZNYX Networks actions that may be applied to packets. These are described in the "Supported Targets" section.

A chain is a checklist of rules. Each rule is checked against the packet header and if a rule matches, action is taken. If the rule doesn't match the packet, then the next rule in the chain is consulted. Finally, if there are no more rules to consult, then the kernel looks at the chain default policy to decide what to do. In a security-conscious system, this policy usually tells the kernel to DROP the packet.

In the ZX5000 product, both the FORWARD chain hook, and the INPUT chain hook (packets destined for the CPU) are implemented in hardware. The rest of the hooks are in software in the Linux kernel. An extension of the FORWARD hook also resides in software. It is important to note that this is in sync with routing being implemented in hardware with software assist for exception handling. Under general circumstances, when routing happens in hardware, only the FORWARD chain is traversed. Under exceptional handling of an incoming packet, one can force the full software traversal. As a router you do not really care about the other hooks except in the situation where you have some special handling., in which case a policy would force the packet to be sent to the CPU for further processing. Note: This is also how one would extend the OA packet munging capabilities (e.g. introduce NAT).

Packet Walk

When a packet comes in via one of the interface ports, the ZX5000 makes a routing decision. If the packet was destined for the ZX5000 itself or if the send to CPU action is specified, it is sent to the INPUT chain for further processing. If there is no valid way to forward the packet, it is dropped. If the switch is configured to forward the packet, it is sent to the FORWARD chain.

Next the hardware FORWARD chain is walked. If there is a rule inserted that matches the packet headers, then it is looked up next. The inserted policy will decide the packets fate.

In essence, a filter rule will be used to scan the packet data for certain characteristics. Upon a match a selected 'target' is executed. The target decides what should happen to the packet.

Filter Rules Specifications

A rule could be added (-A) to a chain, deleted (-D) from a chain, replaced (-R) in a chain or inserted (-I) in a specific position in a chain. Each rule specifies a set of conditions the packet must meet, and what to do if it meets them ('what to do' is referred to as a `target').

Here's an example filter rule:

```
iptables -A FORWARD -p UDP -s 0/0 -d 10.0.0.1/32 --source-port 53 -j
       DROP
```

This adds to the FORWARD chain the rule: "If you see UDP packets (-p UDP) from anywhere (-s 0/0) going to host 10.0.0.1 (-d 10.0.0.1/32) with a source port number 53 (--source-port 53) then the target is to DROP (-j DROP). More details on rule specifications follow.

Specifying Source and Destination IP Addresses

Source (`-s, --source or --src`) and destination (`-d, --destination or --dst`) IP addresses can be specified in four ways. The most common way is to use the full name, such as `localhost` or `www.linuxhq.com`. The second way is to specify the IP address such as `127.0.0.1`.

Netmasks can be applied to IP addresses to specify ranges, like `199.95.207.0/24` or `199.95.207.0/255.255.255.0`. Both specify any IP address from *199.95.207.0* to *199.95.207.255* inclusive. To specify an all-inclusive IP address `/0` can be used, like: `-s` or `-d 0/0`. The example rule we use above applies this trick. Note however that the effect above is the same as not specifying the `-s` option at all.

Specifying Inversion

Many flags, including the `-s` (or `–source`) and `-d` (or `--destination`) flags can have their arguments preceded by `!` (pronounced `not') to match addresses NOT equal to the ones given. For example, `-s ! localhost` matches any packet not coming from *localhost. NOT SUPPORTED*

Specifying Protocol

The protocol can be specified with the `-p` (or `--protocol`) flag. Protocol can be a number (if you know the numeric protocol values for IP) or a name for the special cases of `TCP`, `UDP` or `ICMP`. Case doesn't matter, so `tcp` works as well as `TCP`.

The protocol name can be prefixed by a `!`, to invert it, such as `-p ! TCP` to specify packets that are NOT TCP. *NOT SUPPORTED*

Specifying an Interface

The `-i` (or `--in-interface`) and `-o` (or `--out-interface`) options specify the name of an interface to match. An interface is the physical device the packet came in on (-i) or is going out on (-o). You can use the *ifconfig* command to list the `up' interfaces (i.e., working at the moment).

As a special case, an interface name ending with a + will match all interfaces, whether they currently exist or not, which begin with that string. For example, to specify a rule which matches all *zhp* interfaces, the `-i zhp+` option would be used.

The interface name can be preceded by a `!` to match a packet which does not match the specified interface(s). *NOT SUPPORTED*

Filter Rule Targets

As mentioned above the `-j` construct within a rule specifies which target is to be used in filter rule to define a target.

Supported Targets

The following are the supported targets. The ZX5000 has many additional targets that are software based (example Network Address Translation or generic connection tracking). Please contact ZNYX Technical support if you have additional questions on additional features.

Classical Targets

DROP — This drops the packet.

ACCEPT — Accepts the packet

ZNYX Targets

ZACTION

This is the Znyx Action target.

Parameters for ZACTION:

--use-prio <val> — Use queue priority <val>

-set-tos <val> — Set the IP-Precedence bits in the TOS field of the IP header to <val>

-set-dscp <val> — Set the 6-bit DSCP in the TOS field of the IP header to <val>.

--copy-cpu — Send the packet to the CPU. This will force the full installed chains traversal in software

--accept or *–drop* — Accepts or drops the packet

Options with any of these ZACTION parameters:

--counter <val> — Increment classifier hit counter <val>

--arp — Not an action, match only ARP packets. Only –i, -o, and –d fields are permitted with this option.

ZACTION Examples

Send all tcp packets arriving on zhp5 to the CPU (software).

```
iptables -A FORWARD -i zhp5 -p tcp -j ZACTION --copy-cpu
```

Set the 802.1p priority to 3 on all tcp packets arriving on zhp5.

```
iptables -A FORWARD -i zhp5 -p tcp -j ZACTION --set-prio 3
```

Extensions to the default matches

These are described in the Linux packet filtering HOWTO at:

http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html

ZNYX Networks FORWARDING Chain supports all of them.

NAT/MASQUAREDING and stateful firewalling

ZNYX Networks supports these features, however, they are only implemented in software and therefore will not be very scalable for large requirements.

tc and zqosd

For more advanced queing disciplines, including bandwidth allocation to selected traffic on an egress port and limiting the bandwidth that can be used, additional control is required beyond what can be done with *iptables*. *tc*, which stands for Traffic Control, is a mechanism for enabling Quality of Service on Linux. *tc* uses three functional objects: *queueing disciplines*, which comprise queueing and scheduling algorithms such as FIFO queues, priority queues, RED queues, and token buckets; *classes*, which provide differentiated service to classes of traffic in queueing discipline hierarchies; and *filters*, such as u32 filters, which classify packets into traffic classes.

The functional elements of *tc* may be combined to produce complex QoS rules. For example, a packet may be matched to a filter, metered, policed as in-profile or out-of-profile, remarked, mapped to a FIFO queue, and transmitted by a priority scheduler. *tc* is very flexible in the data paths that it allows.

The ZNYX utility *zqosd* is a daemon that monitors Linux QoS policy and shadows the policy rules into a hardware configuration. When *zqosd* is running, *tc* rules are translated into hardware rules. Note: This document does not detail all of the capabilities of the *tc* command, rather it explicitly mentions only features that are supported by OpenArchitect-based switches.

The examples that follow assume that the switch is running the standard Layer 2 start-up script, `/etc/rcZ.d/examples/S50layer2`, with all ports placed in a single VLAN, *zhp0*. Note that this assumption is implied only by the fact that changes to *zhp0* are shown to configure all ports. Neither *tc* nor *zqosd* is limited by the interface setup. Each utility works on either VLANs (*zhp*) trunks (zrl) or ports (*zre*).

FIFO Queues (pfifo and bfifo disciplines)

The simplest configuration for *tc* involves no classes or filters, and only a single FIFO queue. With *tc*, queue sizes may be specified in bytes or packets. The first example defines a packet-limited FIFO. This example begins with only *tc* and then illustrates *tc* in conjunction with *zqosd*.

As a first step, confirm that no *tc* configuration is active on the switch, by listing any queue disciplines:

```
tc qdisc ls
```

The command should return nothing. Now, add a single packet-limited FIFO queue to *zhp0* and confirm that it has been installed to software:

```
tc qdisc add dev zhp0 handle 100:0 root pfifo limit 32
tc qdisc ls
```

The output should display the following,

```
qdisc pfifo 100: dev zhp0 limit 32p
```

The *tc* command is applied to a device, so `dev zhp0` must be specified. Note that a VLAN, such as *zhp0*, and a port, such as *zre0*, are each treated as devices. Breakdown of the options:

handle 100:0	Defines the handle for the queing discipline. This handle may be used to reference the pfifo queue. Note that the handle is included with the output of the qdisc ls command. (100:0 and 100: are equivalent in tc.) The choice of handle is significant for zqosd.
root	Tells tc that this is the base queueing discipline for the device, not a child of another queueing discipline.
pfifo limit 32	Specifies a packet-limited FIFO queue with an upper bound of 32 packets.

Now, delete the queueing discipline from *zhp0* and confirm that it has been removed:

```
tc qdisc del dev zhp0 root
tc qdisc ls
```

Thus far, *tc* has been used without *zqosd*. It is not sufficient to install software rules on the OpenArchitect switch though, because the normal case is for packets to be switched in hardware. For that reason, *zqosd* must be used to shadow *tc* configuration into hardware. Like *zfilterd*, *zqosd* works with *ztmd*, which provides the actual hardware interaction.

If *ztmd* is not already running, start it:, then initiate the zqosd daemon with no parameters:

```
ztmd
zqosd
```

Now, repeat the same *tc* command as before, to install a packet-limited FIFO queue:

```
tc qdisc add dev zhp0 handle 100:0 root pfifo limit 32
```

When this command is processed, *zqosd* detects the state change and generates messages to *ztmd* to set up the hardware.

The actual limit on the ZX5000 is set in terms of bytes. The program uses an average packet length of 1000 bytes to convert the packet-limit to a byte-limit.

Thus, the byte-limited version of the previous *pfifo* queue is:

```
tc qdisc add dev zhp0 handle 100:0 root bfifo limit 32000
```

Tear down any installed rules before proceeding with the next example:

```
tc qdisc del dev zhp0 root
```

Random Early Detection (red discipline)

Random Early Detection is a queue discipline which manages a single queue, controlling its length by gradually dropping more and more incoming packets as the queue length grows. Once the queue has reached a minimum length packets enqueued have a chance of being dropped. This chance increases linerly until the queue reaches a maximum length. The queue may grow longer than this maximum, when it reaches a limit value, all further packets are dropped. RED prevents synchronous retransmits after a burst in traffic, while allowing a small queue size.

As with the *pfifo* and *bfifo* queue disciplines, *red* can be configured as the root qdisc on a port, or it can be configured on top of one of the classes of a qdisc such as *prio* or *wrr*. Red has many required parameters, not all of which are really needed for the hardware. As an example, to set up red as the root qdisc on a single port, with discard to start at 16000 bytes and continue to discard 5% of the packets when the queue depth is 32000 bytes, the tc command is:

```
tc qdisc add dev zre3 root handle 103:0 red min 16000 max 32000
    probability .05 limit 80000 avpkt 1000 burst 20
```

The value for **limit** is the upper bound for queue length. The parameters **avpkt** and **burst** are required, but not used.

Configuring red qdisc on top of a classful qdisc like *prio* requires the same changes as was done in the pfifo case. The keyword **root** is replace by **parent** <handle>, where <handle> is the id of the class which the red qdisc is to be associated with.

Priority & Weighted Round Robin Scheduling (prio discipline with wrr)

The FIFO examples used a single queue for each interface. In fact, the ZX5000 maintains eight queues of packets for each port, with either priority or weighted round robin (WRR) scheduling, classification of packets into particular queues is based on a priority map.

In *tc*, the *prio* queueing discipline establishes multiple queues and specifies their associated priority map. Although WRR support is not part of the standard *tc* distribution, it has been added to the *prio* discipline by ZNYX.

Although priority scheduling can be implemented using only zcos. The priority mapping is common for all ports, and it is not possible to assign relative bandwidths to the queues. The final example in this document illustrates WRR. A strict priority scheduler is a simpler case that can be constructed easily from this example.

The full set of commands to install four queues, a priority map, and weights is as follows:

```
tc qdisc add dev zre5 handle 100:0 root prio bands 4 priomap 1 2 2 2 3
   3 3 3 1 1 1 1 1 1 2 wrr 1000 2000 4000 6000
tc qdisc add dev zre5 parent 100:1 pfifo limit 32
tc qdisc add dev zre5 parent 100:2 pfifo limit 32
tc qdisc add dev zre5 parent 100:3 pfifo limit 32
tc qdisc add dev zre5 parent 100:4 pfifo limit 32
```

The first command attaches a queueing discipline as the root discipline for *zre5*, with a handle of "100:0," as in the FIFO cases. The "prio" option identifies the type of queueing discipline.

Priority scheduling implies multiple queues and the "bands 4" parameters specify that there are four queues to be used.

The priority map may be read from left to right as *Priority* n *maps to Queue* q, where *n* is the index of the list element (numbering from 0) and *q* is the value specified by that element. So, this example would read:

- Priority 0 maps to Queue 1
- Priority 1 maps to Queue 2
- Priority 2 maps to Queue 2
- Priority 3 maps to Queue 2
- Priority 4 maps to Queue 3

.
.
.

Note that the *tc* priority map applies to a 4-bit field. With the ZX5000, the priority map refers to the IP-precedence portion of the TOS, which is a 3-bit field. When translating this *tc* rule to hardware, only Priorities 0 through 7 are significant; the other eight priorities are ignored.

The parameters `wrr 1000 2000 4000 6000` specify that WRR scheduling is being used and assigns weights to each queue. These weights are treated as relative values. In this example, the sum of the weights is 13000:

- Queue 0 will receive approximately 1000/13000 = 7.7% of the port bandwidth.
- Queue 1 will receive about 2000/13000 = 15.4%.
- Queue 2 will receive about 4000/13000 = 30.8%.
- Queue 3 will receive about 6000/13000 = 46.1%.

The remaining commands each define a packet-limited FIFO queue. As with all previous *tc* examples, these queues are created on the same device as the root queue discipline. However, unlike all previous examples, they are not created as root disciplines for the device. Instead, the "parent" option identifies them as child queues of the *prio* discipline.

For example, "parent 100:1" identifies that queue as the first child of the *prio* discipline (Queue 0), because the *prio* discipline's handle is 100:0.

The queue mappings are straightforward: as noted, they represent the first eight elements in the *tc* priority map. The weights are adjusted to packet values

To test the strict priority case, simply remove the `wrr 1000 2000 4000 6000` options from the first *tc* command. Note that all queue disciplines in this test may be cleared by deleting the root discipline, as before:

```
tc qdisc del dev zre5 root
```

HTB Queuing Discipline

HTB is a classful queuing discipline which provides greater control of the outbound bandwidth than *prio* or *wrr* qdiscs. With HTB, a port can have an arbitrary number of traffic classes. Packets are assigned to classes using a filter which can classify a packet based on its L2, L3 or L4 headers, as well as the port on which it arrives. Each traffic class has a minimum bandwidth guarantee, called its **rate**, and a maximum, or **ceil**, which it is not to exceed. If a port has excess bandwidth available after its classes have received their guarantee, it is used to send additional packets from classes which have not reached their **ceil**, according to weights based on their relative **rate**s and priority, **prio**.

Setting up HTB on a port involves several steps, since each class is set up separately. In addition, a root class is defined to provide a upper bound for the bandwidth of each port. The following commands set up three traffic classes on each port in zhp0:

```
tc qdisc add dev zhp0 root handle 100:0 htb
```

This command attaches the queue discipline HTB to each port of zhp0 and gives it a handle of 100:0.

```
tc class add dev zhp0 parent 100:0 classid 100:1 htb rate 50mbit ceil 50mbit
```

Now the tc class command is used to attach a base class which provides an upper bound on the bandwidth to be used of 50 million bits per second. It is not really necessary to give both **rate** and **ceil** parameters; if only **rate** is given, the same value will be used for **ceil**.

```
tc class add dev zhp0 parent 100:1 classid 100:10 htb rate 5mbit
tc class add dev zhp0 parent 100:1 classid 100:20 htb rate 1mbit ceil 50mbit
     prio 1
```

```
tc class add dev zhp0 parent 100:1 classid 100:30 htb rate 5mbit ceil 20mbit
    prio 2
```

The first class, id 100:10, is guaranteed 5mbit, and is not to exceed that bandwidth. The second class, id 100:20, is only guaranteed 1mbit, but is allowed to use up to the full 50mbit limit for the port. It has a priority of 1 for allocation of excess bandwidth. Class 100:30 is guaranteed 5mbit, and can use up to 20mbit, if available. Its priority of 2 gives it a smaller proportion of the excess bandwidth than class 100:20. No explicit priority was specified for the class 100:10, since it is not allowed to use any bandwidth above its guaranteed rate.

Before this queuing discipline can be used, it is necessary to describe what packets belong in each class. We could use iptables, except that it provides no action to direct a packet into a particular class; instead, the filtering capabilities of tc are used.

The U32 Filter

The u32 filter provides the capability to match on fields in the L2, L3 or L4 header of a packet. Each match rule gives the location of the field to be tested, which is always a 32 bit word, a mask selecting the bits to be tested, and a value which is to be matched by the packet field. Many matches can be specified in one tc filter command. Only if all matches succeed does the filter match. In that case, the **flowid** field identifies the classid of the class this packet belongs in.

The following tc commands put all icmp packets in class 100:10, packets from ip address 1.2.3.4 in class 100:20. Packets for ip address 1.2.3.4 in class 100:20, and arp reply packets in class 100:30. The last filter illustrates using an offset from the beginning of the protocol header, along with a mask, to locate the field to be matched

```
tc filter add dev zhp0 protocol ip parent 100:0 u32 match ip protocol 1 0xff
    flowid 100:10
tc filter add dev zhp0 protocol ip parent 100:0 u32 match ip src 1.2.3.4/32
    flowid 100:20
tc filter add dev zhp0 protocol ip parent 100:0 u32 match ip dst 1.2.3.4/32
    flowid 100:20
tc filter add dev zhp0 protocol arp parent 100:0 u32 match u32 2 0xffff at +4
    flowid 100:30
```

Translation Rules

Some *tc* configurations have no exact correlation in the switch hardware. This section explains the rules that are used for translating *tc* commands into hardware. The rules are listed by discipline.

bfifo — *limit*: The limit is in bytes. The single limit is translated into 2 thresholds; when the queue is less than the first threshold, all packets are kept. When the queue is between the first and second threshold, packets from the "worst" stream are discarded. When the queue is greater than the second threshold, almost all packets are discarded. A limit of 16kB or less is treated as 16KB. If the limit is below 64KB, the first threshold is 16KB and the second is 32KB. Otherwise, the first threshold will be the closest power of two to the limit and the second threshold will be 1.5 times the first threshold.

htb	This queuing discipline is typically specified in three parts, with different parameters used for each part. **htb root:** No optional parameters are used. **htb root class:** *ceil*: rate limit for the ports in the interface. **Htb child classes**: These are not required, but any number of child classes can be defined to limit and prioritize the bandwidth used by packets selected using the **tb filter** commands. *ceil*: The maximum rate the class is allowed. *rate*: The guaranteed rate the class will be given, if there is enough traffic. *prio*: The priority of this class for use of any bandwidth remaining after all classes have received as much of their guaranteed rate as they can use. Allowed values are 0-3. prio 0 is the default, and has the highest priority. Prio and rate/ceil are used to determine the weight, which is used in a round-robin process to allocate actual bandwidth.
pfifo	*limit*: The limit in packets is converted into a limit in bytes by multiplying by 1000. See the translation for *bfifo* above.
prio/wrr	*bands*: The hardware supports up to 8 queues. For any specified value greater than 8, 8 queues are used. *priomap*: Only the lower eight priorities are used. Furthermore, if any priority maps to a queue greater than the number of queues, it is treated as if it maps to the highest number queue defined. *wrr:* The weights are obtained by dividing the byte parameter by the sum of this parameter across all bands to give a bandwidth percentage.
red	*min*: The minimum value is 16kb. Values are adjusted to multiples of 16kb. *max*: Same rules as for *min.* *limit*: All new packets will be dropped when the queue reaches this length. Same rules as for *min*. *probability*: expressed as a decimal number less than one.. *avpkt*: Only used for host cpu processing. Can be set to 1000.

burst: Recommended value is (*min*+*min*+*max*)/3000.

Several combinations of queuing disciplines are possible, to further control bandwidth distribution and packet discard when the available bandwidth and limits are exceeded. The following combinations are supported by the hardware:

Combining Queueing Disciplines

Any of the queue length limiting disciplines can be used with the bandwidth management queue disciplines, by defining them with the handle of one of the classes as their parent. For the htb queueing discipline, each class has an explicit handle specified when it is defined. For the prio queueing discipline, including wrr, each band is a class; their handles are formed from the handle of the prio qdisc by appending a minor number of 1 to n for the n bands. For example, the following commands define two strict priority queues for port zre5, with the lower priority queue limited to 32kb and the higher priority queue using red to discard packets as its length increases above 16kb:

```
tc qdisc add dev zre5 root handle 100:0 prio bands 2 priomap 0 0 0 0 1 1 1 1
tc qdisc add dev zre5 parent 100:1 handle 110:0 bfifo limit 32kb
tc qdisc add dev zre5 parent 100:2 handle 120:0 red min 16kb max 64kb limit 80kb
           avpkt 1000 burst 32 probability .1
```

These translation rules handle conversions of individual rules from *tc* entries into hardware entries. They do not explain the results of creating rules that are individually supported; but which do not make sense in conjunction.

Although the translation rules handle some inconsistency between software and hardware, a user must define a combination of rules that is reasonable in hardware, to ensure predictable results.

Handle Semantics

All examples have illustrated *zqosd* copying *tc* rules into hardware. In fact, the *zqosd* utility also enables the user to add *tc* rules that remain only in software. This selection is based on handles. *zqosd* processes all supported queue disciplines and filters with handles between 100:0 and 200:FFFF. Note that handle values are interpreted as hexadecimal numbers, so digits from 0-9 and A-F can be used.

COPS: Common Open Policy Service

The Common Open Policy Service (COPS) is a protocol for distributing networking policy to devices such as switches and routers. COPS allows a single Policy Decision Point (PDP) to distribute policy to multiple Policy Enforcement Points (PEPs). A PDP acts as a server for PEP clients.

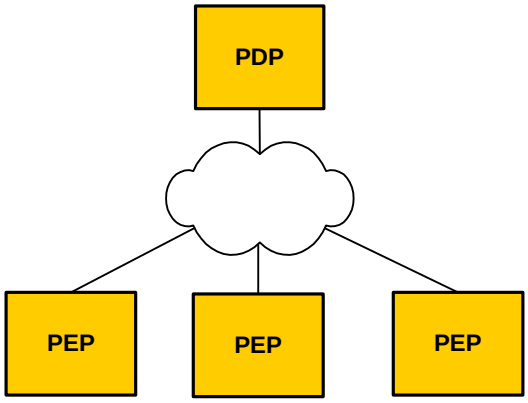


Figure 3-3 COPS Network Architecture

A PDP contains all of the policy rules for its associated PEPs. A PDP typically stores rules in a data and is a dedicated server, not a forwarding device.

A PEP is any network device that has to enforce policy decisions. For example, a switch that restricts network access or prioritizes traffic fits the definition of a Policy Enforcement Point. A PEP makes no policy decision. It simply applies policy that receives from its PDP.

COPS uses a connection-based query and response mechanism. The following scenario illustrates PEP-PDP communication:

- A PEP comes online and opens a connection to its PDP.
- After a connection has been established, the PEP transmits state information to the PDP.
- The PDP uses that state information to determine what policy is applicable for the PEP.
- The PDP sends that policy to the PEP.
- The PEP installs the policy and applies it to future traffic.

As long as COPS is running, a connection between the PEP and PDP should stay open. A PEP could query a PDP at any time asking for a policy decision. Alternatively, an administrator could modify the policy on a PDP, which would then push any policy changes to its PEPs.

Protocol Architecture

The COPS protocol is broken into several components. The base layer is the COPS protocol itself, which defines the messaging format. This protocol defines *how* communication is handled without specifying the details of the message data.

The base COPS protocol is then used by different *client types*. These client types apply the COPS messaging scheme to particular types of data. The currently standardized client types deal with the RSVP model (COPS-RSVP) and provisioning model (COPS-PR).

The COPS-RSVP scheme is designed around the requirement that a PEP will have to query a PDP in response to events. An RSVP PEP is constantly listening for resource reservation requests and relaying those requests to its PDP.

By contrast, the provisioning model is based on longer lasting policy. The expectation is that policy should be administratively defined at the PDP and pushed to the PEPs as needed. OpenArchitect is a COPS-PR client.

The most common use of COPS-PR is for distributing *Differentiated Services* (Diffserv) policy. Diffserv is concerned with such Quality of Service elements as queues and schedulers.

OpenArchitect PEP

The OpenArchitect PEP implementation is known as *pepd*. The *pepd* utility is based on:

- RFC 2478: Common Open Policy Service (COPS)
- RFC 3084: COPS Usage for Policy Provisioning
- RFC 3159: Structure of Policy Provisioning Information
- RFC 3289: Management Information Base (MIB) for the Differentiated Services Architectecture
- Internet Draft: Differentiated Services Quality of Service Policy Information Base (latest version draft-ietf-diffserv-pib-09)
- Internet Draft: Framework Policy Information Base (latest version draft-ietf-rap-frameworkpib-09)

A Policy Information Base (PIB) defines the representation of a particular data set. For example, the Diffserv PIB specifies the structures userd to represent all Diffserv elements. PIBs are functionally equivalent to Management Information Bases (MIBs) such as those used by SNMP. The OA PEP has implemented those portions of the Diffserv and Framework PIBs that are supported by the underlying switch architecture.

The *pepd* utility requires a PDP that has implemented the above RFCs and drafts. Until all draft standards are approved, the certain COPS-PR data types will not be assigned OIDs. *pepd* uses non-standard OIDs for the unassigned values.

Using *pepd*

The *pepd* utility works by connecting to a PDP, informing the PDP of its *roles*, and installing any rules that the PDP has for those roles. Configuration information should be specified in a configuration file, specified on the command line with the –f option.

```
pepd –f <full_path_and_filename>
```

A sample configuration file is listed below:

```
PDP address: 10.0.0.11
   PDP port: 3288
   PEPID: some-id
   Role-If: a zre1,zre2,zre3,zre4
```

where,

PDP address:	The IP address of the PDP. Default is loopback (127.0.0.1)
PDP port:	The destination port on which to open a COPS connection. Default is 3288.
PEPID:	The PEP Indentifier
Role-If:	A mapping of roles to interfaces. The name of the role is followed by a comma-delineated list of interfaces. Multiple role-interface mappings are defined through multiple Role-If declarations.

Chapter 4 High Availability Networking

High availability (HA) networking is achieved by eliminating any single point of failure: Redundant cables, switches and network interfaces for hardware, combined with HA software solutions. ZNYX Networks provides a complete solution with RAINlink on the host and Surviving Partner on the switch.

NOTE: OpenArchitect HA is not part of the standard product. The license for OA/HA is purchased separately. You must have an OpenArchitect/HA Suite RTU license for the chassis to use any OpenArchitect High Availability functionality or component on the switch within the chassis, or use OpenArchitect/Node on any of the host cards. The steps to installing the RTU are included in the sections that follow regarding configuration of Surviving Partner.

Please contact your ZNYX Sales Representative, call (800) 724-0911, or send email to sales@znyx.com, or support@znyx.com.

RAINlink

RAINlink provides *IP Transparent Failover* between two or more ports running in hosts that use ZNYX drivers. RAINlink presents a single, virtual interface to the protocol stack while managing multiple physical links. Figure 4-1 shows the relation of the protocol stack, the RAINlink driver and physical ports.

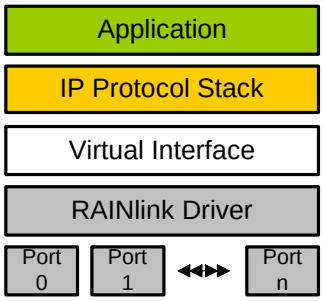


Figure 4-1 RAINlink Architecture

With RAINlink, a failover between physical links can be made very quickly without requiring change to the IP or MAC address of the virtual interface, effectively transparent from the applications point of view. With redundant links from a switch (or switches) to the host, one link is maintained as the ACTIVE link and the other as STANDBY. If the ACTIVE link were to go down, RAINlink switches to the STANDBY, while presenting the same virtual interface to the host.

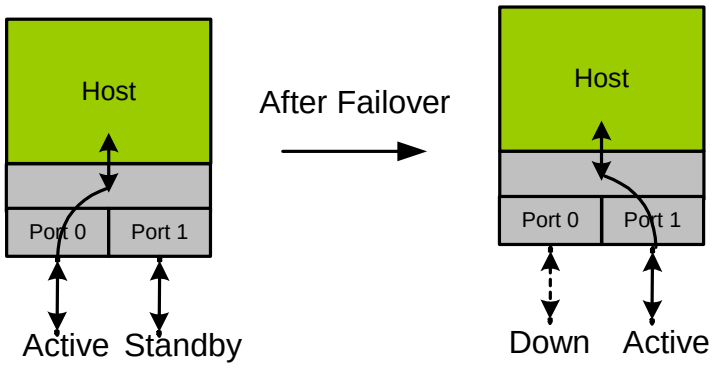


Figure 4-2 RAINlink Failover

Redundant connections with RAINlink can provide an ACTIVE and STANDBY link to a switch, or more likely to provide redundancy between more than one switch. In the case of more than one switch, a complete HA solution requires a switch-based HA solution.

Surviving Partner

Surviving Partner is ZNYX Network's switch-based HA solution. Surviving Partner runs on the switches to provide transition of Layer 2 and Layer 3 switching functionality between two or more switches. Surviving Partner is comprised of many interactive protocols and processes including *VRRP*, *zlmd*, *zlc*, and others.

VRRP

Since most end nodes use default router addresses, the change of the default router address during a switch failover would require the end nodes to reconfigure. Layer 3 switches that failover must maintain the default router address to maintain the end node's IP transparent failover. The Virtual Router Redundancy Protocol (VRRP, RFC 2338) running in the Surviving Partner switches provides transparent movement of the default router address. VRRP maintains the notion of a Master switch and one or more Backup switches. This group of switches presents a virtual router IP address that can be used by hosts on that net as their default route, as shown in *Figure 4-3.*

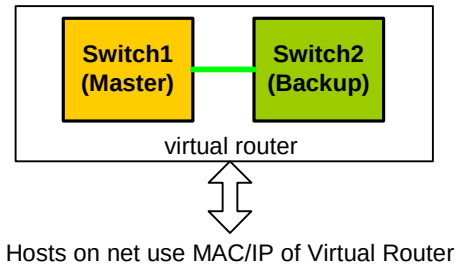


Figure 4-3 VRRP

If a Backup switch determines the Master switch is no longer available, one of the Backup switches will assume the role as Master. Physically, each switch maintains a link to the local network. Only the Master switch answers to the default gateway, and the hosts on that net have no need to relearn the router address.

In an HA configuration, the goal is to avoid any single point of failure. VRRP provides a good mechanism to provide a static route for a local network, but a true HA configuration must also provide redundant connections for the host. Providing a virtual router for the local network is not enough. Take the simple case of two hosts on the local network with a connection to the virtual router. Each host needs a connection to each physical switch participating in VRRP. In the simplest configuration, each host would have one connection to the network as shown in *Figure 4-4*.

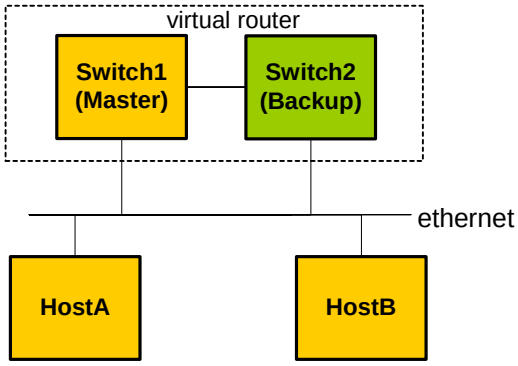


Figure 4-4 VRRP with Single Point of Failure on LAN

The problem with *Figure 4-4* is that it assumes that each host has a single connection into a hub or switch that is then connected to the redundant virtual switch. An HA solution would include redundant connections from each host to each switch in the virtual router, as shown in *Figure 4-5*

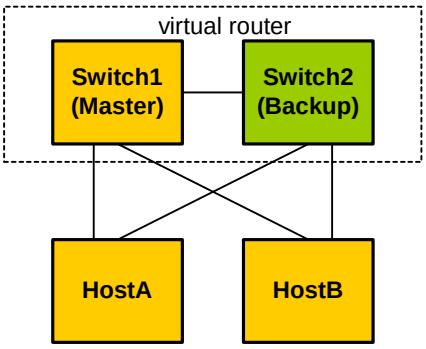


Figure 4-5 VRRP with HA on LAN

Combining the features of Surviving Partner on the switches and RAINlink on the hosts allows implementation of this true HA configuration.

ZLMD

In addition to complete switch failover, single link failure must be properly handled. The ZNYX Link Monitor Daemon *zlmd*, monitors the link status of each port. If a link goes down, *zlmd* communicates with the VRRP daemon (*vrrpd*) to change its priority. Changing the VRRP priority results in movement of switching functionality.

By combining *zlmd* with the *zlc* application, links connected to hosts that have not failed can be deterministically moved to the new master switch if desired. Supported modes:

switch	The switch with the greatest number of UP links becomes the Master for all VLAN's under HA management.
vlan	The switch with the greatest number of UP links in that particular VLAN becomes the Master for that particular VLAN. If the switch has additional VLANs, they each change independently.
port	The Master will remain the Master for that particular VLAN until all ports in that VLAN are down. The Backup then becomes the new Master for that VLAN. Failed links move their connectivity through the Backup Switch and the switch interconnect to reach the Master Switch. This option alleviates the need to move all nodes to a new switch just because a single link goes down

Switch Replacement and Reconfiguration

When a switch fails, it must be replaced. The replacement switch will likely require proper configuration. For transparent switch replacement, the newly replaced switch must learn its configuration from its Surviving Partner.

Figure 4-6 illustrates a simple failover scenario. Host A and Host B are configured with ZNYX RAINlink software, providing IP transparent failover between two host ports, one port connected to Switch A and the other connected to Switch B. Assume Switch A provides connectivity between Host A and Host B. If Switch A fails, RAINlink moves the active link on each host over to the port connected to Switch B. Surviving Partner software on Switch B recognizes that Switch A has failed, and assumes the role of switching traffic between Host A and Host B. When the failed Switch A is replaced with a new Switch A, Switch A will learn its network configuration from the surviving partner Switch B. Replacement Switch A is now ready as a backup to Switch B in case of failure of Switch B.

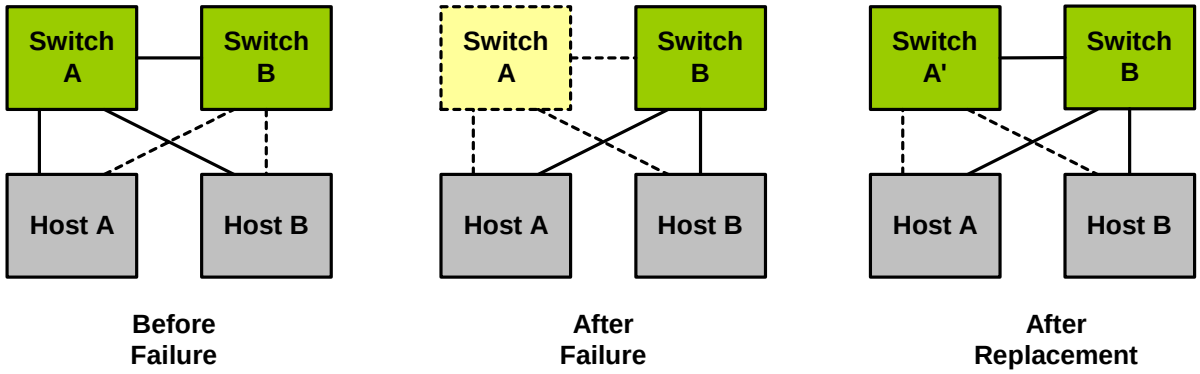


Figure 4-6

This is achieved through the use of DHCP. When a switch becomes a VRRP Master, a DHCP server is started with a pointer to a configuration file that contains configuration information for its partners. The replacement switch comes up running DHCP client to retrieve its configuration.

Configuring Surviving Partner

The `S60SP_startup` script is useful in setting up proper switch replacement. By factory installing the `S60SP_startup` script in replacement switches, the replacement switches will boot looking for a Master switch configuration. The `S60SP_startup` works as follows:

1. It first looks for a local file `/etc/rcZ.d/surviving_partner/zsp.primary.conf`. If this file exists, it is used to configure the switch. Only the originally configured switch or *Central Authority* should contain this file. See Central Authority later in this Chapter for more information.

2. Next it uses *zspconfig –u* to attempt to contact a running Master switch to retrieve the proper configuration. This is the normal case for a replacement switch.

3. Finally, it lets the currently saved `S70Surviving_Partner` script execute. This case would be the case of a power up of an already configured backup switch when the other HA switch is unavailable. This case could occur after losing power to the entire chassis.

Proper configuration of Surviving Partner requires coordinated configuration of many different processes, including *vrrpd*, *zlmd*, *zlc*, and *dhcpd*. The daemon processes run scripts to perform their actions. Because these scripts are complex and inter-dependent, a configuration application called *zspconfig* is used to build them.

NOTE: You must have an OpenArchitect/HA Suite RTU license for the chassis to use any OpenArchitect High Availability functionality or component on the switch within the chassis, or use OpenArchitect/Node on any of the host cards. This includes executing *zspconfig.* Please contact your ZNYX Sales Representative, call (800) 724-0911, or send email to sales@znyx.com, or support@znyx.com.

The basic steps to configuring Surviving Partner are:

- Determine your desired configuration.
- Modify the configuration file (`/etc/rcZ.d/surviving_partner/zsp.conf` is the default) to use as input to the configuration utility (*zspconfig*).
- Configure startup scripts or other scripts such as *gated* routing scripts and *vrrp* configuration scripts.
- Run `zspconfig` on the Master system.
- Run `zspconfig -u` on the Backup/Sibling system(s).

zspconfig

Zspconfig performs the job of building the scripts based on a provided input file locally, or from a remote machine. The result of *zspconfig* is to create several configuration files and runtime shell scripts, and optionally start the Surviving Partner processes. Scripts are generated for configuring VLANs, starting the network, and starting the *vrrpd* and *zlmd* daemons.

Zspconfig can also used by sibling backup switches to retrieve configuration from the Surviving Partner and start the *vrrpd* and *zlmd* daemons. *Zspconfig* is generally only run once to configure Surviving Partner. *Figure 4-7* shows the scripts generated by *zspconfig*.

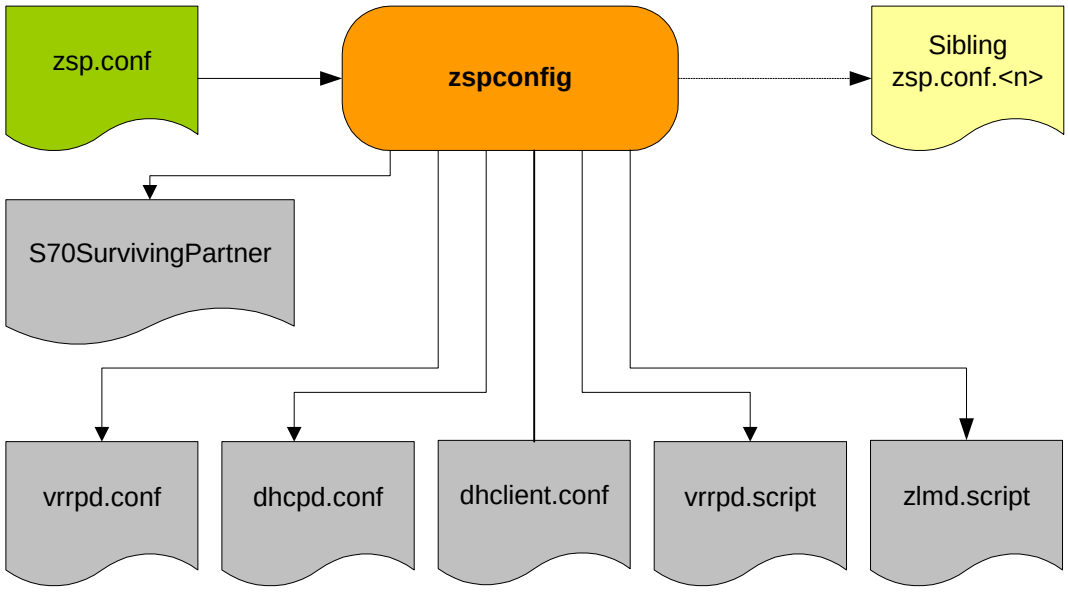


Figure 4-7

The configuration and runtime scripts created are as follows:

S70Surviving_partner

Switch initialization script that is run at boot time. This script will restart the switch with the original configuration given to *zspconfig*. Optionally, *zspconfig* will run this script from the initial invocation.

zsp.conf.<n>

zspconfig configuration file that contains the configuration of the sibling backup switches. The <n> is used to distinguish potentially more than one backup switch. This configuration file is placed in /tftpboot, and is retrieved via DHCP by a replacement switch on boot up.

vrrpd.conf

Configuration script for the VRRP daemon. This configuration is used when the S70Surviving_partner script launches *vrrpd*. There is a line in this file for each virtual router address *vrrpd* will manage.

dhcpd.conf

Configuration script used by *dhcpd* when the switch becomes Master. *dhcpd* is used to serve replacement switches their configuration scripts. Namely a zsp.conf file that can be input to *zspconfig* with the -u flag.

dhclient.conf

If *zspconfig* is executed with the -u flag, a dhclient.conf file is created, and then dhclient is used to retrieve a *zspconfig* configuration file from the /tftpboot area of the Master switch.

vrrpd.script

Runtime script that executes each time the *vrrpd* changes state. This script starts and stops *dhcpd*, and toggles down RAINlink ports to force the RAINlink nodes to a new Master switch.

zlmd.script

Runtime script executed by *zlmd* when a link goes up or down. This script modifies the priority of the *vrrpd* that in turn may cause the VRRP Master to move from one sibling switch to another.

After the scripts are created, *zspconfig* may run the `S70Surviving_partner` script to start the Surviving Partner tasks. The tasks started are *vrrpd*, *zlmd*, and *dhcpd*.

The *vrrpd* and *zlmd* daemons run scripts to perform their actions. When *vrrpd* changes state between Master and Backup, it runs a script that starts and stops *dhcpd*. When *zlmd* sees a link go up or down, it runs a script that communicates with *vrrpd* via *vrrpconfig*.

Figure 4-8 illustrates the tasks running on a pair of switches configured in a Surviving Partner setup. In this illustration, the upper switch is the Master and the lower is Backup.

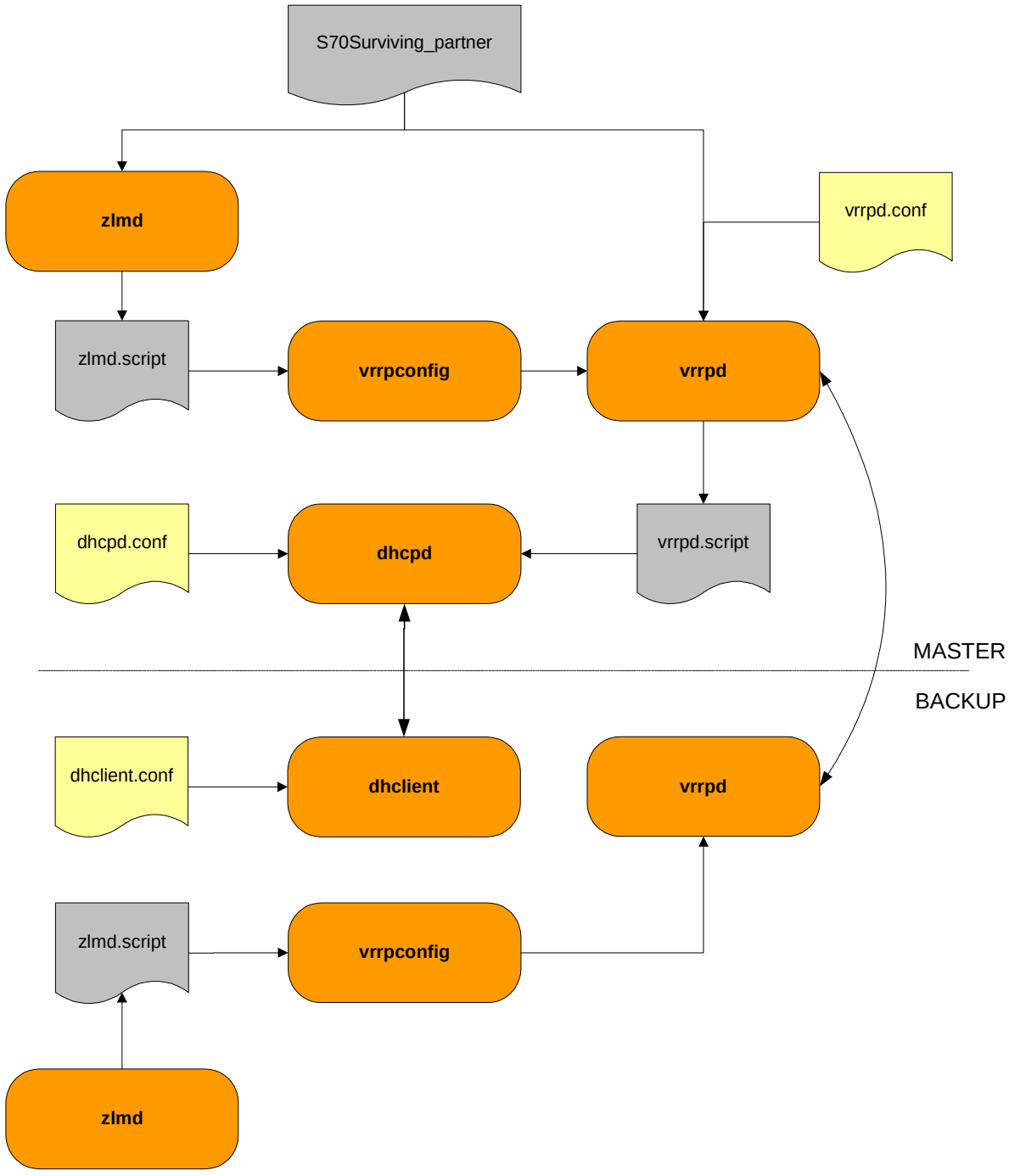


Figure 4-8 Surviving Partner Overview

Example HA Chassis Configuration

The following walks through a basic Surviving Partner configuration typical for an HA setup. Consider an HA chassis with four hosts, such as single-board CPU's, and two switches configured for Surviving Partner. Each of the hosts has two Ethernet ports providing a link to each of the switches. Each switch furnishes an uplink from the HA chassis out of the box.

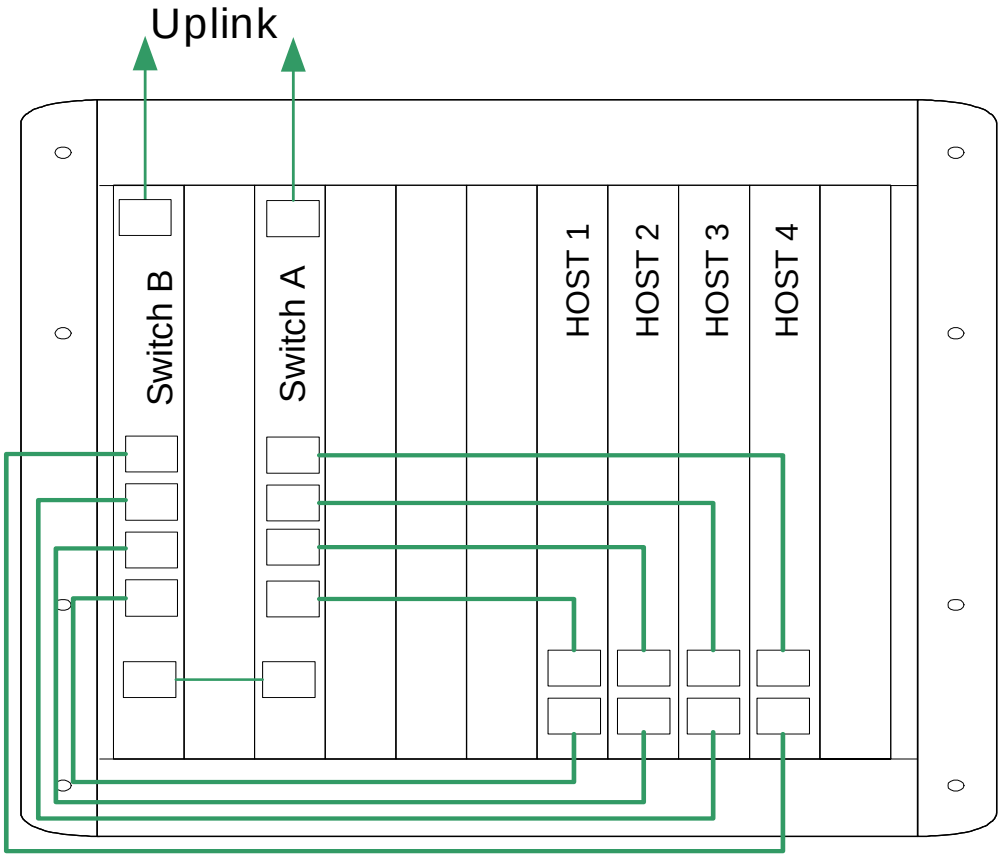


Figure 4-9 HA Chassis

Each host runs ZNYX OA Node software with embedded RAINlink technology with the two ports configured for failover. OA Node provides one virtual interface (zrl0) to the host protocol stack. One physical port (znb) is connected to the Master Switch, and one physical port is connected to the Backup Switch. An interlink provides communication between the switches. *Figure 4-10* shows detail of the two switches and a representative host configuration.

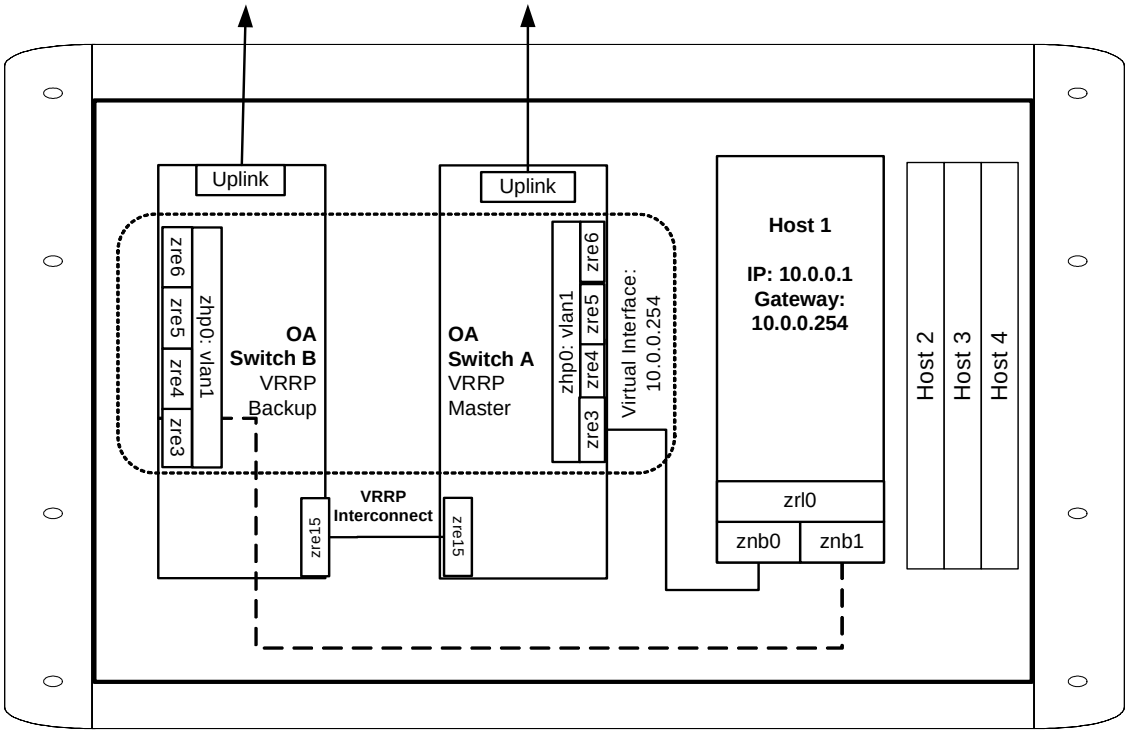


Figure 4-10

In the above configuration, the hosts are all configured on the same VLAN. The OpenArchitect switches will be configured with ports 1-4 in a single VLAN, and the two switches will be configured as Surviving Partners, using VRRP to form a single virtual interface to the hosts. The inter-connect has it's own VLAN. The Uplink ports can be configured many different ways, either as part of the same VLAN, a different VLAN, or a different VLAN running VRRP.

Modifying zsp.conf

An example file for setting up zspconfig is `/etc/rcZ.d/surviving_partner/zsp.conf`. The following will document the settings to prepare Switch A and Switch B of the *Figure 4-10*.

On Switch A (Master), make a backup copy of `zsp.conf`, and edit `zsp.conf`:

```
cd /etc/rcZ.d/surviving_partner
cp zsp.conf zsp.conf.save
vi zsp.conf
```

The first thing you need to do is add your Run Time User's (RTU) license required to utilize the HA features to zsp.conf. Replace the "xxxx" value with your license, and uncomment the line by removing the "#" sign.

```
# Copyright 2002 ZNYX Networks, Inc
```

```
# All Rights Reserved.
#
# You must have an OpenArchitect/HA Suite RTU license for the chassis
# to use any OA high availability functionality or component on the
# switches within the chassis or use OpenArchitect/Node on any of the
# CPU host cards.  This includes executing this utility script.

# Please uncomment the line below and replace 'xxxx' with your
# OpenArchitect/HA license key.  It must be terminated with a ';'

# OAHA_RTU: xxxx;
```

The first section uses *zconfig* to create the VLANs. Create VLAN1 for the Hosts1-4 in the HA chassis (ports 3 through 6; ports 0, 1, and 2 are on the front panel), and VLAN100 for the VRRP Interconnect (port 15). This example uses untagged frames, so untag the ports:

```
zconfig zhp0: vlan1= zre3,zre4,zre5,zre6,zre15;
zconfig zre3,zre4,zre5,zre6=untag1;
zconfig zhp1: vlan100=zre15;
zconfig zre15=untag100;
```

Notice that the interconnect port (zre15) is also a member of VLAN1. If you are going to use *vlan* or *port* switching modes, each VLAN that presents a virtual address must include the interconnect port.

The next section sets up the physical IP addresses to use for the Master and each of the Sibling (Backup) switches. The Master provides the addresses to the Siblings on a first come, first serve basis. Note that the physical IP address should be different from the virtual IP address that spans the entire Sibling (Virtual Router) group. Once configured, the Sibling group appears as one connection point to other hosts on the VLAN. You need to supply an IP address for each interface on each switch, including the interconnect.

```
sibling_addresses: zhp0 = 10.0.0.100, 10.0.0.101;
sibling_addresses: zre15 = 192.168.0.100, 192.168.0.101;
```

Now configure the virtual address for each sibling group. In our diagram, we are going to create a virtual interface across VLAN1 of each switch. To Hosts1-4 on that VLAN, this provides a single point to connect to, or route through. For the setup above,

```
vrrp_virtual_addresses: zhp0 = 10.0.0.254;
```

Next come port definitions, as defined on the *zspconfig* man page. Since our hosts are connected using RAINlink, we will want to choose RAINlink on each of the ports in VLAN1 on each switch, and interconnect for the interconnect port on each switch,

```
interconnect: zre15;
RAINlink: zhp0;
```

Next come special modes for VRRP for use when more than one pair of ZNYX switches are connected to another pair on ZNYX switches in a redundant configuration. The intent of these

modes is to provided Spanning Tree like capabililties eliminating network loops between pairs of Surviving Partner configurations, as well as expedite address learning between the two pairs of switches:

block_crossconnect

Causes the equivalent of Spanning Tree blocking on the cross-connect ports of the VRRP backup. This mode requires that all switches be ZNYX switches.

RAINlink_xmit_on_failover

Requires that the Node Adapter Cards connected to RAINlink ports transmit a packet when failing over to expedite MAC address learning. An example would be the ZNYX Network's RAINlink enabled OANode drivers in SNAP_BCAST_MODE, or a gratuitous ARP.

The next sections are optional. The first determines the failover mode between the Surviving Partner switches. There are three modes:

switch	The switch with the most links becomes the Master (default)
vlan	The switch with the most up links in the VLAN becomes the Master. *NOTE: This requires an in-band port as the interconnect.*
port	The Master will remain the Master until all ports in the VLAN are down. The Backup then becomes the new Master. *NOTE: This requires an in-band port as the interconnect.*

The remaining entries provide a mechanism to propagate files and/or startup scripts to sibling switches. An example might be startup scripts or scripts to configure gated. Example scripts are included to start gated with RIP1, RIP2, or OSPF setup. You must use absolute path names.

Once the configuration files are complete, run the *zspconfig* utility on the Master to configure all the scripts:

```
zspconfig -f zsp.conf
```

Once configuration is complete, insure there are no superfluous S-type startup scripts in /etc/rcZ.d, and *zsync* your switch to save your configuration.

Now go to the backup switch and run *zspconfig –u* to get the appropriate configuration information from the Master,

```
zspconfig -u zhp0
```

Central Authority

Modifications can be made to the `S60SP_startup` script to use a third machine running DHCP that is not part of the Surviving Partner pair. The third machine is referred to as the *Central Authority*.

Setup requires a DHCP daemon configuration file on the Central Authority and a *dhclient* configuration file for each of the two Surviving Partner switches in the pair. The format of the DHCP daemon configuration file is dependent on the machine and operating system being used. An example can be obtained from the Surviving Partner primary switch in the location `/etc/rcZ.d/surviving_partner/dhcpd.conf`.

This configuration will contain configuration for only one of the two Surviving Partner switches. It must be edited. For example:

```
subnet 100.0.0.0 netmask 255.0.0.0 {
        option broadcast-address 100.255.255.255;

        host ZNYX1 {
                    fixed-address 100.0.0.31;
                    option dhcp-client-identifier "ZNYX";
                    option vendor-encapsulated-options "zsp.conf.1";
        }
}
```

A second host entry must be added with unique information.

```
subnet 100.0.0.0 netmask 255.0.0.0 {
        option broadcast-address 100.255.255.255;

        host PRIMARY {
                    fixed-address 100.0.0.30;
                    option dhcp-client-identifier "PRIMARY";
                    option vendor-encapsulated-options
   "zsp.primary.conf";
        }
        host SECONDARY {
                    fixed-address 100.0.0.31;
                    option dhcp-client-identifier "SECONDARY";
                    option vendor-encapsulated-options
   "zsp.secondary.conf";
        }
}
```

The `zsp.primary.conf` and `zsp.secondary.conf` files must be placed in the *tftp* location on the machine, often `/tftpboot`. The `zsp.primary.conf` and `zsp.secondary.conf` files can be retrieved from the Surviving Partner switches. This is the configuration that will be given to the switches. It is recommended that the `zsp.conf` be taken from the primary as follows:

1. The `zsp.conf` file created by hand on the primary is moved to `/tftpboot/zsp.primary.conf` on the Central Authority.

2. Move `/tftpboot/zsp.conf.1` file on the primary created by *zspconfig* to `/tftpboot/zsp.secondary.conf` on the Central Authority.

3. Create `dhclient.conf` files on the Surviving Partner switches. Examples can be found in `/etc/rcZ.d/surviving_partner/dhclient.conf`. As an example:

```
send dhcp-client-identifier "ZNYX";
request vendor-encapsulated-options;
require vendor-encapsulated-options;
```

4. Modify the `dhclient.conf` file on the primary switch as follows:

```
send dhcp-client-identifier "PRIMARY";
request vendor-encapsulated-options;
require vendor-encapsulated-options;
```

5. Modify the `dhclient.conf` file on the secondary switch as follows:

```
send dhcp-client-identifier "SECONDARY";
request vendor-encapsulated-options;
require vendor-encapsulated-options;
```

The last step is to modify the startup scripts that run *zspconfig* to use the -c option. The -c option allows you to provide a `dhclient.conf` script rather then having *zspconfig* create a default. For example, the `S60SP_startup` script line that reads:

```
echo y y n | zspconfig -t 10 -su zhp0 > /dev/null 2>&1
```

Can be modified to

```
echo y y n | zspconfig -c
   /etc/rcZ.d/surviving_partner/dhclient.new.conf -t 10 -su zhp0 >
   /dev/null 2>&1
```

If you use `S60SP_startup`, the `/etc/rcZ.d/surviving_partner/zsp.primary.conf` file sould not exist. This way the `S60SP_startup` script will first look at the Central Authority. If the Central Authority is down, then it will use its current configuration.

Chapter 5 Switch Administration

One of the main benefits of the OpenArchitect switch is that it runs Linux, so much of the switch administration is already familiar to most network or system administrators. ZNYX recommends complementing these instructions with a standard Linux reference guide, such as *Linux Network Administrator's Guide* available from O'Reilly. Below are brief descriptions of some of the more routine administrative task pertinent to the switch.

Setting the Root Password

The switch is shipped with a default user root and no password. To set the root password, use the password command:

```
sh-2.04# passwd
Changing password for root
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Enter new password:
Re-enter new password:
Password changed.
sh-2.04#
```

NOTE: Even when just changing the password, you need to save the file system overlay with the *zsync* command, or you will lose your changes upon reboot.

Adding Additional Users

Additional users can be added with the *adduser* command. Additional users are desirable for connecting to the switch via *ftpd* and other daemons that require a login other than root and a password. To create a user named guest, run *adduser*

```
sh-2.04# adduser guest
Changing password for guest
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Enter new password:
Re-enter new password:
Password changed.
sh-2.04# zsync
sh-2.04#
```

Setting up a Default Route

If you wish to access the switch from some place other than a directly attached network, you may want to setup a default route. Use the route command to set a default gateway.

```
route add default gw 10.0.0.254
```

Put the entry into the `/etc/init.d/rcS` startup script to automatically set a default route upon reboot.

Name Service Resolution

Name service lookups will be done locally using `/etc/hosts`. You can also tell the switch which name server to use by including an entry in `/etc/resolv.conf`.

DHCP Client Configuration

A utility is included to dynamically obtain the IP address of the OpenArchitect switch interfaces. To set the the IP address dynamically, execute the command,

```
dhclient zhp0
```

The default device name, *zhp0*, works with the default configuration of the OpenArchitect switch and will attempt to obtain an IP address from the local DHCP server. To use DHCP to set your IP addresses automatically on boot up, uncomment the the following line in `/etc/init.d/rcS` by removing the # sign

```
/usr/sbin/dhclient zhp0
```

DHCP Server Configuration

The OpenArchitect switch includes a DHCP server. To start the DHCP server, configure `/etc/dhcpd.conf` for your network, and run

```
dhcpd
```

Consult Linux Network administration manuals for more information on DHCP and configuration options.

To use DHCP to set your IP addresses automatically on boot up, uncomment the the following line in `/etc/init.d/rcS` by removing the # sign

```
dhcpd
```

Network Time Protocol (NTP) Client Configuration

NTP is a protocol for setting the real time clock on a system. There are numerous primary and secondary servers available on the network. For more NTP information, and a list of available NTP servers, see the following URL:

http://www.eecis.udel.edu/~ntp

You will need to have your network settings properly configured to reach an available NTP server on your local network or the internet. To set the time and date, execute *ntpdate* with the server of your choice. For example,

```
ntpdate –u ntp.ucsd.edu
```

The *–u* is required if the OpenArchitect switch is operating behind some types of firewalls.

If you wish for *ntpdate* to set your date and time automatically each time you boot, uncomment the example *ntpdate* command line in `/etc/init.d/rcS` by removing the # sign. *ntpdate* returns the Universal Time (UTC, formerly Greenwich Mean Time, or GMT). To display the localtime, set the TZ variable to the appropriate name and the number of hours offset from UTC. For instance,

```
export TZ=PST8
```

for Pacific Standard Time offset from UTC by 8 hours. To set an environment variable, add the entry to /etc/profile. Remember to *zsync* to make your changes permanent.

Network File System (NFS) Client Configuration

The OpenArchitect switch includes an NFS client for mounting remote file systems. You will need to start NFS server processes in order to use NFS. You will need to start the following servers:

```
/sbin/portmap
/sbin/rpc.statd
/sbin/rpc.mountd -r
```

Once the above servers are started, you can mount a remote NFS file system.

```
mount rhost:nfs_file_system local_mount_point
```

If the remote NFS file system you're mounting is on an OA switch, you should mount with caching disabled.

```
mount rhost:nfs_file_system –o noac local_mount_point
```

All the necessary servers are included in `/etc/init.d/rcS` but are commented out by default. To automatically start all NFS client services each time you boot, uncomment the NFS Client servers. Go to the `/etc/init.d/rcS` file.Uncomment the following command lines by removing the # sign.

```
/sbin/portmap
/sbin/rpc.statd
/sbin/rpc.mountd -r
```

You can also include commands to mount remote NFS file systems at boot time. There is an example line included at the appropriate location in `/etc/init.d/rcS`. Uncomment and alter the mount command included for your particular configuration. Note: A “sleep” of 5 seconds is included to allow time for the links to come up prior to attempting the mount.

```
sleep 5
mount 10.0.0.1:/nfs -t nfs -o noac /mnt
```

NFS Server Configuration

The switch also contains an NFS server so that you can mount the switch file system from other systems. To enable the NFS server, first follow the steps to enable the NFS client. Then, edit `/etc/exports` to include the file systems you wish to export. Consult a standard Linux Network Administrator’s Guide (or man pages) regarding options for exported file systems. Generally, an entry in `/etc/exports` looks like the following:

```
/nfs        *.localdomain.com(ro)
```

Now start *nfsd* to export the mount points and begin answering requests from remote clients.

```
/sbin/rpc.nfsd -r
```

To export file systems automatically on boot, edit `/etc/init.d/rcS`, uncomment the `/sbin/rpc.nfsd` command line by removing the `#`.

```
/sbin/rpc.nfsd -r
```

Connecting to the Switch Using FTP

Use ftp to transfer files to or from the switch. See the Linux Reference Guide for details of the ftp command. In general, you can use ftp to connect to any system running an ftp server, including other OpenArchitect switches, to either get (transfer files from the remote host to the switch) or put (transfer files from the switch to the remote host) files.

```
ftp <remote_host>
```

FTPD Server Configuration

The switch itself can also be configured to run an FTP server (ftpd). See the Linux Reference Guide for details of the *ftpd* command. You will need to add a user to the switch in order to connect via ftp from a remote host, since root is not allowed ftp access. See the earlier section in this chapter regarding how to add a user. The ftp daemon is started by default. If you wish to shutdown the ftp daemon, comment out the `betaftpd` line in `/etc/init.d/rcS`.

Connecting to the Switch Using TFTP

Trivial File Transfer Protocol or TFTP, is a very simple protocol used to transfer files. It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP, like user authentication. You can use *tftp* to connect to any system running a tftp server (*tftpd*) including other OpenArchitect switches.

```
tftp <remote_host>
```

TFTPD Server Configuration

The TFTP server is started by *inetd(8)* using the configuration set up in `/etc/inetd.conf`. The use of *tftp(1)* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. The default location of these files is `/tftpboot`.

SNMP Agent

Simple Network Management Protocol (SNMP) is the de facto standard for network management. An SNMP agent maintains a structure of data for a network device in a virtual information database, called a Management Information Base (MIB). A network management station is capable of accessing the MIB of the network device to monitor and configure the network device.

The OpenArchitect switch utilizes the NET-SNMP (formerly UCD-SNMP) agent core. Additional information on the agent can be found at: http://www.net-snmp.com. The OpenArchitect switch agent will respond to SNMPv1, SNMPv2, and SNMPv3 requests.

Protocols supported on the OpenArchitect switch by *gated*, such as RIP and OSPF communicate with SNMP agent via the SNMP Multiplexing (SMUX) protocol.

Supported MIBS

OpenArchitect includes MIB support as documented by each of the RFCs listed. The MIBs themselves are located on the switch in the /usr/share/snmp/mibs directory.

RFC 1155:	Structure and Identification of Management Information for TCP/IP-based Internets
RFC 1227:	SNMP MUX Protocol and MIB
RFC 1493:	Definitions of Managed Objects for Bridges (obsoletes RFC 1286)
RFC 1657:	Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMI-V2
RFC 1724:	RIP Version 2 MIB Extenstion (obsoletes RFC 1389)
RFC 1850:	OSPF Version 2 Management Information Base (obsoletes RFC 1253, which obsoletes RFC 1252, which obsoletes RFC 1248)
RFC 2011:	SNMPv2 Management Information Base for the Internet Protocol Using SMIv2

RFC 2012:	SNMPv2 Management Information Base for the Transmission Control Protocol Using SMIv2
RFC 2012:	SNMPv2 Management Information Base for the User Datagram Protocol Using SMIv2
RFC 2013:	Management Information Base for Network Management of TCP/IP-based Internets: MIB-II (obsoletes RFC 1213, which obsoletes RFC 1158)
RFC 2021:	Remote Network Monitoring Mangement Information Base Version 2
RFC 2096:	IP Forwarding Table MIB
RFC 2571:	An Architecture for Describing SNMP Management Frameworks
RFC 2572:	Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)
RFC 2573:	SNMP Applications
RFC 2574:	User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)
RFC 2575:	View-based Security Model (VACM) for version 3 of the Simple Network Management Protocol (SNMP)
RFC 2576:	Coexistence between Version 1, Version 2 and Version 3 of the Internet-standard Network Management Framework
RFC 2665:	Definitions of Managed Objects for Ethernet-like Interfaces
RFC 2674:	Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions
RFC 2742:	Definitions of Managed Objects for Extensible SNMP Agents
RFC 2787:	Definitions of Managed Objects for the Virtual Router Redundancy Protocol
RFC 2819:	Remote Network Monitoring Mangement Information Base
RFC 2863:	The Interfaces Group MIB (obsoletes RFC 2233, which obsoletes RFC 1573, which obsoletes RFC1229)
RFC 2932:	IPv4 Multicast Routing MIB
RFC 3165:	Definitions of Managed Objects for the Delegation of Management Scripts
RFC 3231:	Definitions of Managed Objects for Sceduling Management Operations
ZNYX Networks Private MIB	Custom ZNYX MIB to support software and hardware features not covered by standard MIBs.
UCD-SNMP Enterprise MIB	UCD-SNMP MIB related to management and monitoring of the LINUX host

Supported Traps

Upon certain events, the OpenArchitect switch can be configured to send notification of the event, called an SNMP Trap, out to a defined recipient/manager or managers. Traps are not issued in real time. OpenArchitect will send SNMP traps for the following conditions:

SNMPv2-MIB:	coldStart

SNMPv2-MIB:	authenticationFailure
IF-MIB:	linkUp
IF-MIB:	linkDown
UCD-SNMP-MIB:	ucdShutdown
RMON-MIB:	risingAlarm
RMON-MIB:	fallingAlarm
VRRP:	vrrpTrapNewMaster
VRRP:	vrrpTrapAuthFailure
EGP (rfc1213):	egpNeighborLoss
BGP4-MIB:	bgpEstablished
BGP4-MIB:	bgpBackwardTransition

SNMP and OpenArchitect Interface Definitions

OpenArchitect, defines three types of devices:

```
zre    physical port

zrl    trunk of ports

zhp    interface consisting of ports (zre's) and trunks of ports
       (zrl's)
```

A `zrl` (trunk device) is treated as an aggregate of its constituent `zre's` (ports). A `zhp` is an aggregate of its immediately contributing sub-interfaces (`zre's` and `zrl's`). The ports that make up a trunk do not contribute to the `zhp`.

The administrative status of a `zre` and `zhp` are independent of each other. If the administrative status is down, then the operational status will be down independent of the underlying link state. You must ifconfig up the `zre's` to see the operational link status for a `zre`. When the administrative status is up, the operational status is dependent on the underlying physical state. For example, if `zhp0` contains `zre1` and `zre2` the following would be true for the operational status given the administrative status is up on `zre1`, `zre2`, and `zhp0`:

Physical Link Status		**SNMP Operational Status**		
zre1	**zre2**	**zre1**	**zre2**	**zhp0**
down	down	down	down	down
down	up	down	up	up
up	down	up	down	up
up	up	up	up	up

The administrative status is directly controlled by ifconfig up/down. The administrative status of the zhp's and zre's do not affect each other.

ifStackTable Entries

The example ifStackTable MIB entries shown demonstrates the relationship:

```
ifMIB.ifMIBObjects.ifStackTable.ifStackEntry.ifStackStatus.0.1 =
    active(1)
ifMIB.ifMIBObjects.ifStackTable.ifStackEntry.ifStackStatus.0.2 =
    active(1)
```

If they are X.Y then

- if X = 0 there is nothing "above" this interface
- if Y = 0 there is nothing "below" this interface

otherwise interface X has interface Y as a logical constituent.

SNMP Configuration

The SNMP agent is called *snmpd* and is started by default from the Linux boot up script `/etc/init.d/rcS`. If you do not wish to start *snmpd*, comment out the `snmpd` line in `/etc/init.d/rcS`.

Configuration of the OpenArchitect switch SNMP agent is the same as configuration of any standard Linux host that uses the NET-SNMP agent. Configuration information for persistent data and security information is kept in `snmpd.conf` under the default SNMP configuration location, which for the OpenArchitect switch is `/usr/share/snmp`. `snmpd.conf` is the location to change “sys” information such as the *syslocation* and *syscontact*, as well as permissions such as the *rocommunity* or *rwcommunity*. Note: For NET-SNMP agents, these objects (sysLocation.0, sysContact.0 and sysName.0) ordinarily are read-write. However, specifying the value for one of these objects by giving the appropriate token in `snmpd.conf` makes the corresponding object read-only, and attempts to set the value of the object will result in a notWritable error response.

The processing for link up and link down traps is now user configurable. As the default, traps conform to RFC2863, meaning the trap contents will include:

```
ifIndex, ifAdminStatus and ifOperstatus
```

You can alter this behavior by specifying:

```
cisco_link_traps on
```

If cisco_link_traps are turned on as described then link up and link down traps will have a cisco-like format and the trap contents will include:

```
ifDescr and ifType
```

Examine and edit `/usr/share/snmp/snmpd.conf` appropriately for your configuration. Information in `/usr/share/snmp/snmpd.conf` is only read at startup - or when the daemon is forced to reread its configuration. See the standard Linux man page for `snmpd.conf` for more details.

SNMP Applications

The OpenArchitect switch includes the *snmpget*, *snmpwalk*, and *snmpset* applications you can use these standard Linux utilities to test your SNMP agent. For example,

```
snmpwalk localhost -c public
```

walks the entire MIB of the localhost (ie OpenArchitect switch) starting at the top of the MIB. See the *Linux Reference Man Pages* for the usage of the SNMP utilities.

MIB values are decoded from their numerical representations into readable text by parsing MIBs located in `the /usr/share/snmp/mibs/` directory. If you need to add a MIB, add it to that directory and *zsync* to save across reboots.

Port Mirroring

Zmirror sets packet mirroring from a given set of ports to a given port. Turning on packet mirroring causes a copy of the packet to be sent to the *mirror_to* port. There can be only a single *mirror_to* port, but there can be multiple *mirror_from* ports. The *zmirror* command overwrites the *mirror_to* port. The *zmirror* command accumulates the *mirror_from* ports. Note, there are performance issues when trying to mirror more bandwidth than is available on the *mirror_to* port. Use the *zmirror* command in the following way,

```
zmirror mirror_from mirror_to
```

After executing the following three commands, packets received on ports 0, 1 and 2 would be mirrored (copied and transmitted) to port 12. This mirroring would be in addition to any Layer 3 or Layer 2 switching.

```
zmirror 0 12
zmirror 1 12
zmirror 2 12
```

To clear the current mirroring use the *-t* option. The *-e* option can be used to indicate that packets being sent on a given port should be copied to the *mirror_to* port. For example if the *-e* option is used as follows, the packets transmitted, as opposed to received, on ports 0, 1 or 2 would be mirrored to port 12.

```
zmirror -e 0 12
zmirror -e 1 12
zmirror -e 2 12
```

Link and LED Control

The *zlc* application sets the link speed and state of individual ports of the switch, or display their current state. It can also set or clear the external fault led, the internal fault led, or to set a port down or up. To force the link on port 0 down,

```
zlc zre0 down
```

To check the status of a link,

```
zlc zre0 query
```

To check the status of all links,

```
zlc zre0..15 query
```

Link Event Monitoring

The *zlmd* application is intended to run as a daemon, waiting for a configured event to occur and then running the program configured for that event. The events monitored are changes in the link status at any of the 16 ports of the switch, the start of removal of the switch from the cPCI backplane, or the cancellation of the removal before it actually takes place. The program can be a shell script that initiates appropriate actions to respond to the event.

Web Interface (thttpd)

Web management is provided for basic configuration and monitoring of OpenArchitect. By default, the *thttpd* daemon is not started. Documentation for this program can be found online at http://www.acme.com.

Starting thttpd

To start the daemon, enter the following command exactly as shown or the web interface will not function properly.

- Change directories to root

```
cd /
```

- Enter the following command

```
/usr/sbin/thttpd -r -u root -c "/http/cgi-bin/*"
```

By adding the previous command to /etc/init.d/rcS, the web server will start automatically when the switch is booted. Once *thttpd* is running, you can connect to OpenArchitect from your browser.

Restricting Access

By default, there is no password protection for the web interface. To setup password protection, follow these steps.

- Change directories to root

```
cd /
```

- Create a password file and a new user

```
htpasswd –c .htpasswd root
```

Notice the "." in front of the file name. Enter and confirm the password for the new user when prompted.

The password protection will only work properly when the password file is located in the root directory. Once a new user is created, reconnect to the web interface through your browser. You will be prompted for a username and password.

Web Interface

Figure 5-1 shows the opening screen for the web interface. There are five buttons available with the following function: From left to right they are System, Ports, Vlan, IP, and Command. The default page loaded is the System page. By clicking on any of the buttons, a page will be displayed.

System	This is the default screen. The system page displays general information about the switch.
Ports	Displays the current link status for all ports. The zre, duplex setting, duplex state, and transmit and receive information is all displayed.
Vlan	Displays the current VLAN configuration on the switch. It displays which ports are assigned to which VLANs, and which ports are tagged or untagged. It also displays any trunks that have been created.
IP	Displays the current ip addresses assigned to interfaces. This information is obtained via SNMP.
Command	Allows the user to enter any commands as if they were at the console. On the left, there is a list of links that are the most common commands. By clicking one of these, the information will be displayed on the right. A command can also be entered manually. To keep someone from executing a command that might cause harm, all commands are checked against a list on the switch. To add more commands, they can be added to the /http/html/comcheck.html file. Only then can they be executed in the web interface.

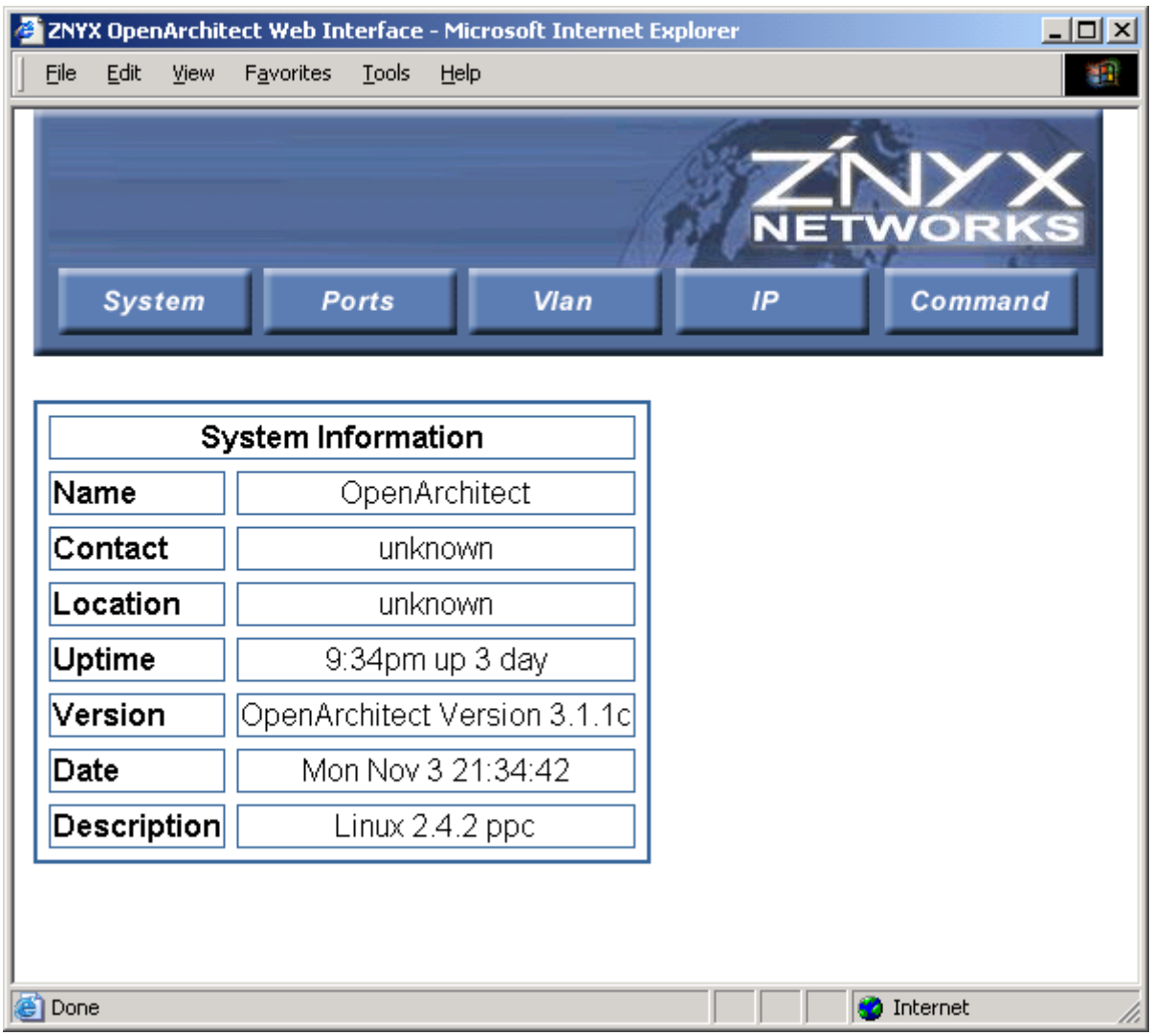


Figure 5-1 Web Interface Main Screen

Customizing the Web Interface

Web pages can easily be added, modified, or deleted from the OpenArchitect web interface. All files are located under the /http directory.

Chapter 6 Switch Maintenance

This chapter includes basic information about the OpenArchitect switch environment including an overview of the file system structure, modifying and updating switch files, upgrading the switch driver and kernel, and implementing a system recovery.

Overview of the OpenArchitect switch boot process

The OpenArchitect switch is equipped with a Random Access Memory (RAM) disk and three Read Only Memory (ROM) devices, including, a boot ROM and two application flash (see Figure 6-1).

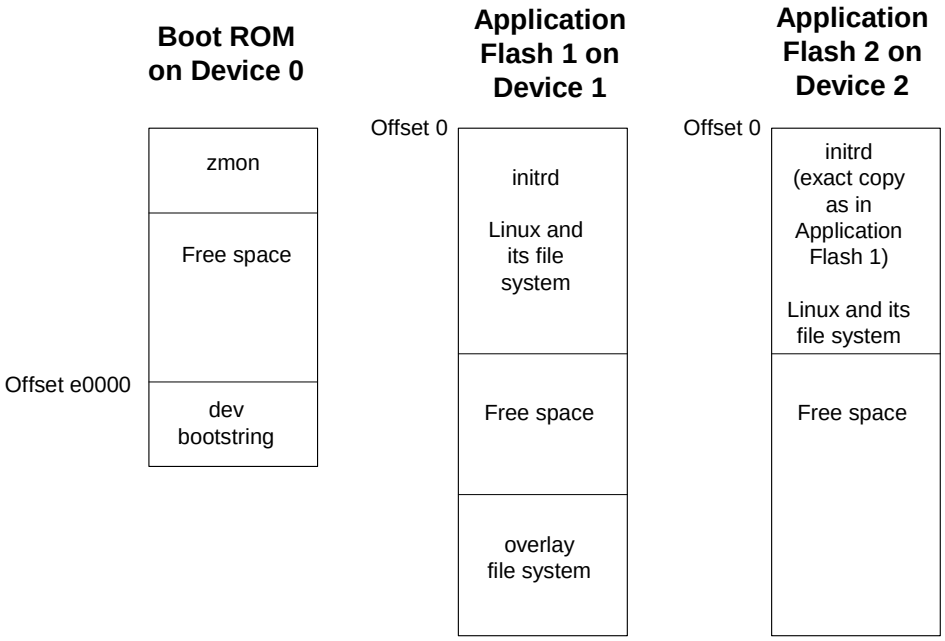


Figure 6-1 ROM Devices in OpenArchitect

The boot ROM is located on device 0 and contains the OpenArchitect *zmon* application that operates as a boot loader and includes a device bootstring. Device 1 contains the application flash 1 image of the Linux operating system and the OpenArchitect overlay file system. Application flash 1 is the primary working image for the switch. Device 2 contains the application flash 2 that is an exact copy of application flash 1. You would only boot from this device if application flash 1 is corrupted and you need to restore the switch to the factory-shipped configuration. See Page 79, *Booting the Duplicate Flash Image*.

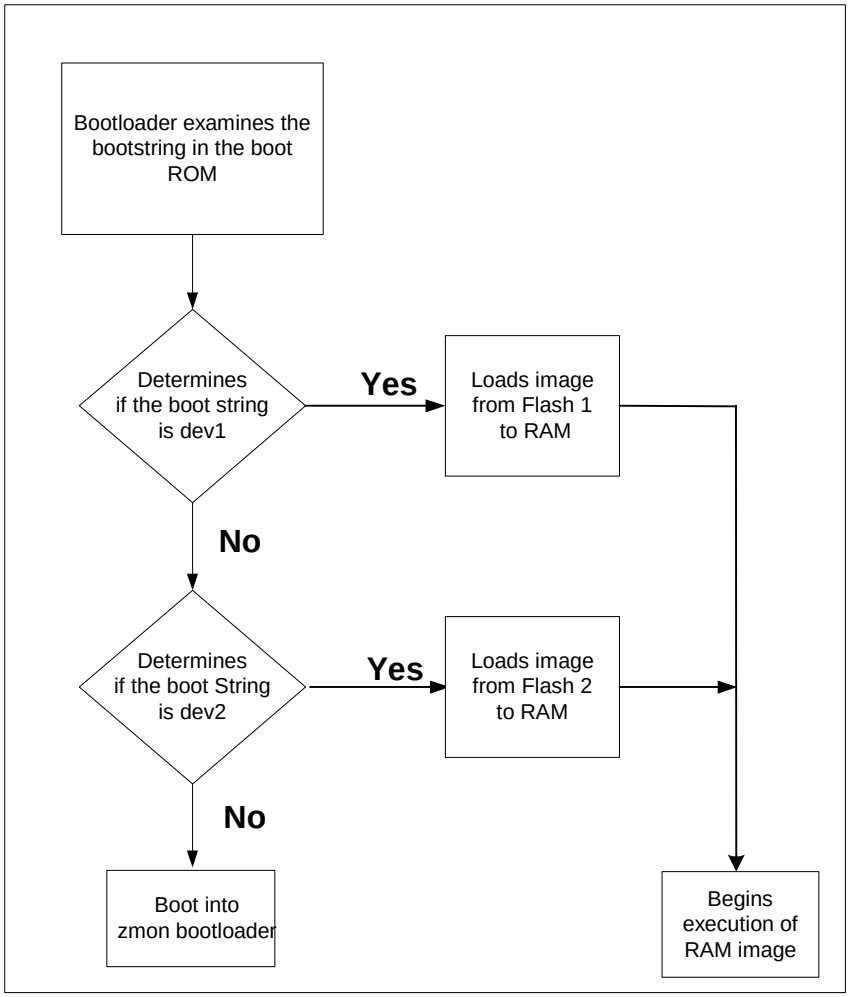


Figure 6-2 Booting up Process Flow

Under normal circumstances, the booting up process follows the process outlined in Figure 6-2. During boot up, the *zmon* bootloader reads the device bootstring to locate and validate the correct application image to load. The bootstring command is in the following format:

boot : X | [<options>] X represents the device value 0, 1 or 2

The boot process opens and uncompresses the *initd* image onto the RAM disk. Then *zmon* begins booting the Linux image. After Linux boots, the *init* process executes the `/etc/init.d/rcS` script which, in turn, executes `/etc/rcZ.d/rc` (see *Figure 6-3)*). The `/etc/rcZ.d/rc` script runs *S** files in `/etc/rcZ.d`, with the start parameter. The S* files are the switch configuration files (e.g., S50layer2).

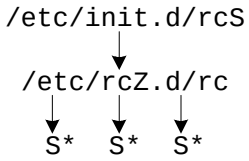


Figure 6-3 Init Script Flow

Saving Changes

Any modifications made to the scripts for your particular configuration must be properly saved or your changes are lost when you reboot. The file system for the switch only exists in memory. A rewriteable overlay is contained within the upper four megabytes of the first application flash.

Modifying Files and Updating the Switch

Any file in OpenArchitect can be added, deleted or modified, with the exception of `/sbin/init`, `/usr/sbin/zmnt`, `/lib/modules/zfm_c.o`, and the `/tmp` directory. Files are saved across a system reboot by running the script *zsync*.

A directory `/.zsync` contains database files used by *zsync* for managing the file system overlaying process. The user should not modify the files in this directory or unpredictable results may occur.

Recovering from a System Failure

If the switch does not function after you initially change or reconfigure the image, you have several options for recovering from an error. First, try to *telnet* into the switch. If you are successful, remember to run *zsync* after fixing your problem.

If you cannot *telnet*, attach a console cable to the switch. You can obtain the correct cable from your ZNYX Networks sales representative or see Appendix C to build your own. Bring down the system and properly attach the console cable, see *Chapter 2,* ***Connecting to the Console Port*** .

System Boots with a Console Cable

After attaching the system console cable, if the system boots, fix the problem that does not allow you to *telnet* to the box, run *zsync*, and reboot. The problem is likely to be in the configuration files contained in `/etc/rcZ.d` In order to telnet into the box, there must be a configured interface with a proper IP address. For example, *zhp0* is configured with the IP address 10.90.90.90 in the factory default configuration.

Booting with the –i option

If you cannot *telnet* into the switch and Linux fails to boot, it is likely that a change saved by *zsync* has left the switch in an inaccessible state. To allow users to recover from mistakes saved in the overlay file system, a boot argument of `-i` passed to the init process will stop the un-tarring of the saved overlay files. As a result, the system boots to the factory-shipped configuration.

- Connect through the console port. During boot up, the system displays the Linux boot string. "Linux/PPC load: " for 5 seconds. During the 5 second pause, enter the boot option "-i" and press Return

  ```
  Linux/PPC load: root=/dev/ram init=/sbin/init -i
  ```

- Initiating the *–i* option of *zbootcfg*.

  ```
  zbootcfg -d 1 -i
  ```

- Reboot the system. After the reboot, clear the –i option from the boot string. Enter the following command:

  ```
  zbootcfg -d 1
  ```

  The reboot command will also take "-i" as an option and pass it to the Linux boot,

  ```
  reboot -i
  ```

- When the system boots, the overlay file system is returned to the factory-installed configuration. At this point, you have a few options.

- Run *zsync* and the factory-installed system will be restored to your flash. NOTE: All changes you have made and saved prior to the *zsync* command will be lost.

- Restore particular files from the existing overlay. Use the *zmnt* command to mount the overlay in a designated directory and copy back just the changes you want to keep from the existing overlay. For example, if you wanted to recover your `/etc/hosts` file from the existing overlay, use *zmnt* to mount the overlay in a designated directory, like `/tmp`, then copy `/tmp/etc/hosts` to `/etc/hosts`. Lastly, use *zsync* to save your changes.

  ```
  zmnt /tmp
  cp /tmp/etc/hosts /etc/hosts
  zsync /etc/hosts
  ```

- Reboot the system.

System Hangs During Boot

After attaching the system console cable, if the system hangs during boot, try booting with the *–i* option as described in the previous section. It is possible that important Linux system files became corrupted and incorrectly saved in the flash overlay. Use *zmnt* as described in the previous section to fix or remove the problem files from the overlay. If the system will

not boot with the *–i* option, you will need to *Booting the Duplicate Flash Image* section in this chapter.

Booting the Duplicate Flash Image

Another recovery method, if Linux fails to boot, is to temporarily boot the factory-installed duplicate image located in the second flash device.

1. Connect through the console port.

2. When you see the following, hit ESC to break into the boot loader. Depending on your terminal emulation application, you may have the enter ESC twice.

```
ZX5000 OpenArchitect Firmware  v1.1.x
Copyright (c) 2001, 2002 ZNYX Networks

Testing memory...... ok.
Copying loader image to 0x01000000

Press ESC to interrupt auto-boot.
```

3. At the monitor prompt, type:

```
boot : 2
```

4. The system should boot from the image in the second device. If you have difficulties booting, contact ZNYX Networks technical support.

5. At this point, follow the *Upgrading the OpenArchitect Image* section to put a new RAM disk image in the application flash 1.

   IMPORTANT: Be sure not to program flash 2, since currently this is your only bootable image.

6. The command to program flash 1 should be similar to the following command. The image name may be slightly different depending on the model of switch and version of the image:

```
zflash –d 1 rdr5000.zImage.initrd
```

Upgrading the OpenArchitect Image

1. Use *telnet*, or preferably, attach a console cable to the switch, and login to the switch. If you are connecting via telnet, be aware that the upgrade process will reset the switch to the default IP address of 10.90.90.90, so you will have to be able to reach 10.90.90.90.

2. Download the OpenArchitect image to a local system.

3. The OpenArchitect image is very close to the limit of free space available on a default system so you may need to clear some space prior to downloading the OpenArchitect

image to the switch. Check for free space with the "df" command.

One of the easiest ways to create free space is to remove `/usr/sbin/gated`. The application will be replaced during the update procedure. Once you have enough free space, proceed.

4. From the switch console *ftp* the new OpenArchitect (rdr) image from the local system to your switch.

5. The switch has two flash devices available: Device 1 and Device 2. Use the *zflash* command to write the new OpenArchitect image into the first flash device.

   NOTE: Make sure that Surviving Partner is not running before using zflash. The delays incurred while zflash writes the flash can cause the Surviving Partner daemons to think there is a failure, resulting in link oscillation.

   ```
   zflash -d 1 <image_file>
   ```

6. The image file will be something named similar to the following,

   ```
   zflash -d 1 rdr5000.zImage.initrd
   ```

Upgrading or Adding Files

Follow the procedure below to upgrade or add a new file to the switch. Place the file you are adding or upgrading into the appropriate location in the file system. Save the file in the overlay directory area on the application flash by running *zsync*.

```
zsync
```

After running *zsync*, the file is saved to the flash for future reboots.

Excluding Saving Files to Flash

Specific files or directories can be excluded from saving to flash by *zsync* by including an entry in `/etc/exclude`. Likewise, existing entries in `/etc/exclude` such as `/tmp` can be removed in order to save those files to flash with *zsync*.

Upgrading the Switch Driver

The switch driver upgrade process is the same as a file upgrade. However, more caution should be taken since the driver module is likely to be the method by which you are logging into the system. If the switch driver has a problem, you will need to have a console cable to recover. To upgrade a switch driver, replace the file `/lib/modules/if_zxe.o`, run *zsync* and reboot.

Using Apt-Get

apt-get is a utility created by the Debian Linux community to allow remote fetching and installation of software stored in a repository in Debian package format. It allows users to keep their software up-to-date with the latest binaries, and install new software without the need to recompile.

ZNYX Networks maintains a repository of updated and add-on Debian packages for OpenArchitect–based switches. This repository resides at openarchitect.znyx.com and is accessable through either FTP or HTTP. The file */etc/apt/sources.list* contains a line-by-line listing of all *apt-get* repositories. Users may also create their own repositories and add entries in */etc/apt/sources.list* for their private access methods.

For example, assume there is a small utility that allows a user to modify the L2 table included in the ZNYX Networks repository called *mac-address*. The utility requires a library called *znyx-lib* to function properly. The Debian packages have been created with the Debian *dpkg* and *debhelper* utilities (see the web sites listed in the *Networking Reference* appendix for more information on creating Debian packages.) To use apt-get to retrieve the utility, first download the list of available packages from the repositories listed in */etc/apt/sources.list,*

```
apt-get update
```

Now that *apt-get* knows about all the packages currently in the repository, use *apt-get* to install the package,

```
apt-get install mac-address
```

The process also determines that mac-address relies on *znyx-lib* and installs the library. Once finished, the mac-address utility can be run on the switch. Use zsync to make the changes permanent.

Removing a package will also remove it’s dependencies, For instance if you were to remove the library,

```
apt-get remove znyx-lib
```

The mac-address utility would also be removed. Finally, because space is at a premium on the switch, it is a good idea to cleanup after managing packages:

```
apt-get clean
```

Appendix A OpenArchitect Command Manual Pages

OpenArchitect applications are implemented above the OpenArchitect libraries and the RMAPI interface. OpenArchitect applications are used for normal operation of the switch, for runtime status and diagnostics, and for prototyping new applications development.

For runtime operation, the OpenArchitect applications perform initialization and configuration, and real-time control and maintenance of the switching tables in the switch silicon. Protocol support is performed by the Linux operating system. In turn the OpenArchitect applications communicate with Linux to determine the appropriate switch table setup.

The initialization of the switch is completed by the *zconfig* application. Through configuration scripts, the user can setup any combination of Layer 2 and Layer 3 switching configurations with VLAN support. Running the *zconfig* command causes network interfaces to be presented to the Linux operating system. These interfaces can be setup for Layer 2 bridging functions such as Spanning Tree Protocol, or Layer 3 routing through the Linux operating system.

zl2d is run as a daemon to monitor the Linux operating system bridging function and update the switch silicon accordingly.

zl3d is run as a daemon to monitor the Linux operating system routing table information and update the switch silicon switching tables accordingly.

For gathering statistics or prototyping applications, there are OpenArchitect applications that allow any register or table in the switch to be read or written. These applications include *zreg*, *zstats*, and *zarl* and all of the different table equivalents (see the Broadcom documentation for additional information).

vrrpconfig

NAME

vrrpconfig – Configure and control the running vrrpd

SYNOPSIS

```
vrrpconfig [-d <level>] -- <vrrpd parameters>

vrrpconfig [-d <level>] [-k] [-a] [-p] [-s <vid>]
```

DESCRIPTION

vrrpconfig provides communication with a running *vrrpd* daemon. The -- option for *vrrpconfig* will pass all parameters to *vrrpd* as would be done when starting the *vrrpd*. Any output generated by *vrrpd* is displayed on the *vrrpconfig* controlling tty. Any action normally taken by *vrrpd* for the given parameter is done so by *vrrpd*. Reference *vrrpd* for *vrrpd* parameters and their usage.

OPTIONS

vrrpconfig also has a set of local options that are not passed to *vrrpd* directly. Many do, however, retrieve information from the running *vrrpd*. The local options are as follows:

-d <level>	Set the debug level. The default debug level is 1. The higher the level, the more debugging output is produced. Debugging output is sent to the controlling tty. This debugging output is from *vrrpconfig*. To set the debug level of *vrrpd*, one would use the *vrrpd* debug level setting option placed after the -- in the *vrrpconfig* command line.
-a	Display in a user readable format, information about the current state of all the Virtual Routers controlled by *vrrpd*.
-k	Kill *vrrpd*. The entire daemon is killed. Running this command will require that *vrrpd* be restarted.
-p	Display relevant SNMP table values.
-s <vid>	Print a numeric representation of the state of the Virtual Router associated with the Virtual Router Identifier <vid>. The numeric representations are 1 = INIT, 2 = BACKUP, and 3 = MASTER

EXAMPLES

Here is an example of using the -- invocation method that changes the priority to 99 for the Virtual Router associated with the Virtual Router Identifier 1:

```
vrrpconfig -- -v 1 –p 99
```

SEE ALSO

vrrpd

vrrpd

NAME

vrrpd – Virtual Router Redundancy Protocol Daemon

SYNOPSIS

```
vrrpd -i ifname -v vrid [-f piddir] [-s] [-a auth] [-p prio]
      [-nhb] [-I ifname] [-d delay] [-m address] [-M ] [-B]
      [-S script] [-c conf_file] [-D level] ipaddr
```

DESCRIPTION

vrrpd is an implementation of Virtual Redundant Routing Protocol (VRRPv2) as specified in RFC2338. It runs in Linux user space. In short, VRRP is a protocol that elects a Master server on a LAN to which the Master answers to a *virtual IP address*. If it fails, a Backup server takes over the IP address.

VRRP specifies an election protocol that dynamically assigns responsibility for a virtual router to one of the VRRP routers on a LAN. The VRRP router controlling the IP address(es) associated with a virtual router is called the Master, and forwards packets sent to these IP addresses. The election process provides dynamic failover in the forwarding responsibility should the Master become unavailable. This allows any of the virtual router IP addresses on the LAN to be used as the default first hop router by end-hosts. The advantage gained from using VRRP is a higher availability default path without requiring configuration of dynamic routing or router discovery protocols on every end-host.

OPTIONS

The following options are supported by vrrpd:

-h	display the usage line
-n	Don’t use the virtual MAC address
-b	Run vrrpd in foreground
-i <ifname>	the interface name on which to run the Virtual Router. Machines connected with the named interface will see the Virtual Router Address move with the Master switch
-I <ifname>	the interface name on which to communicate with other VRRP routers for management of the Virtual Router (default is the -i interface)

-v <vrid>	the id of the Virtual Router Identifier [1-255]. This value must be a unique value, one per Virtual Router. In other words there is a unique vrid to ifname associated with the –i option.
-s	Toggle preemption mode (Enabled by default). Preemption means that a Master switch will go to Backup if a current Backup has higher priority.
-M	Become MASTER when priority is equal. Be sure it is only set on one host or the switches will oscillate. Must set –B option on other hosts (requires preemption mode ! -s)
-B	Become BACKUP when priority is equal. See -M option
-S <script>	script to be called when state change occurs.
-a <auth>	(not yet implemented) set the authentification type auth=(none|pass/hexkey|ah/hexkey) hexkey=0x[0-9a-fA-F]+
-p <prio>	Set the priority of this host in the virtual server (default is 100)
-f <piddir>	specify the directory where the pid file is stored (default is /var/run)

-c <conf>	Configuration read from conf file (required when managing multiple Virtual Routers). The contents of the conf file are lines of command line options. Each line represents a Virtual Router. Parameters given on the command line apply to all Virtual Routers defined by the conf file. So for example, if the command line reads:
	
	`vrrpd –d 50 –c vrrpd.conf`
	
	And the *vrrpd.conf* file contains:
	
	`-v 1 –i zhp0 –I zhp3 10.0.0.42`
	`-v 2 –i zhp1 –I zhp3 11.0.0.42`
	
	vrrpd would be started controlling twoVirtual Routers; one for 10.0.0.42 and the other for 11.0.0.42. They would both get a –d 50 option.

-d <delay>	Set the advertisement interval. Default is 1 second. By default time is specified in seconds. If the delay value ends in a lower case ‘m’ the time is specified in milliseconds. The millisecond specification results in a proprietary use of the VRRP Adver Int field.

-m <address>	Change the virtual MAC address from 00:00:5E:00:01:<vid> to the provided addr. The addr should be input as 6 two digit hex numbers that are colon delimited with no spaces. The –n option overrides the change made with –m. The result of which to use the native MAC address of the interface. Using the –n option is not recommeded.
-D <level>	Set debugging output to the supplied level
<ipaddr>	the ip address(es) of the virtual server

SEE ALSO

vrrpconfig

zal

NAME

zal, zhgtable, zhostmac, zipf, zipmc, zl2mcvlanbc, zl4table, zlinkagg, zmac, zmirror, zpttable, zrule, zvmc, zvrp, zvrpout, - Formatted display of OpenArchitect switch tables.

SYNOPSIS

```
zal [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zhgtable [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zhostmac [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zipf [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zipmc [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zl2mcvlanbc [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zl4table [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zlinkagg [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zmac [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zmirror [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zpttable [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zrule [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]

zvmc [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
    [-v <vlan_id>][-h <host_name>] [-d <level>]
```

```
zvrp [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
   [-v <vlan_id>][-h <host_name>] [-d <level>]

zvrpout [-p <ppa>][-i <index>][-m <mac_address>][-a][-c]
   [-v <vlan_id>][-h <host_name>] [-d <level>]
```

DESCRIPTION

The switch table display functions produce formatted output of the switch tables for display on the user console. The format of the output is table-dependent. Headers describing the column being displayed are printed after every 22 lines of output, which makes it easy to pipe through *more*. The switch has keyed and non-keyed tables. Keyed tables grow and shrink as entries are added and deleted. The non-keyed tables are always the same size, and entries are written into specified locations. By default, the table functions print all entries for non-keyed tables and all active entries for the keyed tables. Several options are available which enable the user to display only selected entries. Additionally, there is an option that clears entries in the table. *zal* functions are not mapped. The tables reflect the true display of the 5416 functions, and bypass mapping. Use *zgr* to display the tables with the ports mapped to the order indicated to other applications and on the front panel.

OPTIONS

-h <host_name> Specifies which hostname to connect. By default, *zal* connects to the locally connected OpenArchitect switch (i.e., the one that is on the local PCI bus).

-d <level> Sets the level of debugging output required by *zal*. The default level is one (1). Setting the debug level higher produces more output. Four (4) is currently the maximum output level.

-p <ppa> Specifies the Physical Point of Attachment (PPA). Each switch that is controlled by the CPU and run by *zreg* is a unique PPA. If there is only one switch, such as when *zreg* is running on the embedded processor, the PPA is zero (0). The default PPA is 0.

-m <mac_address>
Displays entries whose MAC address field matches <mac_address>. Only valid for tables that have a MAC address field. Cannot be combined with -v or -i.

-v <vlan_id> Displays entries whose VLAN ID field matches <vlan_id>. Only valid for tables that have a VLAN ID field. Cannot be combined with -m or -i.

-i <index> Displays the entry at the <index> position in the table. Valid for all tables. Cannot be combined with -m or -v.

-a	Displays the entire table. Do not stop when all the active entries in the table have been displayed. The -a flag can produce a lot of output for many of the tables.
-c	Clears the entries in the table. Any entry that would be displayed if the -c flag had not been used will be cleared. Exercise caution with the –c flag, as clearing entries out of some tables can stop the switch from functioning, particularly when *zl3d* is running.

EXAMPLES

The following example searches for and displays an entry in the L2 table with the specified MAC address:

```
zmac -m 00:c0:95:45:00:00
```

If there is an entry in the L2 table with the MAC address, 00:c0:95:45:00:00, all the fields of that entry will be displayed.

The following command deletes the above entry:

```
zmac -c -m 00:c0:95:45:00:00
```

The following command displays all entries of the PTTABLE:

```
zpttable
```

SEE ALSO

zreg

zbootcfg

NAME

zbootcfg – Modifies the boot parameters of the OpenArchitect switch.

SYNOPSIS

```
zbootcfg -a | -d <device number> [<boot_string>]
```

DESCRIPTION

zbootcfg is used to display or modify the boot parameters on the switch. The boot parameters are utilized by the *zmon* boot loader application to indicate on which device to find a boot image. Care should be taken when changing the boot string. Incorrect procedures can result in a switch that cannot boot.

-d <dev>	Three ROM boot devices are available in the switch. The factory-shipped boot device is 1. The following describes each boot device:
-d 1	Boot image located at offset 0 in the application flash 1. This is the factory-shipped location of the primary OpenArchitect image.
-d 2	Loads an image located at offset 0 in the application flash 2. This is the factory-shipped location for the alternate OpenArchitect image.

Any characters after the -d <dev> parameters are saved in flash memory and passed unchanged to the booting kernel.

OPTIONS

-a	Displays the current boot string. The default factory shipping string is “dev1.”
-d <dev>	Specifies the ROM device from which to boot. The <dev> value must be the number 1 or 2 corresponding to application flash 1 or application flash 2, respectively.
<boot string>	Optionally, a boot string may be provided that is passed to the booting kernel. All characters after the -d <dev> are passed unchanged to the booting kernel.

EXAMPLES

The following example illustrates a command for making the image boot from the second application flash. Typically this is required before updating application flash 1. By booting the alternative image, if a failure occurs during the programming of application flash 1, recovery is easier.

```
zbootcfg -d 2
```

The next example passes the *-i* option to the booting kernel. This is useful when recovering from a mistake saved to the read-write file system or after updating the application flash 1 and doing the first boot. The *-i* option prevents the read-write file system from overwriting the initial RAM disk image.

```
zbootcfg -d 1 –i
```

SEE ALSO

zflash, reboot(8)

zconfig

NAME

zconfig - Configures the OpenArchitect switch.

SYNOPSIS

```
zconfig [-h <host_name>] [-d <level>] [-a] [-t] [{-f <file>} |
   <configuration>]
```

DESCRIPTION

zconfig creates Virtual Local Area Network (VLAN) groups of switch ports or trunks. Each VLAN group forms a Layer 2 switching domain. Each VLAN group has a VLAN Identification number (VID) that can be carried in a tag field, located in the header of packets traveling on that VLAN. The configuration of a port determines whether a packet transmitted from that port includes the VLAN tag. A set of up to eight ports may be configured as a trunk, with all links from these ports connect to the same link partner. For each VLAN group created by *zconfig*, a network interface is also created. After the network interface is started by *ifconfig(1M*), the VLAN group performs Layer 2 switching. The network interface can be used for Layer 3 routing between VLAN groups.

A network interface uses the following format:

```
zhpN (e.g., zhp0)
```

N-is an integer between 0 and 9999. The value of N is not required to be the same as any of the port(s) that are its members. The range 0-4999 is reserved for network interfaces created by users. The range 5000-9999 is reserved for network interfaces created by switch applications.

A trunk uses the format *zrlK*, where K is an integer between 0 and 7.

OPTIONS

-h <hostname>	Specifies the hostname to configure. By default, *zconfig* configures the local OpenArchitect switch.
-d <level>	Sets the level of debugging output produced by *zconfig.* The default level is 1. Setting the debug level higher produces more output. The maximum output level is currently four (4).
-a	Displays the current configuration of the switch.
-t	Tears down the entire switch configuration.

{-f <file>} | <configuration>

Gets configuration information from the specified file. A <file> name of '+' reads configuration data from standard input. If the -f flag is not used, a single line of configuration data can be entered as parameters to *zconfig*.

CONFIGURATION SYNTAX

zconfig takes configuration data from standard input or from a file with the -f option. In either case, the configuration syntax is the same. The *zconfig* configuration data consists of a list of semicolon-delimited statements. Each statement specifies an action to take on an interface. An interface is one of three types: a network interface called *ZNYX host port* (`zhp`); a switch port interface called *ZNYX Raw Ethernet* (`zre`); or a trunk interface called *ZNYX RainLink* (`zrl`).

Comments, spaces and new lines are ignored. Comments begin with the # character and include characters through the next new line.

Trunk Interface Statements

A trunk interface statement begins with the trunk name followed by an equals sign and an action. Trunk interface statements are used to create or tear down trunks or define the rules to determine which member of the trunk should be used to transmit a packet

Trunk interface syntax:

```
zrl0 = <Trunk Interface Action>;
```

Trunk interface actions:

List of ports — Creates a trunk interface with the specified port members. All of the ports specified must not be a part of any other trunk, or be individually included in any network interface. Up to eight ports can be included in a trunk.

A port member is identified with the zre<X> format, where x represents a port number between 0 and 19 for 10/100/1000 ports, and 20 to 23 for 1000FX ports. The Out of Band ports cannot be included in the List of ports.

teardown — Removes the trunk interface, making the ports which were part of the trunk available for configuration in other trunks or VLANs.

all

mac [source_address | destination_address]

ip [source_address | destination_address]

port [source_port | desination_port]

Further specifies the rules for selecting which port in the trunk a packet should be transmitted out of. A comma delimited list is valid to specify more than one criterium. Specifying a particular option only uses that layer's source and destination information. The default is `all`, which combines all criteria for determining the transmit port of the trunk. Specifying both source and destination for a given layer is the same as specifying that layer itself, i.e,

zrl0=ip source_address, ip destination_address

is the same as,

zrl0=ip

NOTE: For the ZX5000, trunk settings are made on a global basis, not a per trunk basis.

Examples of trunk interface statements:

This statement creates a trunk containing three ports:

```
zrl5 = zre11,zre13,zre15;
```

The following statement specifies that packets will be sent out over this trunk using the exclusive OR of the last four bits of their MAC source and destination addresses to select the port:

```
zrl5 = mac source_address destination_address;
```

The teardown statement uses a colon instead of an equals sign:

```
zrl5: teardown
```

Network Interface Statements

The syntax for a network interface statement is the interface name followed by a colon and an action. Network interface statements are used to create or tear down a VLAN group and can consist of one or a list of network interface names; followed by a colon and then an action. For example:

```
zhp0: <Network Interface Action>;
```

Network interface actions may include:

vlan<N> = list of ports or trunks

Creates a network interface and a VLAN group with a VLAN identification number (VID) consisting of specified port members.

<N> is an integer between 1-4095.

list of ports or trunks

A port member is identified with the zre<X> format, where x represents a port number between 0-15 for 10/100/1000 ports. A trunk is identified with the zrl<Y> format, where Y represents the port number between 0-5.

If the network interface and VLAN group already exist, the specified ports or trunks are added to the network interface and VLAN group.

teardown — Deletes the network interface and the associated VLAN group.

zre_list = multicast <mac_address>

Register the multicast <mac_address> on the zre_list ports associated with the given VLAN

multicast_clear — Clear all registered multicast address on all the ports in the VLAN

<list of ports or trunks> teardown

Deletes the specified ports or trunks from the network interface and the VLAN group associated with it. If there are no remaining port or trunk members, then also deletes the network interface and VLAN group.

Examples of Network Interface Statements:

The statement below creates a VLAN group with the VID number 1 and the network interface named `zhp5`. This VLAN includes a single switch port, `zre1`.

```
zhp5: vlan1=zre1;
```

The next statement creates a VLAN group with the VID number 100 and the network interface named zhp1. This VLAN includes four switch ports, zre1, zre10, zre11, zre13.

```
zhp0: vlan100 = zre1,zre10,zre11,zre13;
```

The next statement adds two switch ports, zre1, zre2 and zre3, to an existing network interface and VLAN.

```
zhp0: vlan100 = zre1..3;
```

The next statement deletes two switch ports, zre1 and zre2, to an existing network interface and VLAN.

```
zhp0: zre1..2 teardown;
```

The final example is a teardown action that deletes the VLAN group defined in the previous example, including the network interface.

```
zhp1: teardown;
```

Port Interface Statements

Port interface statements specify a port or trunk name or a list of such names; followed by an equal sign (=) and then the action. Port interface actions may include:

SYNTAX

zconfig <zre_list>=untag<n>

untag<N> Packets sent from this port or trunk for VLAN <N> are transmitted without a VLAN tag. The port or trunk specified must have previously been included in the VLAN group with VID<N>.

zconfig <zre_list> multicast=<forward_type>

<forward_type> Set the ports specified to act as defined by the forward_type for multicast traffic. Possible <forward_types> are:

forward_unregistered (default)
forward_all
filter_unregistered

Examples of Port Interface Statements:

Assuming that zre1 has been assigned to VLAN 1, to specify that packets sent from port 1 for VLAN 1 are transmitted without a VLAN tag, and packets arriving on this port without a VLAN tag are given the VLAN tag with the VID number 1, enter:

```
zre1=untag1;
```

If port 0 is also a member of VLAN 100, packets for VLAN 100 are sent from this port with a VLAN tag as part of their header.

In the next example, the switch ports 10, 11, and trunk 2 are configured as untagged members of VLAN 100.

```
zre10,zre11,zrl2=untag100;
```

This statement is equivalent to the following three lines:

```
zre10=untag100;
zre11=untag100;
zrl2=untag100;
```

In the examples above, since port interfaces can only be untagged for one VLAN group, zre1 cannot also be untagged for VLAN 100. A port or trunk

can be a member of multiple VLANs but can only be designated untagged on one VLAN.

WILDCARDS

Wild card characters can be included to simplify the process of creating larger, more complex configurations. Wild card characters for *zconfig* include:

,	(comma)	Use for creating lists
..	(dot-dot)	Specifies an inclusive range
+	(plus)	Specifies auto-incrementing

Below are some examples for the correct usage of the comma (,) and dot-dot (..). Each line below produces the same results:

```
zhp0: vlan1 = zre1, zre2, zre3, zre4;
zhp0: vlan1 = zre1..4;
zhp0: vlan1 = zre1, zre2..4;
zhp0: vlan1 = zre1..2, zre3..4;
```

The following examples create multiple VLAN groups using a single statement. A list of network interface names are followed by a colon (:) and a list of VLAN actions which are followed by an equal sign (=) and a port list. Each VLAN group is created in turn, along with the corresponding network interface, and all ports listed after the equal sign are included in each group.

The following statement creates 14 VLAN groups with VID numbers 1-14. Each VLAN contains the same switch port, port 1, represented as zre1 **.**

zhp0..13: vlan1..14 = zre1;

The plus (+) wildcard can be used with the last port listed to auto-increment that port number before each VLAN group is created. The following network interface statement creates 14 VLAN groups, with the first group containing port 1, the second group port 2, and so on. The second statement configures all ports as untagged in their respective VLANs.

```
zhp0..13: vlan1..14=zre1+;
zre1..13=untag1+;
```

This is equivalent to:

```
zhp0: vlan1=zre1;
zhp1: vlan2=zre2;
zhp2: vlan3=zre3;
```

```
    .
    .
zhp13: vlan14=zre14;

zre1=untag1;
zre2=untag2;
zre3=untag3;
    .
    .
zre14=untag14;
```

The previous configuration can be used for creating a 14 port Layer 3 switch, with each port assigned to its own VLAN.

In the next example, one VLAN group, with VID number 1, is created that contains 14 ports. The second statement designates the 14 ports as untagged for the VLAN 1 group.

```
zhp0: vlan1 = zre1..14;
zre1..14 = untag1;
```

The previous configuration can be used for creating a 14 port Layer 2 switch, all 14 ports assigned to the same VLAN.

SEE ALSO

zl3d

ZCOS

NAME

zcos – class of service queue control

SYNOPSIS

```
zcos [-p <ppa>][-h <hostname>] [-d <level>]
     [-m q0,q1,q2,q3,q4,q5,q6,q7 [<source>P | <source>Q]]
     [ -o <source list> [QS|QR|PS|PR]] |
     [ -s <# cells> [port list>]] |
     [-q [all | qmap | pmap | qinfo [port list]]
```

DESCRIPTION

The OpenArchitect switch supports eight queues of packets to be sent out each of the 16 ports or forwarded to the CPU. These class of service queues are serviced in strict priority order, with packets from queue 0 sent first, and packets from the other queues sent only when there are no packets to be sent on any lower numbered queues. Packets can be assigned a COS queue based on a number of criteria. The outgoing 802.1p priority can also be set by the switch based on many of these same criteria. The sources for the COS queue priority and 802.1P priority are:

The incoming 802.1P priority value for packets tagged at the input.

The VLAN priority if the packet was not tagged at the input

The priority for the destination MAC address of the packet in the L2 lookup table.

For IP packets, the TOS field, including the IP precedence and DTR bits.

The priority for the destination IP address from the L3 lookup table.

The COS field from a matched classifier rule. This applies only to selecting the COS queue.

The priority for the destination IP multicast group address from the IPMC table.

For each of these sources, except the COS field of the classifier, there are two priority mapping tables, one to map the priority to one of the COS queues, and one to map the priority to the 802.1P priority for packets tagged on output.

Which priority to use is determined by setting the precedence order for each of these sources. Precedence values range from 0 to 6; 6 is the highest priority, 0 disables the source. There are separate priority precedence tables for switched packets and routed packets for both COS queue selection and 802.1P priority setting.

There are additional sources for determining the COS queue used for packets sent to the CPU. Priorities can be set for BPDU packets, packets sent to the CPU because they contain IP oprions, IP-redirect packets, packets which caused errors in the address lookup tables and packets selected by the classifier for copying to the CPU. For normal packets sent to the CPU, an additional table remaps the COS queue selection to the queue for the CPU.

The amount of space used by the queues to hold packets for each of the ports and the CPU can be limited by setting the number of 64 byte cells allowed.

The *zcos* application sets the maximum number of cells used for each port in the switch and provides the mapping to packet priorities and COS queues.

OPTIONS

-p <ppa> Specifies the physical point of attachment (PPA). This is used to uniquely identify the switch being used. By default, this parameter is set to 0 to imply the current OpenArchitect switch where you are running this command. Ignore this option if you are configuring on the OpenArchitect switch on which the zcos command is being run.

-h <hostname> Specifies the hostname of the OpenArchitect switch to be configured. Ignore this option if you are configuring on the OpenArchitect switch on which the zcos command is being run.

-d <level> Sets the debug level. The default is 1. The maximum is 4.

-m q0,..,q7 [<source>P | <source >Q]

Specifies the priority to COS queue or 802.1P priority map for each priority source. The first parameter maps priority 0 to q0, the second maps priority 1 to queue number q1, etc. (valid range is 0 to 7). The default behavior is to use the same mapping for all sources and for both COS queue selection and outgoing 802.1P priority. Optionally, a map may be specified for an individual source by giving the source name followed by P to set the outgoing priority map or Q to set the COS queue map for that source.

source One of the priority sources. The possibilities are: user – 802.1p tag, vlan – vlan default for untagged packets, mac – L2 mac table, l3 – IP forwarding table, mc – IP multicast table, tos – DSCP/TOS field of IP packets.

- s <# cells> [<port list>]
Sets the number of 64 byte cells allowed for holding packets destined for each of the ports listed. Defaults to set the limit for all ports.

port list A comma separated list of ports specified as *zre*<n> (where n is 0-15 for output ports; and *cpu* for the CPU.

-o <source list> [QS | QR | PS | PR]
Sets the precedence order among the different priority sources for

determining the COS queue or outgoing priority. Specify QS to set the order for the COS queues for Switched packets, or QR to set the order for the COS queues for Routed packets. Use PS or PR to set the order for outgoing 802.1P priority.

source list is a comma separated list of <source> s or class, for the COS selection from a classifier rule.

queue list defines a list of queues, comma separated.

-q [all | qmap | pmap | qinfo [port list]]
queries currently configured parameters.

all displays all of the *queue mappings*, *queue configuration* parameters, and *precedence ordering* for all ports and queues.

qmap displays the priority to COS queue mappings. If port list is not specified, then details for all ports will be displayed.

pmap displays the priority outgoing priority mappings. If port list is not specified, then details for all ports will be displayed.

qinfo displays configured details for the egress queues. If port list is not specified, then details for all ports will be displayed.

port list
is a comma separated list of ports specified as *zre*<n>, where *n* is 0-15 for Ethernet ports 0-15 and *cpu* for the CPU.

EXAMPLES

To set all 16 10/100/1000 Ethernet ports to use no more that 500 cells each:

```
zcos -s 500 zre0..15
```

To set the CPU port to use no more than 256 cells (16384 bytes):

```
zcos -s 256 cpu
```

To map 802.1p priorities 0-3 to COS0, 4-5 to COS1, 6 to COS2 and 7 to COS3:

```
zcos -m 0,0,0,0,1,1,2,3 userQ
```

To set the precedence for mapping routed packets to COS queues to prefer the classifier, then IP TOS, then 802.1P or default VLAN priority:

```
zcos -o class,tos,user,vlan QR
```

SEE ALSO

zfilterd, zqosd, ztmd

zfilterd

NAME

zfilterd - A daemon to use the filter hardware of the OpenArchitect switch for filtering based on *iptables(8)* rules.

SYNOPSIS

```
zfilterd [-d <level>] [-p <port>] [-f] [-l] [-i <pid>] [-o <pid>]
```

DESCRIPTION

zfilterd is a daemon that intercepts filtering rules entered by the user, using *iptables(8)*, checks them for validity and then prepares messages for the traffic management daemon *ztmd*, which is responsible for setting up the switch hardware for the filtering rules and actions.

OPTIONS

-d <level>	Sets the level of debugging output required by *zconfig*. The default level is one (1). Setting the debug level higher produces more output. Four (4) is currently the maximum output level.
-p <port>	Set the multicast port to which messages will be sent.
-f	Run *zfilterd* in the foreground, by default, it runs in the background.
-l	Log all diagnostic output to /var/log/zfilterd.log.
-I <pid>	Set our pid used in identifying ourselves to *ztmd*
-o <pid>	set the pid of the *ztmd* process we will communicate with.

SEE ALSO

ztmd, *zrule*, *iptables(8)*

zflash

NAME

zflash Loads images into the flash ROMs on the OpenArchitect switch.

SYNOPSIS

```
zflash -d <dev> [-o|-O <offset>] <image_file>
```

DESCRIPTION

zflash enables you to program the flash ROMs on the switch. The switch contains 3 flash ROM devices: the boot ROM flash, application flash 1 and application flash 2. Care should be taken when flashing new images into the switch. Incorrect procedures can result in a switch that cannot boot; especially when flashing the boot ROM referred to as device 0. See *Switch Maintenance* for updating the flash ROM images.

OPTIONS

-d <dev>	Specifies the ROM device being programmed. Dev must be a number 0, 1, or 2 corresponding to the boot ROM, application flash 1, or application flash 2 respectively.
-o <offset>	Specifies the offset into the device or memory where the image should be loaded. The offset is specified in megabytes. To specify an offset not on a megabyte boundary, use -O. Default offset is 0
-O <offset>	Specifies the offset in bytes. Default offset is 0

EXAMPLES

The following example loads a new initial RAM image into application flash 1 at the default offset of 0:

```
zflash -d 1 rdr5000.zImage.initrd
```

The following example loads a new boot image into the boot rom at the default offset of 0:

```
zflash -d 0 zx5000.img
```

Exercise caution when using this command, as an error can render your switch inoperable. Do not interrupt this process until complete.

SEE ALSO

zbootcfg

zgmrpd

NAME

NOTE: Not implemented in this release

zgmrpd - GARP Multicast Registration Protocol (GMRP) daemon for the OpenArchitect switch.

SYNOPSIS

```
zgmrpd [-d <level>] [-f] [-h <hostname>] [-p <ppa>] [-t <target>]
```

DESCRIPTION

zgmrpd is run after the network interfaces are created and initialized with *zconfig*, and started with *ifconfig(1M)*. *zgmrpd* starts a background task that implements the GARP Multicast Registration Protocol (GMRP) protocol for a specified interface, either a *zhp* or *bzhp*. GMRP provides a Layer-2 mechanism for determining which ports in a VLAN are listening to which multicast addresses. *zgmrpd* updates the multicast table in silicon with this information, so that the multicast traffic is forwarded only to the ports in a VLAN to which listening hosts are attached. This optimizes packet flow through the switch. The background task started by *zgmrpd* continues throughout the life of the Layer 2 network. GMRP is specified in *ANSI/IEEE Std 802.1D, 1998 Edition*.

zgmrpd manages the switch multicast table (MARL). Based on GMRP packets received on the target interface, *zgmrpd* creates, updates, or deletes an entry in the MARL. The key to each MARL entry is a source Ethernet multicast address combined with a VLAN ID. Two port bitmaps are maintained: one that identifies the untagged members of the VLAN, and one which identifies which ports of the VLAN have listening hosts attached.

NOTE: OpenArchitect does not tag BPDU packets. This means that if a port belongs to multiple VLANs, exactly which Multicast MAC/VLAN entry to create, modify, or delete cannot be determined. GMRP should not be used when ports belong to multiple VLANs. If the target is a zhp, it's recommended that the zhp contain all the ports on the switch.

When the maximum number of entries in the MARL is reached, *zgmrpd* deletes a random entry prior to adding the next entry.

By default, while GMRP is running, unregistered multicast traffic is filtered on the target interface. The GMRP protocol provides a mechanism for changing the filtering behavior on a per-port basis. Filtering behavior can be set to filter unregistered multicast traffic, forward unregistered multicast traffic, or forward all traffic. *zconfig* can be used to set the filtering mode of

each port. See *zconfig* . Ports connected to routers should be set to “Forward All” filtering mode manually with zconfig after zgmrpd is up and running. When *zgmrpd* is terminated, filtering behavior is set to forward unregistered multicast traffic on each port in the target interface.

Only the GARP normal registration mode is currently supported.

Multiple instances of *zgmrpd* may run concurrently provided the targets are unique. However, *zgmrpd* cannot run concurrently with *zsnoopd*. See *zsnoopd*.

OPTIONS

-d <level>	Sets the level of debugging output required by *zgmrpd*. The default level is zero (0). Setting the debug level higher produces more output. Five (5) is currently the maximum output level.
-f	Run *zgmrpd* in foreground. Default is to run it in background.
-h <hostname>	Connect to remote host <hostname>.
-p <ppa>	Start *zgmrpd* on switch <ppa>. Default is 0.
-t <target>	Enable GMRP on the set of ports specified by the target, either a *zhp* or *bzhp* interface. There is no default. A target must be specified.

EXAMPLES

In the following example, *zgmrpd* starts a background task that enables the GMRP protocol for the ports in the *zhp0* interface. *zgmrpd* receives and sends GMRP packets, and updates the Multicast Table (MARL) accordingly. This background task continues throughout the life of the Layer 2 network, or until manually terminated.

```
zgmrpd -t zhp0
```

Once you run *zgmrpd*, use *zmarl* to display the contents of the MARL. *zgmrpd* deletes all entries in the MARL when starting up, and when shutting down. Manual changes to the MARL can be made while *zgmrpd* is running. However, all such changes will be deleted when *zgmrpd* is terminated.

SEE ALSO

zconfig, *zsnoopd*

zgr

NAME

zl2, zl2mc, zl3host, zl3net, zvlan – Formatted display of OpenArchitect generic tables.

zl2 displays the abstraction API's layer 2 table.

zl2mc displays the abstraction API's layer 2 multicast table.

zl3host displays the abstraction API's layer 3 host route table.

zl3net displays the abstraction API's layer 3 network route table.

zvlan displays the abstraction API's VLAN table.

SYNOPSIS

```
zl2 [-p <ppa>] [-i <index>] [-m <mac_address>] [-a]
   [-v <vlan_id>] [-P port] [-h <host_name>] [-d <level>]

zl2mc [-p <ppa>] [-i <index>] [-m <mac_address>] [-a]
   [-v <vlan_id>] [-P port] [-h <host_name>] [-d <level>]

zl3host [-p <ppa>] [-i <index>] [-m <mac_address>] [-a]
   [-v <vlan_id>] [-P port] [-h <host_name>] [-d <level>]

zl3net [-p <ppa>] [-i <index>] [-m <mac_address>] [-a]
   [-v <vlan_id>] [-P port] [-h <host_name>] [-d <level>]

zvlan [-p <ppa>] [-i <index>] [-m <mac_address>] [-a]
   [-v <vlan_id>] [-h <host_name>] [-d <level>]
```

DESCRIPTION

The generic table display functions produce formatted output of the abstraction API's tables for display on the user console. The format of the output is table-dependent. Port mapping affects the ports referenced in the generic tables. (Ports listed in order from 0 to 15)

Headers describing the column being displayed are printed after every 22 lines of output, which makes it easy to pipe through *more(1)*. The abstraction layer tables grow and shrink as entries are added and deleted.

Several options are available which enable the user to display only selected entries. Additionally, there is an option that clears user-specified entries in the table.

OPTIONS

-p <ppa> Specifies the Physical Point of Attachment (PPA). Each switch that is controlled by the CPU and run by *zreg* is a unique PPA. If there is only one switch, such as when *zreg* is running on the embedded processor, the PPA is zero (0). The default PPA is 0.

-i <index> Displays the entry at the <index> position in the table. Valid for all tables. Cannot be combined with -m, -P or -v.

-m <mac_address>
Displays entries whose MAC address field matches <mac_address>. Only valid for tables that have a MAC address field. Cannot be combined with -i, -P, or -v.

-a Displays the entire table.

-v <vlan_id> Displays entries whose VLAN ID field matches <vlan_id>. Only valid for tables that have a VLAN ID field. Cannot be combined with -i, -m, or -P.

-P <port> Displays the entries whose port field matches <port>. Only valid for tables that have a PORT ID field. Cannot be combined with -i, -m, or -v.

-h <host_name> Specifies which hostname to connect. By default, *zgr* connects to the locally connected OpenArchitect switch (i.e., the one that is on the local PCI bus).

-d <level> Sets the level of debugging output required by *zgr*. The default level is one (1). Setting the debug level higher produces more output. Four (4) is currently the maximum output level.

EXAMPLES

The following example searches for and displays an entry in the *zl2* table with the specified MAC address:

```
zl2 -m 00:c0:95:45:00:00
```

If there is an entry in the ZL2 table with the MAC address, 00:c0:95:45:00:00, all the fields of that entry will be displayed.

The following command deletes the above entry:

```
zl2 -c -m 00:c0:95:45:00:00
```

The following command displays all entries of the *zl2* table:

```
zl2
```

Be careful, the *-c* option does not ask. The following command deletes all entries in the *zl2* table:

```
zl2 -c
```

SEE ALSO

zal

zgvrpd

NAME

NOTE: Not implemented in this release

zgvrpd - GARP VLAN Registration Protocol (GVRP) daemon for the OpenArchitect switch.

SYNOPSIS

```
zgvrpd [-d <level>] [-f] [-h <hostname>] [-p <ppa>] [-t <target>]
```

DESCRIPTION

zgvrpd is run after the network interfaces are created and initialized with *zconfig*, and started with *ifconfig(1M)*. *zgvrpd* starts a background task that implements the GARP VLAN Registration Protocol (GVRP) protocol for a specified *zhp* interface. GVRP provides a Layer-2 mechanism for dynamically managing port membership in VLANs, including adding and deleting ports, and creating and deleting VLANs. The background task started by *zgvrpd* continues throughout the life of the Layer 2 network. GVRP is specified in *ANSI/IEEE Std 802.1Q, 1998 Edition*. The GARP protocol on which GVRP is based is specified in *ANSI/IEEE Std 802.1D, 1998 Edition.*

zgvrpd updates the switch's VLAN configuration, based on GVRP packets received on the target interface. Specifically, it adds a port to or deletes a port from the VLAN specified in the GVRP packet. If the VLAN does not exist, *zgvrpd* creates it. If *zgvrpd* deletes the last port from a dynamically-created VLAN, it also deletes the VLAN.

When a VLAN is dynamically created, a corresponding zhpN interface is also created, where N is an integer between 5000 and 9999. The value of N is equal to 5000 plus the VLAN identification number (VID). For example, if zgvrpd creates vlan 5, it also creates a zhp5005.

zgvrpd learns the existing (static) VLAN configuration of its target when starting up. When shutting down, *zgvrpd* deletes only the dynamic changes to the VLAN configuration that it has made. Manual changes to the VLAN configuration can be made while *zgvrpd* is running. However, all such changes will be deleted when *zgvrpd* is terminated.

Only the GARP normal registration mode is currently supported. Multiple instances of *zgvrpd* may run concurrently provided the targets are unique. If zgvprd's target is a zhp, it's recommended that the zhp contain all the ports on the switch.

zgvrpd cannot be run concurrently with *zsnoopd*, because *zsnoopd* assumes static VLAN membership.

OPTIONS

-d <level>	Sets the level of debugging output required by *zgvrpd*. The default level is zero (0). Setting the debug level higher produces more output. Five (5) is currently the maximum output level.
-f	Run *zgvrpd* in foreground. Default is to run it in background.
-h <hostname>	Connect to remote host <hostname>.
-p <ppa>	Start *zgvrpd* on switch <ppa>. Default is 0.
-t <target>	Enable GVRP on the set of ports specified by the target *zhp* interface. There is no default. A target must be specified.

EXAMPLES

In the following example, *zgvrpd* starts a background task that enables the GVRP protocol for the ports in the *zhp0* interface. *zgvrpd* receives and sends GVRP packets, and updates the VLAN configuration accordingly. This background task continues throughout the life of the Layer 2 network, or until manually terminated.

```
zgvrpd –t zhp0
```

Once you run *zgvrpd*, use *zconfig -a* to display the current VLAN configuration.

SEE ALSO

zconfig, zsnoopd

zl2d

NAME

zl2d - Layer 2 daemon for the OpenArchitect switch.

SYNOPSIS

```
zl2d [start | stop] [-t <msecs>] [-d <level>] [-f]
        [-p <priority>] <iface..>
```

DESCRIPTION

zl2d is run after the network interfaces are created and initialized with *zconfig*. *zl2d* creates a Linux bridge for each interface using *brctl(8)*. The bridge name is the interface name with a 'b' pre-pended to it. This command is primarily used for Spanning Tree Protocol (STP). Each port associated with the interface is included within the bridge. *zl2d* starts a background task that continues throughout the life of the Layer 2 network.

OPTIONS

start | stop	Starts or stops the *zl2d* daemon.
-t <msec>	Cause *zl2d* to monitor the Spanning Tree state of each port on each bridge every <msec> milliseconds. If unspecified, the default is 500 milliseconds.
-f	Enables Fast Forward on bridge(s) using 0x4000 (16384) as the dynamic root priority.
-d <level>	Sets the level of debugging output required by *zl2d.* The default level is one (1). Setting the debug level higher produces more output. Four (4) is currently the maximum output level.
-p <priority>	Sets the dynamic root priority. <priority> should be specified as a decimal number. A priority of 0 disables root priority change.
<iface…>	The network interfaces on which *zl2d* should operate. These network interfaces must first be created by *zconfig*. *zl2d* does not operate with standard network interface cards. It only works on switch network interfaces created by *zconfig*.

OPERATIONS

zl2d manages the Spanning Tree state fields in the switch of each port within the bridge(s). Based on a timer, *zl2d* reads the port information for each Linux bridge and updates the switch when necessary.

EXAMPLES

In the following example, *zl2d* creates a Linux bridge named `bzhp0` which includes all of the `zre<n>` devices previously associated with the `zhp0` device. *zl2d* then starts a background task that monitors the port information of the Linux bridge every 500 ms. and updates the Spanning Tree state fields in the hardware when necessary.

```
zl2d -t 500  zhp0
```

Once you run zl2d, use brctl(8) to display and alter your Spanning Tree settings.

SEE ALSO

zconfig, brctl(8)

zl3d

NAME

zl3d - Layer 3 daemon for the OpenArchitect switch.

SYNOPSIS

```
zl3d [-h <host_name>] [-t <msecs>] [-b] [-e] [-i D|I] [-l] [-n] [-d
   <level>] <iface ..>
```

DESCRIPTION

z*l3d* is run after the network interfaces are created and initialized with *zconfig*, and started with *ifconfig(1M)*. zl3d listens for Netlink messages from the kernel and monitors the Linux network routing tables for routing updates. When an update occurs, if appropriate, *zl3d* updates the corresponding routing tables in the silicon so that Layer 3 forwarding is done in the switch at line speed. zl3d also ages the switch host route entries.

zl3d attempts to keep the Linux route cache and the switch host route table consistent. When a host route table entry is in use, *zl3d* updates the Linux route cache. Host route table entries are deleted when Linux removes the corresponding entry from the route cache. Similarly, network route entries are removed when the corresponding Linux FIB table entry is deleted.

OPTIONS

-h <hostname>	Specifies which host to monitor. By default, *zl3d* monitors the OpenArchitect switch that is locally connected (i.e., the one that is on the local PCI bus).
-t <msec>	Sets the timeout value. By default, *zl3d* wakes up every 15 seconds (15000 ms) to look for updates to the Linux routing tables and to do aging processing of route table entries on the switch.
-b	Do not background the *zl3d* process.
-e	Enables additions of a default route to the L3 network route table.
-i I | D	Turn off L2 insert(I)/delete(D) notification processing
-l	Leave hardware tables intact on exit.
-n	Do not write the proc table entry. By default, zl3d writes 3/2 of it's timeout value to /proc/net/sys/ipv4/route/route_expires. The Linux kernel uses this value as an expiration time for cache routes updated by zl3d.

-d <level>	Sets the level of debugging output required by *zl3d*. The default level is one (1). Setting the debug level higher produces more output. Four (4) is currently the maximum output level.
<iface…>	The network interfaces on which *zl3d* should operate. These network interfaces must first be created by *zconfig*. *zl3d* does not operate with standard network interface cards. It only works on switch network interfaces created by *zconfig*. It uses the same syntax as *zconfig*.

OPERATIONS

zl3d manages the host route and network route tables located in the switch. Based on a trigger condition, *zl3d* reads the Linux FIB table and the Linux route cache. Each table entry is filtered by *zl3d* according to the following:

- If an entry from the route cache is not a broadcast address or a local address, and *zl3d* is able to resolve the MAC address of the destination, then *zl3d* inserts the entry into the switch host route table.
- If an entry in the Linux FIB table is a host entry and *zl3d* is able to resolve the MAC address of the destination host, then the entry is inserted into the switch host route table. If an entry from the FIB table is a gateway entry, is not local, and *zl3d* is able to resolve the MAC address of the gateway, then the entry is inserted into the switch network route table.

EXAMPLES

Normally, *zl3d* is run as a background task that continues throughout the life of the Layer 3 network.

In the following example, for the three interfaces specified, *zl3d* continuously monitors the Linux FIB and route cache tables looking for updates every three seconds. Entries in the switch host route table are checked for activity every 15 seconds and aged accordingly.

```
zl3d zhp1 zhp2 zhp3
```

SEE ALSO

zconfig

zlc

NAME

zlc – link and LED control

SYNOPSIS

zlc [-p <ppa>][-h <hostname>][-d <level>][-x] <port_list> <action> [on | off]

zlc [-p <ppa>][-h <hostname>][-d <level>][-x] <action> [on | off |clear]

zlc [-p <ppa>][-h <hostname>][-d <level>][-x] [state|query]

DESCRIPTION

The *zlc* application sets the link speed and state of individual ports of the switch, or displays the current state. It can also turn on or off the external fault LED or the internal fault LED.

OPTIONS

-p <ppa>	Specify ppa to connect to switch(default 0).
-h <hostname>	Connect to remote host <hostname>.
-d <level>	set debug level to <level>
-x	expected query value. Creates no output, exit code only. If the port_list contains more than one port, returns the number of ports that match the option.
<port_list>	port or list of ports on which to take action. Port lists are supplied in *zconfig* syntax (e.g. zre1, zre2..4, etc.)
<action>	Set link speed or state to `up`, `down`, `auto`, `1000fd`, `1000hd`, `100fd`, `100hd`, `10fd` or `10hd`. The interface must be down to change the port speed. Set `intfault` or `extfault`. Must supply <option>. Use `query` to return present settings.
on | off	Turn `on` or `off` the specified LED. Only valid for actions `intfault`, `extfault` or `extract`.
clear	No longer globally control the specified LED
state	Return the list of LEDs currently illuminated.
query	Return the list of LEDs currently controlled globally.

EXAMPLES

In the following example, *zlc* forces the line speed of port 1 to 100 Full duplex. The interface must be down to change the speed. Assuming zre1 is part of interface zhp0,

```
ifconfig zhp0 down
zlc zre1 100fd
```

The external fault, internal fault, and ok LEDs can be set on a per port basis or *globally*. To set the external fault LED for a particular port,

```
zlc zre1 extfault on
```

To query the settings of a particular port,

```
zlc zre1 query
```

Global Settings

The external fault and internal fault LEDs are set as a logical OR of all the ports. The LEDs can also be set globally to on, off, or other. If globally set to on or off, the LED will not change when links go up or down ,or interfaces are configured. If set to *other*, the LED resumes its normal operation.

The next example globally turns on the external fault LED.

```
zlc extfault on
```

Additional capabilities are also available by supplying an additional led action:

```
zlc <led_name> on
```

The <led_name> LED is turned on. The LED will not change when links go up or down or interfaces are configured. If `other` is used, the LED resumes its normal operation. <led_name> can be `intfault`, `extfault`, `extract`, or `ok`.

```
zlc query
zlc state
```

`query` lists the LEDs which have been set globally `on` or `off`. `state` shows which LEDs are on at the moment. All LEDs are shown, including the *clk* LED.

SEE ALSO

ifconfig(8)

zlmd

NAME

zlmd – monitor link changes or hot swap events.

SYNOPSIS

```
zlmd [-h <hostname>] [-b] [-d <level>] [-p <ppa>]
   {-f <file>} | <configuration>
```

DESCRIPTION

The *zlmd* application is intended to run as a daemon, waiting for a configured event to occur and then running the program configured for that event. The events monitored are changes in the link status of any of the 16 ports of the switch, the start of removal of the switch from the back plane, or the cancellation of the removal before it actually takes place. The program can be a shell script that initiates appropriate actions to respond to the event.

OPTIONS

-h <host_name> Connect to remote host <host_name>

-b Do not background the zlmd process.

-d <level> Set debug level to <level>.

-p <ppa> Monitor switch <ppa>. Default is 0.

{-f <configuration_file>} | <configuration>

Read configuration from <file>. If file is a '+', configuration is read from stdin. Without the -f option, a single line of input can precede the last flag. (i.e. <Configuration>)

CONFIGURATION SYNTAX

zlmd takes configuration data from standard input or from a file with the -f option. In either case, the configuration syntax is the same. The *zlmd* configuration data consists of a list of semicolon-delimited statements. Each statement specifies an event to monitor followed by an response action.

Configuration commands:

<port-list> = <program>

Run <program> when a fault occurs or clears on a port.

hotswap = <program>

Run <program> when a hot-swap extraction or insertion event occurs.

<port-list> A list of ports in the same forms supported by *zconfig*, e.g. zre1,zre2 or zre10..14.

<program> Path to an executable program or script to be run when the event occurs. Note: An absolute path to <program> is required. The program will be called with the following parameters:

For Link Changes:

<program> <ppa> <port> {external(0)|internal(1)} {off(0)|on(1)}

For Hot-swap Events:

<program> <ppa> {extraction(1)|insertion(2)}

EXAMPLES

In the following example, *zlmd* monitors ports 1 through 4 and runs a script called *prt_change* upon a link change event.

```
zlmd zre1..4=/usr/sbin/prt_change
```

Suppose port 2 were up and you disconnected the cable, *zlmd* would call *prt_change* with the following parameters:

```
/usr/sbin/prt_change 0 2 0 1
```

where 0 is the *ppa*, 2 is the *port*, 0 is an *external fault*, 1 is *on*.

SEE ALSO

zconfig

zlogrotate

NAME

zlogrotate – Rotates log files.

SYNOPSIS

```
zlogrotate [-b] [-t time] [-s segment size] [-n # of files] [-f file
   to rotate]
```

DESCRIPTION

zlogrotate rotates the selected file every [time] seconds if the file is larger than [segment size]. It will keep only the number of files selected. *zlogrotate* is called from /etc/init.d/rcS by default with no parameters.

OPTIONS

-b	Do not background the process - i.e. run in foreground.
-t <time>	the time between logfile checks in seconds (default 60)
-s <size>	the targeted file segment size, in kilobytes (default 256)
-n <# of files>	The number of segments kept on the system (default 4)
-f <file>	The file to rotate (default /var/log/messages)

EXAMPLES

To start zlogrotate with the default values,

```
zlogrotate
```

zmirror

NAME

zmirror - Set packet mirroring on an ingress or egress port

SYNOPSIS

```
zmirror  -a | -t
zmirror  [-e] <from_list> <to_port>
```

DESCRIPTION

zmirror sets packet mirroring from a given set of ports to a given port. Turning on packet mirroring causes a copy of the packet to be sent to the *to* port. Any number of *from* ports can be mirrored to one or more *to* ports.

Note, there are performance issues when trying to mirror more bandwidth than is available on the *to* port.

After executing the following command, packets received on ports 1, 2 and 3 would be mirrored (copied and transmitted) to port 12. This mirroring would be in addition to any Layer 3 or Layer 2 switching.

```
zmirror zre1, zre2, zre3 zre12
```

To clear the current mirroring, use the *-t* option.

The *-e* option can be used to indicate that packets being sent on a given port should be copied to the *to* port. For example if the -e option is used as follows, the packets transmitted, as opposed to received, on ports 1, 2 or 3 would be mirrored to port 12.

```
zmirror -e zre1, zre2, zre3 zre12
```

The *to* port can also be the keyword `cpu` to indicate that packets should be forwarded to the on-board processor. The following example would mirror the contents of port 1, 2 or 3 to the on-board processor:

```
zmirror zre1, zre2, zre3 cpu
```

The *to* port can be a single port or the keyword `cpu`. The from port can be a list consisting of one or more ports. The *from* port cannot be the `cpu`. See the section on wildcards for discussion of from port lists.

zmirror is cumulative:

```
zmirror zre1, zre2, zre3 cpu
```

Is the same as:

```
zmirror zre1 cpu
zmirror zre2 cpu
zmirror zre3 cpu
```

Multiple mirroring setups can be made. The following example will mirror port 1 traffic to port 11 and port 2 traffic to port 12.

```
zmirror zre1 zre10
zmirror zre2 zre11
```

Setting a different *to* port will overwrite the previous setting. Given the last setup the following will change port 1 traffic to be forwarded to port 10.

```
zmirror zre1 zre10
```

Use *zmirror –a* to query the hardware to display the true current configuration.

```
zmirror -a
```

OPTIONS

-a	Display the current mirroring setup
-e	Set egress port mirroring for the specified *from* port
-t	Teardown or disable the mirroring

WILDCARDS

Wildcard characters can be included to simplify the process of creating larger, more complex configurations. Wild card characters for *zconfig* include:

, (comma)	Use for creating lists
.. (dot-dot)	Specifies an inclusive range

Below are some examples for the correct usage of the comma (,) and dot-dot (..). Each line below produces the same results:

```
zre1, zre2, zre3, zre4
zre1..4
zre1, zre2..4
zre1..2, zre3..4
```

SEE ALSO

tcpdump(1M)

zmnt

NAME

zmnt – Expands the read/write files onto the RAM disk.

SYNOPSIS

```
zmnt [-c] <directory>
zmnt [-c] -t <file>
zmnt [-c] -l
```

DESCRIPTION

zmnt expands files from flash onto the RAM disk that have been previously saved with *zsync*. The *init* process runs *zmnt* to expand the files in flash onto RAM file system. The user may use *zmnt* to expand the files at anytime and may place them in any directory. For example,

```
zmnt /tmp
```

will expand the files under the directory /tmp.

Booting the kernel with `-i` instructs the init process to skip the *zmnt* stage After booting in this way, *zmnt* can be used for correcting a problem file in the read-write file system.

The *-t* option can be used to save the configuration of a switch to a tar file. A tar file can be copied to another switch and saved with *zsync -t*. As a result, the configuration of a switch may be cloned to other switches.

The *-c* option is used to mount the custom overlay. See *zsync* for a description of custom verses dynamic overlay.

OPTIONS

-c	Read saved files from the custom overlay
-t <file>	Save the overlay into the specified <file> in tar format.
<directory>	The directory under which the overlay files are expanded, or the file to which the tar image is saved.
-l	List files in selected overlay, do not unpack.

EXAMPLES

In the following example, *zmnt* the current overlay into a tar file called `overlay.tar`

```
zmnt –t overlay.tar
```

The resulting tar file can now be saved on a different host as a snapshot of the overlay at that point in time. Use zsync to restore the overlay on the switch:

```
zsync –t overlay.tar
```

The restored overlay will be used upon the next reboot.

SEE ALSO

zsync

zqosd

NAME

zqosd – monitors *tc(8)* commands to implement classification filters and queuing disciplines in hardware.

SYNOPSIS

```
zqosd [-d <level>] [-p <port>] [-f] [-l] [-i <pid>] [-o <pid>]
```

DESCRIPTION

zqosd monitors commands entered by *tc* which set up queuing disciplines and classification filters for managing traffic in the switch. It supports a variety of queuing disciplines which allow distributing available bandwidth at each output port of the switch among different classes of traffic as well as selecting which packets to drop when bandwidth limits are exceeded.

zqosd does not directly set up the hardware. It prepares messages describing the queuing disciplines and filters and sends them to a hardware specific daemon, *ztmd*. *ztmd* should be started before *zqosd*. Both programs normally run as background processes.

OPTIONS

-d <level>	Set the level of diagnostic information logged. <level> may be 0-4; higher levels produce more output.
.p <port>	Use <port> as the multicast listening port for communication with *ztmd*. Default is 2345.
-f	Run *zqosd* in the foreground. Without this option, it is run as a daemon.
-l	Log diagnostic output to `/var/log/zqosd.log`
-i <pid>	Set the pid for this process.
-o <pid>	Set the pid of the destination process (*ztmd*)

EXAMPLES

Start the traffic management daemon, *ztmd*, then start *zqosd* to monitor *tc* output. Both daemons are run as background processes and log their messages.

```
ztmd –l
zqosd -l
```

SEE ALSO

ztmd, tc(8), zfilterd

zrc

NAME

zrc - Packet rate control

SYNOPSIS

```
zrc  -b | -m | -d | -t | -a [-p <port>] [-v <vlan>] [-g <group>] [-M
   <mac_addr>] [-T <timeout>] [-D <level>] <rate>
```

DESCRIPTION

zrc sets rate control on Broadcast, Multicast and/or Destination Lookup Failure (DLF) packets. The rate is measured in the number of packets per time period. If the number of packets received of the specified type exceeds the specified rate limit, packets are discarded at the ingress port.

OPTIONS

-b	Enable rate control for Broadcast packets
-m	Enable rate control for Multicast packets
-d	Enable rate control for DLF packets
-t	Teardown or disable all rate control
-a	Display the current rate control settings
-p <port>	Enable rate control on this <port>
-v <vlan>	Enable rate control for this <vlan_id>
-g <group>	Enable rate control for this Multicast <group>
-M <mac_addr>	Enable rate control for this Mac <mac_addr>
-T <timeout>	Set time period to <timeout> milliseconds. Default is 1000 (one second).
-D <level>	Set debugging output to <level> when running the program.
<rate_limit>	The number of packets per time period above which Broadcasts, Multicasts and/or DLFs will be discarded. Valid rate limits are any number between 0 and 1048560 inclusive.

SEE ALSO

zstats

zreg

NAME

zreg - Read and write registers and tables on the OpenArchitect switch switching hardware.

SYNOPSIS

```
zreg [-p <ppa>] [-w] [-i <index>] [-t <index>] [-k] [-h <hostname>]
   [-d <level>] [-r 10] <reg>
```

DESCRIPTION

zreg allows a user to read and write direct and indirect registers and tables on the resident switch chip. *zreg* is commonly used for debug, or for prototyping when creating applications that control the OpenArchitect switch. It is also useful when put into shell scripts for displaying hardware status or statistics. Although the -t option allows tables to be displayed, one might find the formatted output of the table functions more useful. See *zal*.

OPTIONS

-p <ppa>	Specify the Physical Point of Attachment (PPA). Each OpenArchitect switch that is controlled by the CPU on which *zreg* is running is a unique PPA. If there is only one OpenArchitect switch, as would be the case when *zreg* is running on the embedded processor the PPA would be 0. The default PPA is 0.
-w	Causes *zreg* to write to the register or table. Data to be written is read from standard input.
-i <index>	Causes *zreg* to access at the indexed register specified by <reg>. See OPERANDS for usage of <reg> with indexed registers. The <index> parameter is used to determine which entry i
-t <index>	Causes *zreg* to access the memory specified by <reg>. The <index> parameter is used as the index into the specified table. Only content addressable memories are accessed using the –t option. All other tables and memories are accessed with the –i option.
-k	Causes *zreg* to access the memory specified by <reg>. The entry accessed is determined using the data from standard input as the search key. Enter 0 for fields that are not part of the search key.
-h <hostname>	Specify the <hostname> to configure. By default *zreg* configures the OpenArchitect switch that is locally connected (i.e., the one that is on the local PCI bus).

-r 10	Sets numeric radix for registers to 10. Default is 16.
-d <level>	Set the level of debugging output produced by *zreg*. The default level is 1. Setting the debug level higher produces more output. The maximum level of output is currently 4.

OPERANDS

<reg>	If no *–i*, *-t*, or *–k* option is specified, <reg> is chosen from the enum ix54_reg_e. With the *–i* option, <reg> is chosen from the enum ix54_xreg_e. With the *–t* and *–k* options, <reg> is chosen from ix54_mem_e.

EXAMPLES

There are three types of accesses performed by *zreg*; scaler register, indexed register, and table. For each of theses access types, values can be read or written. Content addressable memory register access is the default. The following is an example of reading the CONFIG Register:

```
zreg 1
```

When running *zreg* on the embedded processor of the OpenArchitect switch, the <ppa> is always 0, since the embedded CPU processor only controls the directly attached switch chip. Since the default <ppa> is 0 the *-p* option is not needed.

To write a value to a register the *-w* flag is used and the data is read from standard input. The following example writes 0x800000640 to the Aging Time Register:

```
echo 0x80000640 | zreg -w 49
```

If the *zreg* command is typed at the prompt, it waits for input from the user. You may also use File I/O or shell scripts. The following example reads the MAC Packet Length Register for port 7:

```
zreg -i 7 2
```

SEE ALSO

zal, zstats

zrld

NAME

zrld – ZNYX redirector deamon

SYNOPSIS

```
zrld [-d <level>] [-p <port>] [-f]
```

DESCRIPTION

zrld is used for remote management of OA/HA applications. OA/HA applications capable of remote management include *zlc*, *zstats*, *zlmd*.

zrld only allows requests from hosts listed in */etc/rcZ.d/zrld_trusted_hosts*.

OPTIONS

-d <level>	Set debug level to <level>
-p <port>	Specifies TCP port to listen on. Default port is 7000.
-f	Run the daemon in the foreground

EXAMPLES

In the following example, *zrld* starts a background task that listens on the default port 7000 for incoming TCP requests and passes along the request to the OA/HA application,

```
zrld
```

Once started, you can issue supported commands to the host running *zrld* from a remote host. For example, if the host running *zrld* had an IP address of 10.0.0.42, you could use *zlc* to remotely query the status of ports 1 through 5. Remember, the IP address of the switch you are on must be listed in */etc/rcZ.d/zrld_trusted_hosts* on the switch running *zrld*.

```
zlc –h 10.0.0.42 zre1..5 query
```

SEE ALSO

zlc, zlmd, zstats

zspconfig

NAME

zspconfig - configure and start surviving partner

SYNOPSIS

```
zspconfig [-d <level>] [-p <directory_path>] [-u <dhcp_interface>]
   [-c <dhclient.conf>] [-t <timeout>] [-s] [-v] -f <file>
```

DESCRIPTION

zspconfig is part of the OAHA product, which is a separate product from the standard OA. *zspconfig* is used to configure and start the Surviving Partner software. With the *-f* option a configuration file is provided that completely describes the network setup and desired behavior of all of the switches participating in the Surviving Partner. With the *-u* option the interface on which to run *dhclient* to retrieve a configuration file must be provided. The configuration file format retrieved by a *-u* is identical to that supplied with *-f*. It is envisioned that the *-f* option is used for initial configuration, and all subordinate and replacement switches run *zspconfig* with the *-u* option. The *-v* option prints the current version of *zspconfig* and performs no actions.

OPTIONS

-d <level> Set the debug level. The default debug level is 1. The higher the level, the more debugging output is produced. Debugging output is sent to the console.

-p <directory_path>
Set the directory path for where *zspconfig* places the scripts it generates. The default location is /etc/rcZ.d/surviving_partner.

-u < dhcp_interface>
Come up as an unconfigured slave. Use the specified <dhcp_interface> to retrieve the configuration. User confirmation is not required unless the *–s* option is also used.

-c <dhclient.conf>
Use file <dhclient.conf> as the configuration file to *dhclient* when retrieving configuration information. If *–c* is not used, a default configuration file is created and used. Only valid with the *–u* option.

-t <timeout> Time to wait in seconds before giving up on finding a Surviving Partner to retrieve configuration information from. Only valid with the *–u* option.

-s Do not ask for confirmation. Run from a script.

-v	Prints the current version of *zspconfig*.
-f <file>	The provided <file> is used as input to configure the Surviving Partner. See the next section on CONFIGURATION FILE for the syntax of the configuration file.

CONFIGURATION FILE

The configuration file contains commands for controlling the Surviving Partner setup. Commands are single lines end -delimited with a semicolon. Comments, spaces and new lines are ignored. Comments begin with the # character and include characters through the next new line. Comments may be placed on the same line as a command after the semicolon.

All configurations must include the VLAN configuration first by use of *zconfig* commands. *zconfig* commands can be put in the configuration file and will be passed directly to the *zconfig* application. *zconfig* commands start with the keyword `zconfig` and are of the same format as described in the *zconfig* manual page. Here is an example of *zconfig* commands in a *zspconfig* configuration file.

```
zconfig zhp0: vlan1 = zre1..4;
zconfig zhp1: vlan2 = zre5..8;
zconfig zhp2: vlan100 = zre14;
```

In the above example, three VLANs are created: *zhp0* and *zhp1* will be used as connections to high availability nodes; *zhp2* will be used as the inter-connect between two Surviving Partner switches to run the VRRP heartbeat. All VLANs must be created before other *zspconfig* commands may operate on them.

The next section of a *zspconfig* configuration file sets up the IP addresses of the created VLANs. There are two types of addresses in a Surviving Partner setup: *Physical Addresses*, and *Virtual Addresses*. The Virtual Addresses are those that VRRP manages and moves to the current Master switch. The Physical Addresses are the real addresses of the switch, and are used for management only. Physical Addresses are setup using the sibling_addresses command. Virtual addresses are setup using the virtual_address command. The sibling addresses name comes from the fact that we are setting up the addresses for all of the sibings in the Surviving Partner group. So if we have two Surviving Partner switches, the sibling_addresses statement might look like this:

```
sibling_addresses: zhp0 = 10.0.0.30, 10.0.0.31;
sibling_addresses: zhp1 = 11.0.0.30, 11.0.0.31;
sibling_addresses: zhp100 = 100.0.0.30, 100.0.0.31;
```

A `sibling_addresses` statement is required for each VLAN created with the *zconfig* commands. The two addresses in the list indicate there are two switches in the Surviving Partner group. The first address 10.0.0.30 and 11.0.0.30 are assigned to the switch on which the configuration is being run.

The remaining addresses are distributed to the switches that run *zspconfig -u* on a first come, first serve basis.

The `sibling_addresses` command may also take a *netmask* operand similar to that given to *ifconfig*. For example:

```
sibling_addresses: zhp0 = 10.0.0.30, 10.0.0.31 netmask
    255.255.255.0;
sibling_addresses: zhp1 = 11.0.0.30, 11.0.0.31 netmask
    255.255.255.0;
```

Virtual addresses should be setup for all VLANs that connect to *High Availability* nodes. Usually this would include all VLANs except the interconnect VLAN or VLANs connected to upstream routers. It is possible that a setup would have VLANs that are used for management only. Virtual addresses are setup as follows:

```
virtual_address: zhp0 = 10.0.0.42 netmask 255.255.255.0;
virtual_address: zhp1 = 11.0.0.42 netmask 255.255.255.0;
```

Only a single address per VLAN is provided because this single address will move with the current Master switch, and the *netmask* must be the same as that provided in the sibling_addresses statement.

The last required section for the configuration is description of the ports. Particularly we need to specify one of the following for all of the ports participating in the high availability setup. The possible port types are:

interconnect	Ports connected between groups of Surviving Partner switches. VRRP heartbeat messages are sent on the interconnect ports.
crossconnect	Crossconnect ports are ports that are connected to other Surviving Partner switches, that are not part of this Surviving Partner group. Crossconnect ports behave differently then RAINlink ports. The links are not brought down temporarily, and VRRP runs with the native MAC addresses to avoid MAC address duplication with the other VRRP group.
RAINlink	Ports connected to RAINlink enabled nodes. These ports contain virtual addresses managed by VRRP. And during a failover event, the links are toggled down to force RAINlink to failover to the Master switch.
route	Ports connected to upstream routers. VRRP does not manage virtual IP addresses for these links. Routing protocols must be used to instruct up stream routers of a different path to get to the VRRP managed networks. *These port types are currently not implemented*.
monitor_only	Ports that are monitored but do not have a virtual address managed on them. They will not have their links brought down temporarily

during a failover scenario. These ports are only monitored. If a problem occurs on this type of link it will cause a failover scenario.

configure_only — Ports are configured as per the *zconfig* commands, but do not participate in the high availability network. Problems on these links will not cause a switch failover.

Each port that is setup in a VLAN by the *zconfig* commands must have its port type specified. The port type is specified on a physical port bases. That is on a zre basis, but zhp names can be used as a quick way to setup the port type for all ports that are a member of that VLAN. It is possible to make a port a member of more then one VLAN. That is a *zre* can be a member of more then one *zhp*. In such cases, configuring the *zhp*'s as different port types would cause a conflict, and will not work. To handle this setup the individual *zre* commands would be used to setup the port types. Here is an example of setting up the port types as a continuation of our current configuration:

```
interconnect:   zhp2;          # Could also use zre14

RAINlink:       zhp0, zhp1;    # Could also use zhp0..1,
                               # or list the zre's
```

The ".." wild card is supported as in *zconfig* to indicate a range of numbers. The comma is used to indicate a list. The *zre's* that are part of the *zhp* could also be used. Here is a more complex setup. The zconfig commands are also shown to understand the VLAN setup:

```
zconfig zhp0: vlan1 = zre1..4, zre14;
zconfig zhp1: vlan2 = zre5..8, zre14;
zconfig zhp100: vlan100 = zre14;
zconfig zhp0: vlan1 = zre1..4;
zconfig zhp1: vlan2 = zre5..8;
zconfig zhp2: vlan100 = zre14;

#  sibling and virtual address setup omitted
interconnect: zhp100;
RAINlink:     zre1..8;      # Use zre definition to
                            # exclude zre14
```

If *zhp0* and *zhp100* are setup as different port types, there would be a conflict for port *zre14*. In the particular example above, the *zre14* is shared. It is used to pass VRRP interconnect traffic and as a means to pass VLAN 1 and 2 traffic between switches. Since *zre14* is an interconnect, it is not a RAINlink enabled port, and therefore should be setup as an interconnect port type. To accomplish this, the zre ports are listed to avoid conflicting port types.

Note that a single line cannot contain both *zhp* and *zre* definitions. Therefore

```
RAINlink:     zhp1, zre1..4
```

does not work and the definition *zre1..8* is equivalent.

Optional *zspconfig* commands are listed below.

```
vrrp_msg_rate:      100;   # Time in milliseconds.
                           # Default is 100 milliseconds
```

The message rate is the interval between VRRP messages sent over the interconnect link. The time given is in milliseconds. It takes the lack of 3 VRRP messages for the Backup to assume the role as Master. It is recommended with faster message rates to increase the default priority to 254. The higher priority will decrease the latency of the failover.

```
vrrp_def_priority:         254;   # default is 100.
                                  Value from 1 to 254
```

The default virtual MAC address for VRRP is 00:00:5E:00:01:<vid>, where <vid> the virtual router ID. Using this virtual address is problematic in network environments where multiple VRRP instances might be running that are using the same <vid>. To overcome this problem, *zspconfig*, uses a default MAC address derived from the physical address of the switch on which it is running. For the slave switch, the vrrp_virtual_mac_addr command is used to set the MAC address to the same as the Master. This statement is typically not used within the Master switch's configuration. It is used in the *zspconfig* generated Slave switch configuration. And is retrieved by the Slave switch with *zspconfig* using the *–u* option. If in doubt, don't use this command.

```
vrrp_virtual_mac_addr: 00:01:02:03:04:05
```

There are three failover_mode's supported; *switch*, *vlan* and *port*. For *switch* failover, if any managed link fails, the entire switch is failed over to the backup switch. In *vlan* failover, if a link fails in a VLAN, only the links associated with that VLAN are failed over. In *port* failover, only the port that fails is moved to the backup switch. For both *vlan* and *port* failover, the interconnect link will need to be used to maintain connectivity between ports that have failed over and those that have not. For *vlan* and *port* failover modes, the interconnect link must be an inband port, and must be included in the managed VLANs running with VLAN tagging on.

```
failover_mode:  switch;
failover_mode:  vlan;
failover_mode:  port;
```

Additional startup scripts may be included in the configuration using the start_script command. The files in the start_script command will be placed in a location for *tftp* transfer to sibling switches that initialize using the *–u* option. A common use of the start_script command might be to propagate

gated configurations to all members of the Surviving Partner group. Absolute path names must be used. Using multiple commands allows inclusion of multiple scripts. For example:

```
start_script: /etc/rcZ.d/S75gated;
start_script: /etc/rcZ.d/S80static_routes;
```

The vrrpd_script command allows a user defined script to be run when *vrrpd* changes state. This script is called at the end of the *zspconfig* created vrrpd.script. See *vrrpd –S* for a description of when the scripts are called. *zspconfig* sets up *vrrpd* to call vrrpd.script. The vrrpd_script command in *zspconfig* places a call to the user-defined script at the end of vrrpd.script file. The following example would call the my_vrrpd_script each time *vrrpd* calls its *–S* provided script:

```
vrrpd_script:  /etc/rcZ.d/surviving_partner/my_vrrpd_script;
```

NOTE: the my_vrrpd_script is not called from a different process thread. Therefore if my_vrrpd_script crashes or has long delays, it will crash the *vrrpd*, or cause delays in the Surviving Partner failover. To protect against this, write the script to launch a second script in a background shell. The advantage to calling the user provided script in the same process thread is that it gives synchronized control over the failover process for those who want it.

OUTPUT FILES

The output of *zspconfig* is a set of configuration and script files. The configuration files configure *vrrpd* and *zlmd* daemons. The *vrrpd* and *zlmd* daemons combined with the script files run the Surviving Partner. This is a list of all configuration and script files:

/etc/rcZ.d/S70Surviving_partner

The main startup script that starts the Surviving Partner by running *zconfig*, *ifconfig*, *zlmd* and *vrrpd*. *zspconfig* prompts the user to run this script. This file can be saved with *zsync* to automatically start the Surviving Partner at switch boot.

/etc/rcZ.d/surviving_partner/vrrpd.conf

Configuration script for the VRRP daemon. This configuration is used when the S70Surviving_partner script launches *vrrpd*. There is a line in this file for each router address *vrrpd* will manage. Or stated another way, each virtual_address command in the *zspconfig* configuration file results in a line in vrrpd.conf.

/tftpboot/zsp.conf<*n*>

zspconfig configuration file that contains the configuration of the sibling backup switches. The <n> is used to distinguish potentially more than one backup switch. This configuration file is placed in /tftpboot, and is retrieved via DHCP by a replacement switch on boot up.

/etc/rcZ.d/surviving_partner/dhcpd.conf

Configuration script used by *dhcpd* when the switch becomes master. *dhcpd* is used to serve replacement switches their configuration scripts. Namely a zsp.conf file that can be input to the *zspconfig* with the *-u* flag.

/etc/rcZ.d/surviving_partner/dhclient.conf

If *zspconfig* is executed with the *-u* flag, a dhclient.conf file is created, and then *dhclient* is used to retrieve a *zspconfig* configuration file from the /tftpboot area of the Master switch.

/etc/rcZ.d/surviving_partner/vrrpd.script

Runtime script that executes each time the *vrrpd* changes state. This script starts and stops *dhcpd*, and toggles down RAINlink ports to force the RAINlink nodes to a new Master switch.

/etc/rcZ.d/surviving_partner/zlmd.script

Runtime script executed by *zlmd* when a link goes up or down. This script modifies the priority of *vrrpd*, which in turn may cause the VRRP Master to move from one sibling switch to another.

SEE ALSO

zconfig, ifconfig, vrrpd, dhclient, dhcpd

zstats

NAME

zstats – Display statistics and information about switch

SYNOPSIS

```
zstats  [-d <level>] [-i <unit>] | [-m <port>] |
   [-v <vlan id>] | [-t <tgid>] | [-V]
```

DESCRIPTION

zstats displays MIB counters for a selected physical port, trunk group or VLAN. It can also display information about the configuration of the switch and bridge to the PCI bus or the Vital Product Data memory. All output is formatted.

OPTIONS

-m <port>	MIB statistics for specified <port>
-v <vlan id>	MIB statistics for specified <vlan id>
-i <unit>	Information for specified <unit>, 0 is switch, 1 is bridge
-d <level>	Set debug level to <level>
-p <ppa>	Specify <ppa> of switch (default 0).
-t <tgid>	MAC layer statistics for all ports in trunk <tgid>.
-V	Vital Product Data (not currently supported in ZX5000)

EXAMPLES

To display statistics for a particular port on the switch, such as port 0

```
zstats –m 0
```

SEE ALSO

zreg, zal

zsync

NAME

zsync – Saves changes to the flash.

SYNOPSIS

```
zsync [-c][-f][<dir_or_file>]
zsync [-c][-f][-t <file>]
zsync [-c][-f][-z]
zsync [-c][-l]
```

DESCRIPTION

zsync is used to save a snapshot of the current file system to flash ROM. By default, *zsync* creates a compressed tar image of the files that have changed and saves the image in the flash ROM. The saved image is expanded on reboot. The saved compressed tar image is called an "overlay".

If a directory parameter is given to *zsync*, the contents of the directory are saved instead of searching for updated files. The specific purpose of the <directory> parameter is for saving files that have been mounted with *zmnt*. Using the *-t* option allows a tar image created by `zmnt -t` to be saved.

To correct a corrupted file that is saved to flash ROM with *zsync*, first reboot with the *-i* option (*see Switch Maintenance*). Use *zmnt* to put the corrupted file in the /mnt directory, open and correct the file, then *zsync* to the /mnt directory to save your changes and reboot.

There are two overlay areas: *dynamic* and *custom*. The dynamic overlay is the default. The custom overlay is used to create base configurations that are different then the ZNYX base configuration. The intent of the custom overlay is to support OEMs and other special customers that require an overlay different from the standard ZNYX overlay. The custom overlay is written by using the *-c* option.

The *-z* option zeroes the overlay area, returning the switch to the factor configuration.

Specific files or directories can be excluded from saving to flash by *zsync* by including an entry in `/etc/exclude`. Likewise, existing entries in `/etc/exclude` such as `/tmp` can be removed in order to save those files to flash with *zsync*.

OPTIONS

-c	Save files to the custom overlay
-t <file>	Read files to be saved from a tar file.
-z	Zero the overlay area.
-f	Do not confirm with user and do not warn if saving failed. Exit code can be examined to determine success or failure.
<dir_or_file>	Save only the named file, or save the named directory to the overlay. Contents of directories must be created with *zmnt.*
-l	List files that would be written. Do not flash.

EXAMPLES

To *zsync* only the hosts file:

```
cd /etc
zsync hosts
```

If you previously created a snapshot of an overlay to a tar file using *zmnt*,

```
zmnt –t overlay.tar
```

You can use *zsync* to restore the overlay on the switch directly from the tar file,

```
zsync –t overlay.tar
```

The restored overlay will be loaded upon the next reboot.

FILES

```
/etc/exclude, /.zsync
```

SEE ALSO

zmnt

ztmd

NAME

ztmd – traffic management daemon which accepts messages from traffic filtering and quality of service applications and sets up hardware.

SYNOPSIS

```
ztmd [-d <level>] [-p <port>] [-f] [-l] [-i <pid>]
```

DESCRIPTION

ztmd listens for messages on a multicast port. These messages describe packet filters and queuing disciplines that are to be installed in the switch hardware. *ztmd* interprets these messages to set up the switch hardware.

OPTIONS

-d <level>	Set the level of diagnostic information logged. <level> may be 0-4; higher levels produce more output.
.p <port>	Use <port> as the multicast listening port for communication with *ztmd*. Default is 2345.
-f	Run *ztmd* in the foreground. Without this option, it is run as a daemon.
-l	Log diagnostic output to `/var/log/ztmd.log`
-i <pid>	Set the pid for this process.

EXAMPLES

Start the traffic management daemon, *ztmd*, then start *zqosd* to monitor *tc* output and zfilterd to monitor *iptables(8)* output. All daemons are run as background processes and log their messages to files in /var/log.

```
ztmd –l
zqosd –l
zfilterd -l
```

SEE ALSO

zqosd, iptables(8), tc(8), zfilterd

brctl(8)

NAME

brctl - Bridge and Spanning Tree Protocol administration.

SYNOPSIS

```
brctl [options]
```

DESCRIPTION

brctl is used to set up, maintain, and display the bridge configuration in the Linux kernel. *brctl* is a standard command included with Linux bridge support including Spanning Tree Protocol (STP), which has been modified to include ZNYX Fast Forward Bridge (ffbridge), and Fast Forward Port (ffport) options.

A bridge is a device commonly used to connect different networks together, so that these networks will appear as one network to the participants.

Each of the networks being connected corresponds to one physical interface, or port in the bridge. These individual networks are bundled into one bigger logical network. This bigger network corresponds to the bridge network interface.

Multiple bridges can work together to create even larger networks using the IEEE 802.1d Spanning Tree Protocol. This protocol is used for finding the shortest path between two networks as well as eliminating loops from the topology. Bridges communicate with each other by sending and receiving Bridge Protocol Data Units (BPDUs).

brctl(8) can be used for configuring certain spanning tree protocol parameters. For an explanation of these parameters, see the IEEE 802.1d specification for detailed information.

OPTIONS

addbr <bridge>

creates a new instance of a bridge. The network interface corresponding to the bridge will be called <bridge>. For the OpenArchitect switch, bridges are named bzhp0, bzhp1, etc.

NOTE: This option must only be executed by *zl2d*.

delbr <bridge>

deletes the instance <bridge> of an ethernet bridge. The network interface corresponding to the bridge must be down before it can be deleted.

NOTE: This option must only be executed by *zl2d*.

show shows all current bridges.

addif <bridge> <interface>
makes the interface a port of the bridge. This means that all frames received on the interface will be processed as if destined for the bridge. Also, when sending frames on the bridge, the interface will be considered as a potential output interface. For the OpenArchitect switch, <interface> is zhp0, zhp1, …

NOTE: This option must only be executed by *zl2d*.

delif <bridge> <interface>
detaches the interface from the bridge.

NOTE: This option must only be executed by *zl2d*.

showbr <bridge>
shows information for the bridge and it's attached ports.

showmacs <bridge>
shows a list of learned MAC addresses for the bridge.

setageingtime <bridge> <time>
sets the ethernet (MAC) address aging time, in seconds. After <time> seconds of not having seen a frame coming from a certain address, the bridge will time out (delete) that address from the Forwarding DataBase (fdb).

setgcint <bridge> <time>
sets the garbage collection interval for the bridge to <time> seconds. This means that the bridge will check the forwarding database for timed out entries every <time> seconds.

stp <bridge> <state>
controls this bridge's participation in the Spanning Tree Protocol. <state> can be “off” or “on”. When turned off, the bridge will not send or receive BPDUs, and will thus not participate in the Spanning Tree Protocol. If your bridge isn't the only bridge on the LAN, or if there are loops in the LAN's topology, DO NOT turn this option off. If you turn this option off, please know what you are doing.

setbridgeprio <bridge> <priority>
sets the bridge's priority to <priority>. The priority value is an unsigned 16-bit quantity (a number between 0 and 65535), and has no dimension. Lower priority values are better. The bridge with the lowest priority will be elected Root Bridge.

setfd <bridge> <time>
sets the bridge's *bridge forward delay* to <time> seconds.

sethello <bridge> <time>
sets the bridge's *bridge hello time* to <time> seconds.

setmaxage <bridge> <time>
sets the bridge's *maximum message age* to <time> seconds.

setpathcost <bridge> <port> <cost>
sets the port cost of the port to <cost>. This is a dimensionless metric. The path cost is set to 100 for all OpenArchitect switch ports by default. IEEE 802.d recommends the following:

Link Speed	Recommended Value	Recommended Range
10 Mb/s	100	50-600
100 Mb/s	19	10-60
1 Gb/s	4	3-10

setportprio <bridge> <port> <priority>
sets the port's priority to <priority>. The priority value is an unsigned 8-bit quantity (a number between 0 and 255), and has no dimension. This metric is used in the designated port and root port selection algorithms. For the OpenArchitect switch a port is zre1, zre2, …

ffbridge <bridge> <state>
sets the bridge's Fast Forward state to enabled or disabled. If Fast Forward is enabled for the bridge, then whenever STP picks a new Root Port, that port will bypass the Listening and Learning states and transition immediately to Forwarding.

ffport <bridge> <port> <state>
sets the port's Fast Forward state to enabled or disabled. If Fast Forward is enabled for this port, then this port will always bypass the Listening and Learning states and transition immediately to Forwarding. Fast Forward should ONLY be enabled on ports connected to end-stations.

NOTES

brctl(8) replaces the older *brcfg* tool.

SEE ALSO

zconfig, zl2d

Appendix B PENDING Legal Notices and Certification

These are the certifications and approvals targeted for the ZX5000.

FCC Class A Notice: OPENARCHITECT ZX5000 SERIES

Modification to this product not authorized by ZNYX Networks could void the FCC approval and negate your authority to operate the product.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if it is not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Canada Compliance

This Class A digital apparatus complies with Canadian ICES-003.

Cet appareil numeriqué de la classe A est conforme à la norme NMB-003 du Canada.

CE Conformity

The ZX5000 Series is in compliance with the following standards:

EN 55022 (1998) Class A
EN 55024 (1998)
EN 61000-4-11 (1995)
EN 60950 (1992)+ Amendment 1, 2, 3, 4, and 11

THE ZX5000 IS FOR USE ONLY WITH COMPACT PCI BUS COMPATIBLE SYSTEMS THAT HAVE ENCLOSED POWER SUPPLIES WITH SELV OUTPUTS AND INSTALLATION INSTRUCTIONS DETAILING USER INSTALLATION OF CARD CAGE ACCESSORIES.

Manufacturer Name: ZNYX Networks

Manufacturer Address: 48421 Milmont Drive, Fremont, CA 94538

Warranty

EARLY ACCESS UNITS ARE PROVIDED AS IS WITH NO WARRANTY.

The following is the standard warranty that will be provided for the ZX5000 once it is production released.

ZNYX Networks warrants to the original purchaser of any ZNYX Networks compact PCI Ethernet Switch product that is to be free from defects in workmanship and materials, under normal use and service, for a period of one year from the date of purchase from ZNYX Networks or its authorized dealer. In order for this warranty to be valid, this hardware product must remain in its original system and be registered with ZNYX Networks within one year of purchase. ZNYX Networks warrants to the original purchaser of this hardware product that it is to be in good working order for a period of twelve (12) months from the date of purchase from ZNYX Networks or an authorized dealer.

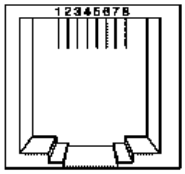
Should this product, in ZNYX Network's opinion, malfunction during the applicable warranty period, ZNYX Networks will, at its expense, repair the defective product or part or, at its option, deliver to the Customer an equivalent product or part to replace the defective item. To prevent damage in transport, the Customer must return the product in its original packaging or, if this is not available, other protective packaging approved in advance by ZNYX Networks. All returned products will become the property of ZNYX Networks. At ZNYX Network's option, replacement parts may be new or reconditioned. Any replaced product or part has a ninety (90) day warranty or the remainder of the initial warranty period, whichever is longer

Appendix C Console Port Pinout

The serial Console Port for the ZX5000 is connected via an RJ45 located on the front panel. The pin out for the connector is shown below.

Connector: RJ-45, pin out defined below

Signal	Pin #
RTS	1
DTR	2
TXD	3
GND	4
GND	5
RxD	6
DSR	7
CTS	8



RJ-45 Jack

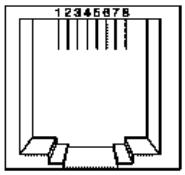
Appendix D Pinout of Front Panel Ethernet Ports

10/100 Management Port Pinout

There are two out of band (OOB) 10/100 Ethernet ports for the ZX5000. The eth0 port is available via an RJ45 on the front panel.

Eth 0 Management Connector: RJ-45, pin out defined below

Signal	Pin #
RX+	1
RX-	2
TX+	3
GND	4
GND	5
TX-	6
GND	7
GND	8



RJ-45 Jack

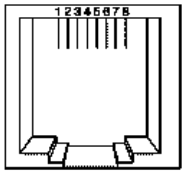
The eth1 connects to the ShMC, which may provide an RJ-45 connection point.

1000Base T Port Pinout

There are three front panel 10/100/1000 Ethernet ports for the ZX5000. The OpenArchitect software nominally refers to them as zre0, zre1 and zre2 and they are available via RJ45 connectors.

1000BaseT Switch Port Connector: RJ-45, pin out defined below

Signal	Pin #
TP0+	1
TP0-	2
TP1+	3
TP2+	4
TP2-	5
TP1-	6
TP3+	7
TP3-	8



RJ-45 Jack

Appendix E Rear Panel Pinouts

The ZX5000 uses ATCA base switch fabric compliant pin outs and are outlined.

Table of pins for Zone 1 Connector

Pin	Description
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	Geographical address bit 0
6	Geographical address bit 1
7	Geographical address bit 2
8	Geographical address bit 3
9	Geographical address bit 4
10	Geographical address bit 5
11	Geographical address bit 6
12	Geographical address bit 7 (odd parity bit)
13	IPMB0 clock
14	IPMB0 data
15	IPMB1 clock
16	IPMB1 data
17-24	No connect
25	Shelf Ground
26	Logic Ground
27	Enable B Voltage
28	-48V Return A
29	-48V Return B
30	-48V Early A
31	-48V Early B
32	Enable A Voltage
33	-48V Input – A
34	-48V Input – B

Table of pins for J23/P23

Row	Interface Designation	A	B	C	D	E	F	G	H
1	Fabric								
2	Channel 2								
3	Fabric								
4	Channel 1								
5	ShMC	ShM-Tx+	ShM-Tx-	ShM-Rx+	ShM-Rx-				
6	Base Ch 2 Fabric Link	Tx1[15]+	Tx1[15]-	Tx2[15]+	Tx2[15]-	Tx3[15]+	Tx3[15]-	Tx4[15]+	Tx4[15]-
7	Base Ch 3	Tx1[3]+	Tx1[3]-	Tx2[3]+	Tx2[3]-	Tx3[3]+	Tx3[3]-	Tx4[3]+	Tx4[3]-
8	Base Ch 4	Tx1[4]+	Tx1[4]-	Tx2[4]+	Tx2[4]-	Tx3[4]+	Tx3[4]-	Tx4[4]+	Tx4[4]-
9	Base Ch 5	Tx1[5]+	Tx1[5]-	Tx2[5]+	Tx2[5]-	Tx3[5]+	Tx3[5]-	Tx4[5]+	Tx4[5]-
10	Base Ch 6	Tx1[6]+	Tx1[6]-	Tx2[6]+	Tx2[6]-	Tx3[6]+	Tx3[6]-	Tx4[6]+	Tx4[6]-

Table of pins for J24/P24

Row	Interface Designation	A	B	C	D	E	F	G	H
1	Base Ch 7	Tx1[7]+	Tx1[7]-	Tx2[7]+	Tx2[7]-	Tx3[7]+	Tx3[7]-	Tx4[7]+	Tx4[7]-
2	Base Ch 8	Tx1[8]+	Tx1[8]-	Tx2[8]+	Tx2[8]-	Tx3[8]+	Tx3[8]-	Tx4[8]+	Tx4[8]-
3	Base Ch 9	Tx1[9]+	Tx1[9]-	Tx2[9]+	Tx2[9]-	Tx3[9]+	Tx3[9]-	Tx4[9]+	Tx4[9]-
4	Base Ch 10	Tx1[10]+	Tx1[10]-	Tx2[10]+	Tx2[10]-	Tx3[10]+	Tx3[10]-	Tx4[10]+	Tx4[10]-
5	Base Ch 11	Tx1[11]+	Tx1[11]-	Tx2[11]+	Tx2[11]-	Tx3[11]+	Tx3[11]-	Tx4[11]+	Tx4[11]-
6	Base Ch 12	Tx1[12]+	Tx1[12]-	Tx2[12]+	Tx2[12]-	Tx3[12]+	Tx3[12]-	Tx4[12]+	Tx4[12]-
7	Base Ch 13	Tx1[13]+	Tx1[13]-	Tx2[13]+	Tx2[13]-	Tx3[13]+	Tx3[13]-	Tx4[13]+	Tx4[13]-
8	Base Ch 14	Tx1[14]+	Tx1[14]-	Tx2[14]+	Tx2[14]-	Tx3[14]+	Tx3[14]-	Tx4[14]+	Tx4[14]-
9	Base Ch 15								
10	Base Ch 16								

Appendix F IPMI

The ZX5000 provides PICMG 3.0 Intelligent Platform Management Interface (IPMI) support. IPMI circuitry provides:

- The communication channel between the Baseboard Management Controller (BMC) and the CPU for management.
- Data storage, SDRR, FRU, access.
- Sensors reading

I^2C Bus Devices

The following table outlines the I^2C bus connections and devices.

Signal	Description
SDA0	Connected to Backplane IPMI Bus0 SDA.
SCL0	Connected to Backplane IPMI Bus0 Clock.
SDA1	Connected to Backplane IPMI Bus1 SDA.
SCL1	Connected to Backplane IPMI Bus1 Clock.
SDA2	Not Used in ZX5000
SCL2	Not Used in ZX5000
SDA3	Connected to Temp-Voltage Sensors & Hot Swap Controller Address 00 – LM77 Temperature Sensor Address 00 – SMH4804 Hot Swap Controller Address 04 – AT24C64 IPMI Eeprom Address 02 – ADM1026 Voltage Sensor
SCL3	Connected to Temp-Voltage Sensors & Hot Swap Controller
SDA4	Connected to MPC8255 & Eeproms. Address 30 – MPC8255 (Set by Zmon) Address A6 – 24C256 OEM Eeprom Address AC – ATC24C02 Hardware Eeprom Address AE – ATC24C02 Mac Eeprom
SCL4	Connected to MPC8255.
SDA5	Not Used in ZX5000
SCL5	Not Used in ZX5000

Interrupt Pins

Interrupts in the IPMI subsystem are connected as outlined in the following table.

Signal	Int #	Description
/TMP_LM77	0	Interrupt from Temperature Sensor.
/THERM_LIMIT	1	Interrupt from ADM1026 Temperature & Voltage Sensor.
/HOTSWAP_DET	2	Interrupt from How Swap Switch
/HECETA5_INT	3	Interrupt from ADM1026 Temperature & Voltage Sensor.
/SMC_XINT4	4	Not Used in ZX5000
/RESET_PB	5	Interrupt from Push Button Reset
/SMC_XINT6	6	Not Used in ZX5000
/CPU_INT	7	Interrupt from MPC8255

More details about IPMI command and feature set will be provided at a future date.

Appendix G ZNYX Private MIBs

This section contains the definition of the ZNYX Private MIBs. The private MIBs reside with the other MIB definitions in /usr/share/snmp/mibs on the switch. There are two ZNYX private MIBs:

ZNYX-H.MIB
: The ZNYX Networks SNMP MIB

ZX5000.MIB
: The OpenArchitect switch product-specific MIB

ZNYX-H.MIB

```
ZNYX-MIB DEFINITIONS ::= BEGIN

IMPORTS

        enterprises, NetworkAddress, IpAddress, Counter,
        Gauge, TimeTicks, OBJECT-TYPE
                FROM RFC1155-SMI;

-- Conventions

DisplayString ::= OCTET STRING
DateAndTime   ::= OCTET STRING (SIZE (8|11))

znyx                    OBJECT IDENTIFIER ::= { enterprises 688 }

-- ZNYX Identifiers

znyxIds                 OBJECT IDENTIFIER ::= { znyx 1 }
znyxExperimentalIds     OBJECT IDENTIFIER ::= { znyxIds 1 }
znyxProducts            OBJECT IDENTIFIER ::= { znyxIds 2 }

-- znyxProducts

znyxBridges             OBJECT IDENTIFIER ::= { znyxProducts 1 }
znyxSwitches            OBJECT IDENTIFIER ::= { znyxProducts 2 }
znyxAdapters            OBJECT IDENTIFIER ::= { znyxProducts 3 }

-- znyxBridges

zx100                   OBJECT IDENTIFIER ::= { znyxBridges 1 }
zx110                   OBJECT IDENTIFIER ::= { znyxBridges 2 }

-- znyxSwitches

znyxOpenArchitect       OBJECT IDENTIFIER ::= { znyxSwitches 1 }
znyxZX4500              OBJECT IDENTIFIER ::= { znyxOpenArchitect 1 }
```

```
-- znyxAdapters

znyxRainAdapters        OBJECT IDENTIFIER ::= { znyxAdapters 1 }
znyxRainARS             OBJECT IDENTIFIER ::= { znyxRainAdapters 1 }
znyxRainACS             OBJECT IDENTIFIER ::= { znyxRainAdapters 2 }

-- ZNYX MIBs

znyxMibs                OBJECT IDENTIFIER ::= { znyx 2 }
znyxExperimentalMib     OBJECT IDENTIFIER ::= { znyxMibs 1 }
znyxMibOne              OBJECT IDENTIFIER ::= { znyxMibs 2 }
znyxProductSpecificMib  OBJECT IDENTIFIER ::= { znyxMibs 3 }

-- znyxSystemMib

-- The system mib describes objects commonly found in most products.

znyxSystemMib           OBJECT IDENTIFIER ::= { znyxMibOne 1 }
zsysHardware            OBJECT IDENTIFIER ::= { znyxSystemMib 1 }
zsysSoftware            OBJECT IDENTIFIER ::= { znyxSystemMib 2 }
zsysLicenses            OBJECT IDENTIFIER ::= { znyxSystemMib 3 }
zsysControl             OBJECT IDENTIFIER ::= { znyxSystemMib 4 }
zsysEventLog            OBJECT IDENTIFIER ::= { znyxSystemMib 5 }

-- Hardware group

zhwCompNumber OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The number of hardware components in this system."
        ::= { zsysHardware 1 }



zhwCompTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF ZhwCompEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "A list of hardware components."
        ::= { zsysHardware 2 }

zhwCompEntry OBJECT-TYPE
        SYNTAX  ZhwCompEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "A component table entry."
        INDEX   { zhwCompIndex }
        ::= { zhwCompTable 1 }

zhwCompEntry ::=
        SEQUENCE {
```

```
                zhwCompIndex
                        INTEGER,
                zhwCompType
                        INTEGER,
                zhwCompOid
                        OBJECT IDENTIFIER,
                zhwCompDescr
                        DisplayString,
                zhwCompSerial
                        DisplayString
        }

    zhwCompIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Component index."
        ::= { zhwCompEntry 1 }

    zhwCompType OBJECT-TYPE
        SYNTAX  INTEGER {
                other(1),       -- none of the following
                rack(2),
                chassis(3),
                module(4),
                baseboard(5),
                extender(6),
                port(7),
                mau(8)
                }
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Component type."
        ::= { zhwCompEntry 2 }

    zhwCompOid OBJECT-TYPE
        SYNTAX  OBJECT IDENTIFIER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Component OID."
        ::= { zhwCompEntry 3 }

    zhwCompDescr OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Component description."
        ::= { zhwCompEntry 4 }

    zhwCompSerial OBJECT-TYPE
  SYNTAX  DisplayString (SIZE (0..255))
  ACCESS  read-only
```

```
        STATUS  mandatory
        DESCRIPTION
            "Component serial number."
        ::= { zhwCompEntry 5 }

        -- Control group

        zctrlState OBJECT-TYPE
                SYNTAX  INTEGER { okay(1), problematic(2) }
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "The systems state. A state of okay indicates
                        the system functions to spec. A state of
                        problematic indicates compromised functionality."
                ::= { zsysControl 1 }

        zctrlReset OBJECT-TYPE
                SYNTAX  INTEGER { running(1), softreset(2), hardreset(3) }
                ACCESS  read-write
                STATUS  mandatory
                DESCRIPTION
                        "This variable always returns running on reads. If
                        softreset is written, the unit will perform a
                        software-only reset if possible, or a hardware
                        reset otherwise. If hardreset is written, the unit
                        will perform a hardware reset if possible, or
                        a software-only reset otherwise."
                ::= { zsysControl 2 }

    zctrlPassword OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (1..255))
        ACCESS  read-write
        STATUS  optional
        DESCRIPTION
            "This object is used to change system passwords. On
            reads, the object returns nothing meaningful.
            If written with a currently valid account and password,
            separated by one whitespace character, this password
            will become once-writable by the specific agent for
            20 seconds. No indication is returned as to whether the
            supplied account/password combination was valid.
            To change the password, once this object has become
            writable, write the account and new password,
            separated by one whitespace character. The password
            gets stored in the system if it satisfies system
            password policies.
            If writable, this object becomes non-writable again
            immediately after being written to.
            Beware: this mechanism is about as safe as a cleartext
            telnet session."
        ::= { zsysControl 3 }

        zctrlSystemUptime OBJECT-TYPE
                SYNTAX     TimeTicks
                ACCESS read-only
```

```
            STATUS     mandatory
            DESCRIPTION
                    "The amount of time since this host was last
                    initialized.  Note that this is different from
                    sysUpTime in the SNMPv2-MIB [RFC1907] because
                    sysUpTime is the uptime of the network management
                    portion of the system."
            ::= { zsysControl 4 }

  zctrlSystemDate OBJECT-TYPE
            SYNTAX     DateAndTime
            ACCESS read-write
            STATUS     mandatory
            DESCRIPTION
                    "The host's notion of the local date and time of day. "
            ::= { zsysControl 5 }

  zctrlSystemProcedureTrigger OBJECT-TYPE
SYNTAX  DisplayString (SIZE (0..255))
ACCESS  read-write
STATUS  mandatory
DESCRIPTION
        "Writing a name to this object will start the
        system procedure identified by the name. No
        action is taken if no system procedure matching
        name is defined. The name written may be followed
        by arguments, separated by whitespace. For
        instance, system procedures may be looked up in
        the file /etc/procedures.conf. Each line of this
        file will list a procedure name, minimum interval
        in seconds, uid, gid, working directory, and the
        command to be executed, including required arguments.
        If the minimum interval has passed since the procedure
        last was triggered, or the procedure has not been
        triggered since last system boot, the snmp agent will
        perform a chroot to the working directory if it is
        not the root directory, change the user id and group
        id to uid and gid, respectively, and then execute the
        specified command. Any arguments supplied with the
        procedure name will get appended, not overriding
        required arguments from the procedures file. This
        mechanism should not use the system() call.
        Care should be taken that supplied arguments like
        ';/bin/rm -rf /*' do not have the intended effect.
        On reads, the object will return the last value
        written, or the empty string if no value has been
        written yet."
::= { zsysControl 6 }

  -- Product specific MIBs

  -- znyxPsMibZX5000 OBJECT IDENTIFIER ::= { znyxProductSpecificMib 3 }


  END
```

ZX5000.MIB

```
ZX5000-MIB DEFINITIONS ::= BEGIN

IMPORTS
        enterprises, NetworkAddress, IpAddress, Counter,
        Gauge, TimeTicks, OBJECT-TYPE
                FROM RFC1155-SMI
        znyxProductSpecificMib
                FROM ZNYX-MIB;

-- Conventions

DisplayString ::= OCTET STRING



znyxPsMibZX5000                 OBJECT IDENTIFIER ::= { znyxProductSpecificMib 3 }

zZX5000GeoPins OBJECT-TYPE
        SYNTAX  DisplayString (SIZE (0..255))
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "This object returns a string identifying
                 the shelf and the slot within the shelf
                 the switch is located in."
        ::= { znyxPsMibZX5000 1 }

zZX5000InternalFaultLedControl OBJECT-TYPE
        SYNTAX  INTEGER {
                off(1),
                on(2),
                other(3) -- driven by other control
                }
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "This object is used to determine how the
                 Internal Fault LED is driven."
        ::= { znyxPsMibZX5000 2 }

zZX5000InternalFaultLedStatus OBJECT-TYPE
        SYNTAX  INTEGER {
                off(1),
                on(2)
                }
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "This object is used to determine the
                 current Internal Fault LED status."
        ::= { znyxPsMibZX5000 3 }

zZX5000ExternalFaultLedControl OBJECT-TYPE
```

```
        SYNTAX  INTEGER {
                off(1),
                on(2),
                other(3) -- driven by other control
                }
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "This object is used to determine how the
                 External Fault LED is driven."
        ::= { znyxPsMibZX5000 4 }

zZX5000ExternalFaultLedStatus OBJECT-TYPE
        SYNTAX  INTEGER {
                off(1),
                on(2)
                }
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "This object is used to determine the
                 current External Fault LED status."
        ::= { znyxPsMibZX5000 5 }

zZX5000OpenArchVersion OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS     mandatory
        DESCRIPTION
                "The Open Architect software version."
        ::= { znyxPsMibZX5000 6 }

zZX5000LinkCtrlTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF ZZX5000LinkCtrlEntry
        ACCESS  not-accessible
        STATUS      mandatory
        DESCRIPTION
        "A table containing link control information for each port
             available on the device."
        ::= { znyxPsMibZX5000 7 }

zZX5000LinkCtrlEntry OBJECT-TYPE
        SYNTAX      ZZX5000LinkCtrlEntry
        ACCESS  not-accessible
        STATUS      mandatory
        DESCRIPTION
        "Link control information for each port available on the
                 device."
        INDEX   { zZX5000LinkCtrlPort }
        ::= { zZX5000LinkCtrlTable 1 }

ZZX5000LinkCtrlEntry ::=
        SEQUENCE {
                zZX5000LinkCtrlPort
                        INTEGER,
                zZX5000LinkCtrlState
```

```
                    INTEGER,
                zZX5000LinkCtrlSpeed
                    INTEGER
        }

zZX5000LinkCtrlPort OBJECT-TYPE
        SYNTAX      INTEGER (1..65535)
        ACCESS  not-accessible
        STATUS      mandatory
        DESCRIPTION
        "The port number associated with this Link Control entry."
        ::= { zZX5000LinkCtrlEntry 1 }

zZX5000LinkCtrlState OBJECT-TYPE
        SYNTAX      INTEGER { up (1), down (2) }
        ACCESS  read-write
        STATUS      mandatory
        DESCRIPTION
        "The link control state of the port."
        ::= { zZX5000LinkCtrlEntry 2 }

zZX5000LinkCtrlSpeed OBJECT-TYPE
        SYNTAX      INTEGER  { auto(1), n100fd(2), n100hd(3), n10fd (4), n10hd(5)}
        ACCESS  read-write
        STATUS      mandatory
        DESCRIPTION
        "The link control speed of the port. Changes to this speed
                require the port to be in the down state."
        ::= { zZX5000LinkCtrlEntry 3 }

END
```

Appendix H Networking References

For further reference information regarding ZNYX Networks products, Linux, switch, and other networking-related subjects, you can browse through the Internet web sites below:

ZNYX Networks — Sales and Technical Support

web site: www.znyx.com

Apt-Get

Apt-Get utility information and man page:

http://www.debian.org/doc/manuals/apt-howto/index.en.html

http://www.fifi.org/cgi-bin/man2html/usr/share/man/man8/apt-get.8.gz

Creating Debian Packages:

http://www.debian.org/doc/debian-policy/

http://people.debian.org/~jaldhar/make_package1.html

Creating Repositories

http://ibiblio.org/gferg/ldp/giles/repository/repository.html

Busy Box

http://busybox.net/

GateD

www.gated.org

Linux Commands

http://linuxcentral.com/linux/man-pages/index.html

Tiny Login

http://tinylogin.busybox.net/

Virtual Local Area Network (VLAN)

http://www.summa4.com/univercd/cc/td/doc/product/lan/cat5000/rel_5_2/config/vlans.htm

Quality of Service (tc, QoS)

Linux Advanced Routing & Traffic Control HOWTO. Bert Hubert.
http://ds9a.nl/lartc/

Differentiated Services on Linux.
http://diffserv.sourceforge.net/

IP Quality of Service on Linux. J. Hadi Salim. July 24, 1999.
http://www.davin.ottawa.on.ca/ols/

Appendix I Glossary

Application Flash ROM

This is code embedded in the Boot ROM which may contain one or more images that are placed there as a value-added operation to generate an appropriate executable, compressed image.

Boot loader

The first executable used to start another program. For the OpenArchitect switch, a boot loader is placed within the Boot ROM device and is used to start OpenArchitect Linux software from the internal MPC8240 processor.

Boot ROM Image

ZNYX Networks-defined code (firmware) that is programmed into the Boot ROM located in the processor (see Boot ROM). In the OpenArchitect switches, the Boot ROM image is also the boot loader.

Boot ROM

A flash ROM device connected to the X bus of the MPC8240 processor. The switches use a Motorola MPC8240 Boot ROM.

CarrierClass™

A term used to identify telecom products that are capable of meeting the requirements of the telecom industry. CarrierClass is a trademark of ZNYX Networks.

daemon

A program that is not invoked explicitly, but lies dormant waiting for some condition(s) to occur. The idea is that the perpetrator of the condition need not be aware that a daemon is lurking (though often a program will commit an action only because it knows that it will implicitly invoke a daemon).

Unix systems run many daemons, chiefly to handle requests for services from other hosts on a network. Most of these are now started as required by a single real daemon, *inetd*, rather than running continuously. Examples are *cron* (local timed command execution), *rshd* (remote command execution), *rlogind* and *telnetd* (remote login), *ftpd, nfsd* (file transfer), *lpd* (printing).

Domain Name System (DNS)

Name resolution software that lets users locate computers on a UNIX network or the Internet (TCP/IP network) by domain name. The DNS server maintains a database of domain names (host names) and their corresponding IP addresses.

In the following hypothetical example, if **www.mycompany.com** were presented to a DNS server, the IP address **204.0.8.51** would be returned. DNS has replaced the manual task of updating HOSTS files in an inhouse UNIX network.

Dynamic Host Configuration Protocol (DHCP)

Software that automatically assigns IP addresses to client stations logging onto a TCP/IP network. It eliminates having to manually assign permanent IP addresses. DHCP software typically runs in servers and is also found in network devices such as ISDN routers and modem routers that allow multiple users access to the Internet. Newer DHCP servers dynamically update the DNS servers after making assignments.

dhclient

Runs the DHCP client application. Is also used by a replacement switch at initialization to get its configuration from the Surviving Partner

dhcpd

Runs the DHCP server. Started when a switch is the VRRP Master to configure replacement switches

ftp

(File Transfer Program) FTP is a command in Linux that sends and receives files interactively from your system hard drive or other remote computer systems. The ftp command includes built-in help.

flash memory

A memory chip that can be rewritten and hold its content without power. It is widely used for digital film and for data and programs for communications and industrial products as well as a variety of handheld devices. Flash chips generally have life spans from 100K to 300K write cycles.

gated

A modular software program consisting of core services, a routing database, and protocol modules supporting multiple routing protocols such as RIP (V1 and V2) , OSPF, IS-IS, BGP, EGP.

HotSwap™

A set of functions that allows you to remove and reinstall a board in a system while the power is on. HotSwap is a trademark of ZNYX Networks.

Network File System (NFS)

NFS is a file sharing protocol in a UNIX network. A file sharing protocol is a high-level network protocol that provides the structure and language for file requests between clients

and servers. It provides the commands for opening, reading, writing and closing files across the network and may also provide access to the directory services. Sometimes called a "client/server protocol," it functions at the application layer (layer 7 of the OSI model).

In order for a client to have access to multiple servers running different operating systems, either the client supports the file sharing protocol of each operating system or the server supports the file sharing protocol of each client.

OpenArchitect

The ZNYX Networks software platform, based on Linux, that enables system administrators and programmers to configure and customize Ethernet switching directly through a switch. OpenArchitect is a trademark of the ZNYX Networks trademark.

Open Shortest Path First Protocol (OSPF)

A routing protocol that determines the best path for routing IP traffic over a TCP/IP network. OSPF is an interior gateway protocol (IGP) which is designed to work within an autonomous system.

Peripheral Component Interconnect (PCI) bus

A peripheral bus commonly used in PCs, Macs, and workstations. PCI provides a high-speed data path between the CPU and peripheral devices (video, disk, network, etc.).

power cycle

The process of rebooting the switch by turning the computer system on and off.

Rear Transition Board (RTB)

The rear panel egress board that is used in conjunction with the Compact PCI adapters. By convention, ZNYX refers to rear panel boards for non "P" models as RTM's. "P" models use rear transition board (RTB)

Rear Transition Module (RTM)

The rear panel egress board that is used in conjunction with the Compact PCI adapters. By convention, ZNYX refers to rear panel boards for non "P" models as RTM's. "P" models use rear transition board (RTB)

reset button

The button on the OpenArchitect switch which causes the switch to reboot.

Routing Information Protocol (RIP)

A simple routing protocol that is part of the TCP/IP protocol suite. It determines a route based on the smallest hop count between source and destination. RIP is a distance vector protocol that routinely broadcasts routing information to its neighboring routers.

routing protocol

A formula used by routers to determine the appropriate path onto which data should be forwarded. Routing protocol also specifies how routers report changes and share information with the other routers in the network that they can reach. A routing protocol allows the network to dynamically adjust to changing conditions, otherwise all routing decisions have to be predetermined and remain static. RIP and OSPF are examples of routing protocols.

tagged

Data packets which are tagged with a specific VLAN identifier indicating the VLAN from which the data came.

telnet

A terminal emulation protocol that enables a user to log onto a remote device and run a program.

untagged

Data packets which do not belong to a specific VLAN and do not carry an identifier tag.

vi editor

(visual interface) A full-screen text editor that can be run from a terminal or the system console.

VLAN

(Virtual Local Area Network) A VLAN is a logical subgroup within a local area network that is created with software rather than physically reconfiguring network hubs and cables.

zhp

(ZNYX host port) A virtual interface which represents a VLAN that includes specific OpenArchitect switch ports.

zre

(ZNYX raw Ethernet) A virtual interface that represents a port on the OpenArchitect switch.

Index

adduser....63
application flash....75
application flash 1....75
application flash 2....75
application flash ROM....164
Apt-Get....80, 163
boot loader....164
boot process....75
boot ROM....75, 164
  image....164
Booting
  Duplicate Flash Image....79
  -i option....78
  recovery....78
bootloader....75
brctl....18
brctl(8)....143
cable requirements....8
carrier class....164
Central Authority....61
Class of Service....28
Combining Queueing Disciplines....45
Common Open Policy Service....46
  COPS-PR....47
  COPS-RSVP....47
  Policy Decision Point....46
  Policy Enforcement Points....46
console cable
  Booting....77
console port, connecting to....8
COPS..... See Common Open Policy Service
daemon....164
Default Route....63
dhclient....64, 165
DHCP....61, 165
  Client....64
  Server....64
dhcpd....64, 165
Diffserv....47
DNS....164
Fast Forward Bridge....19
Fast Forward Port....19
FIFO Queues (pfifo and bfifo disciplines) 39
file system organization....77
flash....79
flash memory....165
ftp165
FTPD Server....66
gated....25, 165
HotSwap....10, 165
HTB Queuing Discipline....42
ifconfig....21
IPMI....153
iptables....33–38
Layer 2 Switch....16
Layer 3 Switch....21
  Using Multiple VLANs....24
LED Control....72
LED Reference....8, 10
libraries....5
link aggregation....See trunking
Link Event Monitoring....72
Name Service Resolution....64
Network File System....65, 165
Network Time Protocol....64
NFS....See Network File System
NTP....See Network Time Protocol
ntpdate....65
OpenArchitect....166
OSPF....166
PCI bus....166
pepd....47, 48
PIB....See Policy Information Base
Policy Information Base....47
Port Mirroring....71
Port Path Cost....18
power cycle....166

Priority and Weighted Round Robin Scheduling(prio discipline with wrr) 40
RAINlink Management API.....5
RAM disk 75
Random Early Detection (red discipline) ..40
read-write directory 77
Rear Transition Board 166
Rear Transition Module.....166
reset 166
RFC 2478 47
RFC 3084 47
RFC 3159 47
RFC 3289 47
RIP.....166
RMAPI, definition.....5
Root Password.....63
route.....63
routing protocol 5, 167
RSVP.....47
S60SP_startup.....61
Saving Changes 77
Scripts
rcS.....64
S50Layer2.....16
s50Layer3.....13
S50layer3 21
S50Layer3.....13
S50multivlan.....13, 24
S55gatedOSPF 13
S55gatedRip1.....13, 25
S55gatedRip2.....13
Scripts, example 12
snmp 67
applications 71
interface details 69
MIBS.....67
traps.....68
SNMP
ZNYX Private MIBs 155
Software Development Kit.....6
Spanning Tree Protocol.....18
Switch Console..... 12
System Failure Recovery..... 77
System Hangs During Boot 78
tagged packets..... 167
tc 38
technical support..... i
telnet 167
The U32 Filter 43
thttpd..... 72
Translation Rules..... 43
trunking
configuring with zconfig..... 93
untagged packets..... 167
Updating the Switch 77
Upgrading
Switch Driver 81
Upgrading or Adding Files 80
Users
Adding Additional..... 63
vi editor..... 167
VLAN 13, 167
vrrpconfig 83
vrrpd 85
Web management 72
zal 88
zbootcfg 78, 91
zconfig 21, 82, 93
zcos 30, 100
zfilterd..... 33–38, 103
zflash..... 79, 104
zgmrpd..... 105
zgr 107
zgvrpd 110
zhgtable..... 88
zhostmac 88
zhp 167
zipf..... 88
zipmc 88
zl2 107
zl2d 82, 112
zl2mc 107

zl2mcvlanbc....88
zl3d....21, 24, 25, 82, 114
zl3host....107
zl3net....107
zl4table....88
zlc....72, 116
zlinkagg....88
zlmd....72, 118
zlogrotate....120
zlxlib....5
zmac....88
zmirror....88, 121
zmirror_4800....71
zmnt....124
zmonitor....75
ZNYX-H.MIB....155
zpttable....88
zqosd....38, 126
zrc....128
zre....167
zreg....129
zrld....131
zrld_trusted_hosts....131
zrule....88
zspconfig....132
zstats....139
zsync....140
ztlib....5
ztmd....142
zvlan....107
zvmc....88
zvrp....88
zvrpout....88
ZX5000.MIB....160