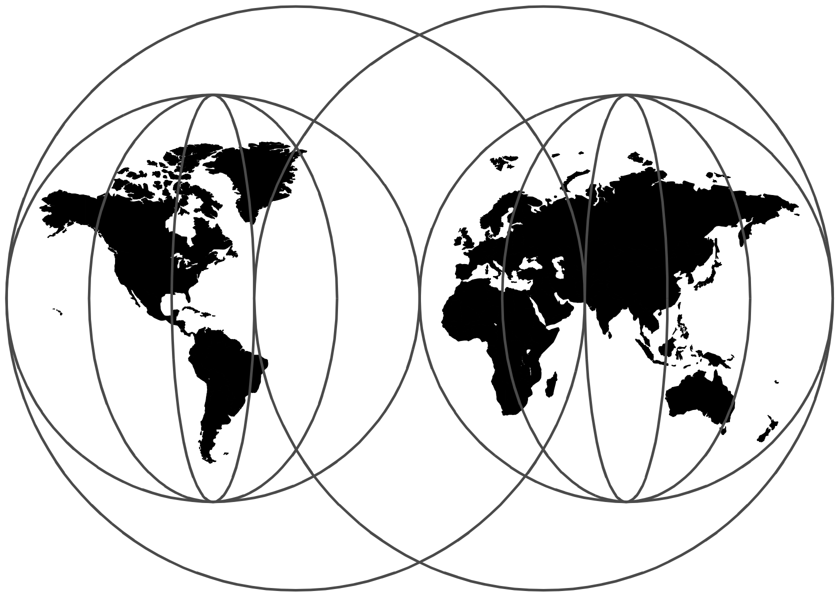
Understanding and Using the SP Switch

Abbas Farazdel, Gonzalo R. Archondo-Callao, Eva Hocks, Takaaki Sakachi, Federico Vagnini



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5161-00

SG24-5161-00

International Technical Support Organization

Understanding and Using the SP Switch

April 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 257.

First Edition (April 1999)

This edition applies to IBM Parallel System Support Programs for AIX (PSSP) Version 3, Release 1 for use with AIX 4.3.2 and the SP Switch.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

Although the IBM RS/6000 Scalable Powerparallel (SP) supercomputer is now more than five years old, and enjoying its second generation of switch technology, the SP Switch is still quite a mystery to many. It embodies leading-edge technology in an area where few people were knowledgeable just a decade ago. This redbook seeks to remove much of the mystery. It is written for system administrators, users and servicers of the SP. The book is organized in five parts and includes the following chapters:

1. Overview of the SP Switch
2. The Switch Hardware
3. Communication Network Hardware
4. Communicating with the SP Switch
5. Planning for the SP Switch
6. Installation of the SP Switch
7. Initialization of the SP Switch
8. Managing the SP Switch
9. SP Switch Problem Determination Tools
10. SP Switch Problem Diagnosis
11. SP Switch Specific Application and Server Tuning

In September, 1993, IBM delivered the first generation of its Scalable Powerparallel (SP) supercomputer for scalable parallel computing. This was the SP1:

> a system comprised of special AIX workstations (nodes) grouped together physically in building blocks (frames) of 16 or less; the nodes may be linked via various communication networks to allow for collaborative problem solving and symmetric maintenance.

The current day SP uses a shared-nothing model, meaning each node has its own memory, disk, CPU, and so on. In order for a group of nodes to work together on a problem, they must share data and status as needed through inter-node messages. These messages are sent by the components of a running application and are delivered through "packets" sent over the communication network chosen for the application.

To enhance the performance of these communication tasks, IBM provides a special hardware medium: the SP Switch. This switch supports a high-speed communication network which provides applications with a low-latency, high-bandwidth path for message passing. Since September, 1996, the SP has been available with its second generation of switch technology, the SP Switch. The switch is considered the heart of the SP.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Abbas Farazdel is an SP technical consultant and a senior project manager at the International Technical Support Organization, Poughkeepsie Center. Before joining ITSO in 1998, Abbas worked in the Global Business Intelligence Solutions (GBIS) group at IBM Dallas as an implementation manager for data mining solutions and in the Scientific and Technical Systems & Solutions (STSS) group at the IBM Thomas J. Watson Research Center as a high-performance computing specialist. Abbas holds an M.S. degree in computational physics and a Ph.D. in computational chemistry from the University of Massachusetts.

Gonzalo R. Archondo-Callao is Manager of the High-Performance Computing Group at the Computer Center of the Federal University of Rio de Janeiro (NCE-UFRJ) in Brazil. He joined NCE-UFRJ in 1977 where he has been working mainly as a systems administrator. He has worked with UNIX for over 12 years and has three years experience with the RS/6000 SP and AIX. He is also a visiting professor at the computer science department of the Federal University of Rio de Janeiro. He holds a M.S. degree in computer science from the University of California, Los Angeles. His areas of interest include operating systems, computer architecture, and networking.

Eva Hocks is a systems administrator for Power Parallel Systems at the Computer Center of the Research Center for High Energy Physics, GSI, in Germany. She joined GSI in 1992. Her areas of expertise include System design and installation as well as interoperability of large systems and workstations operating on VM, MVS and several flavors of UNIX in a high availability environment. Before she joined GSI, she worked at the Technical University of Darmstadt as systems administrator for IBM and Siemens Super Computers. Since 1989, she has been project manager at the SHARE Technology Conference. She became a member of the Internet Society in 1991, working on Internet security issues. Eva holds a B.S. degree in mathematics and computer science from the Technical University of Aachen.

Takaaki Sakachi is an I/T Specialist at the Systems Engineering Division in IBM Japan. He joined IBM Japan in 1989, has been working for technical support in commercial area computing, and has three years experience in the SP field. He holds a degree in chemical engineering from Tokyo Metropolitan University. His areas of expertise include Oracle for SP and VSD.

Federico Vagnini is an I/T specialist at Midrange Technical Support in IBM Italy. He joined IBM in 1995 and has three years experience in the AIX field. He holds a degree in electronic engineering from Politecnico di Milano. His areas of expertise include TCP/IP networking and network security.

We give special thanks to Skip Lundin from the IBM PPS Lab Poughkeepsie for his efforts in reviewing the entire redbook and providing useful information and suggestions.

Thanks also to the following people for their contributions to this project:

Bernard King-Smith

Bob Simon

Nick Rash

Kevin Reilly

Bill Tuel

Jonathon Kaufman

Peter Chenevert

A. Z. Muszynski

Fu-Chung Chang

Aruna Ramanan
IBM PPS Lab Poughkeepsie

Paul Crumley
IBM Watson Research

Terry Barthel, Al Schwab
ITSO Editors, Poughkeepsie

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 279 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users`http://www.redbooks.ibm.com`
  For IBM Intranet users`http://w3.itso.ibm.com`
- Send us a note at the following address:

  `redbook@us.ibm.com`

Part 1. Introduction to the SP Switch

Chapter 1. Overview of the SP Switch

There are four basic physical components of an SP (See Figure 1 on page 4):

frame	A containment unit consisting of a rack to hold computers, together with supporting hardware, including power supplies, cooling equipment and communication media such as the system Ethernet.
nodes	AIX RS/6000 workstations packaged to fit in the SP frame; a node has no display head or keyboard, and so user human interaction must be done remotely.
switch	The medium that allows high-speed communication between nodes.
CWS	The Control Workstation (CWS) is a stand-alone AIX workstation, with display and keyboard, possessing the hardware required to monitor and control the frame(s) and nodes of the system.

SP nodes are available in three form factors: thin, wide and high. The SP is generally available with from 2 to 128 nodes, which are packaged in from 1 to 9 logical frames. The number of physical frames can be more, depending on the types of nodes.

Each frame may contain an SP Switch board. The nodes of the frames are connected to their respective switch board via a special SP Switch adapter and corresponding “switch cables”, and the switch boards of the system are connected via the same type of cables. Thus, a high-speed communication network is formed which allows the nodes to communicate with each other to share data and status. The primary purpose of this high-speed network is the support of solving problems in parallel.

Communication over the switch is supported by IBM software which is shipped with the SP. The switch hardware together with this software is called the Communication Subsystem (CSS). On each node attached to the switch, the CSS software is continuously available to provide the following:

1. a communication path through the switch to other nodes
2. monitoring of the switch hardware
3. control of the switch hardware for startup, and in case of error, execution of any appropriate recovery action

This software is responsible for sending and receiving packets to and from other nodes, on behalf of applications. It also sends packets to switch hardware components as part of its monitoring and controlling functions.

If a component of the switch network (switch board, adapter, cable, node, or software) is not functioning correctly, the CSS software is responsible for recognizing this and reporting this “fault” via the AIX error log. The software will also take recovery action as deemed most appropriate for the health of the system, which may mean removing the offending component from the switch network.

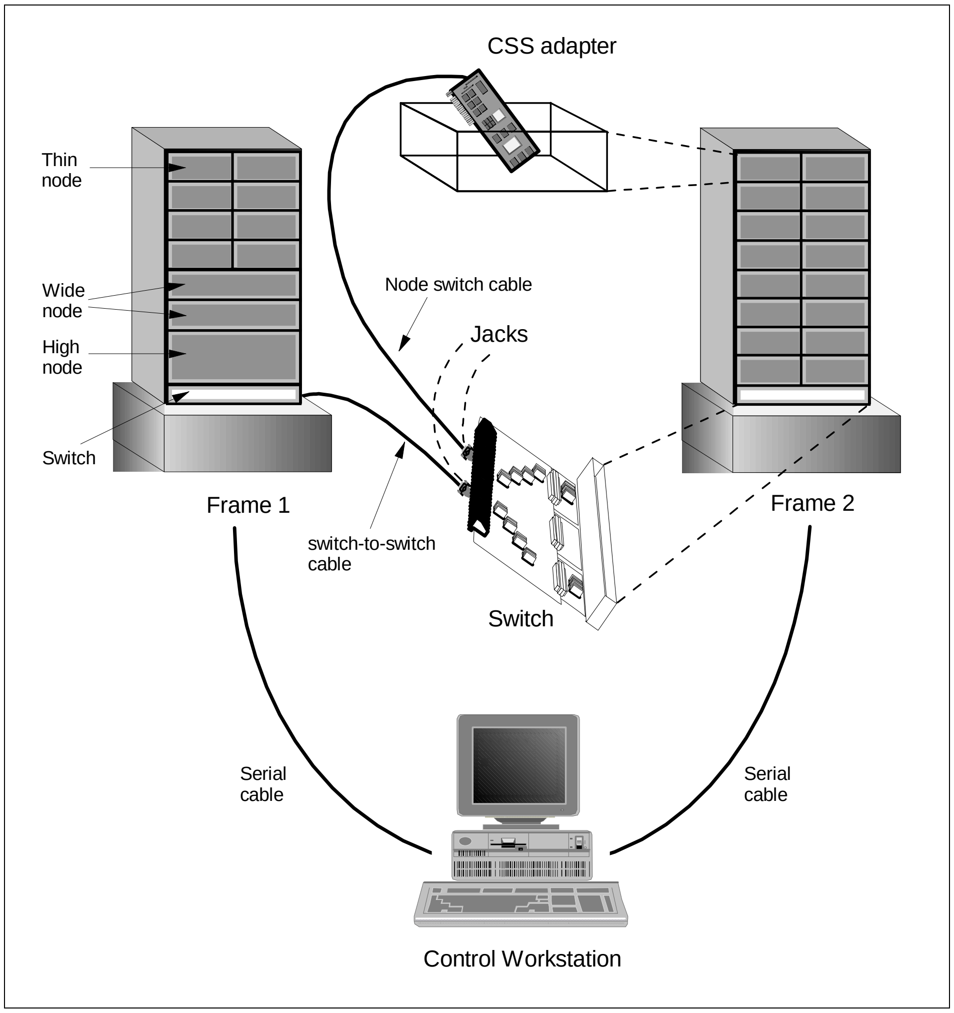


Figure 1. A Two-Frame SP System

Chapter 2. The Switch Hardware

The RS/6000 SP system exploits SP Switch multistage switching technology in order to create a high performance network that connects all computing elements. Each SP system is composed of 2 to 512 RS/6000 machines, hereafter called *nodes*.

In order to support such large quantities of nodes attached to the same SP Switch and also be able to configure systems with lesser numbers (and lower cost), the switch fabric is not monolithic, but is composed of modules. All modules are made in the same way and can be used either alone to create small switching networks, or combined to get bigger networks. It is always possible to add modules to an existing configuration to enlarge the network.

2.1 The SP Switch Module

The basic module of the SP Switch is the *switch board*. It is shown in Figure 2. It is a set of hardware components that create the network and give the system administrator monitoring access to the board.

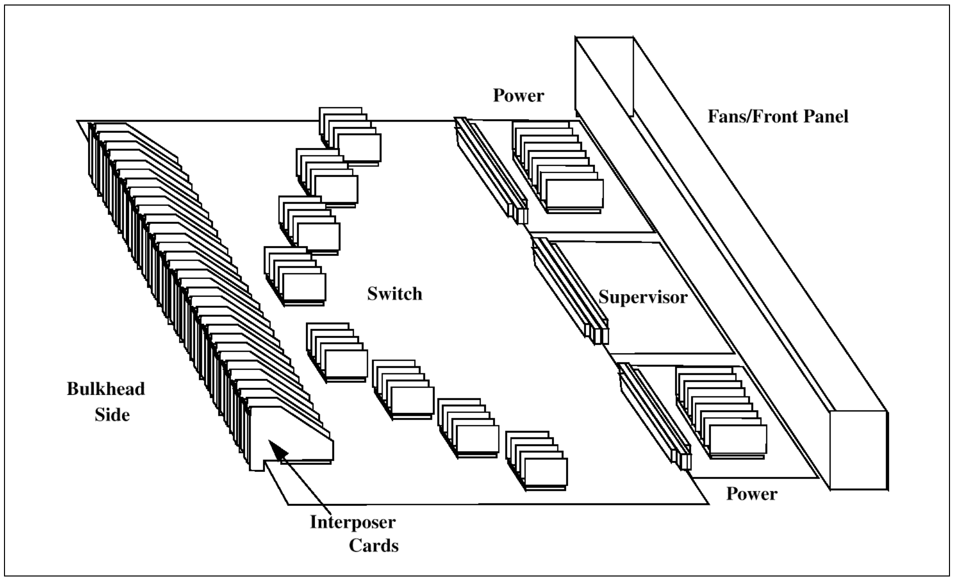


Figure 2. Switch Board Physical Layout

There is a *supervisor card* that constantly monitors the hardware environment and receives configuration messages from the system administrator.

The board is powered by two power supplies that share the load. The power supplies take 48 volts as input and produce 3.3 volts to the switch board. The switch supervisor monitors the availability of the 48 volts, the current usage of each of the power supplies, and the output 3.3 volts. The second power supply is redundant.

The switch board is connected with the rest of the switch complex by *interposer cards*. They provide access to data channels and the supervisor card, and receive the 48-volt power. For each interposer there is a connection, or jack, on the bulkhead side where the appropriate cable can be attached. Bulkhead jacks are numbered: 1 for 48-volt power, 2 for switch supervisor card connection, 3-to-35 for data channels.

There are five cooling fans in the switch assembly. The rotation of these fans is monitored by the switch supervisor. There is an N+1 redundancy in the switch assembly so that a fan may be faulty without affecting the system.

From the network point of view, the switch board is a box that has 32 links with the outside and there are several paths between any pair of them. When connected to nodes, the switch boards have 16 links to nodes and the remaining 16 to other switch boards. To create bigger networks, several switch boards may be interconnected with other switch boards.

The logical structure of a switch board’s network can be seen in Figure 3 on page 7. On the left side are the connections to nodes, while on the right side are connections to other possible switch boards. If the desired configuration does not exceed 16 nodes, the connections on the right side are not used.

All internal and external switch board point-to-point links are bidirectional full duplex. They comprise two channels that can carry data in opposite directions simultaneously, each channel capable of carrying 150 MB of data per second.

Inside the SP Switch there are eight switching devices called *switch chips*. They are the heart of the switch board and are responsible for routing the data from one link to another. They are non-blocking devices, so any two data paths can traverse them in parallel if they do not require access to the same link between switch chips.

Each switch chip has eight *ports* from which it can send and receive data simultaneously. Data arriving at one port can be routed to and transmitted from any other port.

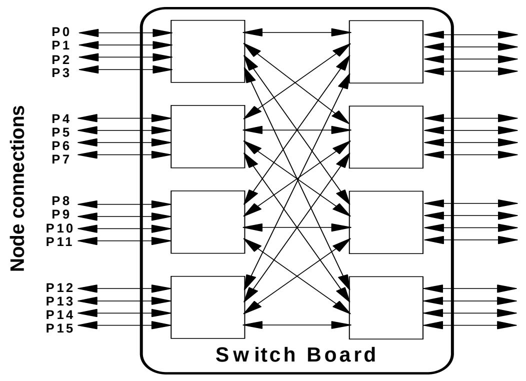


Figure 3. Switch Board Logical Layout

Depending on switch chip configuration, different paths are available to connect the same pair of external devices, making the network highly available. In the case of link failure or switch chip anomalies, there is an internal protocol that handles errors and tries to recover from them. If no correction is possible, a link or a switch chip is marked as unusable and different data paths are used to connect pairs of nodes.

Multiple data flows are supported concurrently, with minimal sharing of communication links. In Figure 4, the two highlighted paths are managed in parallel even though they share a switch chip; the internal structure of the switch chip is able to route data among all its ports in parallel.

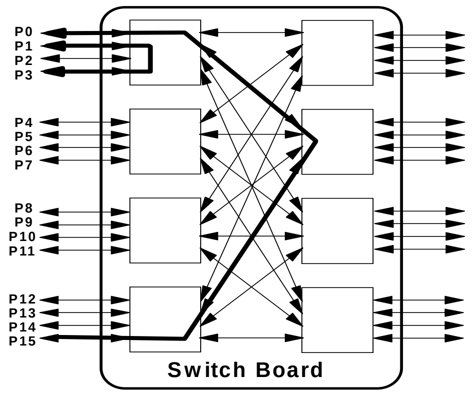


Figure 4. Multiple Parallel Routes

If the communicating nodes are connected to the same switch chip, the data they send does not traverse the SP Switch fabric beyond the chip. If they do not share the chip, they are connected by a set of links and data flows through additional chips to get to the destination. In the latter case a longer path is required, especially if the nodes are not connected to the same switch board. Transmission delays are a little longer but, as we see in 3.7, “Performance Data” on page 47 the differences between shorter and longer paths are not very significant and performance is very good in both cases.

Path redundancy is possible due to the internal network structure of the switch board. The switch chips are divided into two fully interconnected columns, or *stages*, of four switching elements. The first stage is connected to nodes; the second stage is used to support an extended network, which can grow beyond the single switch board. Each chip in one stage is connected to all the chips of the second stage so that two nodes on different switch chips in the first stage can be connected using each of the chips of the second stage.

A representative multiple paths between two nodes is shown in Figure 5. Consider the nodes connected to switch ports P0 and P15, as an example. These two nodes have four separate communication paths between them. These paths are the *shortest paths* available. Other paths are also possible using a greater number of switching devices, but are not normally used except in the unusual case of multiple switch hardware failures.

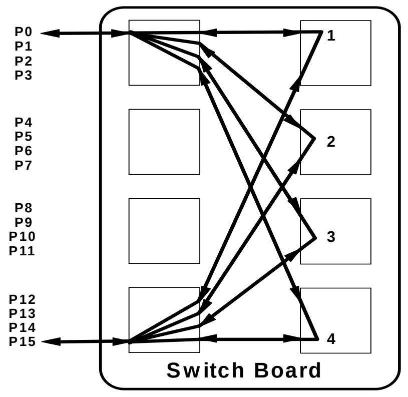


Figure 5. Multiple Paths between Two Nodes

The standard SP Switch network configurations provide at least four shortest paths between nodes that make use of different sets of switch chips, even when multiple switch boards are used. This feature is required to provide high availability and performance across the network.

2.2 Switch Board Topologies

A single switch board is able to interconnect up to 16 nodes. In order to support a larger number, multiple switch boards are used and interconnected in precise fashion. Different topologies are available, and all are structured in order to maintain the basic switch property: at least four independent shortest paths are available between any two nodes connected to different switch chips.

2.2.1 From 2 to 80 Nodes

Up to five switch boards may be interconnected directly in a *star* topology. Data passes through, at most, two switch boards before arriving at the destination. This makes it possible to connect up to 80 machines (16 nodes for each of five boards).

Depending on the number of switch boards involved, different cabling is specified and required to interconnect boards. Figure 6 and Figure 7 on page 10 are two examples of systems with two and three frames.

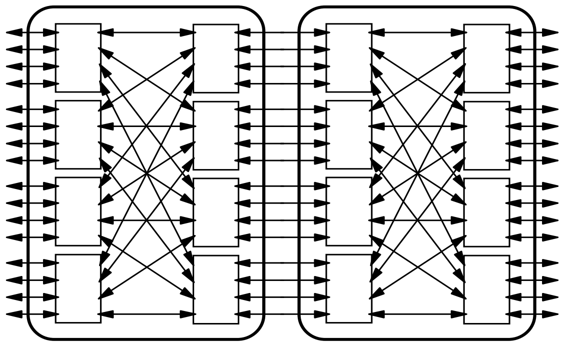


Figure 6. Two Frames Cabling

All switch chip ports are used to create the network. When only two frames are present, there are 16 connections between two frames, while with three frames, only eight connections are possible. In any case, the four shortest paths are always present.

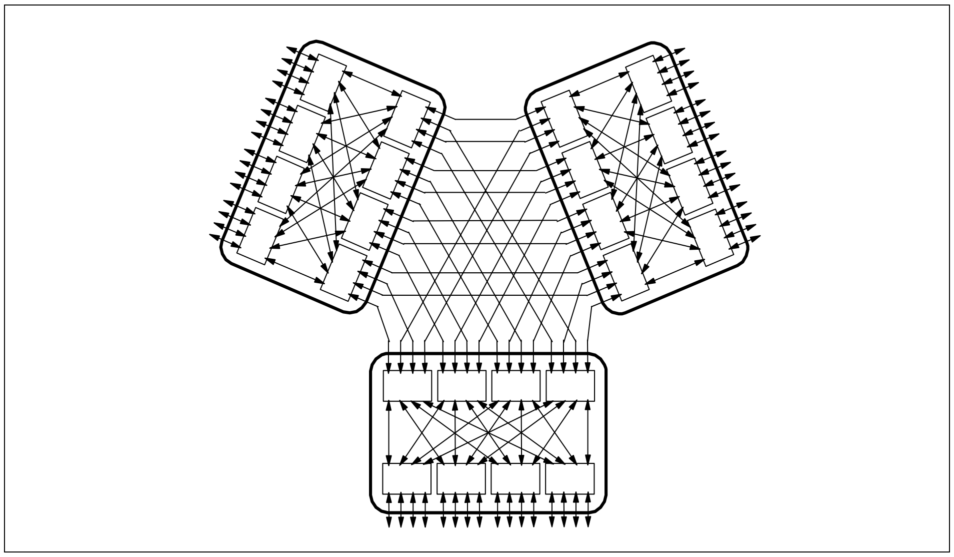


Figure 7. Three Frames Cabling

If the system grows and a new frame is added, all the board-to-board cabling must be redone. This means that the network has to be stopped during the re-configuration. A way to avoid this downtime is to start using the intermediate switch boards we discuss in 2.2.2, even for a system with less than 80 nodes. This is a more expensive solution but the investment may be worthwhile if the system is expected to grow.

2.2.2 From 81 to 256 Nodes

When more than 80 nodes are required, at least a sixth switch board is needed. It is not possible to connect more than five switch boards in a star topology and still have at least four independent shortest paths. An intermediate switch level is required.

Additional switch boards are then added to the network topology, and their task is to provide sufficient links between other switch boards. Since these boards do not connect to nodes, they are called *intermediate switch boards* (ISBs), while the boards that are connected to nodes are called *node switch boards* (NSBs).

ISBs are physically placed in a frame, called a *switch-only frame*, that provides power, management, and connectivity to 4 or 8 intermediate switch boards. In any configuration, the number of required ISBs is always a multiple of 4.

When the number of nodes is from 81 to 128, only 4 ISBs are required, while all eight ISBs in a switch frame are needed to reach the 256-node configuration. An example of a 128-node topology is shown in Figure 8.

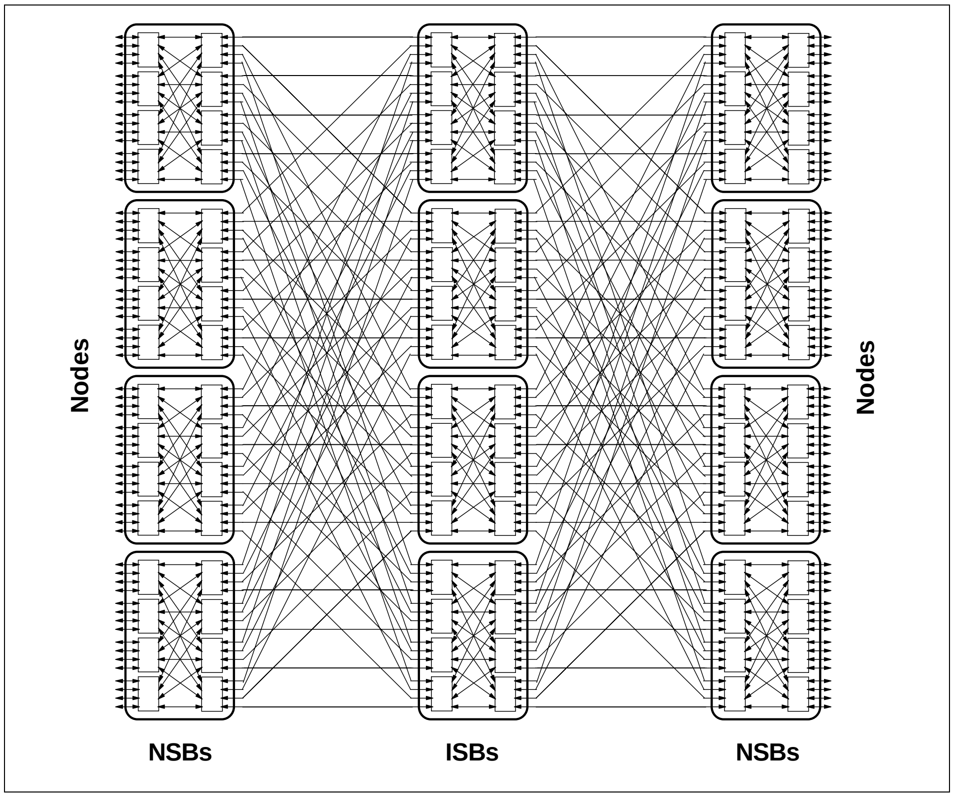


Figure 8. The 128 Nodes SP Switch Topology

2.2.3 From 257 to 512 nodes

In order to reach the 512-node configuration, 16 ISBs are required, using two switch frames. Each NSB connects each port to a different ISB port, getting to the maximum of 32 NSBs: 16 ports times 32 NSB equals 512 nodes.

2.3 The SP Switch-8 Switch Board

Customers who do not need more than eight nodes in an SP system because they have no plan to grow beyond that size can choose a low-cost switch board that has the same features as the switch board we already saw, but which only allows eight connections to nodes and cannot connect to other boards.

The SP Switch-8 switch board is a multistage switched network that has eight bidirectional links that connect *only* to nodes and whose *logical* topology is depicted in Figure 9.

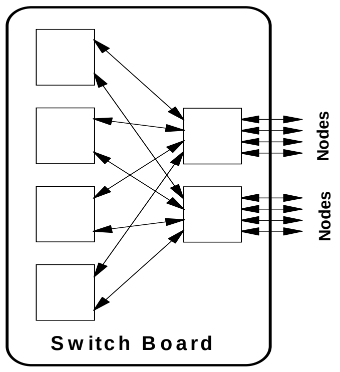


Figure 9. Logical View of SP Switch-8 Switch Board

This switch board is used only in small environments. It has the same components as the full switch frame and all the descriptions we give in the following chapters also apply to this board. However, all future references to the term *switch board* imply the full-featured board comprising eight switch chips.

Chapter 3. Communication Network Hardware

In this chapter we describe in more detail what the components of the SP Switch are and how they cooperate to create the high-performance interconnection network that is the heart of the SP system.

Reliability is of great importance. The communication subsystem detects, locates and isolates all failures. Communication links are protected by error-detecting codes; switch chips are designed using error detection and fault isolation methods. Links and switch chips can be individually disabled to isolate faults, making some communication flow through alternative paths.

3.1 Packet Data Flow

Data is inserted in the SP Switch by a node through the node’s *switch adapter*, which provides for switch network communication. The information that flows into the network is made up of single independent *packets* of data that contain the route information needed to traverse the SP Switch. The route information is used by switch chips in order to decide to which of its ports to route the received packet.

Each packet is destined either to a node or to a single switch chip. In the first case it is typically a *data packet* that has been created by some node application, while in the second case it is called a *service packet* and is used to configure the features of the switch chip. A service packet may also be created by a switch chip to notify a node of some event that is meaningful in network administration, or by a node for node-to-node administrative communications.

Both data and service packets flow on the same network. This choice keeps architecture simple and gives to both packet types similar path redundancy. Service packets do not greatly affect the network performance since they are used basically whenever a switch network configuration is executing or in case of failure recovery. The protocol used is designed to keep service communication low.

Packets are of variable length (application dependent) and are identified by special *beginning of packet (BOP)* and *end of packet (EOP)* control characters. The routing part is also of variable length and contains, for each switch chip the packet reaches, the port number to exit through. As shown in Figure 10 on page 14, the first number instructs the first chip to which of its ports the packet is to be sent, the second number instructs the second chip, and so forth.

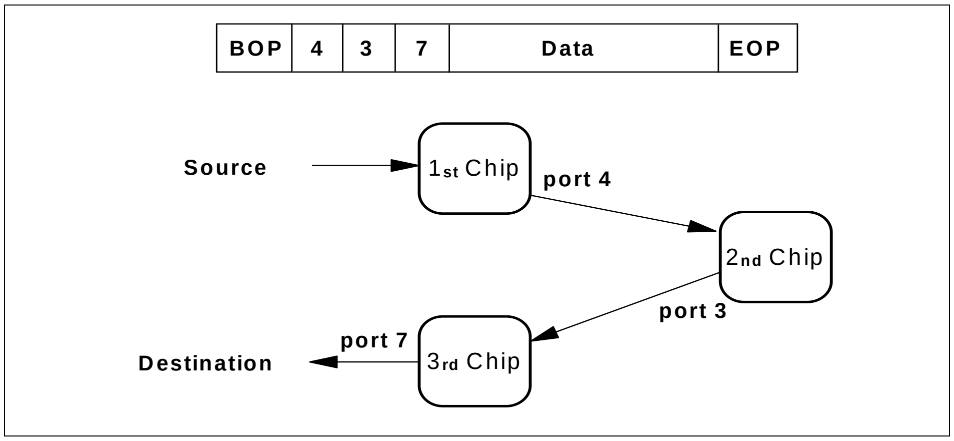


Figure 10. Packet Routing

Each switch chip starts reading the incoming packet starting from the BOP character and then begins scanning the routing part to detect to which of its output ports the packet has to be forwarded. As soon as it detects the port, the bytes are passed through it to the following switch chip or the destination node adapter.

This is *not* a store-and-forward protocol, so incoming bytes of data are sent to the next network stage as soon as possible. The data flow could only be delayed by another packet using the same output port.

Since packet sizes may be very long and minimal buffering is used during transmission time to reduce latency, the data may be strung across the network route.

In order to improve link reliability, all bytes transmitted across the link (packets and control characters) are protected by a time-based, 2-byte *Error Detection Code* (EDC) that gives a checksum value of the transmitted data. Time boundaries, start to end, are called *EDC Frames*, expressed as a programmable number of cycles ranging in power of two from 32 to 256 bytes. EDC errors are used to monitor link data quality and to identify possible link degradation.

Since the EDC Frame is time-based instead of packet-based, the ECD Frame may span a partial packet, a full packet, multiple packets, or no packets at all. It is not a separate entity from a message packet, but they overlap. The EDC Frame has no effect on the message packet except to steal a fraction of the bandwidth to insert the EDC bytes into the data flow.

When traffic on the switch grows, contention for resources may occur inside a single switch chip. Different packets may traverse the same chip at the same time, but if they have to use the same output port, a serialization must occur. The packet that first requests the usage of the port passes through it, while the others are queued on the switch chip.

An example of data flow on a switch board is shown in Figure 11. There are three data flows in the same direction that make use of four switch chips. For each flow, the data bytes are continuously flowing from one switch chip to another. In the same instant, all the following may occur:

- The beginning of the packet may be on one switch chip
- Other packet bytes may be on the same chip
- Still other bytes may be travelling on the link between two chips
- The end of the packet may not yet have even been inserted into the switch network

Two data flows may traverse the same chip in parallel, but if they require use of the same port, one of them is stopped. In the figure, the beginning of the packet of Flow C is buffered in Chip 1, while other bytes are still travelling to Chip 1. When the packet in Flow B finishes, Flow C will proceed.

.

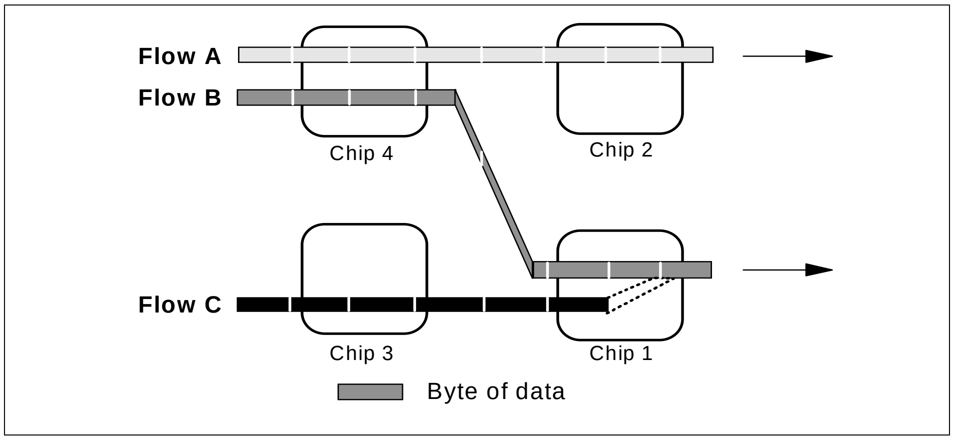


Figure 11. Multiple Data Flows on the SP Switch

A flow control protocol is introduced for each point-to-point link in order to handle events that may reduce the capability of input ports to accept the data sent by output queues, like data buffering on the switch chip, computational overhead on the chip itself in managing packets, or configuration requests.

Each unidirectional part of the point-to-point link between two switch elements (switch chip or switch adapter) is made by 11 signals as described in Figure 12 on page 16.

Ten signals are driven by the transmitter:

- Data (8 bits)
- TAG (1 bit)
- Data Valid signal (1 bit)

One signal is driven by the receiver:

- Token signal (1 bit)

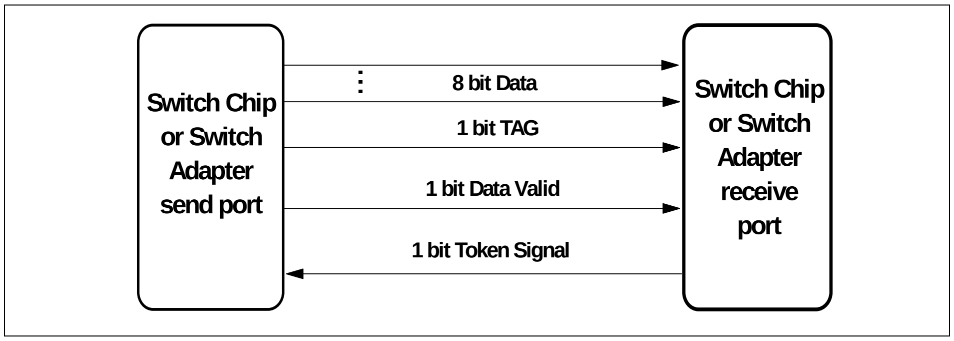

Switch Chip or Switch Adapter send port
8 bit Data
1 bit TAG
1 bit Data Valid
1 bit Token Signal
Switch Chip or Switch Adapter receive port

Figure 12. Switch Channel Signals

The signals can have a different meaning depending on the operational mode of the link, as defined by the sending port. The TAG bit identifies the 8-bit data as a control character or as a data character.

The token signal is used for flow control. When sending and receiving ports are initialized, the sending port gets its token counter set to the size of the buffer present in the receive port. Each time the sending port transmits two bytes of data, it decreases its token counter by one. When the receiving port frees its internal buffer of the same fixed amount of data, it sends a token signal to the sending port. Upon the reception of a token signal, the sending port increases its token counter.

The number of tokens on the sending port reflects the space available on the receiving port queue except for possible data and tokens actually traversing the link. When the counter reaches zero, the sending port does not send any more data and starts buffering the bytes directed to that sending port.

In a situation of high contention, the packet may be partially buffered in different switch chips, waiting for the locking congestion to be removed. As soon as a switch chip is able to receive more data on a link, tokens are transmitted backward, making the data flow continue.

The token protocol is designed to detect and correct token signal errors that may create spurious tokens on the sender or avoid reception of tokens. Sender and receiver periodically check their token counter values and synchronize themselves.

3.2 System and Board Clocking

The whole system is synchronous. All network devices are driven by the same clock signal used as a reference when sending and receiving signals through physical wires. All switch chips and adapters have an internal counter called time-of-day (TOD) that, once initialized, is increased by the system clock to provide a common value on all devices.

Data channels are sampled to detect data bits using this common clock; however, because of the phase difference inherent at each switch chip due to irregularities in the clock distribution network, the data leaving one chip must be synchronized to the clock seen by the receiving chip. Such link tuning is performed for each port of the switch chip using a Self-Timed Interface (STI) macro embedded in the receive section of each switch chip's port. The STI both initially tunes the links at system initialization and dynamically maintains the proper sampling of the received data as the operating environment changes. See 3.3, "STI Timing and Logic Synchronization Process" on page 20 for more details.

Different clock sources may be used. A single switch board can receive its clock from any of 35 sources:

- There are two local 18.75 MHz crystal oscillators on a switch board
- There is a differential clock signal received on each of the 32 switch board's data cables
- There is a connector for receiving a differential clock signal from a source external to the SP system. The switch board can use such signal, but this feature is currently not supported.

Each switch board assembly has a switch supervisor card which is controlled from the control workstation. The switch supervisor cards are used to define the clock signal distribution. Each supervisor sends signals to all its board's switch chips to define how the clock signal is generated and used by all the components of the board. When the clock is configured by PSSP software, the switch supervisor cards are told to set it up.

From the selected source the clock signal is fed to one of the four Phase Locked Loops (PLL) present on the board for clock pulse shaping and to reduce clock jitter. Clock shaping is very critical for data receiving in each switch chip. The clock signal provided by the PLL is then used by all the switch chips on the board (Figure 12 on page 16).

Different topologies for the *clock distribution network* are possible depending on the number of switch boards. The simplest configuration is an SP Switch with only one switch board: one of its internal oscillators is selected as the clock source.

The PLLs are strictly tied to switch chips, as can be seen in Figure 13. Only switch chips SW2, SW3, SW4 and SW5 have PLLs and can receive the clock signal from the oscillators.

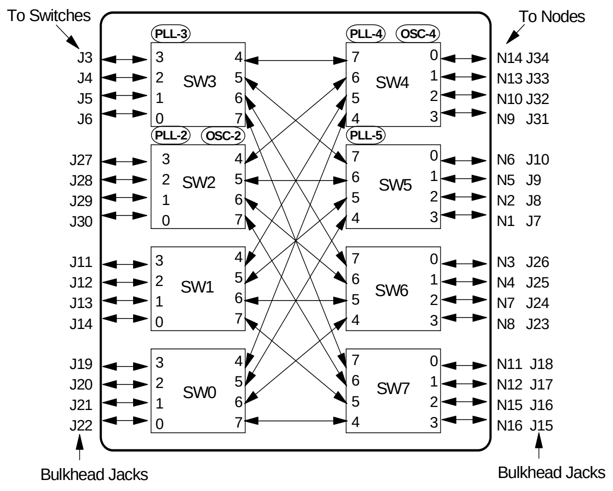


Figure 13. Clock Components Position

On a single-board SP Switch, one of the switch chips with PLL is selected to be the *master chip* for the board. It receives the signal from the selected oscillator, uses its PLL and then distributes the clock to all other switch chips using dedicated clock wires. The standard PSSP 3.1 configuration always chooses SW2 and oscillator number 2, which is physically near SW2.

When more than one switch board is present in the system, one is selected to be the *master board* that uses its internal oscillator and then distributes the clock to all the *slave boards*. On each slave board, one switch chip acts as a master chip receiving the clock from the master board and distributes it to the remaining seven switch chips.

For each slave board, one switch chip is selected to receive the source clock at one port. If it has a PLL, it shapes the signal and re-drives it to all the other switch chips, acting as a master chip for the board, otherwise it does not use the signal but forwards it to the master.

All the data cables that connect a switch board to another board or to a node also carry a clock signal. Each switch chip always sends the clock it uses along with the data. Data and clock are carried by different wires. Therefore, each switch chip on a slave board receives four different clock signals from four sources.

ISBs may receive a clock signal from any port of any switch chip, while NSBs may only receive it from SW0, SW1, SW3 or SW4 since the other switch chips are dedicated to node connections.

Any switch chip's port can be selected to receive the clock signal. However, if you look at clock distribution files, PSSP software always uses jacks J3, J4 or J5. This limitation is for compatibility with the High Performance Switch that has limited clock distribution capabilities.

Clock distribution files only mention in the description part the ports of a switch chip provided with a PLL, but a system administrator can decide to use any other port, and provide a corresponding configuration file.

Both master and slave switch boards can provide a clock signal to other slave boards, creating a clock distribution tree, as you can see in Figure 14.

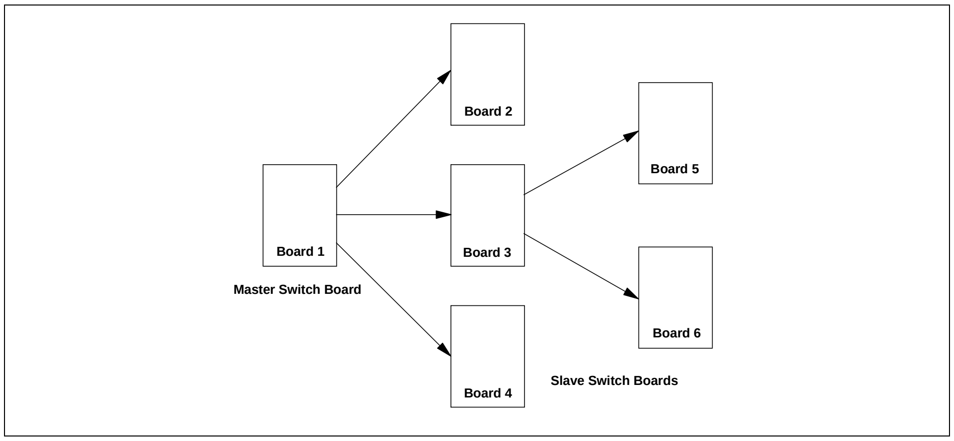


Figure 14. Clock Distribution Tree

Important

The clock signal is of vital importance to the SP Switch.

If a switch board does not receive the clock signal, none of its internal and external data links will be functional and all the switch boards that depend on it for the clock will also fail in the same way.

For example, if the cable from Board 1 to Board 3 in Figure 14 on page 20 is unplugged then Board 3, Board 5, and Board 6 will not receive the clock signal and will stop working.

It is important for the system administrator to know how the clock is distributed and that a problem on a single cable or the power off of a switch board may also affect other switch boards.

3.3 STI Timing and Logic Synchronization Process

Communication between two switch chips, or between a switch chip and a node adapter, must be synchronous, and the two sending and receiving ports must sample data in the channel in the same way. All switching elements and the node adapters share a common clock, but due to propagation time through the links, the clock phase may not be exactly the same.

A hardware phase-tuning procedure is invoked during system initialization to adjust the capture time of each data bit at the receiving side, so that it is sampled at the correct time. This makes adjusting cable length unnecessary while maintaining high link bandwidth.

Each port of a single switch chip has to be independently synchronized with the corresponding port of the neighbor switch chip since each link has to be treated independently.

The synchronization process uses the same wires used by the communication link (see Figure 12 on page 16), and it is run just before data can be transmitted. The receiver is considered the master. It receives data and clock signals from the sender (slave) port on the other end of the link, and returns signals on the token line to the slave unit.

The initialization sequence can be started by either the master or the slave side of the link. The sequence will start in response to a reset of the port due to power on or chip reconfiguration, or by the detection of one error condition on the link. If the synchronization process fails for any reason, it is restarted again.

Once the link has been synchronized, data can flow between the sender and the receiver. The STI macro follows the signal phase and continuously tunes the channel.

A new synchronization process may be needed due to link errors or link delay variations that cannot be tuned out. When the STI macro detects that the phase of the data is nearing the edge of the guard band defined by the current setting of the delay elements, or when the receiver logic determines that there is an error on the link such that the incoming data is suspect, the receiver transmits a synchronization request code to the sender. The sender decodes this code and at the end of the current packet, the synchronization process is started again.

3.4 The Switch Chip

The switch chip is the heart of the switching network. Several chips are connected together in order to create a multistage switching network that allows data to flow from one end to the other with very high performance.

Chips are connected by bidirectional channels that operate at a peak bandwidth of 150 MBytes/second in each direction, while aggregate switch chip bandwidth is 1.2 GBytes/second with minimum or no network contention. The chip latency is less than 300 ns.

Reliability is a crucial consideration in the architecture of large networks. For this reason, the SP Switch is designed to detect and isolate network or chip failures with minimal additional circuitry. Link failures are first handled by the switch chips without propagating them to the rest of the network. Recovery actions are started and normally the link resumes operation. Data that was passing on the faulty link is buffered in the upstream path.

When problems are not recovered by switch chips, administration software (the *fault service daemon*, see 4.7, “Fault Service Daemon” on page 64) disables the single link and re-routes traffic away from it. The use of redundant network topology makes it possible to continue operation even in the presence of multiple channel faults.

Communication subsystem protocols provide the packets with end-to-end protection as well, as described in Chapter 3, “Communication Network Hardware” on page 13.

The switch chip contains 8 receiver modules, 8 transmitter modules and a Central Queue. The switch ports send and receive one byte at 150 MHz frequency. Due to the internal clocking of 75 MHz, the switch chip has to manage two bytes at a time of data coming from the link in order to follow the higher clock source of the link. Each such two-byte quantity is called a *flit*.

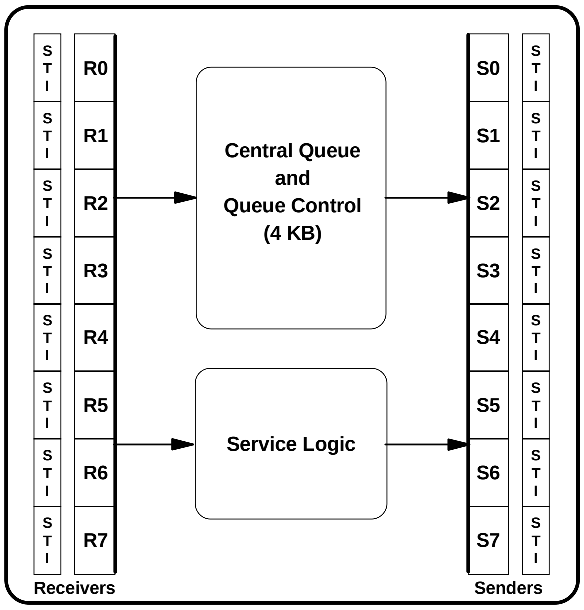


Figure 15. Switch Chip Components

The received flits of a data packet are inserted into the Central Queue from where the sender module can extract and transmit them when it is not busy sending another packet. When more than one packet is directed to the same output port, flits from the first packet are immediately extracted from the Central Queue, while the others remain in the queue waiting for the first packet to finish.

Flits from a service packet directed to the switch chip are routed directly into the chip's service logic.

3.4.1 Receiver Modules

There are 8 receiver modules in the switch chip, each listening to incoming data on its respective link of the chip.

The receiver logic of the switch is responsible for taking data off the link, synchronizing the data with the chip's clock, and routing that data to the appropriate sender port via the Central Queue buffer. The receiver also routes service packets to the chip's service logic if the packets are destined for that chip. The receiver controls the data flow through the link using a token-based flow control implementation, where it transmits a token signal to the sender each time it is able to receive one more flit. The sender counts the tokens received to know how many flits can be sent.

In this section we describe some details of the receiver components. Use Figure 16 as a reference.

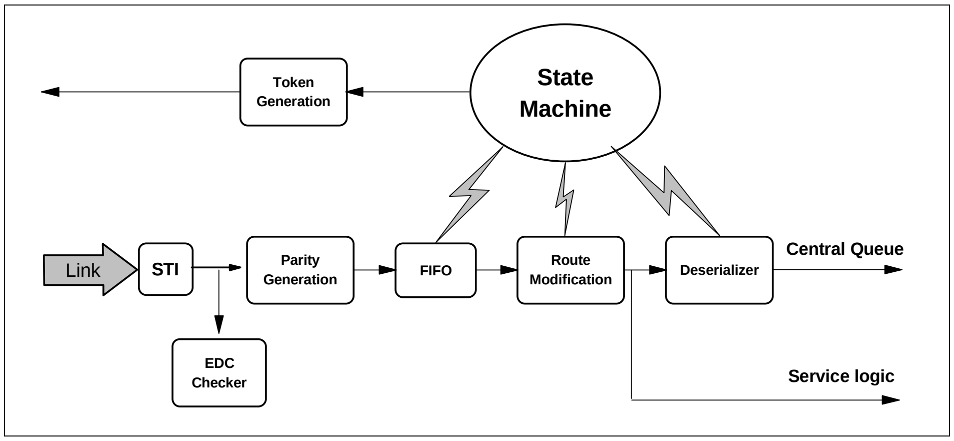


Figure 16. Receiver Modules Components

The data and clock synchronization is accomplished by the Self-Timed Interface (STI) physical receive macro as described in 3.3, “STI Timing and Logic Synchronization Process” on page 20. Data arrives at the STI a single byte wide, but at twice the internal operating frequency. The STI aligns the bits into the proper byte and presents the logical receiver with a flit every internal clock cycle.

As the data enters the receiver, it is checked for correctness. All bytes transmitted across the link (packets and control characters) are protected by a time-based, two-byte Error Detection Code (EDC). The transmitted data and the two-byte EDC code are read from the STI interface and run through the EDC checker. EDC errors can be reported when they happen or when the number of errors has reached a programmable threshold. If the threshold is reached, the link is reset.

A parity bit is internally used by the switch chip to protect each byte of data that flows through it. The bit is generated by the receiver module and checked by the sender module to avoid data corruption inside the chip.

Data read from the STI interface is written into a 64-flit size RAM array with one write and one read port; this is used as a FIFO. The purpose of the array is to buffer the packet data if the flow to the sender is somehow delayed; for example, waiting for arbitration to write into the Central Queue buffer. A receiver state machine controls the reading and writing of the FIFO, as well as all other operations on the module.

The size of the array is determined by calculating the maximum number of token credits that can be outstanding at any given time. The *total* number of tokens in the system must be greater than the *required* number in order to keep the links running at full bandwidth in the worst case.

Each time a single flit is extracted from the FIFO, a token signal is transmitted back to the sending module to inform it that there is one more space available.

Taking into account cable and wire length, propagation delays and clock cycles for STIs and token management on both sides of the link, the optimal number of tokens is 47. In order to allow significant buffer for possible implementations with longer cables and faster links, the array size was chosen at 64 tokens.

If the FIFO becomes full and valid packet data is received on the link despite the token protocol, the FIFO overflows and packet data is lost. The packet currently in transmission is terminated with a packet fail character and the receiver module resets itself, starting a re-initialization of the link with the sender.

A packet ending with a packet fail character may be transmitted over the switch network to the destination node or chip where it will be treated accordingly. There is no other way to handle this kind of packet problem since the preceding data has already been transmitted to the following stage of the network.

In order to prevent an overflow of the FIFO due to link errors, two tokens less than the physical FIFO size are used on the link. In this way the link can gain up to 2 flits through link errors without data loss. When a third flit is gained, then the FIFO overflows.

The *route modification stage* uses the packet's route information to decide if the packet is destined to the switch chip or to which sending module it has to be passed. The route information is also modified to make it usable by the following switch chip.

The receiver is concerned only with the first route flit it encounters, defined as the flit following the BOP control character. Each route byte contains one or two route values. So in each flit, up to four route values may be present. Once the receiver decodes the route value to be used, it invalidates that value.

BOP comes in two forms, BOPa and BOPb, to tell the chip which byte of the following flit is to be used. BOPa is used when the route byte is in the most significant byte (high byte), and BOPb is used when the route byte is in the least significant byte (low byte).

A route byte is depicted in Figure 17, and it contains two route values. Depending on the value of the most significant bit, bits 1 to 3 or bits 5 to 7 will be processed as the next route data.

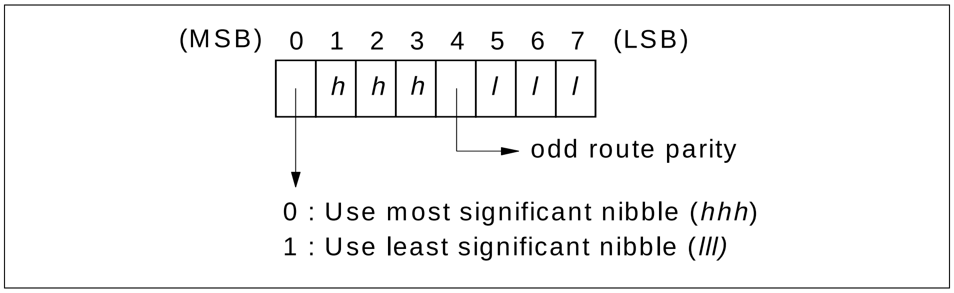

(MSB) 0 1 2 3 4 5 6 7 (LSB)
h *h* *h* *l* *l* *l*
odd route parity
0 : Use most significant nibble (*hhh*)
1 : Use least significant nibble (*lll*)

Figure 17. Route Byte

The parity bit in the route byte is used as an extra check on the correctness on the byte. If a parity error occurs, the receiver would no longer know how to direct the packet, so it simply dumps it without routing it.

When bits 1 to 3 contain the current route value to be used (the MSB bit is 0), that choice must be invalidated. This is done by setting the MSB bit to 1 and passing the modified route byte with the packet. When route modification logic encounters a route whose MSB is 1, bits 5 to 7 are decoded as the route. Since there is no more useful information in this byte, the entire byte has to be invalidated. If it is the first (high) byte of the current flit, the packet will exit from the switch chip with a BOPb character. If it is the second (low) byte, all data in the flit has been used and the flit is not passed on.

As the packet passes through switch chips, all the route bytes get used and discarded. A packet reaches its destination with no route bytes.

As seen in Figure 15 on page 23, after route modification, the data is ready for transmission to the service logic (if the destination is the current switch chip) or to the sending port through the deserializer for buffering into the Central Queue.

The Central Queue is a 4 KByte array with one read and one write port; since there is only one write port and eight receivers requesting its resources, arbitration is required. Those ports requesting Central Queue's resources that are denied access continue to receive packet data while they wait.

The deserializer takes care of arbitration and is constructed to introduce minimal delay for Central Queue access waiting, buffering flits from the route modification circuit and acting very efficiently in sending flits to the Central Queue.

The use of a Central Queue instead of several local buffers on the receivers makes it possible to allocate more buffer space to the most active receiving ports, reducing the probability that a receiver does not have any more buffer space available. As long as the Central Queue is not full, each input port can continue to receive flits at full bandwidth. If the Central Queue eventually fills, input ports with flits destined for the Central Queue are not able to empty their input FIFO queues, and a lack of tokens causes the associated upstream output port to be blocked.

The *receiver state machine* is the brain behind the operations of the receive port. By having a state machine on each port, faults are better isolated and recovered locally, without affecting the remainder of the chip function. All the logic in the receiver is controlled by the receiver state machine. Each state defines what is currently happening within the receiver.

The receiver may be in one of the following major states:

- Disabled
- Tuning
- Operational
- Service

The *disabled* state may be entered for several reasons, among them being a port/chip reset, the link has been disabled, an error occurred or the sender has requested a link synchronization. The reset and link disabling are requested by the supervisor, while errors are internally generated.

A *link enable bit* is associated with each receiver. When the receiver port is not enabled, it does not respond to the synchronization request from its corresponding send port, and actually isolates the send port from the rest of the switch fabric. It defaults to active during switch chip initialization, but may be changed by a service packet.

The disabled state is used by the receive logic to set the state machine in a known state. If the link enable bit is active, the STI starts a a synchronization process (see 3.3, "STI Timing and Logic Synchronization Process" on page

20) with the sending port while internal counters and pointers are reset, and the state machine goes into the tuning state.

In the *tuning* state, the STI proceeds in its signal exchange with the sending port in order to synchronize with it. If any error is detected, the state machine returns to the disabled state and all synchronization restarts from the beginning. As soon as the STI identifies a living sender and it is aligned with its clock phase, the state machine goes into the operational state.

While in the *operational* state, the data packets are received and transmitted to the proper destination. The receiver waits for a BOP character and the route bytes are analyzed to define which sending module has to manage the packet.

If a parity error is detected in the route byte, the error is reported, all flits up to the EOP are discarded, and the port waits for the next BOP.

If a timeout occurs while waiting for the EOP character, the state machine will append a *packet fail* character to the packet and will restart waiting for the BOP. The packet fail character will indicate the end of the packet and will tell you that the packet has to be discarded.

A packet's end is detected either by a EOP or a packet fail character. If a second BOP is received, the receiver first inserts a packet fail character, then proceeds with the second BOP looking for the route byte.

If the received packet is a service packet sent to this switch chip, the receiver module reads a special character after the BOP character. The data portion of the packet is then passed to the chip's service logic and the state machine makes a transition into the *service state*.

3.4.2 Sender Modules

There are 8 sender modules in the switch chip, each taking care of data to be sent out of its respective output port.

The sender logic of the switch is responsible for taking the packet data out of the Central Queue or from the service logic and transmitting it across the cable or board to the receiver on the adjacent switch chip. The sender can only transmit data when it has received valid tokens from its corresponding receiver indicating that the receiver has room to handle it.

In this section we describe some details of the sender components. Use Figure 18 as a reference.

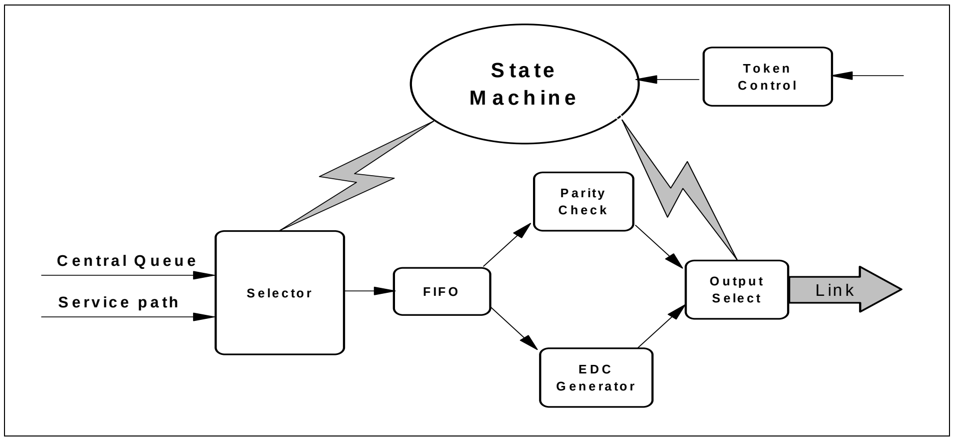


Figure 18. Sender Module Components

The selection stage of the sender module is driven by the state machine that defines from where data has to be taken and sent to the output port. Data from the selector is passed into the sender's FIFO.

The purpose of the FIFO is to smooth out the data flow from the selector into the reminder of the sender logic. It is a 16-byte Register Array with one read and one write port. There are times when the normal data flow through the sender must be stopped for a while, for example, to insert EDC bytes. When these interruptions occur, data is buffered into the FIFO and the source continues to transmit.

Each byte of packet data being transmitted through the switch chip is protected by a parity bit. Parity is checked just before data enters the sender data selector, from which it is launched from the chip. Parity errors that are detected are reported to the service logic to enable problem isolation. The packet is transmitted as it is and data flow is not interrupted. Because of this, parity errors on data will also show up as CRC errors inside the data at the destination processor nodes. Care must be taken when CRC errors are detected to ensure the fault is isolated to the proper place.

The sender module is in charge of creating the EDC frames used to check the correctness of communication on the link. It inserts the EDC values that will be checked by the corresponding receiver module.

The output data selector generates the data to be sent on the link depending on the state of the sender. It will transmit packet data, EDC bytes or control characters to the receiver module.

The *Token-In* control logic is used both on link initialization sequence checking and for normal operation token protocol. The received tokens cause the token counter to be increased by one, and each time a flit is transmitted out into the link, the token counter is decreased by one. The counter is initialized during link initialization. It can count up to the size of the receiver module's FIFO.

If the token counter is at maximum value, a token is detected and a flit is not simultaneously transmitted across the link, the token counter will signal a *token overflow error*. The counter, however, will not overflow and will remain at the maximum count. By design, the counter cannot underflow since a token count of zero will not allow packet data to be transmitted, and transmitted data is what decrements the counter. The token protocol is designed to check if tokens are missing, and to realign sender and receiver counters.

Token transmission errors may occur on the token line in the form of wrong token sequences. If this happens, an internal error counter is incremented and if a pre-defined threshold is reached, a token error character is sent to the receive logic and the link is re-initialized.

The sender's state machine is the "brain" that governs the operation of the send port. By having a state machine on each port, faults are better isolated and recovered locally, without affecting the remainder of the chip function. All the logic in the sender is controlled by the sender state machine, and each state defines what is currently happening within the sender.

The sender may be in the following major states:

- Disabled
- Tuning
- Operational

The *disabled* state may be entered for several reasons including a port or chip reset, disabling of the link, or a request for a synchronization by the receiver.

A *link enable bit* is associated with each sender. When the sender port is not enabled, it does not transmit any valid packet to its corresponding receive port and actually isolates that receive port from the rest of the switch fabric. No token signals are processed when the link is disabled; flits that should use the port are discarded and an *invalid route error* is notified to the service logic. The link enable bit defaults to active during service initialization, but may be changed by a service packet.

The disabled state is used by the receive logic to set the state machine in a known state. If the link enable bit is active, the synchronization process with the receiving port is activated, internal counters and pointers are reset, and the state machine goes into the tuning state.

In the *tuning* state, the synchronization process is active and if any error is detected, the state machine returns to the disabled state and all synchronization restarts from the beginning. When the process ends, the sender module receives a specific signal from the token line and the state machine goes in the operational state.

The *operational* state indicates that the entire link is active and the sender is either waiting for or already busy handling packets coming from the Central Queue or the service path. Tokens received in this state increment the token counter and each flit sent out the port decrements the counter.

The priority for accepting packets to be transmitted is the following:

1. Service packets
2. Central Queue packets

Service packets have priority over all other packets, and other packets have to wait for service to be transmitted. By *service packets* we mean the packets generated by this switch chip and not packets that have been received by a switch chip’s input port, since all data coming from other ports is treated in the same way.

When the service logic is not ready to send any service packets, the source selector checks if there are new packets buffered in the central queue and starts to extract the flits from queue into the FIFO.

If the sender module is transmitting a packet, it continues until the packet is completely sent, even if higher priority packet is waiting. Priority is only checked when deciding which *new* packet has to be sent; it never interferes with packets already on the link.

3.4.3 Central Queue

Any packet from a receiver module is buffered in the switch Central Queue. The queue is a 4 KB RAM with one read port and one write port. On any given cycle a write to the array, read from the array or simultaneous read and write may occur.

The access logic has to arbitrate among multiple read requests from the senders and multiple write requests of the receivers. This arbitration is made using a least recently used algorithm (LRU).

The packet data written to and from the array is grouped into eight *chunks* representing 16 bytes of packet data, and is moved at chunk size to reduce latency of the buffering operations. Each chunk is defined as either a *header chunk* (the first chunk associated with every packet) or a *continuation chunk* (all the chunks following the header chunk inside the packet).

To prevent "starvation", receivers requesting service from the Central Queue are divided into two groups: those offering *critical* chunks and those offering *non-critical* chunks. A critical chunk is one that is ready for immediate forwarding to its destination transmitter. The first chunk of a new packet is critical if no other packets are queued for its transmitter. A continuation chunk is critical if all previous chunks of the packet have been forwarded to the transmitter. Chunks not immediately ready for forwarding are called non-critical.

One chunk of the Central Queue storage is reserved for each transmitter and it is called the *emergency slot*. The remaining Central Queue storage is shared. Whenever a receiver has a non-critical chunk and space is available in the shared pool, the receiver requests service from the Central Queue. Whenever a receiver has a critical chunk, it requests service and the chunk is put in the transmitter's emergency slot. Priority is assigned to the receivers on a least recently serviced basis.

The emergency slots provide a way for the receiver to transmit the packet data the sender is waiting for into the Central Queue buffer, even when the buffer appears full, thus avoiding deadlocks. There are special cases where the Central Queue buffer may become full: the emergency slots ensure that all critical chunks are sent to senders.

3.4.4 Service Logic

Any service packet received by the switch chip from any of its eight ports and destined to the switch chip itself is passed to the service logic.

The service packet looks exactly like any other message packet to a receive port until all the route bytes are used by previous switch chips. In this case the *service frame character* (SVC) is detected right after the BOP and the receiver routes the data of the packet to the service logic (see Figure 19). Each byte transmitted has a tag added to it which identifies it as a data character or a control character.

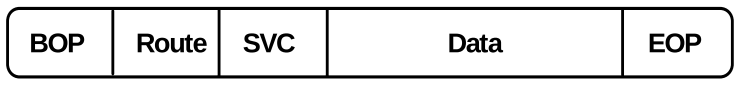


Figure 19. Service Packet Format

Since a service packet can come into the chip through any of the eight receive ports, the service logic must be able to accept the service data from any of these ports. It is assumed that only one receiver module transmits a service packet to the service logic at any given time. If two or more packets are received into the service logic simultaneously, they are all accepted (logical OR of their data), producing an invalid length, invalid command, and CRC error, and an error packet is sent. This causes the service logic to discard the packet as if it has never been received. It is up to the packet's sender to detect that data has not been received and a new packet has to be sent.

Normally the logic is idle, waiting for an incoming command or an internal error that has to be reported. There are five possible incoming messages and only one report message. When an incoming message has been correctly processed by the service logic, a report message is sent like an acknowledge. See Appendix A, "SP Switch Service Interface" on page 227 for more details.

The *initialization packet* contains the information that has to be loaded into the switch chip's configuration registers. It sets information such as which receive and send ports are enabled, the route to be used when sending a report message, and the switch chip identifier.

The *read status* packet is used to make the switch chip send back a complete report message with its current status.

The *reset packet* requests a reset of selected error registers and logic on the switch chip. This message is required to receive notification of secondary error events as described in 3.4.5, "Error Isolation" on page 34.

Two time-of-day (TOD) packets are used to define and propagate to all switch chips a value, the TOD, that is maintained synchronously in the whole SP Switch to make problem determination and error isolation easier. The TOD is increased at each switch chip's internal clock tick.

As a result of incoming service packets, or after the detection of an internal error, the service logic generates an *error/status packet* containing all the error and status information on the switch chip. Two identical packets are sent with the same data but with different routes to a specific node called the primary node. If no previous initialization packet with such routes has ever been received, the packets are not sent.

When the service logic is busy preparing and sending an error/status packet, it cannot process any incoming service packets. They are discarded until the switch chip has sent the message.

Important

An important consideration on internally generated error/status packets is that two of them cannot be sent by a switch chip without an intervening reset packet being received from the primary node. This keeps the switch chips from flooding the switch fabric and the primary node with service packets. It also requires the primary node to manage all incoming signals and explicitly reset them.

3.4.5 Error Isolation

Internal error isolation logic is present on the switch chip to capture the first error that occurs. Subsequent errors are also recorded, leaving an error trail.

Upon detection of the first error, an error/status service packet is sent to the primary node, and succeeding errors are not reported spontaneously until the primary node has handled the error condition and has cleared the error with a reset packet.

There are three multibyte error registers that have to be considered when getting the report from the switch chip. Each bit of each register indicates a separate error present.

The *first error capture register* is where the first occurrence of any error is stored. It marks the *kind* of error present.

The *second error capture register* details exactly *where* the error indicated in the first error capture register occurred, as well as where any subsequent errors occur. It will continue to accumulate errors until it is reset by a reset service packet.

For a bit to be set in the first error capture register, no other bit can be active in the entire register. Once a bit has been set, no other bits may be set until the contents of the register are reset back to logic 0 by a service reset packet.

When an error is detected by the switch chip, the corresponding bit in the second error register is checked. If the bit is already set, the error has already been addressed, so no action is taken. If the bit is not set, the error is added to the register and the first error register is checked. If first error register has a value of zero, this is the first error and the corresponding bit is set; otherwise the register is not changed.

The *pending error capture register* is a shadowed version, bit for bit, of the second error capture register. This register is used to ensure that errors occurring between the reporting of previous errors (in an error/status packet) and the service reset packet that should clear those earlier errors are not lost because of the service reset packet. The contents of the pending error capture register are not transmitted in an error/status packet, but are copied to the second error capture register when servicing a reset packet.

A more detailed description of the error registers content can be found in A.2, “Error Registers” on page 242.

3.5 Switch Supervisor Functions

Each switch board of the SP Switch has its own switch supervisor card from which it is configured and monitored. Each switch supervisor card is connected to its respective frame supervisor to provide the following functions for the board:

- Board clock selection
- Board status monitoring
- Board reset
- Fan rotation sensing
- Power supply monitoring and control
- Board level sensing
- Board configuration sensing

3.5.1 Board Clock Selection

The selection of the clock configuration in the switch board is made by the supervisor card. It defines which is the master switch chip of the board. On master boards it chooses the internal oscillator, while on slave boards it selects which port of which chip is to receive the external clock signal.

The supervisor sends signals to the switch chips in order to configure them according to the requests coming from the Control Workstation via the frame supervisor. When powered on, it sets the internal oscillator (OSC2) as the clock source on the board, until otherwise configured.

3.5.2 Board Status Monitoring

The supervisor monitors the key activities of the switch board and some environmental parameters, providing them to the system administrator as variables that can be displayed with commands like `spmon` and `hmmon`.

3.5.3 Board Reset

The supervisor interface is also responsible for the switch board's power-on-reset (POR). The POR sequence of the switch chip is as follows.

1. The supervisor asserts POR signal to the board
2. The POR opens each switch chip's A-clock and B-clock, flushing the storage elements to a known state (all logic zero)
3. The POR signal goes inactive
4. The switch chip's Built-In Self Test logic takes control of the chip
5. At the end of self test, a synchronous reset is generated internally to the chip, resetting and presetting all registers
6. The Self-Test Complete signal goes active and the POR sequence is complete.

Self-Test takes approximately two seconds to complete.

3.5.4 Fan Rotation Sensing

As previously mentioned, there are five cooling fans in the switch board assembly. The rotation of these fans is monitored by the switch supervisor. If one fan is not working, the switch supervisor will notify the system monitor of an environmental warning. If fans are not enough for switch cooling, the switch supervisor will turn off the power to the switch board assembly, and notify the system monitor of an environmental failure.

There is N+1 fan redundancy in the switch assembly. If a fan fails, the maintenance on the switch is deferred to an acceptable time, instead of causing an entire switch board incident, affecting multiple nodes and possibly other switch boards. The fan assembly is also a Field Replaceable Unit (FRU) of the switch assembly, that is, it can be changed without replacing the entire switch assembly, helping to reduce the maintenance cost of the switch board.

3.5.5 Power Supply Monitoring and Control

As previously mentioned, there are two power supplies in the switch assembly which share the load required by the switch board. The power supplies take 48 volts as an input and produce 3.3 volts to the switch board. The switch supervisor monitors three things:

- The availability of the 48 volts
- The current usage of each of the power supplies
- The output 3.3 volts of the power supplies

If the switch supervisor detects that the 48 volts is available, it enables the power supplies to provide the 3.3 volts to the switch board. If the current levels monitored by the switch supervisor move out of the valid operating range, the supervisor will turn that power supply off. Likewise, if the output voltage sensors indicate the 3.3 volts is not within the valid operating range, the power supplies will be shut off.

The second power supply is redundant in the switch assembly. If one of the power supplies fails, the maintenance on the switch is deferred to an acceptable time, instead of causing an entire switch incident, affecting multiple nodes and possibly other switch boards. Each power supply is also a Field Replaceable Unit (FRU) of the switch assembly.

3.5.6 Board Level Sensing

The switch supervisor card has the ability to sense the current version of the switch board being used. This becomes necessary if the software running in the system is switch board version-specific. Currently there are several levels of the switch board; however, only the last version is shippable to the field.

3.5.7 Board Configuration Sensing

An SP Switch board can be assembled with two different configurations: the full switch has 32 input ports and 32 output ports, while the low-cost version for entry-level systems has 8 input ports and 8 output ports. The switch supervisor senses which configuration is in the system.

3.6 The Switch Adapters

The switch adapters are the interface between the SP Switch and the RS/6000 node. They are connected to a switch chip and the node's internal bus and have the purpose of checking and reformatting data to make software on the node receive and transmit messages with other nodes or switch chips.

There are different kinds of adapters, depending on the bus type on the RS/6000. Their internal structure changes among them due to technology improvements and to bus architecture differences. The functions they perform are, however, the same since the software present on the RS/6000 is the same except for the peculiarities introduced by the system bus.

The adapters known at the time of the writing are:

- TB3, used in microchannel nodes
- TB3MX, for nodes whose adapter is put on the memory bus
- TB3PCI for nodes that have the adapter in a PCI bus slot
- TB3MX2, an evolution of TB3MX

3.6.1 TB3 Adapter

The TB3 adapter internal structure is described in Figure 20.

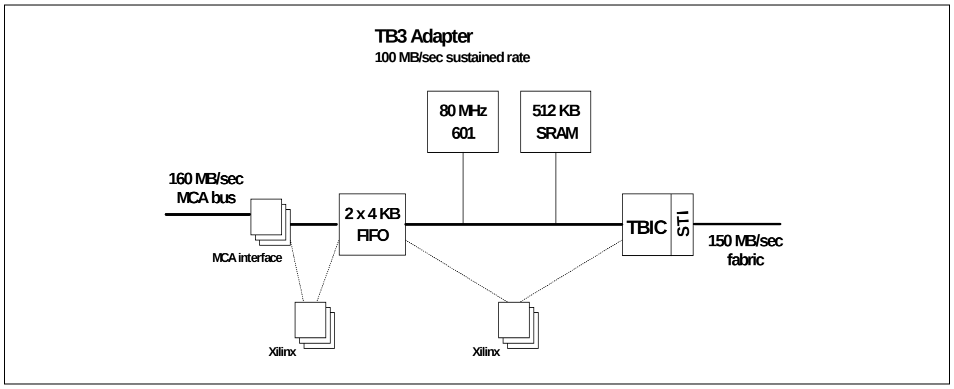


Figure 20. TB3 Adapter Structure

This adapter is used in all microchannel MCA-based RS/6000 nodes. It has a *Trail Blazer Interface Chip* (TBIC) that manages the link with the switch board in the same way a switch chip does, and it connects to the adapter's internal bus. On the other side is the microchannel interface with the node's memory.

A bidirectional FIFO in the middle of the bus is used to buffer data while DMA transfers occur between the microchannel bus (*left-hand DMA)* and the TBIC (*right-hand DMA)*.

The heart of the adapter is an 80 MHz 601 PowerPC with 512 KBytes of SRAM.

The sustained data rate for this adapter is 100 MB/sec.

3.6.2 TB3MX Adapter

The TB3MX adapter is an adapter used in PCI nodes that can plug into the memory bus to gain higher performance. It is represented in Figure 21.

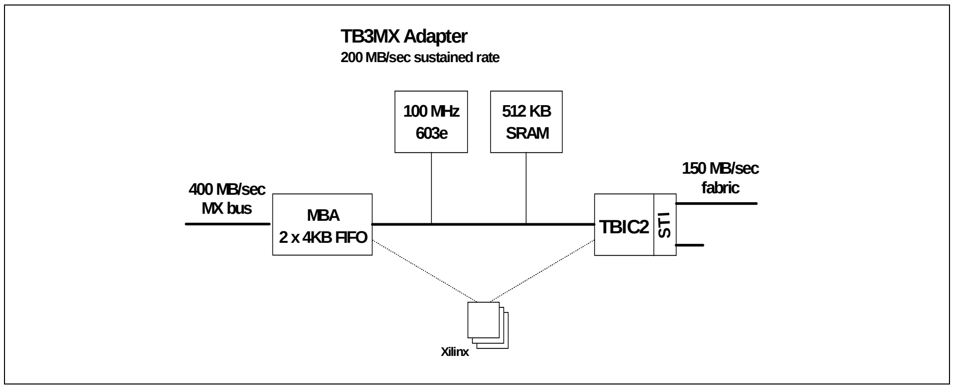


Figure 21. TBMX Adapter Structure

It is a redesign of the TB3 adapter for the MX bus of the node. The most significant difference is the replacement of the MCA interface with an MX bus interface achieved with the MX Bus ASCI chip (MBA). Other significant differences are:

- 100 MHz PowerPC 603e (instead of the 80 MHz PowerPC 601)
- Internal bus operates at 50 MHz (instead of 40 MHz)
- New TBIC-2 chip replaces old TBIC
  - 50 MHz 60x interface
  - 4 MB send buffer
  - IP checksum generation logic

The MBA chip provides both the MX bus interface and buffering for DMA operations between the MX and 60x buses.

3.6.3 TB3PCI Adapter

This adapter is used in those systems that are connected to the switch board using a PCI adapter. The design of the adapter is shown in Figure 22.

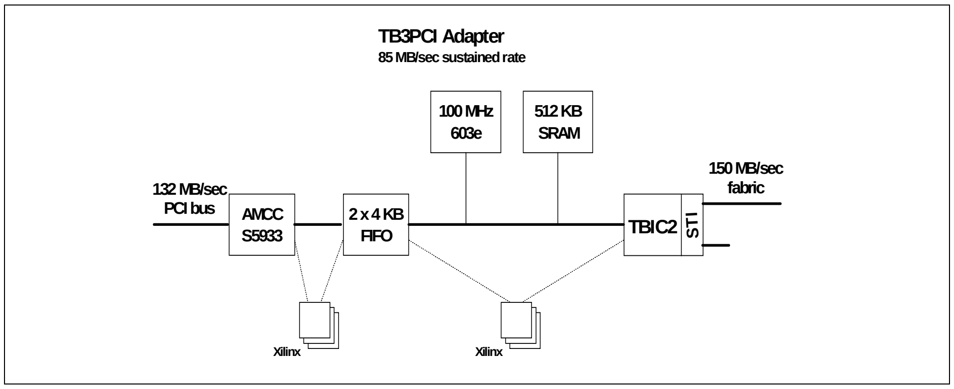


Figure 22. TB3PCI Adapter Structure

The interface with the PCI bus is made by an AMCC chip, and a TBIC-2 is used to connect to the switch board. An internal bidirectional FIFO is used as in the TB3 adapter. The internal components are:

- 99 MHz 603e PowerPC
- 512K SRAM
- Single eight-byte 33 MHz bidirectional internal bus

Due to limited bandwidth in the PCI bus, the sustained bandwidth is 85 MByte/s.

3.6.4 TB3MX2 Adapter

TB3MX2 is the latest version of the adapter to be used in nodes implementing an MX-type bus connection to the node's processor and system memory. Its design is shown in Figure 23 on page 41.

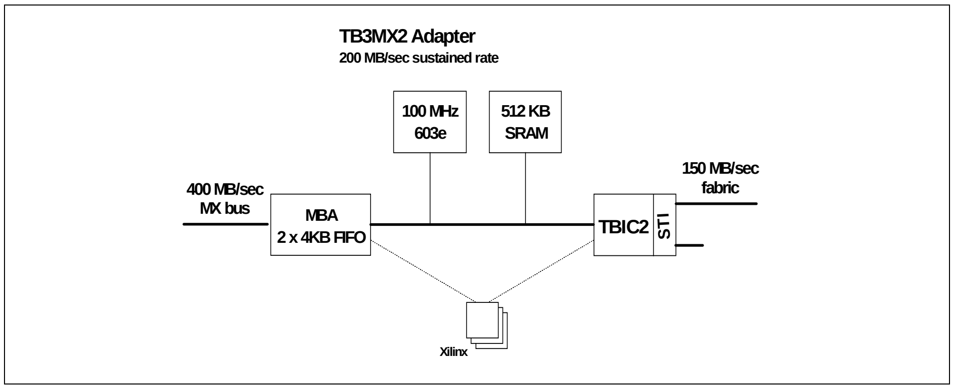


Figure 23. TB3MX2 Adapter Structure

This adapter is a modification of the original TB3MX adapter. It includes more robust MX bus drivers within the MBA chip and allows the node to use either a 64-byte or a 128-byte cache line. The change allows the adapter to operate with a 66 MHz MX bus rather than the 50 MHz figure specified in the original design.

It also enables attachment of additional devices on the MX bus. The adapter may be used in any node implementing an MX bus.

Its main characteristics are:

- 150 MB/s port to SP Switch (each direction)
- MX bus attachment to node
  - 8-byte 66 MHz bidirectional multiplexed address/data interface
  - DMA uses real address translation by adapter microcode
- Its internal components are:
  - 100 MHz 603e PowerPC
  - 512K SRAM
  - TBIC-2 logic using CMOS 5L 10.9mm chip
  - MBA logic using CMOS 5L 10.0mm chip
  - Single 8-byte 50 MHz bidirectional internal bus

3.6.5 Data Flow

The adapter’s hardware and microcode interact with the communication subsystem (CSS) software on the RS/6000 node. Data is transferred through the MCA, MX or PCI busses in both directions using DMA, and several data structures on the adapter are directly read and modified by CSS using *ioctls*.

On the RS/6000 there are several independent environments called *window*s that send and receive data to and from the switch. There are three different kinds of windows:

- IP
- Service
- User space

The IP window is responsible for the IP communication among nodes; the service window manages configuration and monitoring of the switch network; the user space windows permit high-speed data communication among user applications. In PSSP 3.1 there are up to four user space windows available for each node.

Each window has its own *receive FIFO* (rFIFO) and *send FIFO* (sFIFO) in the node’s main memory. The window uses these FIFOs to store application packets before they are received from (rFIFO) or sent to (sFIFO) the switch adapter. These structures are also available to the adapter’s microcode, which uses them to move packets to and from the RS/6000 main memory to the adapter’s TBIC FIFO, using DMA.

Each window has a set of variables that describe the status of its FIFOs, like the position of first available packet and the number of packets. These variables are available to both windows and adapter microcode and are used to properly transfer data to and from the window’s FIFOs and the adapter.

The internal structure of different adapters may vary, but they all share a common basic structure as shown in Figure 24 on page 43. The adapter has a bidirectional FIFO where packets from the windows’ send FIFO are put before these packets are sent to the adapter’s TBIC FIFO on their way to the switch network. All data transfers are made by DMA engines on the adapter under the control of the adapter’s microcode.

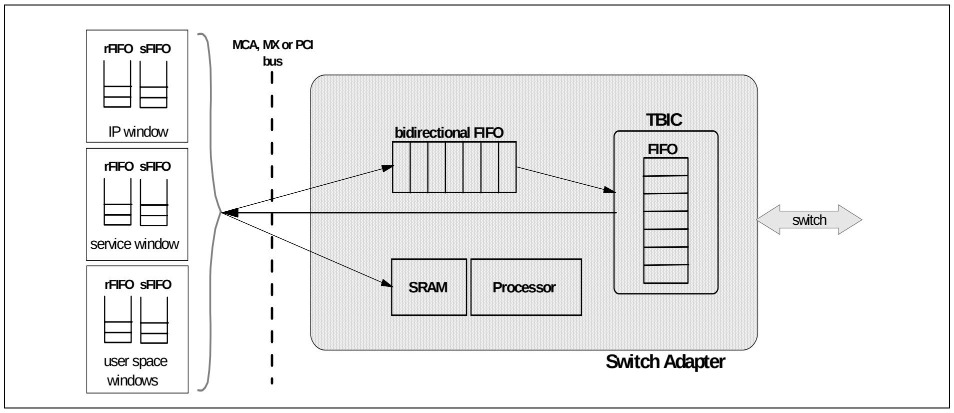


Figure 24. Adapter Logical Structure

After packets arrive from the SP Switch, another DMA engine moves the data from the TBIC FIFO directly to the windows' receive FIFO. The microcode uses its internal tables to create the correct switch routing information for outgoing packets, and to send incoming packets to the correct window on the SP node.

The packet coming from the sFIFO of a window has a 1 KByte maximum size and has the following structure:

- Header
    - Destination node identifier
    - Destination route
- Data

The data contained in the packet is formatted according to the layer that manages the communication at the sFIFO level (see Chapter 4, "Communicating with the SP Switch" on page 49 for more details about User Space).

The header is used by the microcode to create the real route bytes that are used by the switch chips to transport the data. A set of tables in the SRAM of the adapter contain all the information needed to identify the destination and the correct route, depending on the window type.

When a *user space window* packet is passed into the adapter's FIFO, the destination node of the header is a *logical identifier* and not a node number. The user process does not have to know which the SP system topology is or where the destination is: it requires to communicate with a given process that has a given identifier.

The distribution of jobs in the SP system is handled by special job managers, such as Load Leveler for AIX. The presence of such managers makes it possible to better distribute loads on several nodes and to simplify the application processes. The job manager is responsible for defining in which nodes the processes that have to exchange messages are started, and tells the adapter's microcode how these job are distributed. Before starting any job, the manager creates for each user space window a table in the adapter's SRAM that gives, for each logical destination of the window, the following information:

- Physical destination node
- Destination process' window
- Key

The pair destination node and window make it possible for the receiving node to correctly dispatch the incoming packet. The key field is a unique value that is used by the communication between two windows. No other windows, including IP and service, have the same identifier.

The use of the key ensures that a process never receives a packet from a wrong sender. A new job uses the window that was previously allocated to another job and processes may be not aware yet that the window has changed identity. So wrong packets may be sent to the new job, but since they have a different key, they are discarded by the adapter.

The route field of the header is used to select one of the four shortest paths to the destination. The message libraries that user processes use do not know the actual route, but they are aware that four different choices are available, so they make use of all them, creating packets that continuously change the route number. This helps maximize throughput in the network.

The adapter has a complete route table that gives the four shortest paths to each available node in terms of switch chip ports that are used to create the packet's route to destination. This table is created and updated by the *fault service daemon* that runs on the node.

Once a user space packet is put on the adapter's FIFO and is selected to be transmitted, the microcode sends the SP Switch packet feeding the TBIC first with the all the information needed for the routing of the packet, and then with the data that is present on the adapter's FIFO.

The structure of the packet sent by the adapter is shown in Figure 25. It contains not only the usual BOP, route and EOP characters to delimit the packet and to make it reach its destination, but also the physical sender node identifier, key and destination window. A CRC checksum of the packet is also added: it will be used by the receiving side to check the incoming packet.

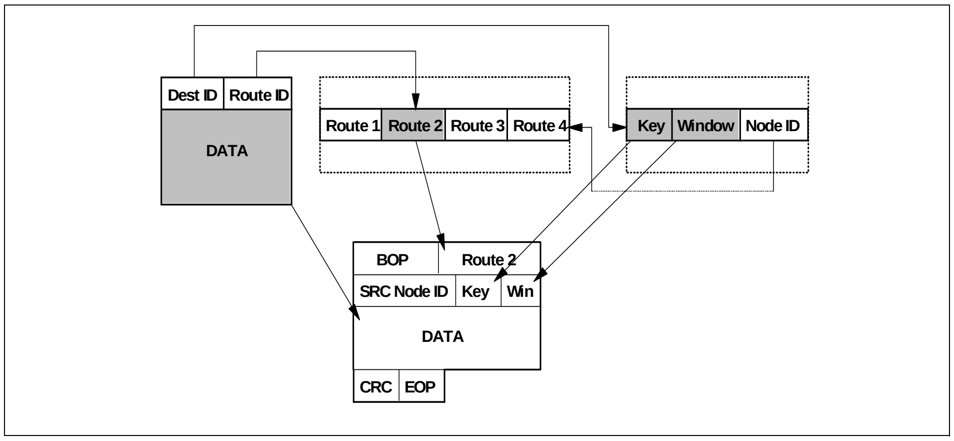


Figure 25. Switch Packet Creation for User Mode

If a *service window* packet is present on the adapter's FIFO, the meaning of the header is slightly different. The process that is sending the packet is the *fault service daemon* and it knows exactly the topology of the SP Switch. It may want to send a packet to a switch chip, and this information is not present in the adapter's SRAM. Instead, the route information coming from the window is the sequence of bytes that must be used in the switch packet.

The adapter adds the BOP, the CRC and the EOP and just copies the route and the data from the packet it has received (see Figure 26).

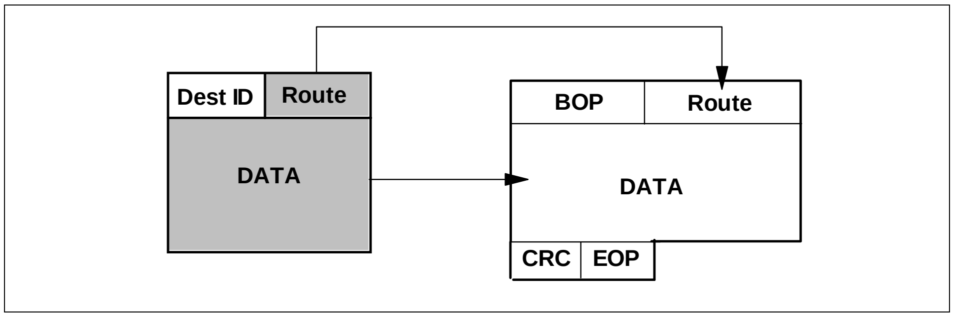


Figure 26. Switch Packet Creation for Service

The last possible case is when the packet in the adapter's FIFO is an *IP packet* (see 4.6, "IP Layer" on page 59 for more details).

IP kernel extension does not know the SP Switch topology but only the physical destination node, so it is the adapter's microcode that has to specify the correct routing. This is done by looking into a table that contains the routes to every node in the system. The table has been supplied to the adapter by the *fault service daemon*. In order to better use the switch, the microcode changes the routes used to reach the same IP destination for each IP packet.

IP packets may have different sizes, up to 64 KBytes, but the maximum packet size in the adapter is 1 KByte. IP is not aware of this limitation and provides the entire IP datagram. The adapter's microcode has to split the data into packets recording the offset of each of them inside the datagram so that the receiving adapter's microcode can reassemble them.

Switch packets are then created as in the previous cases: BOP, route, the key field, the window, the data, CRC and EOP (see Figure 27).

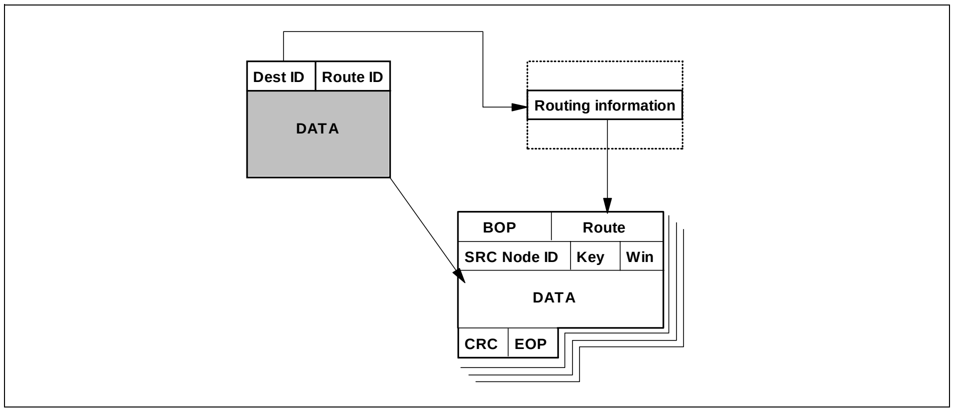


Figure 27. Switch Packet Creation for IP

On the *receiving side*, when a packet is received by the TBIC, the microcode looks at the header received, detecting the sending node identifier, the key and the destination window. Then it checks for the CRC. If any checksum problem arises, the packet is discarded and an Error/Status error message is sent.

If the destination window is for User Space, the source and key values are compared with the expected ones in the table in the adapter's SRAM to check if the packet has to be forwarded. If the packet is for IP, it may need to be assembled with other incoming packets before the IP kernel extension can use it. In any case, the packet is put on the adapter's FIFO and then copied to the correct window in the node's main memory using DMA.

Whenever a packet is ready for the corresponding window, the window's counter of available packets is increased by the adapter. The software on the RS/6000 polls that counter to see if there are packets available. The adapter may issue an interrupt to signal the presence of packets if the counter becomes greater than a given threshold.

The adapter's microcode is designed to follow, as fast as possible the incoming traffic from the switch in order to avoid being the bottleneck of the network. When it does not have incoming packets to handle, it manages all the packets that are ready on the send queues on the node, servicing the windows in a round robin fashion.

3.7 Performance Data

Performance on a network is described in terms of bandwidth and latency. *Bandwidth* is the amount of data that is transferred in a unit of time. In a multistage network, it is defined by the section that is able to move data with the lowest rate. *Latency* is the time needed to transport data from sender to receiver. It is contributed to both by the physical propagation time along the transport media and by the overhead in data management in all the devices involved in the network.

In the SP system, the raw performance is given in Table 1.

Table 1. SP Switch Raw Performance

Number of SP Nodes	Latency (microsec)	Bandwidth (MB/sec)	
		Unidirectional	Bidirectional
up to 80	1.2	150	300
81 to 512	2.0		

All the links in the network are able to transfer 150 MB/sec in each direction, so the bandwidth is constant. The latency is very low compared to other networks and is constant within the ranges shown. When more than 80 nodes are involved, intermediate switch boards are needed to provide the required path redundancy; this new network stage causes a small increase in latency.

The data in Table 1 on page 47 represents the best raw performance the network can give. The performance that may be measured at the application layer will be lower and it will depend on the overall SP system configuration, on the communication protocol used and on the overhead introduced by the application itself. Several parameters may need to be tuned to achieve the best results for each system, and this tuning may be different for each system.

Chapter 4. Communicating with the SP Switch

Software applications on the RS/6000 SP system make use of the SP Switch network, profiting from this low latency and high bandwidth communication channel in different ways.

IP packets can be sent over the SP Switch. The IP kernel extension takes care of feeding data into the SP Switch, and any application that makes use of IP can be run on the system and transfer data through the switch network.

The use of IP requires the application to issue system calls, thus switching to kernel mode in order to send and receive data. Mode switching and IP communications bring significant overhead that affects overall performance. Network degradation may be a factor, especially for parallel processing where a large number of computing nodes are involved and many messages are exchanged.

Better performance can be achieved when an application directly accesses the adapter to send and receive data to other nodes in the network, avoiding mode switching and the overhead of the TCP/IP protocol stack. Applications can use *User Space* communications through standard message passing libraries, obtaining a very efficient use of the switch network.

4.1 Protocol Overview

A communication protocol stack is provided to help applications access the switch adapter. Several components are present to hide the physical details of the adapter and to provide the user application a set of standard libraries that simplify the message passing interface. The protocol stack manages data and provides reliable communications.

As seen in 3.6.5, “Data Flow” on page 41**,** the switch adapter takes care of the physical transmission of data through the SP Switch network and transfers packets of data to and from the node’s main memory. The packets have a specific format and are always sent to the correct environment (*window*) or process that uses them.

Several interfaces are needed to provide both user applications and kernel access to the switch adapter for communication and switch management purposes. The Communication Subsystem (CSS) protocol software and its interaction with other components is shown in Figure 28 on page 50.

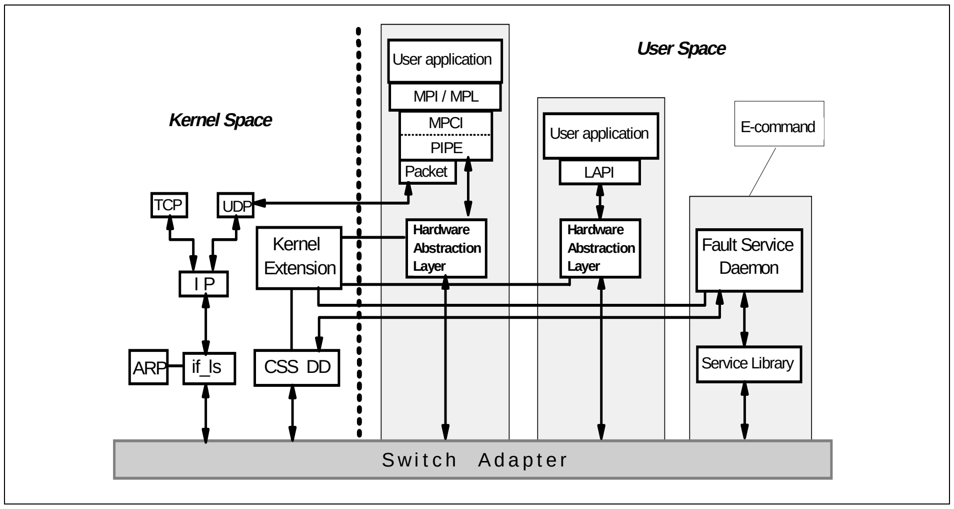


Figure 28. Communication Subsystem Components

User Space applications make use of functions provided by message passing libraries that provide a reliable exchange of data between computing nodes. These libraries make use of a protocol stack described in this chapter. Data is passed directly from and to the adapter and process data structures, with almost no actions from the kernel.

In PSSP 3.1, up to four User Space applications on a single node may use message passing libraries to exchange data with the switch adapter. If more than four applications are required, the communication has to rely on UDP/IP communications over the switch. Performance is reduced, but no application change is required. The UDP protocol can also be used to exchange messages with machines that are not connected to the switch network but are reachable using other TCP/IP communication links.

A special process that runs on each switch-connected node of the SP system is the *fault service daemon*. It accesses the switch using a special service library in order to monitor and configure the SP Switch. Every command issued to configure the switch network (an *E-command*) is directed to this daemon.

IP communication over the switch is made via a kernel interface layer between IP and the adapter, that manages the data transfer with the physical transport medium.

4.2 Message Passing Interface (MPI) Layer

The Message Passing Interface (MPI) library provides a complete set of message passing functions, as defined by the MPI Forum, for point-to-point message passing, collective communication such as broadcast, and environment specification and control. The library provides: functions which can be bound to C (C++) and FORTRAN programs; include files with predefined data types, MPI constants, function prototypes, and so forth.

The library does not take care of allocation of computing resources and starting of tasks on appropriate computing nodes. The MPI message passing library expects all tasks to be started at the same time, and does not deal with situations where the number of tasks varies during a parallel job. A run time environment, like Parallel Environment for AIX (PE), must be used for this purpose. We do not describe the run-time environment. Parallel Environment supports MPI and MPL (IBM's proprietary and practically obsolete Message Passing Library).

Two versions of the MPI implementation are provided:

- Non-threaded library, also called the signal-handling library
- Threaded library, which is thread-safe and uses POSIX threads

The principal function of the message passing subsystem is the reliable exchange of data between computing nodes; the actions performed are send, receive, wait and test. A message is said to be complete when the user application buffer is no longer being used by the message passing subsystem; thus a send is complete when the user may reuse the application data area. A receive is complete when the user may reliably read the contents of the application data area.

Sends and receives may be blocking or non-blocking. A blocking operation does not return from the function call until the requested function is complete. A non-blocking operation returns promptly to the caller, who is restricted from accessing the application data area until the operation is completed via a wait (blocking) or successful test (non-blocking) call.

A non-blocking send or receive is represented by a request handle. The request is created by the send or receive call and the handle is returned to the caller. The request is later completed by a wait or test call using the request handle. The wait or successful test nullifies the handle.

The MPI API defines a number of data types, topology and environment functions. These are implemented within the MPI layer of the stack, along with calls to capture trace data if requested, MPI profiling entries, and specific language bindings. The MPI layer also maintains information about MPI communicator structures and contains the appropriate algorithms for implementing collective communications, which are based on estimates of latency and bandwidth obtained when the library is initialized.

All MPI functions involving actual communication with other tasks are mapped into point-to-point operations implemented via the MPCI library.

4.3 Message Passing Client Interface (MPCI) Layer

Message Passing Client Interface (MPCI*)* is the primary interface to the point-to-point message passing protocols that use the SP Switch. MPCI supports communication via UDP/IP and direct User Space access to the family of switch adapters.

The UDP/IP implementation of MPCI provides a general purpose solution that handles multiple tasks (processes) per node and communication over any network, while the User Space implementation of MPCI supports up to four tasks per node. The User Space provides the best performance by allowing direct access to the adapter, avoiding the cost of switching to kernel mode to send and receive messages.

The MPCI layer provides functions to MPI and MPL, giving them access to the lower PIPE layer. It also serializes message passing over its window, since only one thread at a time may access a window.

MPCI deals with messages sent and received by the user application, relying on the PIPE layer for a reliable, byte-stream physical transport medium between tasks. Each user task has two PIPE FIFOs for each other task it speaks to, as shown in Figure 29 on page 53, one for sending and one for receiving messages. Whenever one task inserts a set of bytes on its send FIFO, it expects that the destination task receives the same bytes in the same order.

The PIPE layer hides the physical position of tasks. Once the library environment is set up and all the PIPE FIFOs created, the MPCI layer is not aware of whether it is using UDP or direct access to the switch adapter to send the messages, or if the tasks are on the same node or on a different machine. MPCI just continuously uses the transport layer given by PIPE structures.

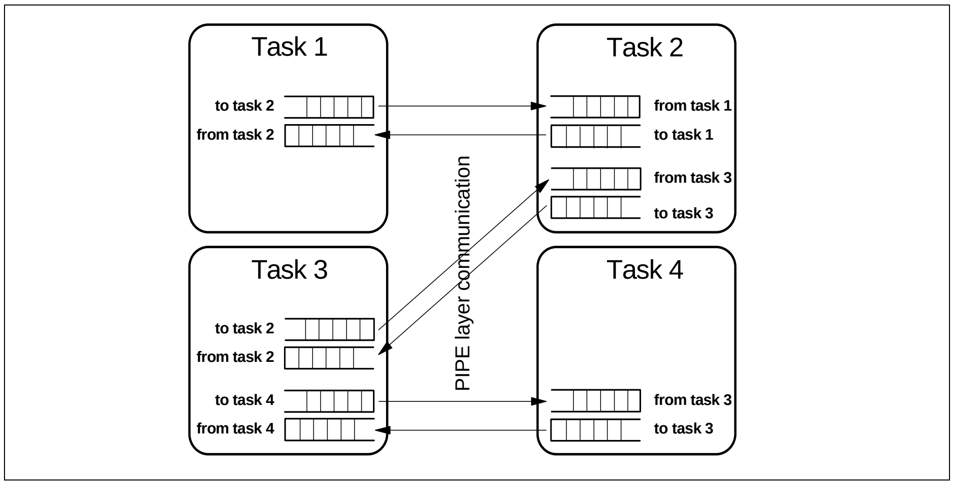


Figure 29. PIPE FIFOs among Tasks

When the user application sends a message, it specifies the buffer where the message lies, the destination task and several sending options. MPCI selects the appropriate PIPE structure and inserts data into it.

If the user message is smaller than *MP_EAGER_LIMIT* (4 KB by default), a message header and the actual message data are inserted into the PIPE FIFO. If the user message is bigger than MP_EAGER_LIMIT, only a header is created and inserted into the PIPE FIFO. The message is not transmitted until the receiving task is ready to use it. When the receiver requests the message, it is sent.

If the message is contiguous and 32 KB or larger, the user buffer is directly read by the PIPE layer when sending it into the switch network, avoiding the penalty of a memory-to-memory copy into the PIPE FIFO.

On the receiving end, the MPCI layer continuously looks into all the PIPE FIFOs for the task. It searches the header sent by the corresponding MPCI layer to detect the beginning and the length of a message. It then waits for the PIPE layer to fill the receiving buffer with all the data sent. When the message is in the FIFO, MPCI extracts it. If the task has requested the message, the data is directly copied in the user’s buffer; otherwise, it is copied in an *Early Arrival Table* while waiting for the task to request it.

If the header is received for a message whose size is greater than the MP_EAGER_LIMIT, MPCI first checks if the task has already requested it. If the message has been requested, MPCI sends back a message asking for data transfer, otherwise it stores the header, waiting for the task to make the get call. This procedure avoids filling MPCI's Early Arrival Table with many big messages.

Once the message is extracted from a PIPE, the space is freed and the PIPE can accept other bytes from the sending task.

4.4 PIPE Layer

The PIPE layer provides a transport layer that is:

- Byte-stream oriented
- Reliable

Each PIPE structure may be seen as a FIFO where data is inserted by one task and removed by the other task with no data loss during transmission. It is the PIPE layer that takes care of using the correct transport layer (UDP or direct access to switch adapter) and of performing recovery actions in case of data loss during data transfer.

The PIPE layer takes the bytes from a send FIFO, assembles them into one or more packets that fit the physical transport layer and sends them to the destination task. When MPCI asks to transmit data with a size greater than the MP_EAGER_LIMIT, the PIPE layer creates the packets directly from the user buffer.

All packets have a sequence number and a base address that are used to reconstruct the byte stream when packets are received out of order.

The receiving PIPE acknowledges the delivery of the packets and implements packet level token flow control. If a packet has been dropped or otherwise unsuccessfully received, the PIPE layer also retransmits that packet.

Each task has one receive and one send PIPE for each task it exchanges messages with. The corresponding sending and receiving PIPEs on the two tasks synchronize themselves in order to provide a continuous data flow to the MPCI layer.

The data packets created by the PIPE layer are transmitted to the receiving PIPE either using UDP or through direct access to the switch adapter. If UDP/IP is used as the transport layer, a switch to kernel mode is required to execute the IP code both on the sending and on the receiving side to make packets flow from sender to receiver.

All interactions between the PIPE layer and the switch adapter are carried out through the Hardware Abstraction Layer (HAL) library. The library provides access to the adapter's resources hiding their physical details, thus making improvements on the adapter hardware and its interface to the system easier to introduce.

Direct access to the switch adapter is the most efficient way to communicate with the SP Switch. Each task has one output and one input DMA buffer that is shared among all PIPEs. Packets are copied from each send PIPE of the task to the output DMA buffer, where the adapter gets them using DMA. The adapter sends the packets into the switch network as described in 3.6.5, “Data Flow” on page 41. On the receiving side, the adapter moves the packets using DMA into the destination task's input DMA buffer, from where it is copied into the correct receive PIPE. In Figure 30 you can see a representation of data flow.

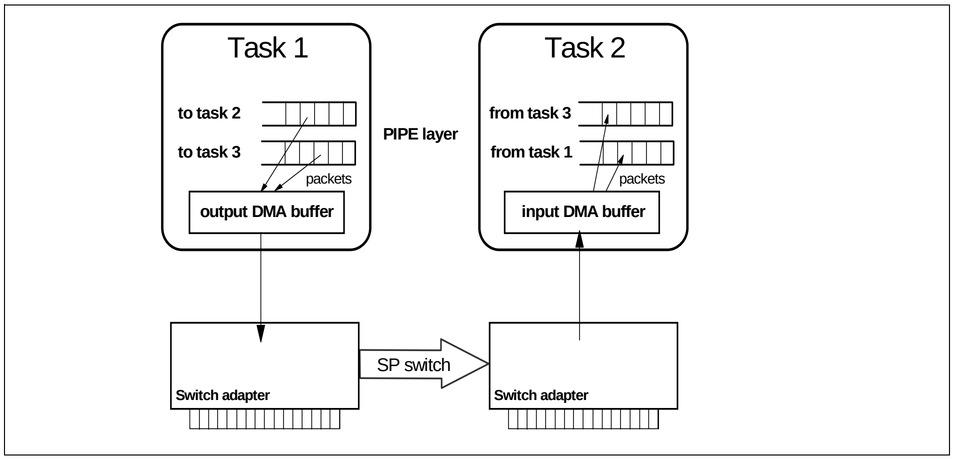


Figure 30. PIPE-to-Adapter Data Flow

Messages already requested by the receiving task and larger than MP_EAGER_LIMIT are put by the PIPE layer directly into the task´s buffer, avoiding the intermediate copy to the PIPE FIFO.

Packets from all send PIPEs are evenly copied into the output DMA buffer. If there is no more space in the output DMA buffer, the PIPE layer stops transmitting until space is freed.

The packet structure, when direct access to the switch adapter is used, is described in Figure 31 on page 56. The first part of the packet contains the destination task logical identifier and a route indicator that tells which of the four possible routes to the destination should be used. The PIPE layer does not know which are the physical routes, but it is aware that four of them are always available: for each packet it sends to the adapter, a different route number (0, 1, 2, or 3) is selected to distribute the load in the SP Switch network. The adapter manages to send the packet to the correct node using the destination field and the route number.

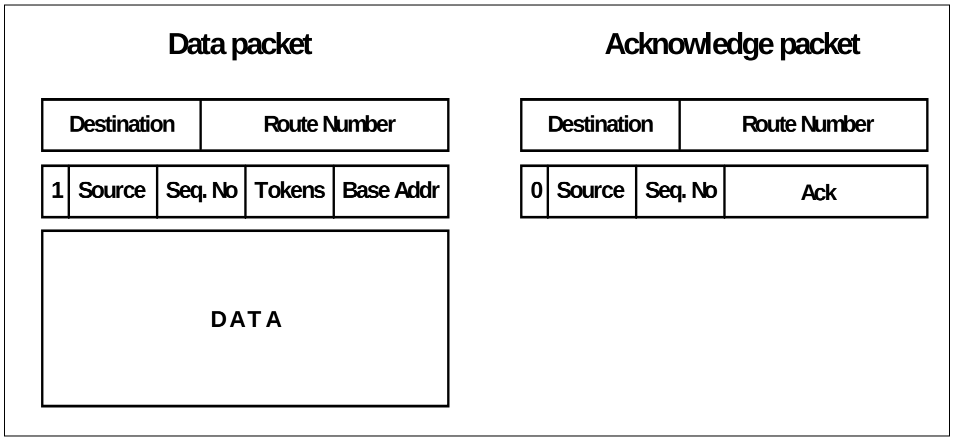


Figure 31. Packet Created by PIPE Layer

The second part of the header is used for synchronization between PIPE layers on different tasks. The adapter sends packets to the DMA buffer of the destination task and the source field is used by the PIPE layer to identify the receive FIFO to which the data belongs. Then the sequence number and base address are used to define in which position of the FIFO the packet's content has to be put, creating the FIFO abstraction for the MPCI layer.

The *token* field is used for flow control. It contains the number of packets the sender task can receive from the destination task. In this way each PIPE buffer is aware of how many packets can be sent, thereby avoiding receiving buffer overflows.

When packets are removed by MPCI from the input buffer, the PIPE creates an acknowledge packet for the received sequence number that is sent back to the sender. When the sender receives the acknowledge, it frees the space held for the sent packets and accepts more data from MPCI.

If sent packets are not acknowledged within a given period of time (less than one second), all outstanding packets are retransmitted and will be placed in the correct place in the byte stream of the receiver.

4.5 Low-Level Application Programming (LAPI) Layer

User Space communication helps to achieve improved performance by avoiding the system calls, context switch and extra copy overhead associated with communication interfaces that have a path through the kernel. However there is overhead associated with the User Space message passing implementation that impacts performance. For instance, in order to satisfy the semantics required of the MPI and MPL specifications, the implementation often needs to keep multiple copies of the data.

Another communication interface, the Low-Level Application Programming Interface (LAPI), which is part of the SP software, addresses some of the performance concerns of MPI and MPL, and adds new features.

LAPI is an asynchronous communication mechanism intended to provide users the flexibility to write parallel programs with dynamic and unpredictable communication patterns. It is deemed to be an efficient interface, with low latency and high bandwidth, and includes data communication as well as synchronization and ordering primitives.

The design object of LAPI includes:

- Performance: eliminate some of the protocol overhead required for implementing MPI and MPL
- Flexibility: provide an alternate programming model for the SP
- Extensibility: provide users of LAPI the ability to extend its functions

LAPI is also intended for exploitation by subsystem developers who want to extract performance from the SP system. The design of LAPI is based on the Remote Memory Copy (RMC) model. RMC is a one-sided programming paradigm quite similar to the load/store model of shared memory programming. The model is one-sided in the sense that the target process does not have to take explicit action for the active message to complete (for example, no explicit read). The RMC interface eases some of the difficulties of the send/receive model used in message passing parallel programs. As part of LAPI, an *active message* style interface is provided to enable the user to easily add functions using the LAPI infrastructure.

An active message includes the address of a user-specified handler. When the active message arrives at the target task, the specified handler is invoked and executed in the address space of the target. We use the term *origin* to denote the task that initiates an LAPI operation, and the term *target* to denote the task whose address space is accessed by the LAPI operation.

Optionally, the active message may also include some data from the originating process. Buffering, beyond what is required for network transport, is not required because storage for arriving data (if any) is specified in the active message, or is provided by the invoked handler.

When the active message brings data from the originating process, LAPI requires the handler to be written as two separate routines:

- A *header_handler* function, which is the function that is specified in the active message call. It is called when the message first arrives at the target process (actually the first packet of the message arrives), and it provides the LAPI dispatcher (the LAPI layer that deals with the arrival of messages and the invocation of handlers) with:
  - An address where the arrival data of the message must be copied
  - The address of the optional *completion handler*
- A *completion handler* which is called after the whole message has been received, meaning that all the packets of the messages have reached the target process.

The separation of the handler into a header handler and completion handler in the active message infrastructure allows multiple independent streams of messages to be sent and received simultaneously within an LAPI context.

LAPI supports messages that can be larger than the size supported by the underlying network layer. This implies that data sent using an active message call will arrive in multiple network packets, possibly out of order. This places some requirement on how the handler is written.

When the first packet is received, the LAPI dispatcher receives it, identifies it as a new message, and reads the header handler to be called. The handler returns a buffer pointer where incoming data is to be copied and the address of the completion handler. The LAPI library then moves all the received packets to the specified buffer and when the whole message is received, the completion handler is executed.

Other characteristics of the LAPI interface are:

- Basic data transfer operations are memory-to-memory copy operations that transfer data from one virtual address space to another virtual address space.
- The operations are semantically unilateral. One process initiates it, and the completion of the operation does not require any other process to take some complementary action. This is unlike a send/receive where a send

requires a complementary receive with matching parameters to be posted for completion, and vice versa.

- The initiating process specifies the virtual address of both the source and destination of the data, unlike send/receive where each side specifies the address in its own address space. This means that the process must know the virtual address of all the objects it will access in address spaces of the processes it will communicate with.
- Since data transfer is unilateral, and no synchronization between the two processes is implied, additional primitives are required for explicit process synchronization when that is necessary for program correctness.

The data operations that LAPI supports are Put and Get. They are unilateral operations. They transfer data between the initiating task and another task specified by the initiator. Put transfers data from the local virtual memory of the initiating task to the virtual memory of the other task. Get transfers the data in the opposite direction. Concurrent use of a buffer which will be modified by multiple asynchronous operations has to be controlled by the user.

The Put function in LAPI enables the *push* mode of communication: write data into the address space of another task in the parallel job. The Get function enables the *pull* operation: read from the address space of another task in the parallel job.

4.6 IP Layer

The SP system was created to provide a powerful parallel processing environment and initially did not use IP to provide communication over the switch. IP was then introduced to make applications that rely on the TCP/IP protocol work on the SP system and to provide a communication layer for tasks that exchange messages with machines not in the SP system.

The IP protocol is handled by AIX kernel extensions and it manages the interaction with all the interfaces present on the node, including the switch interface, as can be seen in Figure 32 on page 60.

Each IP datagram that a node wants to send to a network is passed to the corresponding *interface layer*. There are different layers, depending on the network type, each of which knows exactly how to manage the I/O with the physical medium. For example, if_ls *(*originally called the *light speed interface)* is responsible for the SP Switch network.

What is special for the switch is that it is not the device driver that manages the data transfer with the adapter, but it is the adapter itself that directly reads and writes to node memory.

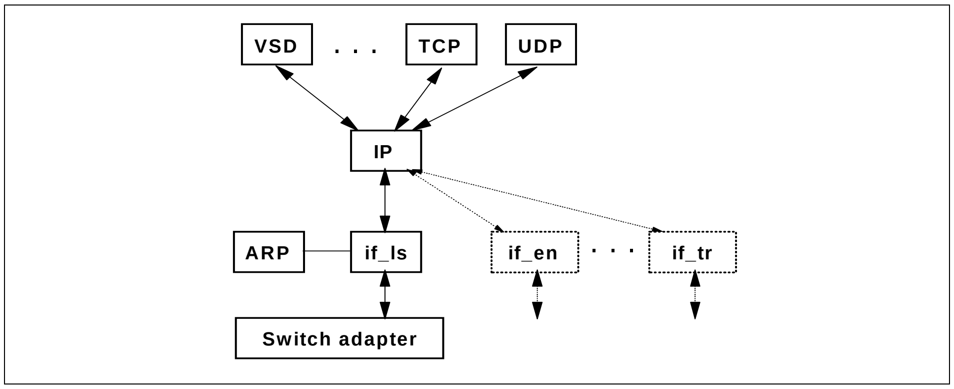


Figure 32. IP Kernel Extension

4.6.1 Address Resolution Protocol (ARP)

The destination of an IP datagram is defined by the IP address, but the switch network uses *switch node numbers* to identify destinations. The if_ls interface layer then has to translate from the IP address to a switch node number in order for data to get passed through the switch network.

The most common method used in the TCP/IP protocol stack to map IP addresses into network identifiers is ARP. The ARP protocol maintains an *ARP table* with the mappings. If an entry does not exist for a given IP address, then the protocol issues an ARP request: a broadcast on the network asking the node with the requested IP address to reply giving its network identifier. The node with the requested IP address responds back. ARP table entries are deleted periodically (the default is 20 minutes).

The SP system uses the ARP protocol. The SP Switch does not support broadcast but only point-to-point communications. In order to support ARP, the interface layer has to translate an ARP request into several messages, one for each node in the switch network.

In order to reduce network bandwidth usage due to ARP packets, when if_ls is configured in a node, it “broadcasts” a gratuitous ARP response packet. In this manner, the ARP tables for all nodes get the mapping for the incoming node.

If ARP is not used, a special IP numbering scheme must be used (section 5.4, “Planning the Switch IP Network” on page 81 for more details). If there are no special application requirements (for example, the HACMP environment requires ARP) and a more flexible IP address assignment is not needed, ARP can be turned off for all the switch adapters in the network. In this configuration, it is the *fault service daemon* of each node that provides a mapping function to each interface layer, translating the IP address into the destination node number. Note: This function can be used only by the interface layer.

4.6.2 Send Data Flow

An IP datagram is copied to the kernel using several data structures, the number depending on the size of the data. The smallest element is the *mbuf* structure that can contain up to 228 bytes of data. If the datagram is larger, the mbuf structure can contain a pointer to an *mcluster* structure, or the datagram can be put in a linked list of mbufs, each containing an mcluster (see Figure 33 for an example).

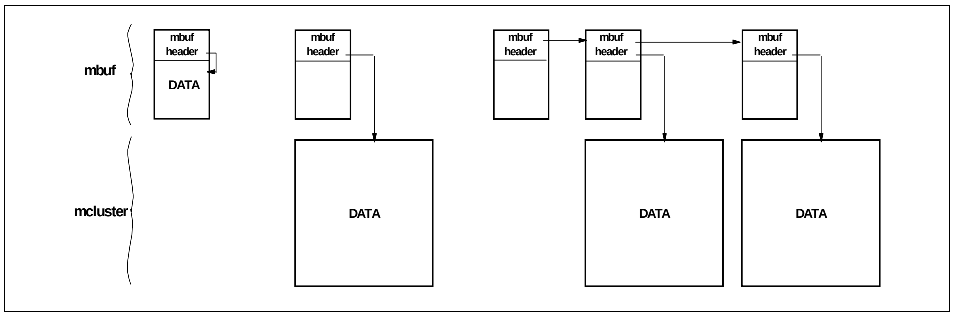


Figure 33. IP Kernel Data Structures

The interface layer works directly with the switch adapter. It can write some variables to the adapter’s SRAM, and some kernel memory segments are read and written by the adapter. Using these shared areas, the kernel and the adapter can synchronize themselves and exchange data.

A send FIFO (sFIFO) structure is part of the interface layer. It has 512 entries of 256 bytes each. If an outgoing IP datagram consists of only one mbuf, it is copied to the sFIFO and the mbuf is freed. Each entry of the sFIFO contains a header that tells to which physical node the data has to be sent.

The switch adapter microcode copies the data from the sFIFO using DMA when it is ready to send IP datagrams. The data in the sFIFO is then put in a single switch packet and sent to the destination.

When sending a larger datagram, too much overhead would occur if it had to be split into smaller pieces and copied into the sFIFO. To improve performance, a *Send Pool* buffer is used in pinned memory that is shared with the adapter. The Send Pool's size can be from 512 KB to 16 MB.

For IP communications over the switch, the kernel allocates mclusters from the Send Pool instead of the standard pool. The cluster size can be 4, 8, 16, 32 or 64 KB (the switch MTU is 64 KB). The interface layer then receives an mbuf structure where all the mclusters are already in a memory zone that can be read by the adapter, so it creates a new entry in the sFIFO containing pointer(s) to the Send Pool.

The adapter reads the sFIFO, detects that there is more data on the Send Pool and starts reading it using DMA. In this case the IP datagram is larger than the switch packet: the adapter's microcode splits the datagram while it is being read from the Send Pool, creating switch packets whose headers contain an offset within the mcluster. When all the data is inserted into the switch, the space in the pool is freed.

The two cases are described in Figure 34.

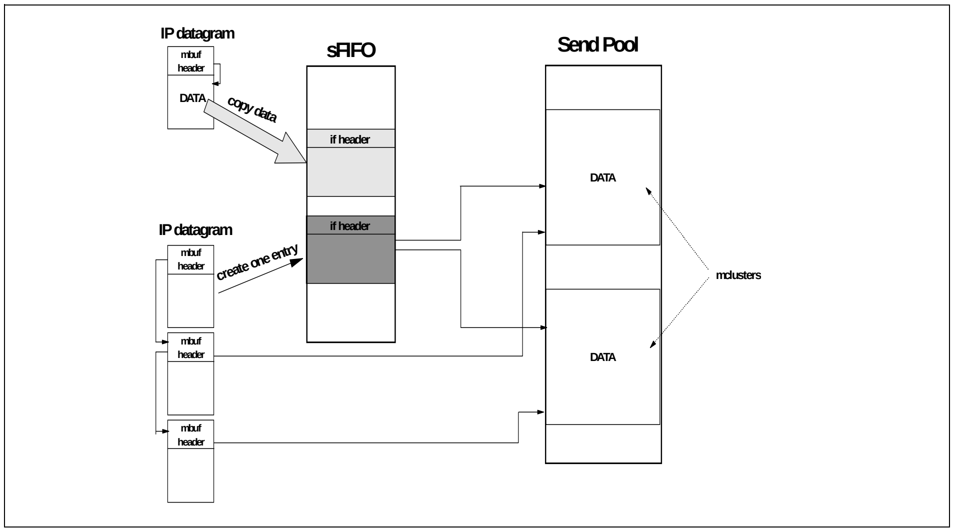


Figure 34. IP Send FIFO

If no space is available in the Send Pool, the kernel allocates an mcluster from the standard pool, where the maximum size of an mcluster is 16 KB. When the datagram has to be sent, the data has to be copied into an area that can be read by the adapter. If possible, the Send Pool is used, otherwise a Reserved Send Pool is used. Memory-to-memory copy cannot be avoided in this case.

4.6.3 Receive Data Flow

The receive data flow is similar to the send data flow. In the kernel there is a receive FIFO (rFIFO) and several internal data structures, available also to the adapter's microcode, that make possible the transfer of data from the switch network to the node.

When a packet containing a complete IP datagram arrives at the adapter, it is added using DMA into the rFIFO. The interface layer reads from the rFIFO, copies the data into an mbuf, and then gives the mbuf to the IP layer for the usual TCP/IP handling.

When the IP datagrams sent are larger than 228 bytes, a *Receive Pool* pinned memory segment shared with the adapter is used. The microcode on the adapter reads the packet, looks into the header for the datagram's size, allocates a buffer on the Receive Buffer, and finally transfers the data into the correct position of the buffer using DMA. When the last packet is received, the microcode adds an entry in the rFIFO that points to the data in the Receive Buffer. The space in the Receive Pool is released after the receiving application reads the data into its memory space.

The if_ls interface is notified of the presence of new data in the rFIFO using interrupts. The number of IP packets received that trigger an interrupt varies depending on the load, in order to reduce the total number of interrupts and increase the performance.

If a packet that is part of a large IP datagram is lost in the switch network, the buffer will never be filled completely. A timer is set up upon the receipt of the first packet and if it expires before the entire datagram is received, the partial datagram is dropped and the space in the Receive Buffer is freed. If_ls, as any other IP interface layer, takes no recovery action.

The use of a Receive Pool improves the overall performance of the system since the data coming from the adapter is not copied into any other intermediate buffer before being dispatched to the user application. The only drawback is that applications may not be fast enough to release the buffer quickly and the pool may fill up.

In order to avoid Receive Pool saturation, a *high-water mark* is present. When the pool fills up to the water mark, all new buffers allocated by the microcode are copied into a kernel mcluster as soon as they are filled, and then the buffer in the pool is freed. There is a performance degradation due to the memory-to-memory copy, but this prevents the pool from being filled up.

4.7 Fault Service Daemon

The fault service daemon is a special User Space process that monitors and configures the SP Switch. The same code runs on each SP node, but not all of its functions are active on all nodes. Each daemon has one of the following personalities:

- Primary
- Primary backup
- Secondary

The node on which the daemon is running is also referred to differently depending on the personality of the daemon: primary node, primary backup node, secondary node.

The *primary* daemon is the one that is in charge of configuring and monitoring the SP Switch’s behavior. Exactly one primary daemon is present in every SP Switch partition. Its fundamental functions are:

- Initialization of SP Switch
- Recovery actions for SP Switch faults
- Generation and update of route tables

The *primary backup* daemon is a secondary daemon, but it also has the goal of checking that the primary daemon is up and running. If it detects that the primary daemon is no longer functional, it changes its personality to primary and chooses the new primary backup from among the available secondary daemons.

The *secondary daemons* only take care of the route table generation and update, in response to the data passed by the primary daemon.

All the fault daemons are responsible for local switch adapter configuration, initialization and monitoring. They download routing information to the adapter’s SRAM, start the microcode and handle adapter faults.

In the following paragraphs we introduce the basic concepts of the fault service daemon's functions that are described in more detail in 7.3.3, "Phase One of Switch Initialization" on page 124 and in the following sections.

4.7.1 Initialization of SP Switch

In order to operate, the SP Switch requires that all its hardware components are checked for anomalies and configured with proper values. The primary fault service daemon is able to exchange information with each single switch chip in the switch network and uses all data collected to recover any faults that may occur.

A topology file is provided by the system administrator to the daemon describing the overall switch network. Based on this overall description of the network, the primary daemon activates its *Worm* code, which scans the entire network looking for any anomalies. Service packets are sent to each switch chip and node, and every link specified in the topology file is traversed. At the end of this process, faulty components (such as links, switch chips and switch adapters) are identified.

All the switch chips receive initialization packets that instruct them on how to manage events and what kind of operation they are enabled to perform. Individual receiving and sending ports are enabled or disabled according to the network configuration and fault isolation. Each chip receives two different paths along the switch network to be used by the error/status notifications generated by the chip.

After the network scan is completed, all the fault service daemons compute the routes from their node to all the other nodes, using the information collected by the primary daemon.

4.7.2 Network Recovery Actions

All notifications received by the primary daemon from the switch chips are used to handle network faults. Normally the switch chips try to automatically recover from errors, but if they are not successful, the daemon is notified so it can isolate the fault.

Depending on the events received, the daemon may decide to isolate a network link or a switch chip. It sends service packets to the appropriate switch chips in order to disable sending and receiving ports. All secondary daemons are notified of such configuration changes so they may update their view of the network topology.

The primary daemon does not completely rely on asynchronous reports from switch chips. It performs a periodical scan of the network every two minutes, requesting the status of all switch chips and the primary backup node. If any component is found with a bad status, recovery actions are started that may end in detecting a faulty component.

During the periodic scan, the primary daemon may detect that a node previously removed from the network because it was not responding has started working properly; for example, a node may have been turned off and then back on. The node is then automatically inserted (automatic Eunfence) into the network.

4.7.3 Generation and Update of Routing Tables

The fault service daemon is responsible for configuring the local switch adapter routing tables. It has to compute four paths to each other node in the switch network and to download all this routing information to the adapter.

Each daemon uses the same network topology file to create the routing information. The primary daemon is in charge of updating the topology view of all the secondary daemons. When an update is received, all daemons compute a new set of paths to all nodes and update the adapter's internal tables.

Routing table updates are done at the initialization of the switch and also each time the primary daemon enables or disables a network component after a switch fault or a specific request of the system administrator.

4.7.4 Administrator Commands

A system administrator can issue a set of commands (the so-called *E-commands*) to alter the network topology and to perform maintenance operations on the SP Switch.

Many of commands are directed to the primary daemon that physically performs the required action. It sends appropriate service packets to the involved switch chips and notifies all other daemons of any topology changes, so they can update their routing tables.

4.7.5 Fault Daemon Recovery

The primary fault daemon is a vital part in the SP Switch software. If it does not work properly, no switch fault will be handled and the system administrator cannot modify the switch configuration.

To avoid all these problems, a backup daemon (the *primary backup daemo*n) is always present which behaves exactly like all other secondary daemons, but also checks the health of primary daemon.

The primary backup listens to all the network scans that the primary performs. For each scan it starts a 2.5 minute timer. If no activity from the primary is received before the timer expires, another 2.5 minute timer is activated. If it expires without the scan detection, the primary daemon is considered faulty.

Upon detection of a primary problem, the primary backup assumes the personality of a primary daemon. The node that was running the former primary daemon assumes the personality of a secondary daemon to prevent the danger of the presence of two primary daemons. All the switch chips and adapters are then reconfigured to report all their events to the new daemon, providing the routing information to the primary node. Finally, the daemon selects a new primary backup.

4.8 Kernel Extension

The functions of the CSS kernel extension code include:

- Acting as the switch code’s second level interrupt handler, waking up the fault service daemon to perform fault handling and daemon commands, such as *Estart* and *Eunfence*
- Setting up shared memory DMA to allow both kernel and User Space clients to interact with the switch adapter
- Providing run-time support for the User Space requirements that have to be done in kernel space
- Handshaking between the fault service daemon and User Space jobs

4.8.1 Initialization

The CSS kernel extension is loaded during system boot time by the switch adapter configuration method. Its initialization is done in two phases. In the first initialization phase, at system boot time, AIX recognizes the kernel extension. In the second initialization phase, the fault service daemon executes DMA setup code that has to be run by a process.

4.8.2 Client Windows

The switch adapter support code recognizes three types of clients: User Space jobs, the IP driver, and the switch code’s Service Library. Each client is granted access to certain parts of the adapter address space (a *window*) in order to share some status variables (not data), such as the number of packets ready to be sent to the adapter or the number of packets present on the receive queue of the client.

Clients have a local data structure that has to be synchronized with the data present on the adapter. Data structures are mostly updated by clients. These updates are made using kernel services, system calls and, in some cases, direct reads and writes to the adapter’s address space.

4.8.3 User Space Client Initialization

The User Space client requires access to the adapter calling a library function that uses the Kernel Extension to initialize the client’s environment. A window and job identifier are used during initialization and a complete description of the job must have been provided by a Job Manager.

The Kernel Extension validates the values provided by the client to be sure that the window is not already in use by a running job, and checks the job identifier for consistency.

If data from the client is valid, a shared memory segment is allocated for DMA data transfers to and from the adapter. It contains the send and receive FIFOs of the client. The memory is pinned and attached to the User Space’s address space. All necessary steps to configure DMA operations are performed.

4.8.4 Second Level Interrupt Handlers

Interrupts received by the device driver’s first level interrupt handler are redirected to one of three second level interrupt handlers: one for IP, one for User Space and one for the fault daemon. They all reside in the Kernel Extension except for the interrupt handler (IP) that resides in the IP interface layer.

The adapter generates interrupts when packets arrive or when switch faults occur. The Kernel Extension is configured to signal the correct process that data has arrived, or to activate the fault daemon for fault handling.

4.8.5 Switch Fault Handling

When a switch fault occurs, the device driver passes control to the kernel extension, which in turn wakes up the fault service daemon. The daemon receives the switch error packet, and passes the packet to the switch recovery code. This code parses the packet and takes appropriate action to recover and/or reconfigure.

Attempted recovery requires the resetting of switch ports and the clearing of errors. Reconfiguration occurs if recovery fails, and may cause links and chips to be disabled. If a node link or chip is disabled, the corresponding node gets fenced. This incident usually causes the corresponding jobs to terminate.

4.8.6 Client DMA Buffer Management

A DMA buffer for User Space tasks is set up as AIX shared memory and is created with read and write permission by the owner only. When a client requests DMA access, the Kernel Extension makes the client task the owner of the shared memory. When the task exits, either normally or with an error, the shared memory is removed.

The Service Library, running as a user client, allocates the DMA buffer from the fault service daemon process heap.

IP allocates the DMA buffer from the kernel pinned heap.

4.8.7 Interaction with the Adapter

There are two tables that are downloaded to the adapter using kernel functions.

The *Switch Routing Table* identifies the physical routes to each node of the system. This is generated by the fault service daemon code and downloaded to the switch adapter's microcode as part of switch fault handling or the microcode loading process. For each node four different paths are provided that are used in a round-robin fashion by User Space and IP packets. After each switch fault, the Kernel Extension is used to update the adapter's partition information.

The switch table, also called the Job Switch Resource Table (JSRT) or *partition table*, maps logical task destination IDs to switch node numbers and window IDs. The JSRT is created by the LoadLeveler or the Resource Manager from Parallel Environment requests of nodes for parallel jobs. When a parallel job starts, the Kernel Extension is invoked to load the JSRT into the adapter.

4.9 Device Driver

The device driver interface provides the functions needed to access the switch adapter resources. It is basically used by the CSS Kernel Extension.

Unlike many other communication adapter device drivers, it does not directly support sending and receiving messages.

The kernel uses the device driver to register the routines that have to be called whenever the adapter issues an interrupt to signal that a DMA operation has completed, that is, that new packets are available. Depending on the window involved, a separate routine is called.

Several *ioctl*s are supported to read and write data from and to the adapter's registers and SRAM. They are used to update information needed by the adapter to manage the incoming and outgoing switch packets, and to interact with the adapter's microcode.

Part 2. SP Switch Operation

Chapter 5. Planning for the SP Switch

In this chapter, we describe some basic areas you need to consider when planning for the switch of your SP system. The tips and hints discussed here are not meant to be a replacement for the two comprehensive planning guides *SP: Planning, Volume 1, Hardware and Physical Environment*, GA22-7280 and *SP: Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281. Consultation with these planning guides is strongly recommended.

5.1 Choosing a Switch

Current SP installations may have one of two types of switch, depending on their age: newer systems have the SP Switch, while the oldest may still use the High Performance Switch (HiPS). For both, there are two models:

- An 8-port model without inter-switch connection
- A 16-port model with inter-switch connection

The High Performance Switch technology predates the SP Switch series. Currently, HiPS networks are not available except for maintenance reasons. If you plan to add any of the nodes introduced since PSSP 2.4, or to install or migrate your Control Workstation or any node to PSSP 3.1, you must convert all your switch boards to the SP Switch.

Although levels of PSSP prior to 3.1 support both the SP Switch and HiPS, the two switch networks are not physically compatible and cannot be mixed within an SP system, not even in a partitioned SP system.

If you expect your system to eventually have more than eight nodes or to have more than two partitions, or you plan to connect it with another frame, you must not choose SP Switch-8. Its internal configuration does not provide the scalability and flexibility of the full 16-port switch.

5.2 Switch Node Numbering

In this section we overview some of the numbering schemes used in an SP system. Refer to *SP: Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281, for a more complete explanation.

5.2.1 Frame Numbers and Switch Numbers

All frames in the SP system are numbered according to the serial port to which they are connected. The frame connected to device tty0 is frame number 1, the one connected to tty1 is frame 2, and so on.

The switch boards are also numbered and they get their number from the order they appear in the sequence of frames. That is, if all frames are switched frames (frames with a switch board) the switch number of the switch board in a frame will be the same as the frame's number. But if the system has non-switched frames (which are frames without a switch board) the frame and switch numbers will not necessarily match, as shown in Figure 2.

Table 2. Frame Numbers and Switch Numbers

Serial Port	Switched Frame?	Frame Number	Switch Number
tty0	yes	1	1
tty1	no	2	—
tty2	no	3	—
tty3	yes	4	2

The two non-switched frames in the preceding table are *expansion frames* for frame 1. That is, the nodes in frame 2 and frame 3 are connected to switch board number 1. There are two aspects about expansion frames that should be mentioned:

1. Expansion frames have to follow the switched frame they are complementing.
2. A switched frame and its expansion frames cannot have any combination of nodes.

Intermediate switch boards are numbered using a different rule. Switch-only frames (frames with intermediate switch boards) have to be the last frames in the system. And the switch boards in them are numbered consecutively, starting from number 1001, according to their drawer in the frame.

5.2.2 Slot Numbers and Node Numbers

A node frame has 16 slots available to nodes (numbered from 1 to 16) and 1 slot (number 17) for the switch board. A switch-only frame has 8 slots.

The *slot number* of a thin node is clearly the number of the slot it occupies. For wide and high nodes, the slot number is the number of the lowest slot the node occupies.

A node's *node number* is defined by the number of the frame where it is located and by its slot number, as shown in Figure 35, where Frame 1 contains wide nodes, and Frames 2 and 3 contain high nodes.

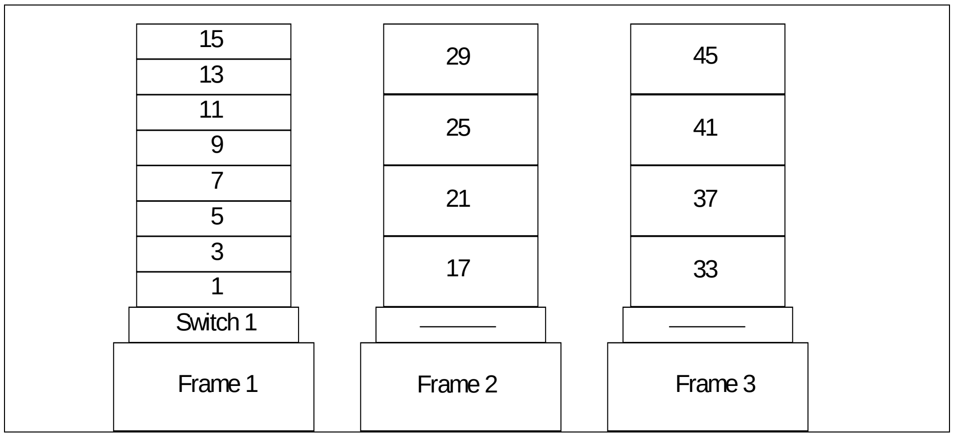


Figure 35. Node Numbers

When first configuring a multi-frame system, if you expect to eventually expand your system through expansion frames, you should consider reserving the serial ports and frame numbers these frames will eventually occupy. Otherwise, when the expansion frames are installed, all existing nodes in frames placed after the expansion frames will be renumbered, and you would need to reconfigure them.

5.2.3 Switch Node Numbers

A node is normally connected to the switch according to its slot number. In Figure 36 on page 76 we show a complete switch board with both internal and external connections. On the figure's right side are the node switch chips, whose ports connect to the nodes. These switch ports are labeled N1 to N16, and a node uses the switch port that corresponds to its slot number. For example, the node in slot 3 uses connection N3, that is, it is connected to the switch through jack J26.

.

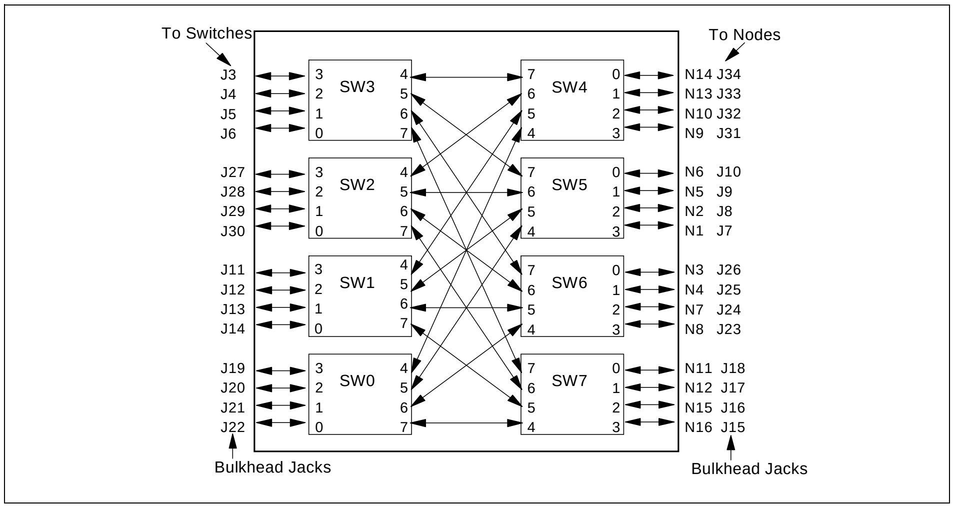


Figure 36. Chip Interconnection and Slot Assignment

SP Switch-8 has only 8 ports and the connection rule is different from that of the standard SP Switch. Nodes are connected to the switch board in the order they are placed in the frame, independently of their slot number.

A node's *switch node number* or *switch port number* is an identification of its connection to the switch fabric. A switch node number is defined by the number of the switch board and the switch port to which the node is connected. But, in contrast to node numbers, switch node numbers are zero-relative. For example, assuming all frames are switched:

- A node connected to slot 1 in frame 1 has switch node number 0.
- A node connected to slot 10 in frame 1 has switch node number 9.
- A node connected to slot 1 in frame 2 has switch node number 16.
- A node connected to slot 6 in frame 3 has switch node number 37.

Note that in the example, the switch node number of the nodes is always the node number minus one. This will always be the case, *unless* the system has expansion frames. Without going into the details of how nodes in expansion frames are connected to the switch board, we show in Figure 37 on page 77 the switch node numbers of the system previously shown in Figure 35 on page 75. Compare the node numbers with switch node numbers.

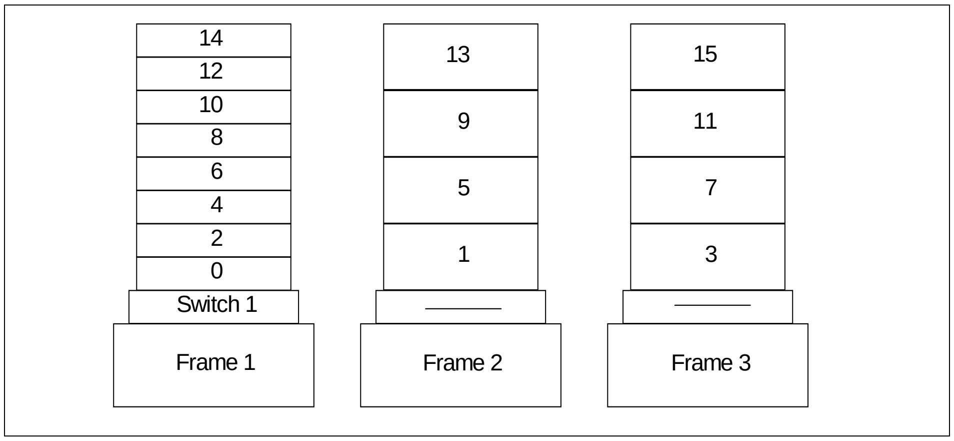


Figure 37. Switch Node Numbers

5.2.4 Inter-Switch Connection Considerations

As shown in Figure 36 on page 76, each NSB has 16 ports for connecting to other switches. For example, if a system has 3 switch boards, any board can have 8 ports connected to each of the other two.

In configurations that require up to five switch boards, all the boards can still be connected directly to each other; there are 4 connections between each pair of boards. We call this configuration a single-stage environment.

Direct connection of five frames thus provides only four separate paths between two nodes, with no backup path. This means that any problem on a link on the switch will cause the packet traffic to flow on one of the other three already used paths, causing a network performance degradation. In this situation, it may be better to introduce an Intermediate Switch Board (ISB), which provides a higher number of possible routes and makes the transition to a 6-switch board system easier.

With more than five NSBs, ISBs are required to provide at least four paths for every two nodes. ISBs are installed in a switch frame in which you can include only switch boards. When ISBs are installed, NSBs are no longer directly connected. NSBs are connected only through the second level of staging provided by the ISBs. This results in 4 ISBs being included for a 6-NSB system, and 8 ISBs for a 12-NSB system.

It is highly recommended to use ISBs in case of more than three switch boards. This makes more backup data paths available and it is easier to expand the system. When no ISBs are used, a complete switch board recabling is needed in order to add a new switch board, and switch communication is stopped while the recabling is taking place. When ISBs are used, introduction of a switch only requires additional connections to the ISBs from that new switch.

5.3 External Connection Plans

Some SP nodes are not physically inserted in an SP frame. They are called Extension nodes and are divided into two types:

Attention

Originally, GRF was available as an extension node, so documents published before PSSP3.1 use the term extension node only for GRF.

- SP Switch router (dependent node)
- Self-framed node

There are two models of SP Switch router product, GRF-400 and GRF-1600, while RS/6000 Enterprise Server Model S70 is currently the only self-framed node.

5.3.1 SP Switch Router

GRF offers high-speed access to and from other systems. If you need many network resources for an SP system, you should choose GRF as a router. The use of RS/6000 nodes as routers is limited by the number of available I/O slots and is usually expensive, providing a low price/performance ratio.

A valid unused node switch port in the SP system is required to attach the SP Switch router to an SP switch, that is, a switch port that meets the rules for configuring frames and switches. You do not put your GRF in the frame, but you need a reserved space on the switch.

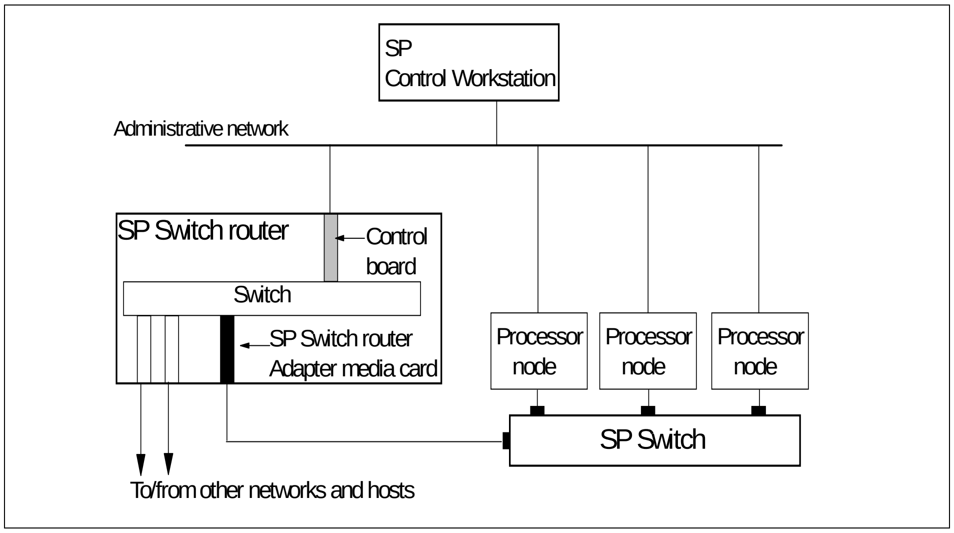


Figure 38. SP Switch Router Connection to the SP System

To attach an extension node to an SP switch, configuration information must be specified on the Control Workstation. Communication of switch configuration information between the Control Workstation and the SP Switch Router takes place over the SP administrative Ethernet.

A GRF supports multiple connections to a switch or to multiple switches. With this function, you can plan more flexible configurations, as follows:

- Communication among partitions via GRF
- Backup path to GRF (multiple paths to one partition)
- Communication between switch boards using GRF

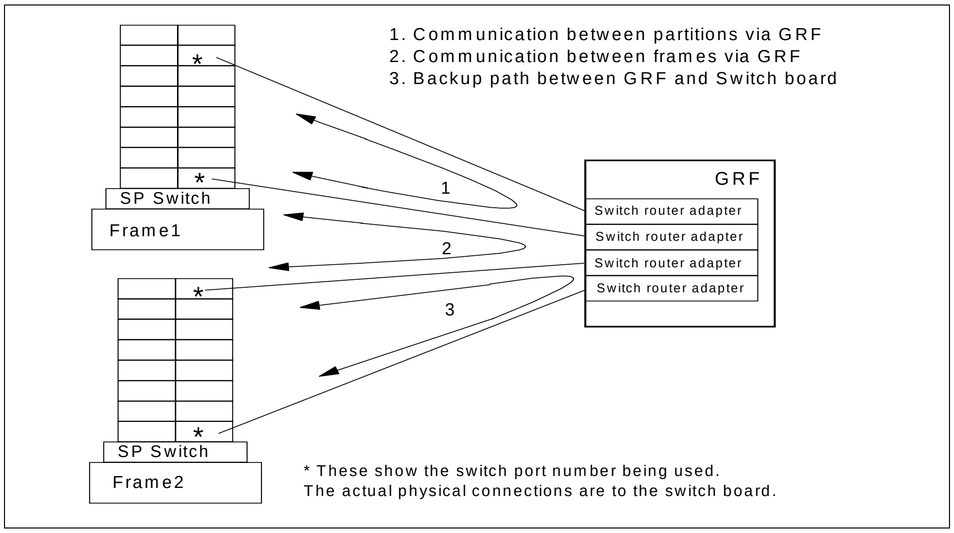

1. Communication between partitions via GRF
2. Communication between frames via GRF
3. Backup path between GRF and Switch board
GRF
Switch router adapter
Switch router adapter
Switch router adapter
Switch router adapter
SP Switch
Frame1
SP Switch
Frame2
1
2
3
*
* These show the switch port number being used.
The actual physical connections are to the switch board.

Figure 39. GRF with Multiple Switch Router Adapter

For a more detailed explanation of GRF and its external connections, refer to the redbook *PSSP 2.4 Technical Presentation*, SG24-5173, and to *SP Switch Router Adapter Guide*, GA22-7310.

5.3.2 Self-Framed Nodes

Currently, only the RS/6000 Enterprise Server Model S70 may be added as a self-framed node. It is a high-end, RS/6000, PCI-based, 64-bit SMP workstation that supports concurrent 32- and 64-bit applications.

This self-framed node has the following SP characteristics:

- It is installed as a standard SP node and runs all PSSP software. It can be viewed and managed by all standard PSSP system management commands and subsystems like parallel commands, node groups, Group Services, Event Management, and so forth.
- It is connected to the SP Control Workstation via the SP administrative network.
- It is connected to the SP Switch through a TB3PCI adapter.
- It is physically located in an SP frame.
- It is viewed like a frame in the SP, but it cannot be the first frame.
- It has no frame or node supervisors, so there is only limited hardware control and monitoring from the Control Workstation.

- 64-bit processing is not exploited by PSSP, but you can run 64-bit applications that do not require any PSSP services.

5.4 Planning the Switch IP Network

To enable IP communication over the switch, each node needs to have an IP address and name assigned for its switch interface, the css0 adapter. If hosts outside the SP switch network need to contact the switch nodes in the SP, they must have a route to the switch network through one of the SP nodes, preferably the SP Switch router.

Like an Ethernet or Token-Ring network, the SP Switch supports the use of ARP to resolve IP addresses to hardware addresses. See Section 4.6.1, “Address Resolution Protocol (ARP)” on page 60 for a description of ARP in the SP Switch.

But unlike other networks, you do not need to use ARP over the switch. If you do not enable ARP, you need to specify the switch network subnet mask and the IP address of the first node in the switch. The IP addresses for subsequent nodes are calculated using the switch node numbers of the nodes. It is the fault service daemon that provides the IP interface layer with a mapping function that translates IP addresses into switch node numbers, as depicted in Figure 40. Note that with ARP disabled, you can have a single IP subnet in your switch.

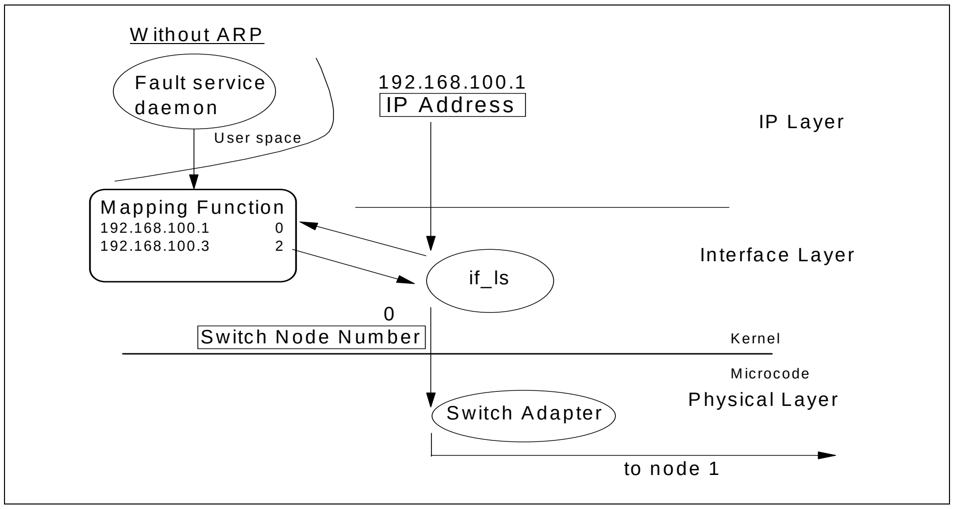


Figure 40. Switch Network Without ARP

If you want to assign IP addresses freely to your switch as you do with other adapters, you must enable ARP for the css0 adapter. Then you may use whatever IP addresses you wish, and those IP addresses do not have to be in the same subnet. When ARP is enabled for switch communication, you should be aware that to resolve IP addresses into switch node numbers an ARP broadcast has to be issued, which means that the same message has to be sent to all nodes. Of course, ARP maintains a cache to reduce the number of broadcasts needed. The use of ARP is depicted in Figure 41.

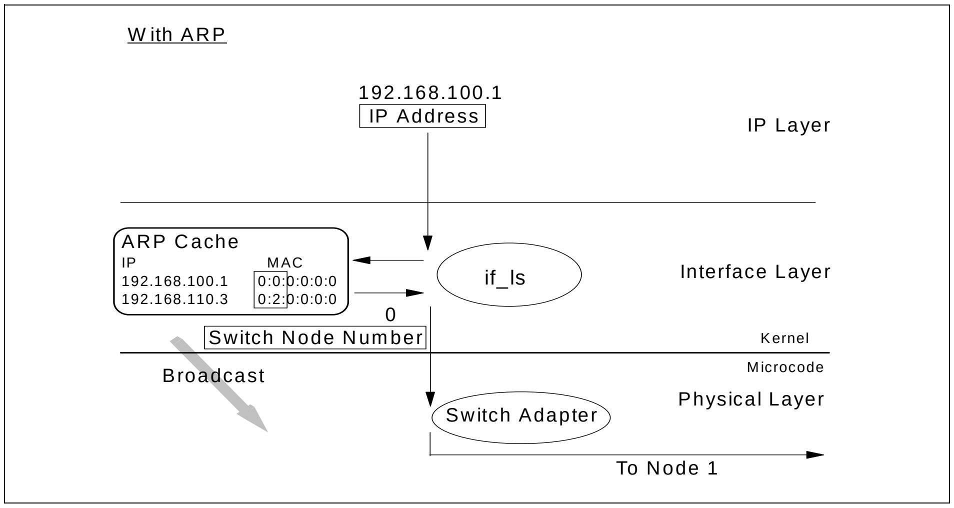


Figure 41. Switch with ARP

See 6.2, “Configuring the SP Switch Adapters” on page 92 for more information about how to configure the IP addresses of the switch adapters.

5.5 Planning a Partitioned SP System

You can use system partitioning to divide your system nodes into multiple groups in order to make your system more efficient and more tailored to your needs. It is internally done by subdividing the SDR and SP service subsystems and by disabling communication paths over the switch between nodes of different node groups.

There are some advantages and disadvantages to this approach. For example, you can partition the system to have different versions of PSSP completely separated, making sure that no switch communication is possible between them. On the other hand, VSD and GPFS cannot work across partitions, since the subsystems they would need on different partitions cannot communicate with each other.

Partitioning is an optional feature. If you do not make use of it, a single default system partition containing all the nodes is created when the system is installed the first time, named with the hostname of the Control Workstation.

Attention

Some configuration procedures can only be done on the default system partition. If you are using multiple partitions on your SP system, you have to reconfigure your system to the default single system partition. As an example, when you add a switch board, you need to have an unpartitioned system.

5.5.1 Partitioning Rules

A new system partition is formed by taking nodes from an existing system partition and collecting them as a new group. Links among partitions are disabled and the groups cannot exchange data through the SP Switch. You can choose from many partitioning templates provided by IBM, or you can define your own.

There are some rules that must be followed to create good partitions that guarantee a minimal availability. The concept is that *at least two disjoint paths are required between two nodes in the same partition*, instead of the four paths that are always present in a unpartitioned system.

When no intermediate switch boards are involved, two rules apply:

1. Each switch chip belongs to only one system partition.
2. Any node switch chip that is part of a multi chip system partition must be connected to at least two link switch chips on the same node switch board in the same system partition.

In a switch board (recall Figure 36), a *node switch chip* is a switch chip that can be connected to nodes, while a *link switch chip* is a switch chip whose ports can be connected only to other link switch chips of other switch boards. In node switch boards, there are four node switch chips and four link switch chips, while in intermediate switch boards, all switch chips are link switch chips.

The switch chips are the building blocks of partitions. If a node switch chip belongs to a partition, then so do all nodes attached to it, see Figure 42 on page 84. The smallest possible partition is one node switch chip, together with any of its attached nodes — a maximum of 4. Any number of partitions that make use of only one switch chip are allowed.

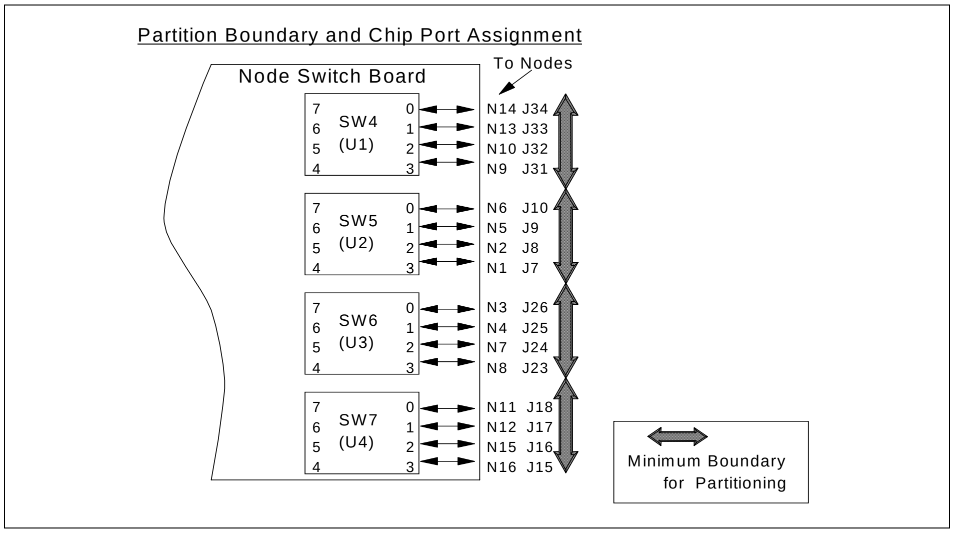
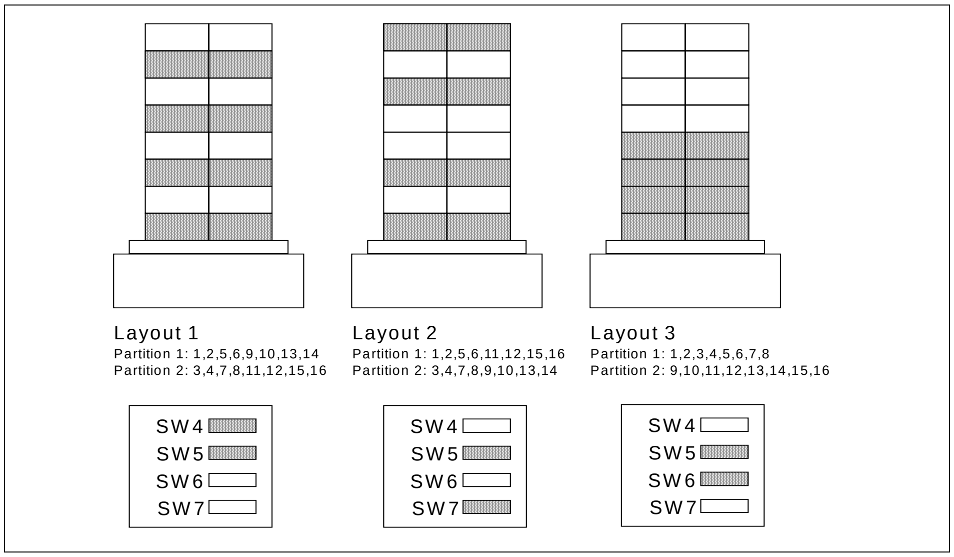


Figure 42. Partition Boundary and Chip Port Assignment

When more than four nodes must be in the same partition, then more node switch chips are involved. To make those switch chips exchange data, link switch chips are required. In order to have at least two disjoint paths, at least two link switch chips are then used.

As a consequence of the second rule, only two partitions that make use of two or more node switch chips can be defined in the same node switch board: each such partition reserves for itself two link switch chips, leaving no more link switch chips for other partitions.

As an example, Figure 43 on page 85 shows all the possible choices for a single switch board system divided into two partitions of eight nodes each (the 8_8 pattern). Three choices (layouts) are possible, depending on which switch chips are selected to group the eight nodes.



Layout 1
Partition 1: 1,2,5,6,9,10,13,14
Partition 2: 3,4,7,8,11,12,15,16
Layout 2
Partition 1: 1,2,5,6,11,12,15,16
Partition 2: 3,4,7,8,9,10,13,14
Layout 3
Partition 1: 1,2,3,4,5,6,7,8
Partition 2: 9,10,11,12,13,14,15,16
SW4
SW5
SW6
SW7

Figure 43. Partition Layout and Chip Assignment

An example of a multi chip partition for a two-switch board system is shown in Figure 44. In order to have Node A and Node B in the same system partition, all the highlighted switch chips must be included to provide at least two disjoint paths.

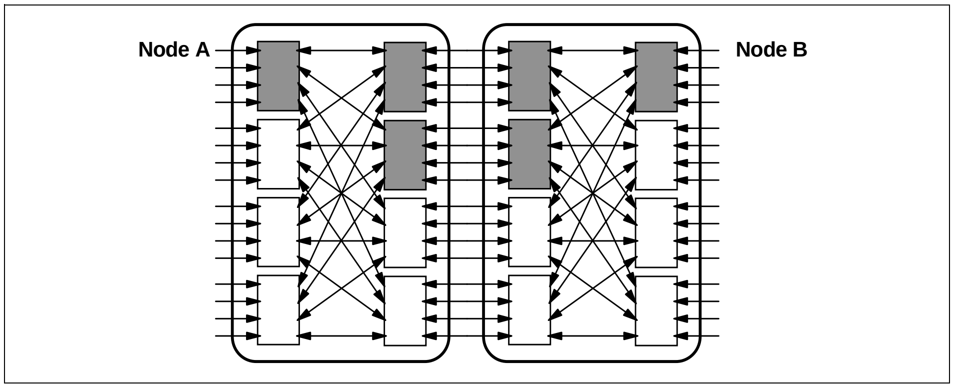


Figure 44. Partitioning in a Two Switch-Board System

As you can see, several switch chips are already reserved for the partition, limiting the topology of any additional partitions. Creating many partitions when using several switch boards may be tricky, and you should use a scheme like the one in this example to ease your work.

When intermediate switch boards are involved in a system partition, complexity increases since the switch chips on the ISBs are also used.

5.5.2 Partitioning Aid

IBM does not provide all partitioning combinations because of their huge number, which grows exponentially with the number of nodes. Inside the PSSP software only the simplest and most common partition configurations are provided, covering most customer needs.

If you do not find a suitable pattern in the supplied templates, you can create your custom partitioning scheme very easily with the Partitioning Aid GUI tool, which also makes the needed consistency checks on the defined partitioning scheme, applying partitioning rules.

For detailed information on Partitioning Aid, refer to *SP: Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281.

5.6 VSD and GPFS

Virtual Shared Disk (VSD) and General Parallel File System (GPFS) are shared disk applications that exploit the SP Switch network. They make use of IP but require the high bandwidth and reliability of the switch to work properly. They enable you to share common data among multiple nodes with very high performance by using the underlying switch fabric.

5.6.1 Virtual Shared Disk (VSD)

VSD is an application on the switch network that makes disk sharing among multiple nodes possible. The characteristics of VSD are:

- High performance shared disk system on SP (PSSP V2.1 or later).
- Single point management for multiple nodes usage, using `smit`.
- Provides data striping function (Hashed Shared Disk, or HSD).
- Device support is limited to raw disk access, and an external locking function is needed for data integrity.

VSD’s architecture is shown in Figure 45 on page 87.

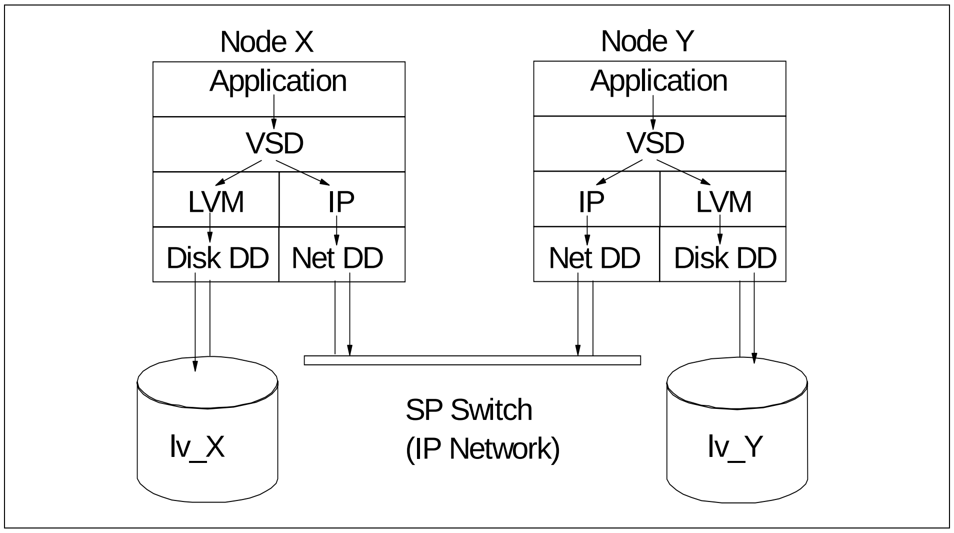


Figure 45. VSD Architecture

5.6.2 Recoverable VSD (RVSD)

RVSD is an optional feature that runs on VSD to provide high availability functions and recovery in case of node failure:

- It provides takeover by a secondary server in case of failure of a server node.
- It is used with twin-tail disks or SSA disks.
- It is able to co-exist with HACMP/ES.

An example of an RVSD configuration is shown in Figure 46 on page 88.

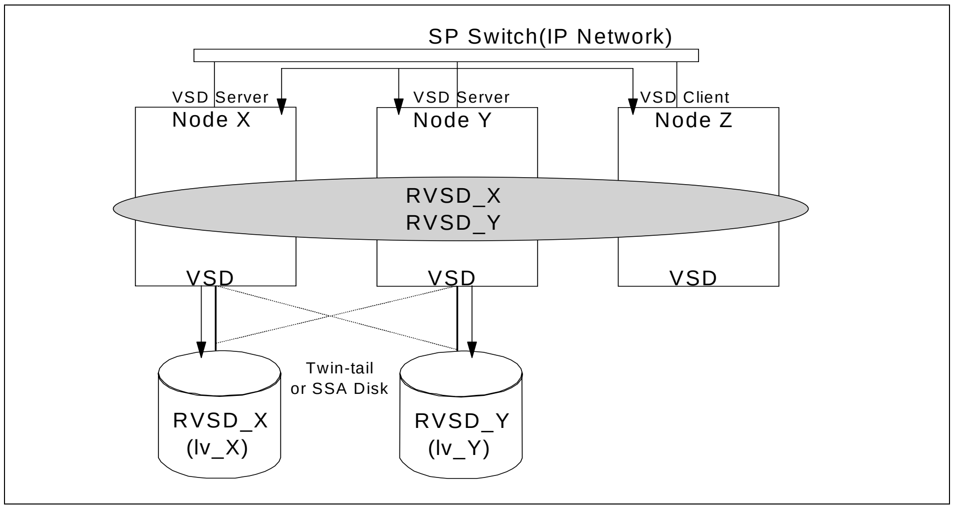


Figure 46. RVSD Configuration

If Node X is down for maintenance or because of a failure, the backup server Node Y takes over and provides access to data on the disk previously managed by the failed node (see Figure 47).

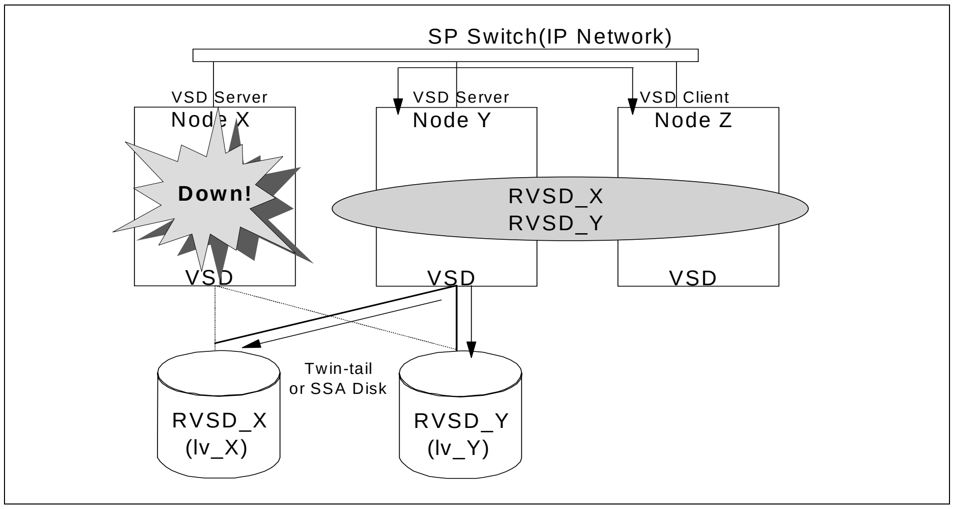


Figure 47. RVSD Takeover

For more detailed information for VSD and RVSD, see *PSSP: Managing Shared Disks*, SA22-7349.

5.6.3 General Parallel Filesystem (GPFS)

GPFS is a relatively new file system application that runs on the SP Switch network. It relies on VSD functions and behaves almost like a common UNIX file system, with some limitations. The file system is used by multiple nodes, requires single-point management and provides high performance. Its main characteristics are:

- Requires PSSP V2.3 or later
- High performance file system over the Switch network
- Requires VSD and RVSD (V2.1 or later)
- Single-point management for multiple node usage using SMIT
- Provides data striping function
- Is treated as a file system, and you can use common UNIX commands like `mkdir`, `cp` and so forth. Data locking is provided by GPFS.

Figure 48 shows the structure of GPFS.

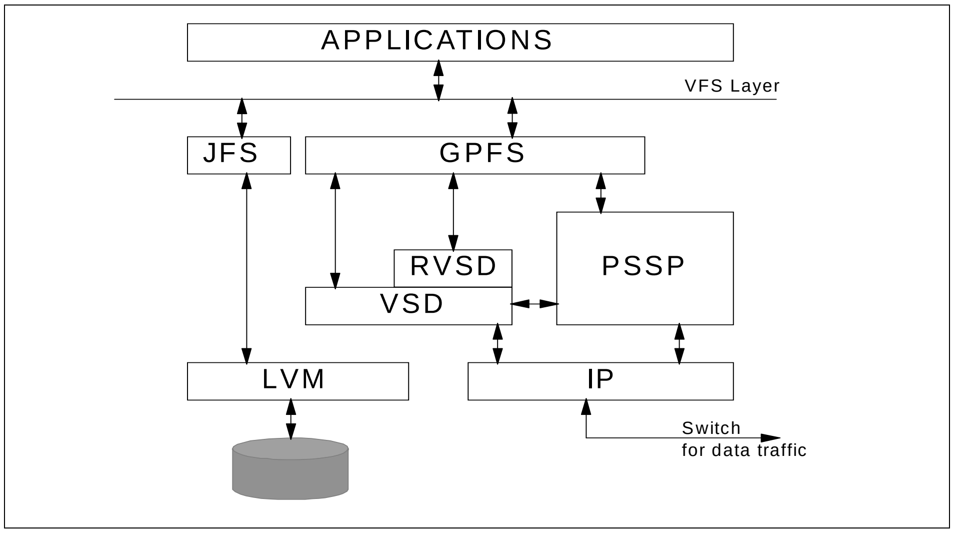


Figure 48. GPFS Overview

For planning and installation, refer to the redbooks *GPFS: A Parallel File System*, SG24-5165.

Chapter 6. Installation of the SP Switch

In this chapter we discuss some of the installation issues for the SP Switch, as described in *PSSP Installation and Migration Guide*, GA22-7347. It is not our objective to discuss all the installation steps of an SP system, and we assume you are consulting the installation manual to know the context where the actions described here are to be taken.

In this chapter we also discuss aspects of system partitioning, and describe some of the installation verification tools available to you. When reading this chapter, refer to *PSSP Command and Technical Reference*, SA22-7351, for the full syntax of the commands mentioned.

6.1 Installing the SP Switch

In the following three sections we go over the essential installation steps for the SP Switch, which follow:

1. Configure the switch adapters in the SDR.
2. Select the switch topology file, annotate it, and store it in the SDR.
3. Determine the clock topology file.

You may also partition your system, as more fully discussed in 6.5, “Setting up System Partitions” on page 103.

After your nodes are installed, you are ready to start the switch through the `Estart` command (refer to 7.3, “Starting the SP Switch” on page 119). Before that, you may also want to specify your primary and primary backup nodes (refer to 8.1, “Selecting the Primary and Primary Backup Nodes” on page 131).

Figure 49 on page 92 depicts the switch installation steps and their interaction with other system components.

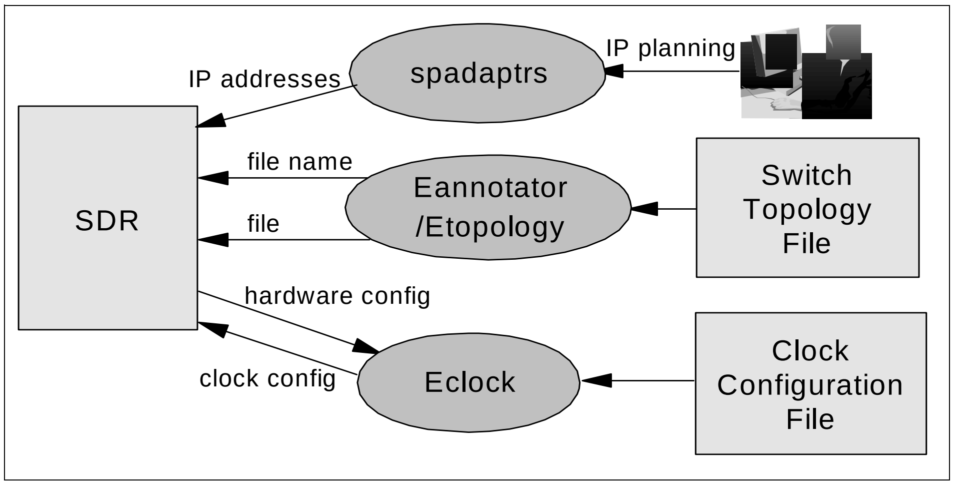


Figure 49. Installation and Configuration: Processes and Resources

6.2 Configuring the SP Switch Adapters

The `spadaptrs` command is used to configure the “additional” adapters of the system nodes. By additional, it is meant all adapters other than the ones connected to the administrative Ethernet network. The SP Switch adapters are configured with this command, and special attention is needed.

When you configure SP Switch adapters to your system you have to define the following thee options through `smit add_adapt_dialog` or the `spadaptrs` command:

- Skip IP Addresses for Unused Slots (`-s` flag).
- Enable ARP for the css0 Adapter (`-a` flag).
- Use Switch Node Numbers for css0 IP Addresses (`-n` flag).

Table 3 shows all allowed combinations of those three options.

Table 3. IP Assignment Options

Objective	Use Switch Node Numbers?	Skip IP Addresses for Unused Slots?	Enable ARP?
Assign IP addresses according to switch node numbers	Yes	No	Optional

Objective	Use Switch Node Numbers?	Skip IP Addresses for Unused Slots?	Enable ARP?
Assign IP addresses according to node numbers	No	Yes	Yes
Assign IP addresses sequentially	No	No	Yes

Note the following observations about this table:

- Remember that if you have expansion frames, the switch node number is *not* necessarily the node number minus one.
- Assignment of IP addresses according to switch node numbers can only be specified for *all* nodes. This means that you cannot have different IP subnetworks on your switch with this option.
- The preceding item implies that if you want to subnet your switch (in a partitioned system, for example), you have to enable ARP.
- You can assign IP addresses according to node numbers or assign them sequentially to all nodes, a subset of nodes, or a single node. If you assign them sequentially, it really does not matter whether you select Skip IP Addresses for Unused Slots or not.

The `spadaptrs` command stores the additional adapter information in the Adapter class of the SDR. You can check the settings of the additional adapters by issuing:

```
splstdata -a
```

6.3 Specifying the SP Switch Topology File

When configuring the SP Switch, the first thing you should do is to specify the files that define the switch topology of your respective system partitions. A topology file contains all switch components that are part of that partition’s switch fabric and how they are interconnected. Any component or link not present in the topology file, even if physically present, will not be included in the network by the fault service daemon, as described in 7.3.3, “Phase One of Switch Initialization” on page 124.

6.3.1 File Naming Rule

The standard topology of a switch network is uniquely defined by the SP hardware configuration.

The standard configuration files for *unpartitioned* systems can be found in the /etc/SP directory of the Control Workstation. These files comply with the following naming convention:

    expected.top.<num_nsb>nsb.<num_isb>isb.<type>

As you can see, the name of the file reflects the SP hardware configuration by using the following three variables:

- <num_nsb>: number of node switch boards in the configuration
- <num_isb>: number of intermediate switch boards in the system
- <type>: type of topology, usually 0

The only exception to this convention is for the topology file of an SP Switch-8 system. The name of that topology file is:

    expected.top.1nsb_8.0isb.1

Actually, the topology files in the /etc/SP directory are symbolic links to the directory /spdata/sys1/syspar_configs/topologies. In the initial releases of PSSP, they were located in /etc/SP, while in later releases file placement was changed and links were introduced to maintain compatibility.

6.3.2 Inside a Topology File

The topology file is the representation of:

- All frames used by the system partition
- All nodes in the system partition
- All switch boards used by the system partition
- All switch chips in the system partition
- The board, switch chip, and chip port to which each node is attached
- The chip-to-chip connections within each board, including chip port data
- Any interboard connections, including switch chip and switch port data

The same topology file can be used in both SP Switch systems and the older HiPS systems. The only important difference between these systems, with respect to the topology files, is in the use of external connections (jacks) of the node-to-switch and switch-to-switch cables.

The topology files shipped with PSSP specify the jack connections for HiPS systems. Thus, to make use of the topology files in finding out which physical external connection is being used or is showing a problem, we need to change the standard topology files, a process called *annotation*. Note that, if you do not annotate the topology file, everything should work, but you could have some trouble in finding, for example, which faulty cable to replace.

The topology files are annotated through the `Eannotator` command. In all following discussions, unless otherwise indicated, the topology files shown have been annotated for an SP Switch.

The topology files are plain text files. Each non-comment line in the file represents a point-to-point link in the SP Switch network. There are two types of connections: between a switch chip port and a node, and between two ports of different switch chips. Typical connections are shown in Figure 50.

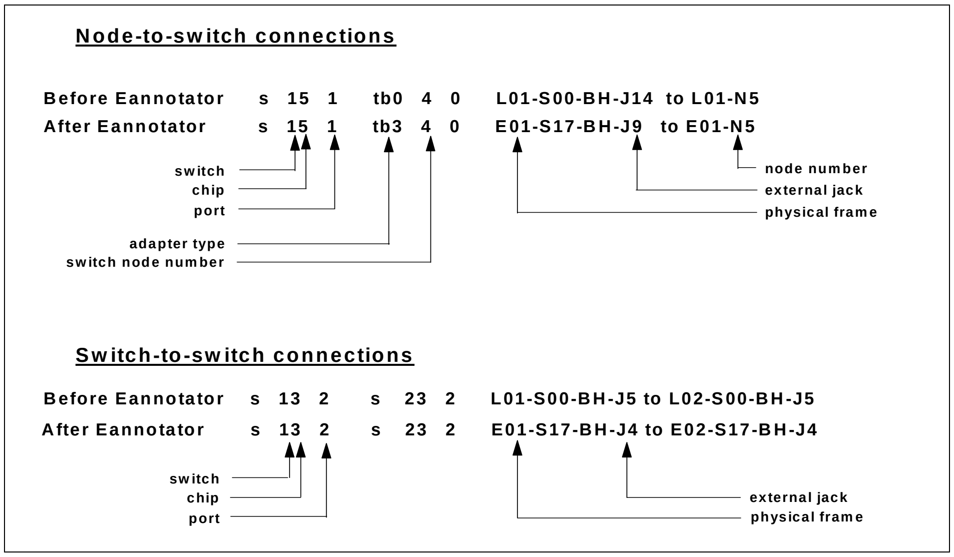

Node-to-switch connections
Before Eannotator s 15 1 tb0 4 0 L01-S00-BH-J14 to L01-N5
After Eannotator s 15 1 tb3 4 0 E01-S17-BH-J9 to E01-N5
switch
chip
port
adapter type
switch node number
node number
external jack
physical frame
Switch-to-switch connections
Before Eannotator s 13 2 s 23 2 L01-S00-BH-J5 to L02-S00-BH-J5
After Eannotator s 13 2 s 23 2 E01-S17-BH-J4 to E02-S17-BH-J4
switch
chip
port
external jack
physical frame

Figure 50. Topology File Nomenclature

Each end of a link in the topology file is defined by the following three fields:

<device type > <device ID> <port>

The <device type> field represents to which device the link is connected. Possible values are:

- s, for switch chip
- tb3, for switch adapters

The name tb3 is generic and does not indicate the actual adapter's type; it is just an indication that this is a system with an SP Switch.

The <device ID> value changes depending on the device type:

1. Switch chip:
   - Most significant digits: the switch board number (If the board is in an ISB, the value 1000 is added.)
   - Least significant digit: the switch chip number
2. Adapter:
   - The switch node number (There are entries for every possible node connected to the switch, even if the node is not actually present.)

The <port_number> is always 0 for adapters. For switch chips, the <port_number> represents the chip's port.

After the identification of both sides of a link comes the physical connection information. There is one entry per side of the link. For the switch side we have:

- The number of the frame where the switch board is installed
- The frame slot where the switch board is installed
- An indication if this is an intraboard connection (SC) or an external connection (BH)
- If this is an external connection, the external jack for the corresponding port

On the node side, the physical connection information contains:

- The number of the frame where the node is installed
- The node number

If there is no physical node connected, the frame number and node number appear as xx.

Important

The topology files should not be changed, because for the resulting configuration may be one that is not supported. All RS/6000 SP systems are cabled in a standard way that matches the information in the topology files. The cabling or the contents of the topology files should only be changed for diagnostic purposes on the advice of an IBM engineer.

Here we describe two topology files. The full content of the two files can be found in B.1, "Example of a Switch Topology File" on page 247.

The first example shows an annotated `expected.top.1nsb.0isb.0`. The first excerpt shows some links between switch chips and nodes:

```
# Node connections in frame L01 to switch 1 in L01
s 15 1  tb3 4 0            E01-S17-BH-J9 to E01-N5
s 15 0  tb3 5 0            E01-S17-BH-J10 to Exx-Nxx
s 16 2  tb3 6 0            E01-S17-BH-J24 to Exx-Nxx
s 16 3  tb3 7 0            E01-S17-BH-J23 to Exx-Nxx
s 14 3  tb3 8 0            E01-S17-BH-J31 to E01-N9
s 14 2  tb3 9 0            E01-S17-BH-J32 to Exx-Nxx
s 17 0  tb3 10 0           E01-S17-BH-J18 to Exx-Nxx
```

The second excerpt shows some links between switch chips:

```
# On board connections between switch chips on switch 1 in Frame L01
s 14 7    s 13 4           E01-S17-SC
s 14 6    s 12 4           E01-S17-SC
s 14 5    s 11 4           E01-S17-SC
s 14 4    s 10 4           E01-S17-SC
s 15 7    s 13 5           E01-S17-SC
s 15 6    s 12 5           E01-S17-SC
s 15 5    s 11 5           E01-S17-SC
s 15 4    s 10 5           E01-S17-SC
s 16 7    s 13 6           E01-S17-SC
```

The links corresponding to the highlighted lines in the preceding file are shown in Figure 51 on page 98.

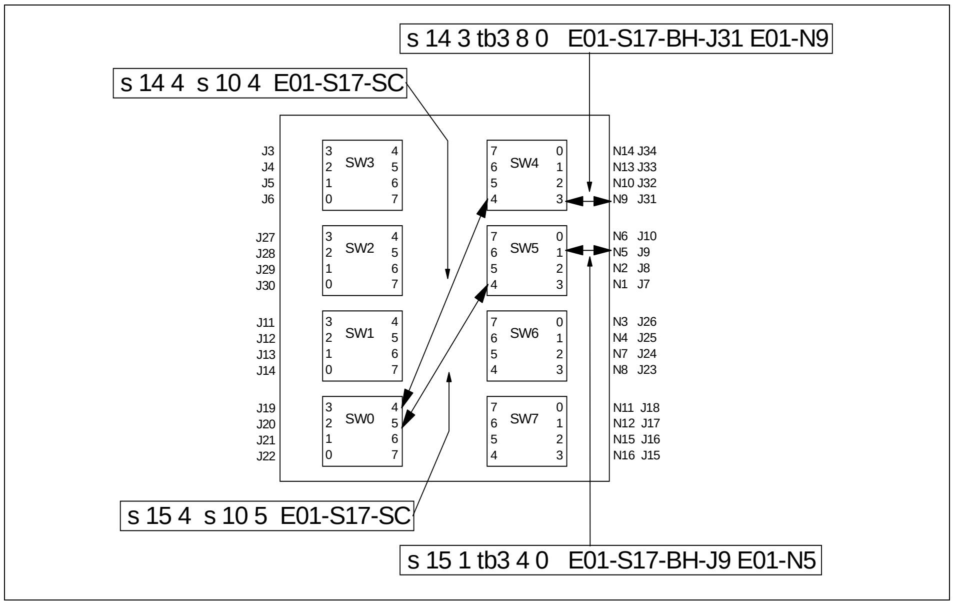


Figure 51. Links in Switch Board

As an example of connections between two switch boards, we show an excerpt from the annotated version of the file `expected.top.2nsb.0isb.0`. On an SP system with two frames, all four ports of the four link switch chips in the first frame are connected to the corresponding ports on the second frame; see the following example.

```
# switch 1 to switch 2
s 13 3    s 23 3         E01-S17-BH-J3 to E02-S17-BH-J3
s 13 2    s 23 2         E01-S17-BH-J4 to E02-S17-BH-J4
s 13 1    s 23 1         E01-S17-BH-J5 to E02-S17-BH-J5
s 13 0    s 23 0         E01-S17-BH-J6 to E02-S17-BH-J6
```

6.3.3 Storing the Topology File in the SDR

You should annotate the appropriate topology file by using `Eannotator`. After annotating the topology file, it must be stored in the SDR with the `Etopology` command. The `Eannotator` command has an option to automatically call `Etopology` after the annotation, so you do not need to run both commands. You can choose any name for the annotated topology file.

As explained in 6.5.2, “Applying a Partition Configuration” on page 105, you do not need to run `Eannotator` (and, optionally, `Etopology`) when you partition your system. The appropriate annotated topology files are automatically stored in the SDR for you. (Note that the PSSP software could do the same for you when you have an *unpartitioned* system, since the appropriate standard topology file is uniquely defined by the system configuration.)

The annotated topology files, one per partition, are stored as SDR files. These files are kept in the /spdata/sys1/sdr/partitions/<ip address>/files directory, where <ip address> is the IP alias that identifies the partition. The name of the topology file is stored in the Switch_partition SDR class.

When a topology file is stored in the SDR, the actual name stored is the name you specify followed by a version number. For example, if you specify the name of your topology file as expected.top.annotated, the actual name stored would be expected.top.annotated.1. If you again store a topology file in the SDR for that partition, the name specified will be appended with ".2", and so on.

6.3.4 The Topology Files for a Partitioned System

The topology files present in /etc/SP are for *unpartitioned* systems, and all physical connections of the switch fabric are listed in them. In a partitioned system, however, there is a topology file for each system partition. The topology file for a partition only lists the links that are part of the switch network for that partition.

The topology files for partitioned systems, as well as the other files that define the partitioning, are in the directory `/spdata/sys1/syspar_configs`. Figure 52 on page 100 shows the directory structure.

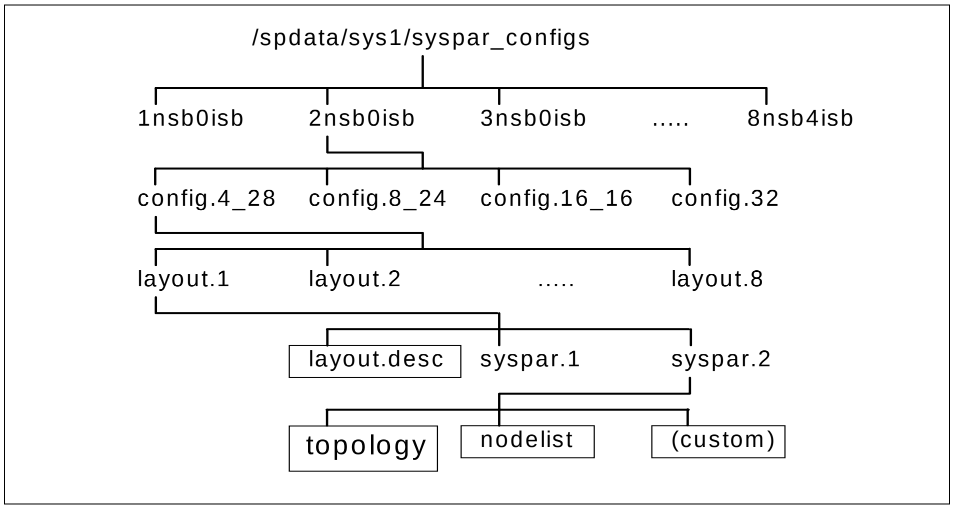


Figure 52. Directory Structure for Partition Configuration

In this figure we highlight the location of a topology file (whose name is topology) for a two-frame system with two partitions, one with 4 nodes and the second with 28 nodes. The topology file shown is for the second partition (syspar.2) of the first possible layout (layout.1).

6.4 Specifying the SP Switch Clock Distribution Tree

The *clock topology* file identifies the master switch board and how the clock is distributed to all other switch boards. The master switch board generates the synchronous clock using an internal oscillator that drives the whole switch fabric. Any other board, a slave board, gets the clock directly or indirectly from the master board through a switch-to-switch cable, thereby defining a distribution tree. For more information on the clock distribution process, see 3.2, “System and Board Clocking” on page 17.

The clock topology files are located in the /etc/SP directory. The appropriate standard clock topology file is uniquely determined by your hardware configuration. The naming convention used for the clock topology files is the same one used for the switch topology files, with two exceptions: the filenames start with "Eclock" instead of "expected", and the clock topology file for an SP Switch-8 systems has a type of 0.

You may use the `Eclock` command to specify your clock topology file. You can explicitly specify the name of the distribution file, using `-f <clock config`

`file`>, or you can let the command figure out the correct name of the file by specifying option `-d`.

Each clock topology file starts with a brief explanation of the contents of the file. The full text of a sample clock topology file can be found in B.2, “Example of a Clock Topology File” on page 250. Then two distribution trees follow: the standard distribution and an alternate distribution. An alternate distribution is provided so you can work around a clock distribution problem, for instance when your master switch board is down. The distribution tree is defined using a relatively simple syntax. For instance, an excerpt from the file Eclock.top.7nsb.4isb.0 follows:

```
#  Switch number
#  |  Clock multiplexor (mux) value
#  |  |    Clock receiver jack number (High Performance Switch) / (SP Switch)
#  |  |    |     Clock source switch number
#  |  |    |     |    Clock source jack number (High Performance Switch)
#  |  |    |     |    |   Clock source jack number (SP Switch)
#  |  |    |     |    |   |
   1  1  J3/J3   1001 J3  J3
   2  1  J3/J3   1001 J5  J4
   3  1  J3/J3   1001 J7  J5
   4  1  J3/J3   1001 J9  J6
   5  1  J3/J3   1001 J4  J34
   6  1  J3/J3   1001 J6  J33
   7  1  J3/J3   1001 J8  J32
1001  0  xx/xx      0 xx  xx
1002  1  J3/J3      1 J5  J4
1003  1  J3/J3      1 J7  J5
1004  2  J5/J4      2 J9  J6
```

The important values are the first and second columns. All the other columns simply document the clock distribution tree.

The first column identifies the switch board and the second column identifies its clock source. The clock source is defined by the board’s *mux* value. The mux value defines the clock source in the following way:

- mux 0: the clock source is the internal oscillator, that is, this is the master switch board.
- mux 1: the clock source comes from external jack 3.
- mux 2: the clock source comes from external jack 4.
- mux 3: the clock source comes from external jack 5.
- mux *n* (between 4 and 34): the clock source comes from external jack *n*. (Specifying that the clock source comes from external jacks 7-10, 15-18, 23-26, and 31-34 only makes sense for intermediate switch boards.)

The values 1, 2, and 3 for mux are a remnant from HiPS, thus explaining the non-straightforward rule of formation. As with the switch topology files, a design objective of PSSP was to have the same set of clock configuration files for both SP switches.

Note that all other values in the file can be derived from the mux value and the standard SP cabling for the system in question. The remaining columns are ignored by `Eclock`, except for the clock source switch (board) number, which is used to define the order in which the switch boards' clocks are configured: first the master switch board, then the first level of the distribution tree, and so forth.

Figure 53 shows how the clock is propagated in the aforementioned clock configuration file.

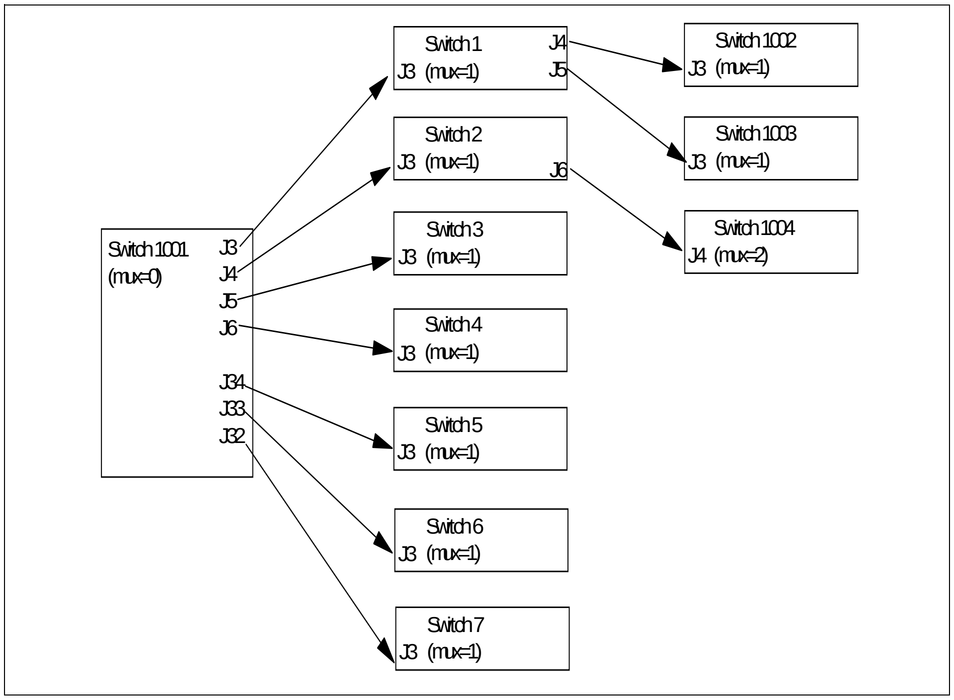


Figure 53. Standard Clock Distribution for a System with 7 NSBs and 4 ISBs

The `Eclock` command is further discussed in 8.2, "Establishing the SP Switch Clock" on page 134. As described there, and unlike the `Etopology` command, which only updates the SDR, this command not only updates the SDR with the clock distribution tree, but also sets the clock multiplexors. In order to do that, the whole switch is brought down and all the switch boards are reset. Therefore, you should only issue an `Eclock` when strictly necessary.

Another difference between `Eclock` and `Etopology` is that Etopology must be executed only once during installation, while Eclock must be executed whenever the frames are powered on.

6.5 Setting up System Partitions

In this section we go over the internal changes when you partition your system. For the actual steps you should take to partition your system, refer to *PSSP Administration Guide*, SA22-7348, where the step-by-step procedures are clearly explained.

In Figure 54 we show an overview of the partitioning steps. The main ones follow:

- Define the IP aliases.
- Select an existing partition configuration or generate a new one.
- Apply your configuration.

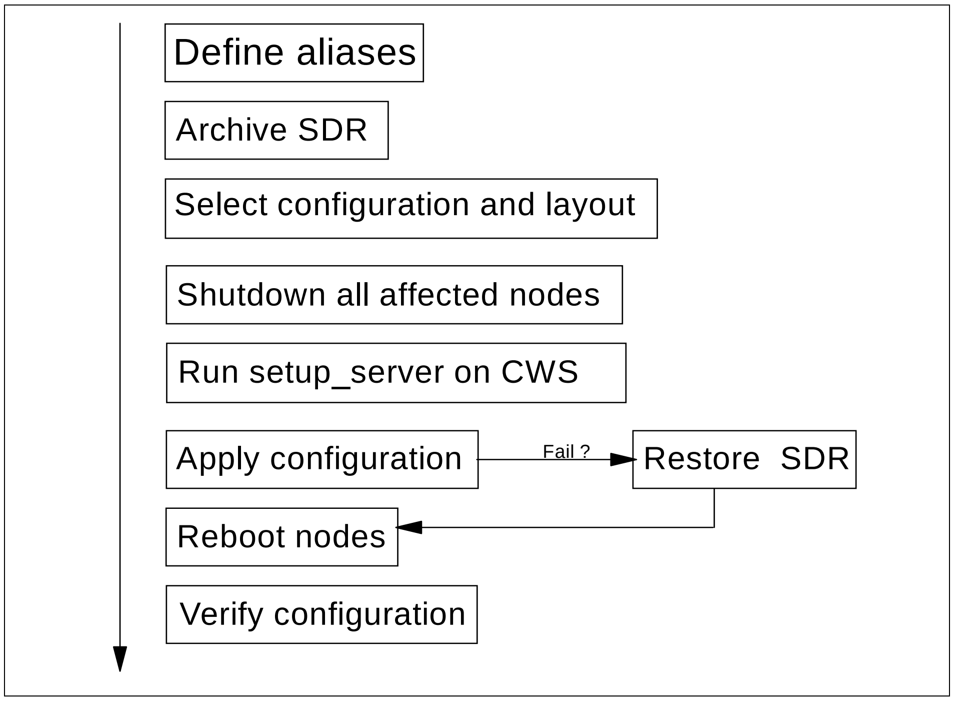


Figure 54. Overview of Partitioning

6.5.1 Defining the IP Aliases

It is important for you to understand how the IP aliases are used in partitioning.

Each partition is associated with a different IP address. All these IP addresses must be aliases for the *primary network interface* in the Control Workstation. The primary network interface is defined by the Control Workstation's *primary hostname*. The primary hostname is the IP name returned by the `hostname` command. This hostname resolves to an unique IP address. And, in its turn, this IP address is associated with one network interface: the primary network interface.

Every partition-sensitive command, like `Estart`, uses the environment variable SP_NAME to determine the partition to be acted upon. The variable must be set to the partition´s IP address or the IP name that resolves to that address. If SP_NAME is not set, the default system partition will be used, which is defined by the name returned by the `hostname` command.

6.5.2 Applying a Partition Configuration

After defining your partition configuration, you should apply it using the `spapply_config` script. The main steps of this script are described in *PSSP Administration Guide*, SA22-7348.From the switch point of view, the main steps are:

1. Issue the `Eprimary` command to create the Switch_partition object for the partition being created, as well as to specify the default primary and primary backup nodes.
2. Issue the `Eannotator` command to annotate the partition's topology file.
3. Issue the `Etopology` command to store the partition's topology in the SDR.

After applying your configuration, you should verify it by issuing the following commands:

```
spverify_config
splstdata -p
syspar_ctrl -E
```

6.5.3 Repartitioning the SP System

To *change* the partition configuration of your SP, you need to follow almost the same steps as taken to partition a system. However, an additional critical step is needed. For every partition to be repartitioned, you should issue the `Eunpartition` command, *before* you apply the new configuration.

The `Eunpartition` script sends a request to the primary node for it to enable the partition's boundary chip ports. Refer to 7.3.3, "Phase One of Switch Initialization" on page 124 for an explanation of why this is necessary. After the ports have been enabled, the primary and primary backup fault service daemons are restarted.

If you do not run `Eunpartition` before applying the new configuration, you may face several `Estart` problems. To solve these problems, you should restore the previous partitioning by restoring the SDR. If you do not have a backup of the SDR, recovery can be accomplished by the following sequence:

1. Issuing `Eclock` to reset the switch, which will take down the switch in *all* partitions, even those not being affected by the repartitioning.
2. Rebooting all the nodes or issuing an `rc.switch` command on all nodes.
3. Issuing `Estart` in each of the system partitions.

6.6 Verifying the Installation

In this section we summarize some of the commands that you can use to check the status of the switch installation and customization.

6.6.1 Verification Commands

PSSP has several commands to verify the installation and configuration of your system.

6.6.1.1 CSS_test

The `CSS_test` command checks the following:

- Whether each node is alive or not
- It performs a switch IP check via ping
- The level of the ssp.basic and ssp.css software in each node

The result of `CSS_test` is written to the /var/adm/SPlogs/CSS_test.log, besides being shown at the terminal. Note in the following excerpt that one node is being reported as not having IP connectivity to the switch.

```
CSS_test:  Beginning Switch IP test of the nodes in partition sp5cw0.
CSS_test:  Following nodes failed Switch IP test:
sp5n05x (192.168.15.5)
CSS_test:  CSS Installation Verification Test completed on
                 Tue Jul 21 10:03:36 EDT 1998.
```

6.6.1.2 spmon

The `spmon` command operates the system controls and monitors system activity. It has many flags. One of the most popular ones is `spmon -d`, which shows the state of all nodes in the partition. Adding the option `-G`, you can get global status information.

6.6.1.3 splstdata

The `splstdata` command is used to list the system's configuration data. It has many flags, one for each type of data maintained in the SDR.

The options that are switch-related follow:

`splstdata -a` shows the configuration of all node adapters.

`splstdata -s` shows the switch connection for each node.

`splstdata -p` shows the partitioning information.

6.6.2 SDR Information

With `SDRGetObjects`, you can get information directly from the SDR. All flavors of `splstdata` simply organize and show the SDR contents.

Following are some of the most relevant SDR classes for switch verification and debugging.

6.6.2.1 Adapter

This class contains the adapter information shown by `spethernt` and `spadaptrs`.

6.6.2.2 switch_responds

This class shows the current status of the switch. Arguably it is the most consulted class while debugging switch problems. In the following sample of the contents of the class, the first two nodes are off the switch:

```
# SDRGetObjects switch_responds
node_number  switch_responds autojoin     isolated adapter_config_status
          1            0            1            1 css_ready
          5            0            1            1 css_ready
          9            1            1            0 css_ready
         13            1            1            0 css_ready
```

As discussed in Chapter 10, “SP Switch Problem Diagnosis” on page 179**,** many switch problems can be overcome by changing the switch_responds attributes directly through the command `SDRChangeAttrValues`. However, this is a delicate activity, and should be avoided; archive the SDR before trying this.

6.6.2.3 Syspar_map

The Syspar_map class is one of the most important in the SDR. It is created early during the installation process. It is referenced by many commands to create other object class and to execute SP-related tasks. You should verify the objects in this class after installation or hardware reconfiguration.

```
# SDRGetObjects Syspar_map
syspar_name  syspar_addr  node_number  switch_node_number used node_type
sp5cw0       9.12.1.150             1            0           1 standard
sp5cw0       9.12.1.150             2            1           0 standard
sp5cw0       9.12.1.150             3            2           0 standard
sp5cw0       9.12.1.150             4            3           0 standard
sp5cw0       9.12.1.150             5            4           1 standard
sp5cw0       9.12.1.150             6            5           0 standard
sp5cw0       9.12.1.150             7            6           0 standard
sp5cw0       9.12.1.150             8            7           0 standard
sp5cw0       9.12.1.150             9            8           1 standard
sp5cw0       9.12.1.150            10            9           0 standard
sp5cw0       9.12.1.150            11           10           0 standard
sp5cw0       9.12.1.150            12           11           0 standard
sp5cw0       9.12.1.150            13           12           1 standard
sp5cw0       9.12.1.150            14           13           0 standard
sp5cw0       9.12.1.150            15           14           0 standard
sp5cw0       9.12.1.150            16           15           0 standard
```

6.6.2.4 Syspar

Syspar is a partition class and contains several attributes for the partition: partition name, IP alias, system version, authentication methods, and so on.

6.6.2.5 Switch

This class contains the main hardware-related switch attributes, as follows:

switch_number
frame_number
slot_number
switch_partition_number
switch_type
clock_input
switch_level
switch_name
clock_source
clock_change

6.6.2.6 Switch_partition

This class is used to store the switch starting values for the partition. Its attributes follow:

switch_partition_number
topology_filename
primary_name
arp_enabled
switch_node_number_used
primary_backup_name

oncoming_primary_name
oncoming_primary_backup_name
num_nodes_success
switch_max_ltu
switch_link_delay

6.6.3 Logs

To verify installation or customization problems, you should also check the logs maintained by CSS.

You should initially check the AIX log through `errpt`, and if more information is needed, look at the log files generated by the switch software. These files are located in the /var/adm/SPlogs/css directory.

Refer to 9.2, “SP Switch Log Files” on page 150 for more information about the logs and their use in solving switch-related problems.

Chapter 7. Initialization of the SP Switch

The SP Switch is initialized in three distinct steps as depicted in Figure 55. First, the switch adapter is configured as described in 7.1, “Configuration Method of the SP Switch Adapter” on page 111. Next, the fault service daemon is started as explained in 7.2, “Running the SP Switch Daemon” on page 113. Finally, the SP Switch is started through the execution of the `Estart` command, whose main steps are detailed in 7.3, “Starting the SP Switch” on page 119.

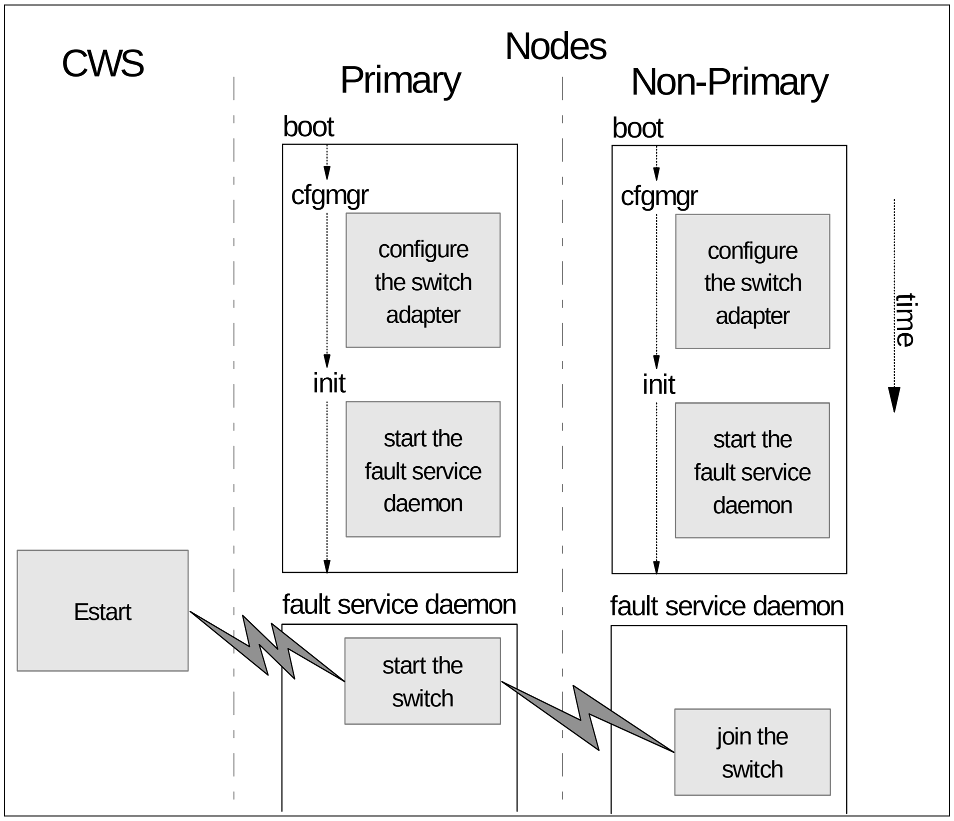


Figure 55. Initialization of the SP Switch

7.1 Configuration Method of the SP Switch Adapter

Early in the node’s boot process, AIX configures all detected hardware. AIX’s `cfgmgr` configures the SP Switch adapter, device *css0*, during phase 2. As specified by the Configure method in ODM’s PdDv, `cfgmgr` executes the `cfgtb3` program, which, like all internal switch-related files, is located in /usr/lpp/ssp/css. One important note is that there is only *one* configuration method, device driver, kernel extension, and fault service daemon to support *all* SP Switch adapters.

The configuration method takes the following actions:

- Defines the css0 device in ODM's CuDv, removing any previous conflicting definition.
- Verifies the adapter's physical location and resolves any bus conflicts, unless the node uses an SP Switch MX Adapter or a SP Switch PCI adapter. When the node is PCI-based, this step is taken care of by the node's firmware. PCI nodes (for example, the 332 MHz SMP nodes and the S70/S7A external node) employ the Common Hardware Reference Platform (CHRP) architecture, and so firmware walks the buses and constructs a device tree: AIX obtains that device tree during boot, and uses it to build the CuDv and CuAt ODM entries.
- Creates the special file /dev/css0.
- Loads the device driver, cssdd3, and sets the device to available.
- Loads the adapter's firmware, which is dependent on the adapter type. It loads the xilinx_file3 file for the MCA adapter, the xilinx_file3mx file for the MX adapter, and the xilinx_file3pci file for the PCI adapter.
- Marks the device as available in ODM's CuDv.
- Loads and initializes the CSS kernel extension, fault_service_SP.
- Executes the POST diagnostic for the adapter. This diagnostic puts the TBIC in reset after the diagnostic runs, disabling the adapter.

If all the configuration steps have been completed successfully, the method sets the css0's adapter_status attribute to css_ready in ODM's CuAt. Otherwise, the attribute is set to an appropriate value describing the point where configuration failed. For example, a value of make_special_fail indicates a failure while creating the special file. You can find the list of values this attribute can assume in *PSSP: Diagnosis Guide*, GA22-7350. This value is eventually transferred to the SDR and used from there. This intermediate step is necessary because, at this point in the boot process, the SDR is inaccessible.

If the adapter was not configured successfully, you can check the node's boot log by running the `alog -f /var/adm/ras/bootlog -o` command.

Attention

Be aware that the use of utilities as described here is not foolproof. Success depends on how busy the device driver is.

When trying to resolve a node or adapter problem, you can, short of rebooting the node, try to reconfigure the adapter. First, try to stop all users of the adapter:

- The Event Management subsystem:

```
stopsrc -s hats
```

- The fault_service_Worm_RTG_SP daemon:

```
/usr/lpp/ssp/css/css_cdn
```

The `ucfgtb3` utility is used to unconfigure the switch adapter. This utility sets the device css0 as defined and marks the adapter as not_configured in the SDR. This utility used to terminate and unload the kernel extension and device driver, but changes in AIX's IP implementation (as of AIX 4.1) prevent the utility from doing so.

The utility should be invoked as follows:

```
ucfgtb3 -v -l css0
```

The `cfgtb3` utility is used to configure the SP Switch adapter after it has been unconfigured. This utility is the configuration method for the css0 adapter.

The utility should be invoked as follows:

```
cfgtb3 -v -l css0
```

After running these utilities you must restart the fault service daemon by running `rc.switch` or `css_restart_node`, as described in the following section.

7.2 Running the SP Switch Daemon

The next step in the switch initialization process is to start the fault service daemon on all nodes of the SP system. The fault service daemon, sometimes called the Worm, is responsible for initializing and monitoring the switch. This daemon can have different *personalities*, that is, different roles in supporting the switch. The type of a node defines the personality of the daemon that is running on that node. A node can be one of the following types:

Secondary	A node which has no special role in managing the switch. All nodes in an SP partition, except for the primary node and the primary backup node, are secondary nodes.
Primary	The node which is responsible for managing the switch. The primary node starts the switch, monitors the switch, and implements recovery from node, link, and chip failures. There can only be one primary node per partition.
Primary Backup	The node which is responsible for monitoring the primary node. Whenever the primary node becomes unreachable, the primary backup initiates a primary node takeover: it becomes the partition's primary node. During normal operation, there is one primary backup node per partition.

This daemon is the fault_service_Worm_RTG_SP program and is started by the `rc.switch` script. You should always start the fault service daemon via `rc.switch` so that the appropriate parameters are passed to the program. The program's name comes from the three main functions of the daemon:

1. The daemon is responsible for servicing the Error/Status packets sent by the switch fabric when reporting faults in the fabric, and, in response to those packets, implements recovery actions for node, chip, and link outages. Only the primary node receives service packets from the switch fabric. The other nodes only deal with adapter faults, and communication with the primary.
2. During switch initialization, the daemon runs the Worm code, which determines the currently usable switch topology. This code is run only by the primary node.
3. The daemon, during initialization or whenever there is a topology change, calculates the routes between nodes using the Route Table Generation (RTG) code. This code, in contrast to the Worm code, is run by all standard nodes.

The last two functions are detailed in 7.3, “Starting the SP Switch” on page 119.

7.2.1 The Daemon Initialization Script

The fault service daemon is started in the `rc.switch` script. This script is run by the init process, as specified in the *fsd* inittab entry. The script's main actions are:

1. It kills any previous instantiation of the fault service daemon.
2. It zeros the switch_responds attribute in the switch_responds class in the SDR.
3. It checks, using the Switch_partition class in the SDR, whether this node is the current primary or the current primary backup node. If the node *is* one of these, the script changes the SDR indicating that there is no longer a primary or a primary backup node.
4. It gets the value of the adapter_status attribute out of ODM and updates the node's adapter_config_status attribute in the switch_responds class in the SDR. If the attribute's value is not css_ready, the script exits with an error.
5. It configures the IP interface css0. The interface's IP address, its netmask, and whether the switch network uses ARP are taken out of ODM's CuAt. The interface is marked as down.
6. The fault service daemon is started. It receives as parameters the node's switch node number and to which switch board, switch chip, and port it is connected. This information is also taken from ODM's CuAt, which contains the following attributes:

```
# odmget -q name=css CuAt | grep -E "attr|val"
        attribute = "switch_node_num"
        value = "0"
        attribute = "switch_number"
        value = "1"
        attribute = "switch_chip"
        value = "5"
        attribute = "switch_chip_por"
        value = "3"
        attribute = "arp_enabled"
        value = "no"
        attribute = "netaddr"
        value = "192.168.15.1"
        attribute = "netmask"
        value = "255.255.255.0"
        attribute = "state"
        value = "up"
```

   These attributes are added during node installation or customization by `pssp_script`. The appropriate values come from the Node, Adapter, and Switch_partition SDR objects. Note that the switch board number, switch chip, and switch chip port values are a hard-coded function of the node's switch node number. And, in turn, the switch node number is usually a hard-coded function of the node number. These values are computed by the `/usr/lpp/ssp/install/bin/SDR_config` script, which handles a change to the association between node number and switch node number if specified in the /etc/switch.info file.

Be aware, though, that the incorrect use of such a file may introduce several switch connection problems and, most important, its use is *not* supported by IBM.

7. Just before exiting, the script runs the `usconfig` utility. This utility predefines some User Space window parameters.

7.2.2 The Fault Service Daemon

The fault service daemon, when launched, starts the adapter's microcode and takes the TBIC out of reset, enabling the adapter.

The daemon then enters into its main loop, waiting for service packets from the primary node, service packets from the switch fabric (if this node is the primary node), interrupts from the adapter, or E-commands. E-commands, like `Estart`, send requests to the fault service daemon through the `i_stub_SP` or `sp_fs_control` programs, which queue those requests in the fault service work queue maintained by the fault_service_SP kernel extension. Requests from the adapter are also put in the same work queue.

Whenever an error is detected in a chip, that component sends an Error/Status service packet to the primary node. Such packets arrivals are *asynchronous* with respect to the fault service daemon, since they are unsolicited. Upon receiving an asynchronous Error/Status packet, the primary daemon starts a recovery procedure, which initially tries to reset the error. If the error cannot be reset, which means there is a permanent error in some component, the daemon disables the faulty node, chip, or link and initiates the update of the switch topology, as described in 7.3.5, "The Generation of Routes" on page 127.

The primary node's fault service daemon does not depend on the switch fabric successfully notifying it when a problem arises. It periodically scans the whole switch fabric to check for link and chip failures. The switch scan is executed every two minutes, sending Read Status service packets to all active switch chips. If errors are reported or no response is received, the faulty component is disabled and the topology is updated. The primary backup node is also scanned. If its daemon is not responsive, an error entry is cut in the AIX error log and a primary backup takeover takes place, where a new primary backup node is selected.

If no errors were detected in the first part of the scan, the daemon checks for nodes waiting to be automatically unfenced. Any node that is off the switch and has its autojoin attribute set in the SDR is checked. If there are no errors in the corresponding node-to-switch link, an unfence is tried. If there are errors, nothing is done, and the daemon will recheck the link to the node in the next scan. If the actual unfence fails for three consecutive periods, probably due to an intermittent problem in that node's adapter, then the primary daemon turns off the autojoin attribute for the node. Automatic node unfencing is a new feature in PSSP 3.1.

Every daemon in the system handles error conditions that occur locally. They handle adapter hardware errors, bad packets received, and microcode errors. If an error is not recoverable or reoccurs beyond its specified threshold, it is considered permanent. A permanent error leaves the TBIC in reset, effectively removing the node from the switch network. When a permanent error is detected the autojoin attribute in the SDR is turned off, so that the node will not unfence at the next scan.

When the fault service daemon terminates abnormally with an unexpected software error condition, or a SIGTERM signal, or a SIGBUS signal, it stops the microcode and puts the TBIC in reset, disabling the adapter. It also turns off its switch_responds attribute in the SDR.

An interesting situation arises when the daemon is killed with the SIGKILL signal. The node does not have a fault service daemon running but continues to be part of the switch fabric. All protocols continue to run normally. The daemon is only needed, and its absence then causes a fence of the node, when the switch configuration is changed (an unfence of another node, for instance), or when `Estart` is run.

The fault service daemon, besides cutting error entries in the AIX log, generates several log files of its own: daemon.stderr, daemon.stdout, worm.trace, fs_daemon_print.file, cable_miswire and flt. The first two files are not used much in PSSP 3.1. The worm.trace and fs_daemon_print.file files trace many of the daemon's activities. The file cable_miswire reports any node-to-switch or switch-to-switch connection apparently miswired. The last file, flt, is definitely the most important log file generated by the daemon. It contains a summary of major events and errors encountered by the daemon.

Also, whenever the daemon faces a serious error, it automatically invokes the `css.snap` utility, which collects the aforementioned logs and other files generated by the daemon, as well as the output of some diagnostics utilities into a single compressed tar image. The description of the fault service daemon's log files, as well as the use of the `css.snap` command, can be found in Chapter 9, "SP Switch Problem Determination Tools" on page 149.

7.2.3 Managing the rc.switch Script

In PSSP 3.1, `rc.switch` is started well into the node's initialization process—more precisely, right after the SP configuration script. You must put any configuration script that expects the switch to be up and running *after* the fsd inittab entry. In particular, you have to be aware that all usual AIX network configuration scripts, like rc.net, rc.tcpip, and rc.nfs, are started before `rc.switch`. This means that you cannot reference the switch in the standard network configuration scripts. For instance, any switch IP route configuration cannot be put in the ODM through `smit mkroute`, but must be added to a later script.

However, Inserting the inittab entry appropriately is not enough. By default, the fsd entry is created with action *once*, which means that it will be run exactly once; and, more importantly, that init will continue with the following inittab entries, without waiting for `rc.switch` to finish. If you want to apply any configuration or to start an application that depends on the switch, besides inserting the corresponding entry after the fsd entry, you need to change the latter's action to *wait*. For example, you could execute `chitab` on all the nodes with the following command:

```
dsh -a chitab \"fsd:2:wait:/usr/lpp/ssp/css/rc.switch\"
```

Whenever the fsd entry has a *wait* action, the script changes its behavior and waits for the node to join the switch before exiting. Changing the fsd entry in all the nodes in itself is not enough, either, since the `rc.switch` does not start the SP Switch, just the node's daemon. For a node to join the switch, the switch has to be running with at least one node, the primary node. The script does not have any logic to start the switch, it assumes that the `Estart` command has already been executed. To prevent the initialization process from being blocked indefinitely when the node is unable to join the switch or the node is the primary node, `rc.switch` waits for a maximum of five minutes for the node to join the switch, after which it exits.

This feature of PSSP 3.1, allied with the automatic starting of the switch described in 8.7, “Automatic Management of the SP Switch” on page 142, effectively guarantees that in most situations, any later configuration scripts will only run after the switch is up.

Attention

The `rc.switch` script generates an rc.switch.log file with the output of its execution. This file, as well as all switch-related log files, can be found in the /var/adm/SPlogs/css directory.

You can directly execute the `rc.switch` script to restart the fault service daemon. This script may be run in a node where the connection to the switch has been lost. When the daemon is restarted, it resets the adapter, which may solve some adapter problems, like, for example, losing the clock after an `Eclock`. Another way to restart the daemon is to execute the `css_restart_node` script, which includes a call to the `rc.switch` script.

7.3 Starting the SP Switch

After the SP Switch adapters have been configured and the fault service daemons have been started, the system is ready for the third and last step in the switch initialization process: the execution of the `Estart` command.

The `Estart` command should be executed from the Control Workstation to start the switch for the current system partition. It can be executed manually or automatically. The features that allow you to automatically start the switch are discussed in 8.7, “Automatic Management of the SP Switch” on page 142.

The `Estart` command initially does some sanity checking:

1. It checks whether any switch board has lost its clock source. It actually checks whether the clock_change attribute in the Switch class has a value of yes whenever a change is sensed in some clock-related hardware variables. The error message output by `Estart` follows:

```
# Estart
Estart: 0028-070 Unable to Estart, the clock source for one or more
switch boards has changed.  Eclock must be run to re-establish the clock
distribution of the switch clock network.
```

2. It checks whether the oncoming primary and oncoming primary backup nodes are up. For a complete discussion of current and oncoming primary and primary backup nodes, see 8.1, “Selecting the Primary and Primary Backup Nodes” on page 131.
3. It checks if the fault service daemon is running on both the oncoming primary and backup nodes.

4. It checks that the oncoming primary node is not isolated. If the oncoming primary backup is isolated, it prints a warning message and another node will be selected as primary backup.

If all the tests pass, `Estart` invokes, using `rsh`, the `Estart_sw` script on the oncoming primary node.

7.3.1 Distributing the Topology File

The `Estart_sw` script starts running in the current partition's oncoming primary node. Its first step is to distribute, if necessary, the file containing the switch's expected topology to all the nodes of the partition.

If there is a file named /etc/SP/expected.top on the primary node, it is used as the partition's topology file. This file can be created to debug the switch, but *must* be removed from the primary node after all tests are done. If such a file does not exist, the primary node extracts the topology file from the SDR and copies it to disk. The name of the file is the one specified in the `Etopology` command with a sequence number appended. You can find out the name of the topology file for all partitions with the command:

```
SDRGetObjects -x Switch_partition topology_filename
```

Then the script checks if the topology file needs to be distributed to the nodes in the partition. This verification is done by matching the number of nodes with switch adapters to the value of the num_nodes_success attribute in the Switch_partition SDR class. This attribute saves the number of nodes that successfully received the current topology file in the last distribution. If the values do not match, the topology file is distributed to all the nodes.

If the topology file is distributed with some node down, the num_nodes_success does not count that node. On the next `Estart`, the topology file is tentatively distributed to all nodes. If the node is still down, the attribute in the SDR is still down by one. This continues until that node is up and running, and thus, the attribute becomes equal to the number of nodes.

Attention

The attribute num_nodes_success is reset by the `Etopology` command to force the distribution of the new topology file. You can also reset this value when you need to redistribute the topology file. This could be useful, for example, when the topology file in one of the nodes has been lost or become corrupted. To reset this attribute, you can use:

```
SDRChangeAttrValues Switch_partition num_nodes_success=0
```

The topology file is distributed using the parallel copy command pcp. To accelerate the distribution of the topology file in large systems, the boot/install servers are used to help in the distribution. The primary node copies the topology file to each boot/install server, and then each server copies its copy to the nodes for which it is responsible. For instance, in a 128-node system, the topology file distribution finishes in approximately 10 seconds. If a boot/install server is unavailable, the distribution of the topology file to that server’s nodes will be done by the Control Workstation.

The boot/install servers check whether there were any distribution errors by checking the timestamp and size of the files. They also inform the primary node which nodes have successfully received a copy. The primary node collects these values and updates the num_nodes_success attribute in the SDR. The distribution process creates a log file on the primary node, named /var/adm/SPlogs/css/dist_topology.log.

In PSSP 3.1, if there is a problem during the distribution of the topology file, the switch initialization continues, giving the following warning message.

```
# Estart
Estart: 0028-061 Estart is being issued to the primary node: sp5n09.msc.itso.ib\
m.com.
dist_to_bootservers: 0028-178 Received errors in distribution of topology file f\
rom bootserver to at least one node.
See /var/adm/SPlogs/css/dist_topology.log on primary node for details.
dist_to_bootservers: 0028-075 Could not distribute the topology file to these no\
des.
They may not come up on the switch network.
sp5n05.msc.itso.ibm.com
Switch initialization started on sp5n09.msc.itso.ibm.com.
Initialized 15 node(s).
Switch initialization completed.
#
```

It is even possible that the problem that caused the distribution failure (an authentication problem, for instance), does not prevent the node from joining the switch: the node tries to use an existing copy of the topology file. But if the topology file is not present or is corrupted, the node will fail to join the switch and its fault service daemon will terminate.

7.3.2 Starting the Worm Code

After the expected topology file has been distributed, the `Estart_sw` script checks to ensure that the oncoming primary node (the node where the script is running) *is not* the partition’s current primary node. If it *is*, `Estart_sw` changes the personality of the fault service daemon running on the current primary node. This action causes the daemon in that node to stop acting as the partition's primary node and start behaving as a secondary node. The current primary backup, if any, also has its personality changed to avoid conflicts between a possible primary node takeover and the `Estart`. If a takeover is currently in progress, the `Estart` finishes with an error.

Finally, the fault service daemon is signaled so it can start the switch. At this point in time, the message `Switch initialization started on <primary node>` is sent to the Control Workstation.

The `Estart_sw` script now waits for the fault service daemon to complete the switch initialization. The initialization is considered complete when the daemon creates the act.top.<pid> file, where <pid> is the PID of `Estart_sw`. This file is created in the usual /var/adm/SPlogs/css directory and its contents are as follows:

```
Number of active node(s) seen by the Worm:
4
Number_of_linksbad: 0
The primary backup node is:
12
The following switch node(s) are active:
8
12
4
0
The topology file used by the Worm:
/etc/SP/expected.top.annotated.4
```

Note that the nodes are identified by their switch node numbers.

Attention

The act.top file is generated by the fault service daemon whenever it initializes the switch. One instance of switch initialization is the execution of the `Estart` command, but it is not the only one. The primary node’s daemon reinitialized the switch under the covers in several situations, generating a different file name for each situation, as follows:

1. When an unrecoverable error occurs during a scan (act.top.0)
2. When a primary node takeover takes place (act.top.1)
3. When an asynchronous error cannot be recovered (act.top.2)
4. When a fence or an unfence faces an unrecoverable error (act.top.3)

Therefore, you may sometimes use the presence of such act.top.n files as an indication of an error in the switch, and use its contents to check the result of the switch reinitialization. We should emphasize that this under-the-covers switch initialization is *not* a reexecution of the `Estart` command, but solely the reexecution of the Worm code described in the following sections.

After the act.top.<pid> is created by the fault service daemon, the `Estart_sw` script reads that file and, from its contents, updates the SDR with the current primary and primary backup nodes. It also displays the number of initialized nodes and the number of uninitialized links, if any, at the Control Workstation. As its last step, the script renames the act.top.<pid> file to topology.data.

The script verifies whether there was a problem during the switch initialization in two ways:

1. It waits a limited amount of time for the creation of the act.top.<pid> file.
2. It checks whether the fault service daemon is still active.

If either condition fails, the script sends the appropriate message to the Control Workstation.

Attention

Although `Estart_sw` sets the time-out in accordance to the size of the system, under extreme conditions that time limit may be insufficient for the primary fault service daemon to finish the switch initialization (for example, if the primary node is heavily loaded). If you run into this problem, you should choose another primary node.

7.3.3 Phase One of Switch Initialization

When signaled by `i_stub_SP`, the primary node's fault service daemon executes the Worm code. This code is divided into two phases: in phase one, the Worm finds the actual topology of the switch fabric; in phase two, the Worm initializes all functional chips and nodes with runtime parameters and routes.

The Worm uses a Breadth First Search (BFS) algorithm to discover the topology of the switch. BFS is a very well-known algorithm to visit all nodes in a graph. The basic idea behind the algorithm is to have a First-In-First-Out queue containing vertices to visit and to mark all visited vertices, as follows:

1. Pick an initial vertex and put it in the queue.
2. Get a vertex from the queue. If the queue is empty, we are done.
3. Mark the current vertex as visited.
4. Traverse a link from the current vertex. If there are no more links to traverse, go to step 2.
5. Check whether the vertex on the other side of the link has already been visited.
6. If it is has not been visited yet, put it at the end of the queue.
7. In any case, go to step 4.

The Worm uses the expected topology file to build the graph. Both switch chips and processor nodes are the vertices of the graph. The initial vertex is the primary node. The node-to-chip connections and the chip-to-chip connections are the links of the graph.

It is important to recognize that only the connections present in the topology file are used to build the graph, and, therefore, will be part of the partition's switch network. This is the basic idea behind the construction of the topology files in a partitioned SP system. Understanding how this works can also be very useful in case you are having trouble with some links or chips. You can disable them by appropriately changing the topology file.

Nodes that are *customer-fenced*, that is, that have the SDR attribute isolated equal to 1 *and* the autojoin attribute equal to 0, are removed from the list of vertices to be visited. These nodes will not join the switch.

For example, consider an SP with a single frame and 8 wide nodes, as shown in Figure 56 on page 125. Suppose that the primary node is node 9.

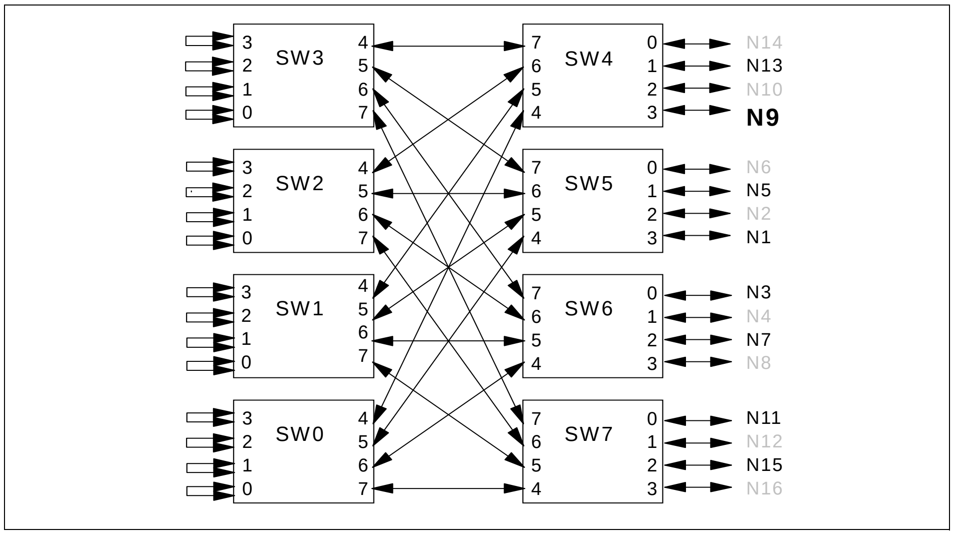


Figure 56. The Switch Board (faint nodes fenced or down)

The Worm code visits the switch board in the following order:

From Node 9 to Chip 4
From Chip 4 to Node 13, Chip 0, Chip 1, Chip 2, Chip 3
From Chip 0 to (Chip 4), Chip 5, Chip 6, Chip 7
From Chip 1 to (Chip 4), (Chip 5), (Chip 6), (Chip 7)
From Chip 2 to (Chip 4), (Chip 5), (Chip 6), (Chip 7)
From Chip 3 to (Chip 4), (Chip 5), (Chip 6), (Chip 7)
From Chip 5 to Node 5, Node 1, (Chip 0), (Chip 1), (Chip 2), (Chip 3)
From Chip 6 to Node 3, Node 7, (Chip 0), (Chip 1), (Chip 2), (Chip 3)
From Chip 7 to Node 11, Node 15, (Chip 0), (Chip 1), (Chip 2), (Chip 3)

Note that switch chips are visited more than once. These revisits, shown within parentheses, are needed to check all links between chips and to check for miswires in multiboard configurations.

The Worm initializes a switch chip on its first visit. A single route to the primary node is given to each, and all error reporting is disabled, except for Re-Timer and Link Synchronization errors. The Worm also informs each chip of its device ID.

If the Worm cannot talk to a chip or a node, the Worm considers the link to that component to be faulty and the corresponding predecessor chip port is disabled. Wrap-plugs that should not be there are detected and the user informed. Also, all chip ports not present in the topology file are disabled.

Attention

Note that if the Worm finds a defective link, it disables *all* ports on each side of the link (unless, of course, other chip or link problems prevent it from getting to both chips). In subsequent switch reinitializations, the Worm will always find the link down, even if that link is then usable (perhaps a faulty cable was replaced). In the Worm's BFS, one of the chips is necessarily visited first. Since the other side of the link is disabled, it will disable the current port. When it gets to the chip on the other side, it will enable its port, but will find the opposite side down, and will disable this port, too.

Thus, to enable a link that was down and then fixed, you need to reset the switch chips by running the `Eclock` command. A similar problem occurs when two partitions are joined without a previous `Eunpartition`. The Worm will fail to reclaim some of the links because ports on the other side were left disabled.

During a revisit, a switch chip returns its device ID to the primary node, and if it is not the expected one, a miswire is suspected. A miswire involving a node is detected during the first visit to the node, since the node's device ID is already known and reported by that node's fault service daemon. During this visit, each node also receives the name of the partition's expected topology file name, as well as a single route back to the primary node.

The Worm now knows the actual topology of the network. It constructs the out.top file, which contains the expected topology file with annotations describing the status of links and devices as detected by the Worm's first phase. You can find the list of possible device and link status in 9.2.4, "The out.top File" on page 160. With the actual topology known, the Worm generates the two disjoint service routes from all nodes and chips back to the primary node.

7.3.4 Phase Two of the Switch Initialization

The Worm's second phase starts by resetting all outstanding errors in all switch chips. Then it reinitializes all chips and nodes, sending the two disjoint routes to the primary node, as well as enabling error reporting and initializing of all thresholds.

The Worm then initializes the time-of-day (TOD) counter in all chips and adapters. Before the TOD initialization proper, the Worm quiesces the fabric, broadcasting to all nodes a command to stop sending packets since such might foil the TOD update algorithm's timing. After finishing the initialization of TOD, traffic on the switch is resumed.

Important

The TOD counter is called a "switch clock" in some SP documentation, and should not be confused with the oscillating signal that drives the switch fabric. It should also not be mistaken with the time of day maintained by AIX, which is synchronized in an SP system through the use of the NTP

The fault service daemon initializes the TOD in all components with *almost* the same value. The TOD counters do not necessarily have the same value because the algorithm used in the SP Switch estimates the distance (therefore, the delay) between switch chips. The values of the counters, though, do not drift apart since the same global oscillator increments them. The switch's TOD is not used by the PSSP and AIX software, but by CSS application debug. The global time implemented by these counters allows a precise (for all practical purposes, at least) ordering of switch events, mainly fault occurrences. Applications have access to the TOD counter through an ioctl.

7.3.5 The Generation of Routes

The Worm's second phase continues with the distribution of the actual switch topology to all the nodes. To reduce the amount of information broadcast through the switch, this distribution is done by sending one or more DB Update service packets to all the nodes. These packets contain only the links that were found down. Each DB Update packet is acknowledged by each node. In SP systems larger than 64-way, the DB Updates distribution is also done in a hierarchical fashion. The highest-numbered node in each frame is used as a distribution server: each of them receives the DB Updates from the primary node and redistributes the updates to the nodes in its frame.

The fault service daemons on all nodes use the DB Updates to create an out.top file by updating the expected topology file. The update will fail and the node will be fenced if the expected topology file (whose name was distributed during the Worm's first phase) does not exist on that node. The acknowledgment of the last DB Update packet is only sent after the node has managed to create out.top. The absence of this acknowledgment packet informs the primary fault service daemon that there is a problem.

All participant nodes now have the actual topology file and are ready to run the RTG code to generate the routes between the nodes.

The goal of the RTG code is to generate four best routes between each pair of nodes. The algorithm uses (a logical) BFS to find all the shortest paths between each pair of nodes. The algorithm counts how many times each link in the network is used by the already chosen paths. For each new pair of nodes, the algorithm chooses four among all the shortest paths so that the *use* of the links is balanced. Therefore, RTG generates balanced, static, shortest routes. If traffic between nodes is random or uniformly distributed, the load on links and chips will be balanced. If RTG cannot generate four different shortest routes between two nodes, it replicates a route(s) found, so the adapter always receives four routes.

Note that the algorithm calculates the best routes between all pair of nodes. This is necessary to obtain a global load balance. Every daemon runs RTG, calculating all best routes. But only the routes between the local node and the other nodes are loaded to the adapter. This could seem a waste of computing cycles, since one node could compute the routes for every other node and distribute them. This used to be the case with the predecessor High-Performance Switch. But the approach was abandoned because of the larger overhead of distributing the routes through the switch, mainly in larger SP systems.

The actual loading of the routes is only done when the Worm broadcasts a Load Routes service packet. Each node then loads the calculated routes into its adapter and sends an acknowledgment back to the primary node.

Dependent nodes are an exception to the preceding discussion. For dependent nodes, the routes are generated by the primary node and sent to the dependent node’s adapter.

The distribution of DB updates and the generation and loading of routes are also carried out whenever the topology of the switch changes; that is during, fences, unfences, and recovery from permanent errors.

During the Worm’s second phase, error recovery is not in effect. The topology of the switch has to be stable so it can be distributed to all nodes and the routes between nodes can be generated. An asynchronous error or the absence of an acknowledgment is considered unrecoverable: the node or the chip in error is disabled and the initialization is retried. The primary fault service daemon attempts to initialize the switch four times. If the switch cannot initialize, the daemon terminates.

The switch initialization is now finished. If the specified primary backup node did not come up, the primary node chooses a new primary backup node and notifies the fault service daemon on that node to change its personality. The primary node updates the switch_responds class in the SDR for each node successfully started: the switch_responds attribute is set to 1 *and* the autojoin attribute is set to 1. Setting the autojoin attribute to 1 after a successful `Estart` is a new feature of PSSP 3.1.

Meanwhile, all fault service daemons enable the IP and User Space protocols for their nodes. In particular, the IP interface css0 is configured as up.

Chapter 8. Managing the SP Switch

In this chapter we go through the actions you take to manage the SP Switch during normal operation. To verify the complete syntax of the commands mentioned in this chapter, refer to the *PSSP: Command and Technical Reference*, SA22-7351.

8.1 Selecting the Primary and Primary Backup Nodes

The *primary node* initializes the switch fabric for its partition, monitors it, processes switch operations, and handles errors reported from the switch fabric. Therefore, you should choose the primary node carefully. Your primary node should not be a node with high switch network traffic. Service traffic in an SP Switch has no priority over application traffic, which means that service traffic can be delayed enough to start generating time-outs in the fault service daemon. In consequence, nonexistent errors in the switch fabric may end up being reported, as well as nodes and chips unnecessarily being taken out of the switch.

The primary node is not a single point of failure on an SP system. There is also a primary backup node for each partition.

The *primary backup node* passively listens for activity from the primary node. The daemon checks every two and a half minutes whether it has been scanned by the primary node. If the primary backup node is not contacted by the primary node during two consecutive check periods, it considers the primary down and assumes its role. This takeover will happen at most seven minutes after the primary node failure (one scan time of two minutes plus two detection periods of two and a half minutes). The primary node takeover involves:

- Selecting itself as the oncoming primary node
- Reinitializing the switch fabric
- Selecting a new primary backup node

The new primary backup node is selected so that it is as far as possible from the primary node, so the chances of a simultaneous failure is as low as possible. This translates into the following:

1. Select a node attached to a different switch board.
2. If there is a single switch board, select a node attached to a different switch chip.

3. Otherwise, select any node.

Attention

The reinitialization of the switch during a primary node takeover is not an `Estart`, as explained in 7.3.2, “Starting the Worm Code” on page 121. A noticeable difference is that, in this case, the daemon generates an act.top.1 file, instead of the usual topology.data file generated by the `Estart` command. You can trace a sequence of primary node takeovers by checking the existence and timestamp of such files, using, for example, the command:

```
dsh -a ls -o /var/adm/SPlogs/css/act.top.1
```

The primary node also watches over the primary backup node. If the primary node detects that the primary backup node can no longer be contacted on the switch fabric, it initiates a primary backup node takeover, selecting a new primary backup node and using the same criteria previously described.

During the period of time between the failure or loss of the primary node and the takeover by the primary backup node, the switch fabric continues to function. Failures that happen while there is no active primary node are detected during the reinitialization of the switch fabric by the new primary node.

Using the `Eprimary` command, the administrator may select two nodes, one to be used as the primary node and the other as the primary backup node. Note that a dependent node cannot be a primary or primary backup node for the SP Switch. The `Eprimary` command selects a default oncoming primary or oncoming backup primary node if one is not specified. The default oncoming primary is the lowest numbered node in the partition, and the default oncoming primary backup is the highest numbered node in the partition.

Until the next `Estart` is issued, the nodes specified in the `Eprimary` command are referred to as the oncoming primary and oncoming primary backup nodes, and are recorded in the oncoming_primary_name and oncoming_primary_backup_name attributes of the Switch_partition SDR class. Once `Estart` completes, the primary_name attribute is updated based on the oncoming_primary_name attribute and the primary_backup_name attribute is updated based on the oncoming_primary_backup_name attribute.

The primary_name and primary_backup_name attributes are also automatically updated when primary node or primary backup node failures occur. Therefore, the primary_name and primary_backup_name attributes reflect the current state of the system.

The following scenario, taken out of the *PSSP: Administration Guide*, SA22-7348, describes the interaction between the current values and the oncoming values. To simplify the example we use node numbers instead of reliable hostnames, which are the actual values stored in the Switch_partition class.

Initially, the primary node and backup primary node fields have the value *none*. In a 16-node system, the oncoming primary node field has the default value of 1 and the oncoming primary backup has the default value of 16.

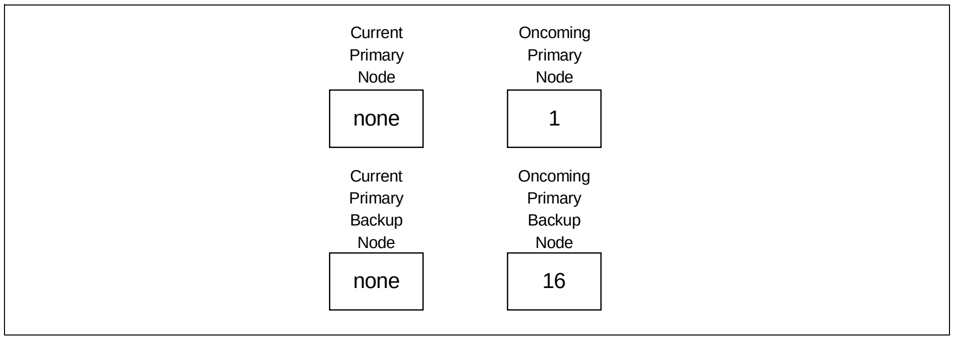


Figure 57. Initial Values

When `Estart` is executed, the node specified in the oncoming primary field becomes the primary node. The node specified in the oncoming primary backup becomes the primary backup.

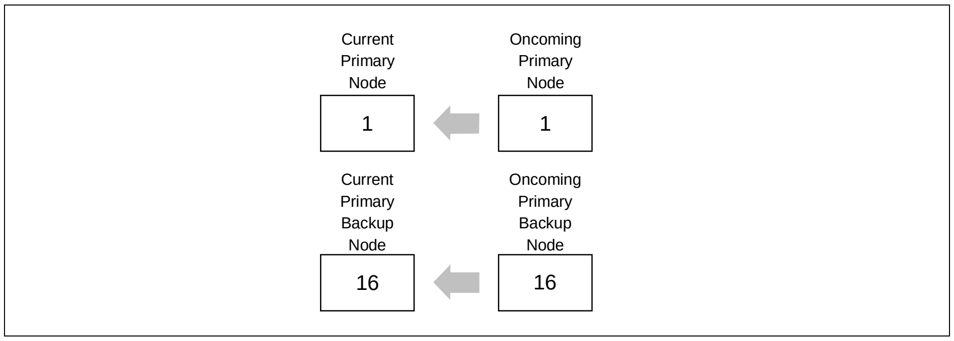


Figure 58. Values after Estart

If the primary backup node fails, the primary node automatically selects a replacement.

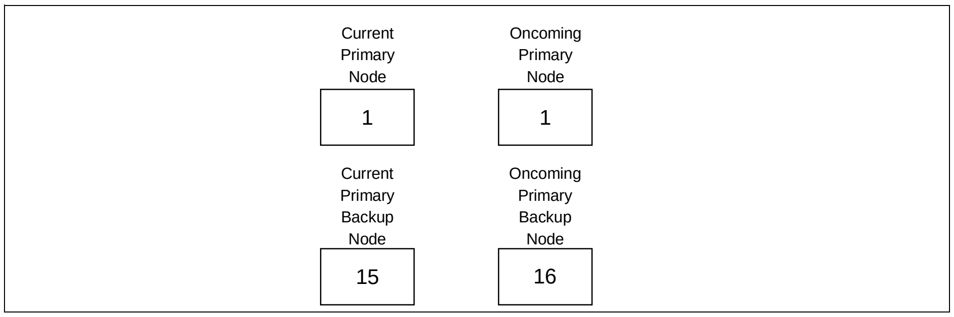


Figure 59. Values after Primary Backup Node Takeover

If the primary node fails, the primary backup node automatically becomes the primary node and a new primary backup is selected.

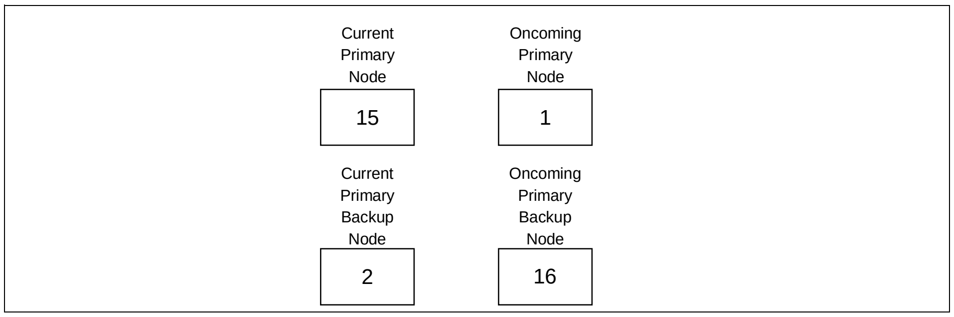


Figure 60. Values after Primary Node Takeover

In summary, primary and primary backup fields reflect the current state of the partition, and the oncoming fields are not applicable until the next invocation of the `Estart` command.

8.2 Establishing the SP Switch Clock

Before starting the SP Switch, you must make sure that the data clock distribution tree is correctly established.

The `Eclock` command establishes a clock distribution tree after the system is powered up or when an alternate clock tree must be selected. It sets the clocking sources that provide the clocking of each switch board. If the current clock distribution tree is not functional, for instance when the current master switch board is down, an alternate clock distribution tree must be selected and the switch reinitialized.

The standard clock configuration trees are specified in clock configuration files. The syntax and contents of such files are explained in Section 6.4, “Specifying the SP Switch Clock Distribution Tree” on page 100. There is one configuration file for each type of SP system. One alternate configuration is also provided in the shipped files.

8.2.1 Verifying the Clock Distribution Tree

All `Eclock` options that change the clock distribution tree, save the info in the Switch SDR class. The relevant content of this class for a two-frame system follows:

```
# SDRGetObjects Switch switch_number clock_input clock_source

switch_number clock_input  clock_source
           1 0                    0
           2 3                    1
```

The clock_input attribute contains the bulkhead jack that is carrying the clock signal that is driving the corresponding switch board. A value of 0 indicates that this board is the master board. The clock_source attribute indicates which switch board is generating the clock signal. This value is a function of the jack being used and the standard SP cabling for the system in question. It is saved in the SDR to help the `Eclock` script in correctly ordering the setting of the clock sources.

The information in the SDR indicates what the clock distribution tree should be. To get the actual clock distribution tree you should issue the `spmon -G -d` command and consult the frame information. A sample (partial) output for the same two-frame system follows:

```
4.  Checking frames

        Controller   Slot 17  Switch   Switch     Power supplies
Frame   Responds     Switch   Power    Clocking   A   B   C   D
----------------------------------------------------------------
  1       yes         yes      on         0       on  on  on  on
  2       yes         yes      on         3       on  on  on  N/A
```

You can see that the Switch Clocking column matches the values of the clock_input attribute of the SDR.

8.2.2 Using the Eclock Command

When you first install your system or after adding a new switch board you should run the `Eclock` command with the `-f` or `-d` flags.

Whenever you need to reestablish your clock distribution tree, for example after powering up your system, you may use the `Eclock -r` command, which extracts the clock distribution data from the SDR. Using `Eclock` with such a flag, instead of the better known `-f` or `-d` flags, will reestablish the most recent clock distribution tree, which may not be the standard clock distribution. Assume that you are having a clock distribution problem, and to work around it you established an alternate distribution. A subsequent `Eclock -d` would not reestablish the appropriate distribution tree: your switch network will not come up complete, or may not come up at all; and you will need to issue another `Eclock`. By having the habit of using `Eclock -r` you would not run into such problems.

When you have a single switch board system, you do not need to run `Eclock` after powering up your system, since there is only one possible clock source, the default. You should be aware, though, that the master switch chip and PLL chosen by the `Eclock` script are not the same as the power-on defaults. This could lead to a behavior difference in the extremely unlikely event of a single PLL or chip failure.

If you are having any hardware problem that is affecting your clock distribution tree, you need to establish a different distribution tree. Your first option is to try the alternate distribution that comes in the clock configuration files. You invoke the alternate distribution by issuing `Eclock -a <clock config file>`.

Important

You should use the following information with extreme care and at your own discretion. The `Eclock` command does take your switch network down, and its incorrect use may leave your network unusable. If you are facing serious clock distribution problems, we advise you to get help from IBM Service.

If the standard alternate tree does not work around the hardware problem you are facing, then you may want to create your own clock configuration file or change the clock source settings of individual switch boards.

For example, assume you have a 112-way system, that is, a system with 7 node switch boards and 4 intermediate switch boards, and that node switch board number 2 is down. Looking at the clock configuration file for such a system (listed in B.2, “Example of a Clock Topology File” on page 250), you can find out that switch board 2 distributes the clock to switch board 1004 in the default configuration. This means that both switch boards 2 and 1004 will be unavailable. The first is down due to whatever hardware problem it is having, the second because it has lost its clock source.

If you run `Eclock` with the alternate clock configuration, you will find out that switch board 1004 is up but there are now two other switch boards with no clock source, intermediate switch boards 1001 and 1003. Now you are in worse shape than before.

Therefore, before trying to establish the alternate clock configuration, you should check whether it will circumvent the hardware that is failing or not. The best solution in this situation is not to use the shipped alternate configuration. Assume, then, that we do not run `Eclock -a`.

The first thing to do is to look into the *annotated* switch topology file for your system, assuming you have an unpartitioned system. Find the external links of the clockless switch board, which is 1004. Then pick the first link that connects that board to an on-line switch board. For instance, the first link in the file for switch board 1004 will do:

```
s 10043 3 s 13 0     E08-S07-BH-J3 to E01-S17-BH-J6
```

So we have a cable running from node switch board 1 jack 6, to switch board 1004 jack 3. At this point we are only interested in where the cable connects to the clockless switch board, which is jack 3. We must transform that value into the corresponding mux value, which is 1. See 6.4, “Specifying the SP Switch Clock Distribution Tree” on page 100. Now we are ready to issue the appropriate `Eclock` command:

```
Eclock -s 1004 -m 1
```

Switch board 1004 should be receiving the clock signal by now, but we are not done yet. `Eclock -s` does not update the clock_source attribute in the SDR. Therefore, it is still indicating that board 1004 is getting the clock signal from board 2, not 1. In this case, and probably in many others, this discrepancy will not make any difference, since this value is used just to order the changes to the clock sources. But to make sure the SDR is correct, we should issue the command:

```
SDRChangeAttrValues Switch switch_number==1004 clock_source=1
```

Even if the switch network was up during the previous procedure, an `Estart` is needed for the primary daemon to recognize the returning intermediate switch board. But because of the way the Worm initializes the switch, not all links of the ISB will be initialized. See 7.3.3, "Phase One of Switch Initialization" on page 124. Therefore, you need to issue an `Eclock -r` before reinitializing the switch. This additional step is only needed because the returning switch board is an intermediate switch board. If it were a node switch board, you would just need to run `Estart` after `Eclock -s`.

Some final notes about changing the clock distribution tree:

- Although the switch hardware allows the use of any tree configuration, `Eclock` does not allow all possible configurations. Namely, the script only works with trees with three levels or less (which in practice is no hindrance).
- Option `-s` works correctly if you are changing the clock source of a switch board that is feeding its clock signal to other boards: it resets not only the board itself but also all its slave boards. But it may not work correctly if some of these slave boards feed the clock signal to other boards. If by any chance you run into this kind of problem, you should know that the correct order to set the sources is: down the tree in a breadth-first-search (BFS) manner.
- If you would like to save your current clock distribution tree in a file, do *not* use `Eclock -c`. The `-c` flag does not work correctly with the SP Switch. What you need to do is construct the clock topology file by hand. To reduce your work, remember that only the Switch Number, Clock Multiplexor (mux) value, and Clock Source switch number columns are actually used by the software.

8.2.3 The Actions of Eclock

Independently of the option you used to specify the clock distribution tree, the `Eclock` command takes the following steps:

1. Kills the fault service daemons on *all* the nodes of the specified switch boards. All options, with the only exception of `-s`, act on *all* switch boards in the system.
2. Sets the clock multiplexors (that is, the clock sources).
3. Starts a power-on-reset of the switch boards and switch chips.
4. Restarts the daemons.
5. Checks whether the affected switch board is receiving a clock signal.

Attention

Since `Eclock` resets all chips, you can use it to solve some `Estart` problems. For instance, if you had a faulty switch-to-switch cable that was eventually replaced, any subsequent `Estart` will still find that link not operational (see 7.3.3, “Phase One of Switch Initialization” on page 124). To solve this problem you run the `Eclock` command, which returns the entire switch fabric to ground zero.

In large SP systems, you could try to avoid bringing down your entire switch network by running `Eclock -s` for the switch board connected to that cable and farthest away from the primary node, that is, farthest away with respect to the BFS algorithm. This could be complicated to figure out, depending on your system configuration. You may try, then, the following approach:

1. Your system does not have ISBs: change your primary node to a node in one of the switch boards connected to the cable; `Eclock` the switch board on the other side of the cable; run `Estart`.
2. Your system has ISBs: change your primary node to a node that is *not* in the node switch board connected to the cable; `Eclock` the node switch board connected to the cable; run `Estart`.

Observe that with the preceding step we are avoiding the `Eclock` of an intermediate switch board. We leave as an exercise to you to figure out why the `Eclock` of an ISB will *never* do you any good.

A final note: always keep in mind the clock configuration tree. If the switch board you are resetting feeds its clock to other boards, these other boards have to be reset, too. So, in some situations, a piecewise approach may not bring you any significant gain.

8.3 Starting the SP Switch

Once you have established the clock source and your primary and primary backup nodes are running, you can start the SP Switch by issuing the `Estart` command. When you are using the Switch Admin daemon, described in 8.7, “Automatic Management of the SP Switch” on page 142, `Estart` is automatically invoked by that daemon in most situations.

Starting the switch with only the primary and primary backup nodes up is enough in PSSP 3.1 because of its automatic unfence feature. Unless an irrecoverable error occurs or you explicitly fence a node, all nodes automatically join the switch shortly after their fault service daemon is started.

You can issue the `Estart` with the `-m` option, which specifies that the Emonitor daemon should be started. But in PSSP 3.1, the Emonitor subsystem is no longer needed, because of the new Switch Admin daemon and the automatic unfence features. However, if you do not run the Switch Admin daemon you may still wish to use the Emonitor subsystem. And if you are using a primary node at PSSP 2.4 or earlier in a coexistence environment, you may still need to use the Emonitor subsystem.

8.4 Removing a Node from the SP Switch Network

The `Efence` command isolates nodes from the switch network. Once a node is fenced, it cannot communicate with other nodes on the switch network, nor cause errors on the network. In PSSP 3.1 you only need to fence a node when you need that node to be running, but out of the switch network (for instance, to do some software maintenance). Before PSSP 3.1, you needed to fence a node that was going to be rebooted or powered off to prevent such nodes from impacting other nodes. This is no longer necessary since the fault service daemon, when exiting, disconnects from the switch fabric by putting the TBIC in reset.

The `Efence` has an `-autojoin` option, which allows the node to automatically join the switch once the node is again operational. In PSSP 3.1, this option is no longer needed due to the automatic unfence feature. Note further that in PSSP 3.1, a node that is fenced with autojoin will automatically join the switch within two minutes. However, if the primary node is at PSSP 2.4 or earlier, the autojoin option enables the nodes in the argument list to be fenced and to automatically rejoin the current switch network if the node is rebooted, the fault service daemon is restarted, or an `Estart` command is issued.

8.5 Adding a Node to the SP Switch Network

The `Eunfence` command adds a node to the switch network a node that was previously removed from it. If your primary node is at PSSP 3.1, you only need to unfence a node in two situations:

- The node was previously removed through the `Efence` command.
- The node was previously removed because an unrecoverable error was detected.

Important

Coexistence Requirements:

Note that the automatic unfence is a feature of the fault service daemon running in the primary node and works even when all other nodes are at PSSP 2.4 or earlier. But for a pre-PSSP 3.1 fault service daemon to be able to coexist with this new feature, the node should be at one of the following PTF sets:

- PSSP 2.1 - PTF set 30 (warning: a coexistence environment with nodes at PSSP 2.1 is not supported)
- PSSP 2.2 - PTF set 17
- PSSP 2.3 - PTF set 9
- PSSP 2.4 - any

The fix provided in the aforementioned PTFs makes the fault service

If your primary node is at PSSP 2.4 or earlier, you also need to unfence a node when the node has been rebooted or the fault service daemon has been restarted without a previous `Efence -autojoin` command. But if you are running the Emonitor subsystem, it automatically attempts to unfence a node when the node is rebooted.

`Eunfence` first distributes the current topology file to the nodes before they are unfenced. The `Eunfence` command, unlike `Estart`, unconditionally distributes the topology file to all nodes being unfenced. If the topology file distribution to a node fails, that node is not unfenced.

8.6 Stopping the SP Switch

There is no single command to stop the SP Switch. If you need to take down the whole switch network without shutting down your nodes, you must kill all the fault service daemons in the partition. You can use an undocumented command, `css_cdn`, to do that from the Control Workstation:

```
dsh -a /usr/lpp/ssp/css/css_cdn
```

If your primary daemon is at PSSP 2.4 or earlier, the preceding procedure could leave some of your nodes fenced. You should run the `Equiesce` command to avoid this problem. For the same reason, you should also run the `Equiesce` command before shutting down one or more nodes if your primary and primary backup nodes are not the first ones to be shut down. You must be careful, for example, with cluster shutdowns.

The `Equiesce` command changes the personality of the primary and the primary backup nodes to secondary, effectively disabling switch error recovery and primary node takeover. After an `Equiesce,` data still flows over the switch.

You must issue an `Estart` to reestablish switch recovery and primary node takeover after running `Equiesce`.

8.7 Automatic Management of the SP Switch

In this section we discuss the use and internals of the Switch Admin daemon. This daemon is a new feature of PSSP 3.1 and, together with automatic unfence, replaces the Emonitor daemon available in previous releases. We start this section briefly discussing the use of Emonitor, followed by a thorough presentation of the PSSP 3.1 Switch Admin daemon. We finish the chapter describing situations where there is still a need for your intervention to keep the switch running.

8.7.1 Managing the Switch Before PSSP 3.1

The Emonitor daemon was released in PSSP 2.2 to solve two common problems in administering the SP Switch:

1. Nodes were automatically fenced after a shutdown without a previous `Efence -autojoin`. You had to explicitly unfence the node after it was started.
2. Switch initialization was not automatic. If you had your primary and primary backup nodes down, you had to explicitly `Estart` the switch after the nodes had come up.

The pre-PSSP 3.1 Emonitor daemon runs on the Control Workstation and monitors host_responds through `spmon`. Whenever a node comes up, it sets a three minute timer, waiting for other nodes to come up. Each time another node comes up, the timer is reset to three more minutes, up to a maximum of 12 minutes. When the timer triggers, it checks whether any node listed in the /etc/SP/Emonitor.cfg file has host_responds on and switch_responds off. If there is any, it tries to bring it back to the switch network. If the primary node is coming back and the primary backup shows as active in the SDR, Emonitor takes no action—a primary node takeover should take place. If the preceding condition is false and the primary or the primary backup nodes are coming back, the monitor attempts an `Estart`. If a secondary node that has autojoin off is coming back, the daemon attempts to unfence the node.

One problem that Emonitor does not address is the need for an explicit `Estart` after your system is powered up. The Emonitor daemon itself can only be started through the `Estart -m` command. After this manual `Estart`, nodes rebooting automatically join the switch, at least in most normal situations.

Also, if you need to keep a node fenced after a reboot, you need to edit Emonitor's configuration file and remove the node from it. Otherwise the node will get unfenced after the reboot, even if you fence it without `-autojoin`. A further inconvenience of the Emonitor daemon is that you have to set up its configuration file before using it.

In PSSP 3.1 the behavior of the Emonitor daemon is slightly changed when your primary daemon is at PSSP 3.1. The Emonitor daemon is aware of the new automatic unfence feature and does not attempt to unfence a node in any situation. Thus, a node that is explicitly fenced, either by you or by the system after an unrecoverable error, remains fenced, even if the node is in the daemon's configuration file.

The PSSP 3.1 Emonitor also checks whether the Switch Admin daemon is active, and if so, does not attempt an `Estart` when one is needed.

8.7.2 The Switch Admin Daemon

The objective of the Switch Admin daemon is to automate the setup, the startup, and the handling of events that affect the switch. The ultimate objective behind this implementation is to make the switch behave, from a management point of view, more and more like a LAN. In PSSP 3.1, the Switch Admin daemon handles the following situation:

Start the switch in any partition where no node is acting as the primary node. This occurs after a power up or reboot of all nodes in a partition, or anytime the primary and primary backup are included in the set of nodes that are rebooted. The Switch Admin daemon runs `Estart` whenever the oncoming primary node shows itself as available through host_responds.

The Switch Admin is the program cssadm and runs on the Control Workstation. The Switch Admin daemon is started out of inittab and is SRC controlled. The subsystem name for the daemon is swtadmd. There is a single subsystem for the entire SP system, that is, a single daemon administers all system partitions.

The Switch Admin daemon has a configuration file, /spdata/sys1/ha/css/cssadm.cfg. It is shipped enabling node recovery with the following contents:

```
Node 1
```

If you need to disable the daemon, you should edit the file, changing the `1` to a `0`. Next you should stop the daemon with the command:

```
stopsrc -s swtadmd
```

You may also stop the subsystem and remove the swtadmd entry from inittab.

The daemon generates several log files:

- cssadm.debug contains entries for each event received and how it was handled.
- cssadm.stderr contains unexpected error messages received by the daemon while performing an external command, like `Estart`.
- cssadm.stdout contains information messages received by the daemon while performing external commands.

8.7.3 The Implementation of the Switch Admin Daemon

The daemon uses the High Availability Event Management subsystem to get notification about node-related events. The daemon registers for events for the following resource variables:

- IBM.PSSP.Response.Host.state, which indicates if a node has connectivity over the en0 LAN adapter, as determined by the High Availability Topology Services subsystem. A value of 1 indicates connectivity; 0 indicates no connectivity. This variable represents the host_responds attribute in the SDR.
- IBM.PSSP.Membership.LANAdapter.state, which indicates if a LAN adapter in a node has connectivity, as determined by Topology Services. A value of 1 indicates connectivity; 0 indicates no connectivity. This variable, instantiated for adapter css0, does not represent the switch_responds SDR attribute, but only represents IP connectivity over the switch. In most situations this causes no behavioral difference, but you should always keep in mind the distinction between those states.

The daemon registers the following events with Event Management:

1. A node comes online: IBM.PSSP.Response.Node.state (NodeNum=*) == 1.

2. A node loses connectivity over the switch network, for instance when the node is shut down or the fault service daemon terminates: IBM.PSSP.Membership.LANAdapter.state (NodeNum=*;AdapterType=css;AdapterNum=0) == 0.

By monitoring these events, the daemon is able to intervene in many of the cases where normal primary node takeover is not effective.

The following are the tests and resulting actions triggered by the occurrence of any of the above events:

1. If the current primary node has just lost its IP connectivity or there is no current primary node in the partition (primary_name==none in the Switch_partition SDR object):
   - If the primary backup node has switch_responds on, the daemon takes no action as normal primary takeover should occur.
   - If the primary backup node has switch_responds off and the oncoming primary node has host_responds on, the daemon attempts an `Estart`.
   - If the oncoming primary has host_responds off, no action is taken. The daemon waits until the oncoming primary comes up.
2. If the oncoming primary node has just come on-line:
   - An `Estart` is attempted.

If the `Estart` fails, the daemon takes no recovery action. It waits for another significant event to occur.

In addition, when the daemon starts, it verifies, in every partition, whether the current primary node has no switch connectivity or there is no current primary node. If so, it simulates the corresponding event and executes the algorithm previously described.

Note that the actions taken when test 1 above is true are also taken whenever any node has come up and there is no current primary node. For example, assume you are doing a cluster start-up and your oncoming primary node comes up before your oncoming primary backup node. The daemon is notified that the oncoming primary node is up and attempts an `Estart`. The command fails because the oncoming primary backup is still down. But when the oncoming primary backup comes up, the first condition still holds and the daemon attempts a second `Estart`, which this time should succeed.

Notice also that an `Estart` is attempted whenever the oncoming primary node comes on-line. At a first glance, it could seem unnecessary to reinitialize the switch if it is up and running. But if the switch is up and the oncoming primary node comes online, a primary node takeover may have taken place. And one would want to have the chosen primary node to reassume its functions after solving whatever problem caused the takeover to happen in the first place.

Also, by issuing an unconditional `Estart`, the daemon overcomes some possible anomalies between the actual state of the system and the SDR. Namely, the attribute primary_name (or primary_backup_name) is *not* set to none when the fault service daemon on the current primary (or primary backup) node goes down, only when `rc.switch` is run in that node. Thus, just checking the SDR attributes is not enough to assure that the switch is really up. A simpler solution, instead of going over several checks, is just to `Estart` when the oncoming primary comes up.

8.7.4 Management Tasks Not Yet Automated

You should realize that the Switch Admin daemon is intended to be neither a switch diagnosing nor a problem solving tool. It does not test for any condition that could lead to an `Estart` failure nor does it have any built-in logic to recover from a switch failure. Following are some events not handled by the daemon:

1. If the oncoming primary node does not come on-line or its fault service daemon does not run, the switch does not initialize. The daemon does not take any steps to detect and overcome such problems, like choosing another primary node.
2. The daemon does not check whether the system was just powered on and, therefore, an `Eclock` may well be needed before the `Estart`. Therefore, when powering up your system, you should: power on the frames and switches; run the `Eclock` command; power on your nodes; and let the Switch Admin daemon `Estart` your system.
3. The daemon does not check for any clock distribution problem in the system, for instance whether the master switch board is down.

Part 3. SP Switch Problem Determination

Chapter 9. SP Switch Problem Determination Tools

In this chapter we describe some of the tools available for problem determination of the SP Switch. We briefly discuss material discussed elsewhere, like the use of the AIX error log and the monitoring tools available in PSSP. But we also go into detail describing the contents of several SP Switch log files and diagnosing utilities.

9.1 Error Logging

Error logging in the SP System uses the AIX Error Log facilities, the Berkeley Software Distribution (BSD) syslog, and several other log files. Going through all those logs to diagnose a problem could be quite a challenge. But the latest releases of PSSP have been improved so all detected errors for the SP Switch and the SP Switch adapter have an entry in the AIX error log.

In addition, some of the SP Switch logging events may trigger the execution of the `/usr/lpp/ssp/css/css.snap` script to generate a snapshot of the log files on the node where the error was reported. See section 9.4, “SP Switch Utilities” on page 175 for more information on the `css.snap` script.

Especially in larger systems, having to go though the AIX error log on all nodes and on the Control Workstation is not an easy task. To make problem determination easier, PSSP 3.1 introduces a centralized error log in the Control Workstation that contains a summary of all switch-related error log entries, as described in 9.2.1, “The Centralized Switch Error Log” on page 152. This summary log file should be used as the starting point for any SP Switch problem solving.

9.1.1 Viewing Error Log Information

Enter the following command to view all the SP Switch adapter error log entries in all nodes:

```
dsh -a errpt -a -N css
```

Enter the following command to view all the SP Switch error log entries in all nodes:

```
dsh -a errpt -a -N Worm
```

Enter the following command to view all the SP Switch adapter diagnostics log entries in all nodes:

```
dsh -a errpt -a -N css0
```

A table with all switch-related error log entries can be found in *PSSP: Diagnosis Guide*, GA22-7350. Each entry contains a brief explanation of the error, possible causes, and what actions could be taken to recover from it.

You can also use the AIX Error Notification Facility to be notified of errors reported in the AIX error log as soon as they occur. This facility causes the execution of an ODM-defined method when a particular error occurs. For example, you could create a method which would send you an e-mail whenever the fault service daemon cuts an error log entry (en_resource="Worm"). Refer to the *PSSP: Diagnosis Guide*, GA22-7350, for more information.

9.1.2 Log Filesystem Size Consideration

All SP log files are kept in the /var filesystem. The size of the log files grows with time, and if the /var filesystem reaches 100%, the system runs into problems. To prevent this situation, you should monitor the usage of /var. For instance, you could use the High Availability Problem Management subsystem, as exemplified in the `/usr/lpp/ssp/install/bin/pmandefaults` script, generating an e-mail whenever the filesystem is more than 95% full.

To reduce the usage of /var you may trim log files or remove old log files. AIX has a default crontab entry that removes all hardware error entries after 90 days and all software error entries after 30 days. SP-related log files in /var/adm/SPlogs are cleaned up on the nodes by the script `/usr/lpp/ssp/bin/cleanup.logs.nodes`. Log files are cleaned up on the Control Workstation by `/usr/lpp/ssp/bin/cleanup.logs.ws`.

When you are frequently running some diagnosing scripts, make sure you save the log files in /var/adm/SPlogs/css. They may be automatically overwritten or deleted to generate free space for new log files. The script `css.snap`, for example, checks the filesystem size at startup and cleans up old files.

9.2 SP Switch Log Files

The SP system also uses some SP-specific log files. Some logs reside on the Control Workstation (CWS) only, and some reside only on the SP nodes.

Others reside on both. Table 4 summarizes and shows the location of the logs you can reference when diagnosing SP Switch problems.

Table 4. SP Switch Log Files

Log File	Type of Message	Location
/var/adm/SPlogs/css/act.top.n	`Estart_sw` output, renamed to topology.data after SP Switch initialization	Primary
/var/adm/SPlogs/css/cable_miswire	Miswire information	Primary
/var/adm/SPlogs/css/dist_topology.log	System error messages occurring during the distribution of the topology file to the nodes	Primary
/var/adm/SPlogs/css/dtbx.trace	SP Switch adapter diagnostics output	Nodes
/var/adm/SPlogs/css/Eclock.log	Eclock related information	CWS
/var/adm/SPlogs/css/Ecommands.log	Log entries of all E-commands	CWS
/var/adm/SPlogs/css/flt	Hardware error conditions found on the SP Switch, recovery actions taken by the fault service daemon and general operations that alter the SP Switch configuration	Primary
/var/adm/SPlogs/css/fs_daemon_print.file	Information from the fault service daemon	Nodes
/var/adm/SPlogs/css/out.top	SP Switch link information	Nodes
/var/adm/SPlogs/css/rc.switch.log	fault service daemon Initialization messages	Nodes
/var/adm/SPlogs/css/router.log	SP Switch routing information	Nodes
/var/adm/SPlogs/css/summlog	Summary records for CSS events	CWS
/var/adm/SPlogs/css/worm.trace	Initialization information	Nodes
/var/adm/SPlogs/css/logevnt.out	Output of css.logevnt, invoked as an ODM error notification object method	CWS
/var/adm/SPlogs/SPdaemon.log	Messages generated by system daemons, including hardware errors	CWS
/var/adm/SPlogs/spmgr/spgrd.log	SP SNMP Agent messages	CWS and Nodes

Table 5 shows the message identifiers used by PSSP for switch-related error or informational messages. Further information about error messages in the SP log files may be found in the *PSSP: Messages Reference*, GA22-7352.

Table 5. SP Switch-Related Message Identifiers

Identifier	Message
0028-nnn	Switch support
2510-nnn	Switch fault service daemon
2511-nnn	Switch table support
2543-nnn	Switch admin daemon

9.2.1 The Centralized Switch Error Log

A new daemon, css.summlog, runs on the Control Workstation and listens for log change events generated by the ODM method running on all nodes, independent of partition boundaries. The daemon is SRC-controlled and its subsystem name is swtlog.

Use the summary CSS error log as the *primary starting point* to diagnose SP Switch problems. The centralized summary log resides in the /var/adm/SPlogs/css/summlog file on the Control Workstation. Any SP Switch or SP Switch adapter error that has an entry cut in the AIX error log in any node triggers the generation of a summary record. Each record is appended in the order received by the Control Workstation. The fields of the summary record are shown in Table 6 on page 153.

The summlog file allows you to identify all nodes that are affected by a fault, and shows the failure symptoms on each node. The log is formatted for processing by user scripts. It will be the basis for automated error analysis mechanisms in future releases. The log cleaning scripts include this log to keep it to a reasonable size.

Important

You should check the summlog periodically to find out problems on the switch that were automatically recovered or do not have any perceptible consequences.

An example of the summlog file follows:

```
080310081998 sp5n05 N sp5cw0 281 SP_SW_RCVLNKSYNC_RE
080311181998 sp5n01 Y sp5cw0 298 TB3_LINK_RE
080319091998 sp5n09 N sp5cw0 195 SP_SW_RSGN_BKUP_RE
080319091998 sp5n05 N sp5cw0 285 SP_SW_RSGN_PRIM_RE
080319101998 sp5n09 N sp5cw0 197 SP_SW_UNINI_NODE_RE
080319101998 sp5n09 N sp5cw0 198 SP_SW_UNINI_LINK_RE
080319101998 sp5n09 N sp5cw0 199 SP_SW_SDR_FAIL_RE
```

Table 6 provides descriptions of the fields that compose each summary record:

Table 6. Summary Record Fields

Name	Description
time	The error log time for the entry
node	The reliable hostname of the source node
`css.snap` dump	Y indicates that a `ccs.snap` dump was taken, N indicates no dump was taken.
syspar_name	The system partition name
index	The error log index from the reporting node
label	The error log entry label field

9.2.2 The flt File

The flt file is found in the /var/adm/SPlogs/css directory on any node that is or was a primary node. The flt file is used to log hardware error conditions on the switch, recovery actions taken by the fault service daemon and general operations that alter the switch configuration.

The messages in the flt file can be informational (i), notification (n), or error messages (e).

In the following paragraphs we show some examples of the contents of the flt file. One general observation is about the *device id*, shown in several messages. It is either a switch node number or the chip id plus 100000. For example, an id of 100015 indicates chip 5 in switch board 1. The description in this section and the following ones assume an understanding of the workings of the SP Switch. Refer to Chapter 3, “Communication Network Hardware” on page 13, Chapter 4, “Communicating with the SP Switch” on page 49, and Chapter 7, “Initialization of the SP Switch” on page 111 for more information.

1. switch chip or ports disabled

   During switch initialization the fault service daemon tries to communicate with all switch chips and nodes in the configuration. If for any number of reasons a switch chip fails to respond or returns an error, the daemon disables the switch chip. This is noted in the flt file with the following entry:

   ```
   (i) 07/24/98 17:34:32 : 2510-798 Disabling Switch chip - device_id =
   100016
   ```

   In this example the entire device is disabled (switch chip 16). Often only the port or ports that are reporting problems are disabled. When this happens you find the following entry in the flt file (in this example, port 3 on switch chip 5 is disabled):

   ```
   (n) 07/30/98 17:12:52 : 2510-743 Disabling port 3 (jack 7) of chip 5 on
   the switch in slot 17 of frame 1
   ```

2. Switch initialization error status

   When a switch chip reports an error to the fault service daemon during switch initialization, it places the following entry in the flt file:

   ```
   (n) 07/29/98 17:38:02 : Switch and Adapter Error bits found during
   switch initialization.
   (i) 07/29/98 17:38:02 : Device ID = 100014
   (i) 07/29/98 17:38:02 : 2510-793 First  Error Capture Register  =
   000008.
   (i) 07/29/98 17:38:02 : 2510-741 Second Error Capture Registers =
   00000000 00000040 00000000 00000000 00000040 000000
   ```

   In this example, switch chip 14 reported the error status packet. Decoding the error capture registers should give you some idea of what types of errors the switch chip is detecting.

   In the following example, the value in the First Error Capture Register is 0x000001. This signifies that this is the first `Estart` since the switch was reset.

   ```
   (i) 07/29/98 17:36:02 : Device ID = 100010
   (i) 07/29/98 17:36:02 : 2510-793 First  Error Capture Register  =
   000001.
   (i) 07/29/98 17:36:02 : 2510-741 Second Error Capture Registers =
   00000000 00000000 00000000 00000000 00000000 000001
   ```

   See 3.4.5, “Error Isolation” on page 34 for the explanation the error registers. See also A.2.1, “First Error Capture Register” on page 242 and A.2.2, “Second Error Capture Register” on page 243.

3. Switch error recovery

Recovery is also part of the fault service daemon. When Error/Status packets are received at the primary node during normal switch operations, the daemon will attempt to clear up the problem on the switch chip. This involves sending initialization packets back to the switch chip in an attempt to reset the error. The reset sequence can be found in the flt file as shown:

```
(i) 08/05/98 14:39:58 : 2510-744 SP Switch error recovery initiated.
(i) 08/05/98 14:39:58 : 2510-793 First  Error Capture Register  =
000008.
(i) 08/05/98 14:39:58 : 2510-741 Second Error Capture Registers =
00000000 00000000 00000000 00000000 00000010 000000
(i) 08/05/98 14:39:58 : 2510-740 Packet Sequence Number = 219 Switch
Time-Of-Day = 0x800005a11fa0dd1f
(i) __Date__ __Time__ _Msgid__ Reg ___Location___ Ch Po
__Type_of_SECR_Error__
(n) 08/05/98 14:39:58 : 2510-778 FEC E01-S17-BH-J07  5  3 Send Link Sync
Failure
(i) 08/05/98 14:39:58 : 2510-759 Error count threshold has been
exceeded, initiating recovery action(s).
(n) 08/05/98 14:39:58 : 2510-743 Disabling port 3 (jack 7) of chip 5 on
the switch in slot 17 of frame 1
(n) 08/05/98 14:39:58 : 2510-749 Turning off switchResponds bits for
node 0 in the SDR
(i) 08/05/98 14:39:58 : 2510-742 Transmitting a Reset Error packet;
        Route = 0x4500000000000001,
        Recv Error Resets = 0x0000, Sender Error Resets = 0x0002,
        Central Queue Resets = 0x00, Service Logic Resets = 0x00
```

Attention

It is not uncommon to find packets with error capture registers containing all zeroes. They are responses to the recovery code when it resets errors on the switch chips.

4. Broadcast failure

During switch initialization, the primary fault service daemon broadcasts several commands to all nodes. When the node finishes executing the command successfully, it sends an acknowledgment to the primary node. The primary then tracks the acknowledgments to determine if everyone has responded.

One of the possible errors during a broadcast operation follows:

```
(i) 07/28/98 16:24:08 : 2510-606 A switch Error/Status service packet
was received during a broadcast operation.
```

It says that while fault service daemon was looking for node responses it received an "error status" packet from some component. This indicates that the switch fabric is not stable and must be reinitialized.

Another possible error is the following:

```
(i) 07/28/98 16:24:08 : 2510-607 Timed-out waiting for acknowledgments
from broadcast operation.
```

This message tells you that one or more of the nodes failed to acknowledge the node command sent to it. This should lead you to believe that there is some type of problem with the node or nodes that failed to respond.

5. Switch initialization as a recovery action

   The switch may be reinitialized by the fault service daemon to recover from error conditions. The following message tells you that the fault service daemon made a decision to reinitialize the switch fabric:

   ```
   (i) 07/28/98 16:24:08 : 2510-816 Switch recovery timed-out waiting for a
   Error/Status packet from switch. Estart will be executed.
   ```

   Note that, contrary to what the message says, the actual `Estart` script will not be executed, but only the fault service daemon code that initializes the switch.

6. Switch Initialization as a command (`Estart`)

   Whenever the `Estart` command is issued, it is noted in the flt file. The entry for this is as follows:

   ```
   (i) 07/24/98 20:11:59 : 2510-744 Estart initiated.
   (i) 07/24/98 20:11:59 : The date and time  = Fri Jul 24 20:11:59 1998
   ```

   `Estart` contacts the fault service daemon on the primary node to start the switch.

7. Node drops off the switch

   The Receiver Link Synchronization Failure messages are generated whenever the primary daemon detects a node dropping off the switch.

   ```
   (i) __Date__ __Time__ _Msgid__ Reg ___Location___ Ch Po
   __Type_of_SECR_Error__
   (n) 08/12/98 11:06:47 : 2510-767 SEC E01-S17-BH-J07  5  3 Recv Link Sync
   Failure
   ```

8. Primary node takeover

   When the backup node takes over control of the switch, the following entry is placed in the flt file:

```
(i) 07/24/98 17:34:08 : 2510-812 Backup node starting Primary node
take-over.
(i) 07/24/98 17:34:09 : The Primary backup is node sp5n01
```

The message also indicates what node was chosen as the new primary backup. In the case shown the switch node name is sp5n01. When the takeover is successfully completed, the following entry is noted in the flt file:

```
(i) 07/24/98 17:34:35 : 2510-813 Primary backup node completed Primary
node take-over.
```

9. `Eunfence` operation

   There are three phases to an unfence operation. The first is the reenablement of the switch port connected to the node. The second is reinitializing the node onto the switch fabric. The third is broadcasting to all nodes that this node has joined the switch. Once the switch port is enabled, the following message is placed in the flt:

   ```
   (i) 07/24/98 20:20:31 : 2510-815 Port enable for unfence of node sp5n01
   completed.
   ```

   This should tell you that the switch hardware successfully unfenced the switch port on the chip and the SP Switch adapter is synchronized with the switch fabric. Once the other two phases complete, the following entry is placed in the flt file:

   ```
   (i) 07/24/98 20:20:36 : Node sp5n01 UnFenced.
   ```

   If any of the operations fail, the following message is shown in the flt file:

   ```
   (e) 07/24/98 20:11:59 : 2510-894 Error found in handleUnfence()
   ```

10. `Efence` operation

    Just like the unfence operation, the fence operation has three phases. The first is to disable the node switch adapter. The second is to disable the associated switch chip port. The last piece to be done is to notify the operational nodes that the node is no longer attached to the switch. Once phases 1 and 2 are completed, the following entry is made in the flt file:

    ```
    (i) 07/27/98 09:53:11 : 2510-814 Port disable for fence of node sp5n13
    completed.
    ```

    When phase 3 is completed, the following message is added to the flt file:

    ```
    (i) 07/27/98 09:53:14 : Node sp5n13 Fenced.
    ```

11. Switch scan failure

    The fault service daemon on the primary node checks the health of the switch network every two minutes. It sends a *read status* packet to all switch chips and the backup node to verify their status. If the daemon receives a bad error status from any of these devices, recovery actions are taken to remove the faulty component. When this happens the following entry is noted in the flt file:

```
(i) 07/28/98 16:24:08 : 2510-906 Scan detected a problem with device
100015
```

12. Primary node switch port disabled

    Whenever `Estart` is run and the fault service daemon finds its switch port disabled it issues the following message and terminates:

```
(n) 08/06/98 19:19:13 : 2510-820 Primary's link to the switch network is
not in the initialized state. Estart could not be executed.
(n) 08/06/98 19:19:13 : 2510-823 The fault service daemon process has
exited.
```

13. Node personality changes

    Some reasons for personality change follow.

    If a new primary was assigned, a subsequent `Estart` will show the following on the new oncoming primary node:

```
(i) 07/24/98 20:01:14 : 2510-811 Fault service daemon's personality has
been changed to Primary.
```

    If the primary fault service daemon determined that it could not control the switch fabric and resigned from being the primary node, the following will be informed:

```
(i) 07/24/98 18:38:05 : 2510-913 Resigning from being Primary node.
(i) 07/24/98 18:38:05 : 2510-810 Fault service daemon personality
changed to Secondary.
```

    The primary fault service daemon was unable to change a node's personality to backup, generating the following messages:

```
(i) 07/24/98 19:38:55: 2510-819 Changing remote node's personality to be
the Primary backup failed. Return code equals 1. Estart will be
executed.
```

14. Route generation

    The fault service daemon generates two types of routes, processor routes and service routes. Processor routes are used for node-to-node communication. Service routes are used for switch chip to primary node communication. The following message in the flt file denotes failure in the service route generation process on the primary node:

```
(e) 07/30/98 10:55:30 : 2510-712 generate_service_routes() failed with
rc= 123
```

The following message could appear on any node when there is a failure on the node-to-node route generation:

```
(e) 07/30/98 11:49:30 : 2510-714 generate_processor_routes() failed with
rc= 124
```

In either case the route generation return code should help determine what type of problem is at hand. See C.2, "Return Codes from Route Table Generation" on page 255. The following entries indicate that the fault service daemon was able to generate the specified routes:

```
(i) 07/30/98 10:49:30 : Service routes generated.
```

```
(i) 07/30/98 10:49:30 : Processor routes generated.
```

Another route generation-related message follows. It indicates that the daemon was able to successfully download the route table to the switch adapter.

```
(i) 07/30/98 10:49:31 : Processor routes down loaded successfully.
```

15. Fault service daemon signals (SIGBUS, SIGTERM, SIGDANGER)

    The fault service daemon processes a number of different signals. The following may be found in the flt file:

    ```
    (n) 07/30/98 11:14:57 : 2510-195 The fault service daemon got a SIGTERM
    signal.
    ```

    ```
    (n) 07/30/98 11:14:57 : 2510-196 The fault service daemon got a
    SIGDANGER signal, probably because the system is starting to get low on
    pg space.
    ```

    ```
    (n) 07/30/98 11:14:57 : 2510-197 The fault service daemon got a SIGBUS
    signal.
    ```

    The first message is the most common, and probably indicates that `rc.switch` was run on the node.

16. Initialization retried

    The switch initialization portion of the fault service daemon is a two-phase process. The first is the discovery phase (BFS), the second the actual initialization of the switch fabric. During phase two, any time that a component of the switch fabric is determined to be faulty, it is removed from the configuration and the initialization is retried. The following entry is accordingly put in the flt file:

    ```
    (i) 07/24/98 19:40:05 : 2510-821 The second phase of the switch
    initialization will be retried.
    ```

    There is also a retry limit of three sets on reinitializing the switch. When this limit is met, the following entry is placed in the flt file:

```
(e) 07/24/98 19:40:05 : 2510-822 The second phase of switch
initialization reached retry limit. Estart failed.
```

9.2.3 The rc.switch.log File

The rc.switch.log file is found in the /var/adm/SPlogs/css directory and exists on each node in an SP system with an SP Switch. The file is created everytime the `rc.switch` script is issued on that particular node. The previous rc.switch.log is saved to the rc.switch.log.previous file. The file is self-explanatory; an example follows:

```
Fri Jul 24 20:32:25 EDT 1998
hostname is sp5n13.msc.itso.ibm.com
node_number is 13
adapter_config_status = css_ready
Enodes does not exist - using ODM and SDR.
switch_node_number is 12
switch_chip is 4
switch_board is 1
switch_chip_port is 1
IP_switch_netaddr is 192.168.15.13
IP_switch_netmask is 255.255.255.0
IP_switch_ARP_enabled is yes
adapter is TB3
IP_switch_offset is 1
/usr/lpp/ssp/css/ifconfig css0 inet 192.168.15.13 netmask 255.255.255.0 down off
set 1 tb2 arp
/usr/lpp/ssp/css/ifconfig css0 down offset 1
/usr/lpp/ssp/css/fault_service_Worm_RTG_SP -r 12 -b 1 -s 4 -p 1 -a TB3 -t 22
main(): (parent) fork() successful, child PID (daemon PID) = 19224
main(): (parent) parent returns 0, child (daemon) continues ...
main(): (child) signal handlers (SIG_IGN) set up successfully
main(): (child) setsid() successful, daemon process group ID = 19224
/etc/inittab entry specified as "once" for the fault service daemon. \
/usr/lpp/ssp/css/usconfig
rc.switch done - Fri Jul 24 20:32:27 EDT 1998
```

9.2.4 The out.top File

The fault service daemon annotates the network topology file with the status of links and nodes (devices) and generates the /var/adm/SPlogs/css/out.top file.

If the link or node is operational, no annotation is done. Otherwise you may find something like the following:

```
s 15 2  tb3 1 0           E01-S17-BH-J8 to Exx-Nxx   -4 R: device has been
removed from network - faulty (link has been removed from network - fenced)
```

SP Switch chip ports that have no connection are usually wrapped. This is noted as follows:

```
s 13 3    s 13 3          E01-S17-BH-J3 to E01-S17-BH-J3    2 L: initialized
(wrap plug is installed)
```

The possible node error status present in the out.top file is shown in Table 7.

Table 7. SP Switch Device Status

Error Code	Description
-4	Device has been removed from network - faulty
-5	Device has been removed from network by the system administrator
-6	Device has been removed from network - no AUTOJOIN
-7	Device has been removed from network for not responding
-8	Device has been removed from network because of a miswire
-9	Device was not reachable through the network

The possible link error status present in the out.top file is shown in Table 8.

Table 8. SP Switch Link Status

Error Code	Description
-2	Wrap plug is installed
-4	Link has been removed from network or miswire - faulty
-6	Link has been removed from network - no AUTOJOIN
-7	Link has been removed from network - fenced
-8	Link has been removed from network - probable miswire
-9	Link has been removed from network - not connected

9.2.5 The act.top and topology.data File

The fault service daemon creates the act.top.n file when it starts the SP Switch. The file exists only on the primary node or any node that was at one time the primary node. However, only the act.top.n file on the current primary node contains valid information. The "n" in the filename can assume one of the following values:

<pid>	where pid is the process ID of the `Estart_sw` script run during switch initialization. You should never see this file since it is renamed to /var/adm/SPlogs/css/topology.data when the initialization terminates.

0 created during a switch initialization spawned by a scan.

1 created during a switch initialization spawned by a primary node takeover.

2 created during a switch initialization spawned by an unrecoverable error.

3 created during a switch initialization spawned by an `Eunfence`.

You should check the files' timestamp across the nodes to verify the current one. The information in this file is self-explanatory; an example follows:

```
Number of active node(s) seen by the Worm:
2
Number_of_linksbad: 0
The primary backup node is:
8
The following switch node(s) are active:
4
8
The topology file used by the Worm:
/etc/SP/expected.top.annotated.2
```

9.2.6 The worm.trace File

The /var/adm/SPlogs/css/worm.trace file is written by the fault service daemon and exists on all nodes. The messages in this file relate to those in the flt file but give more detail.

On a secondary node the important information to look for is the initialization of the adapter, as shown:

```
Entering handleNodeInitPacket.
isTODvalid: 2510-930 Time-Of-Day is now: 80000000944980e8.
```

The messages tell you the initialization packets were received at the node. The node is able to talk over the switch.

The worm.trace file on the primary node contains messages tracing the switch initialization process. For instance, one of the first actions during an initialization is to clean the receive FIFO, discarding any not yet received error/status packets.

```
(i) print_the_time_worm: The date and time  = Mon Aug  3 19:10:04 1998
TBSswitchInit: Switch network Initialization Started!
(i) TBSswitchInit: The Primary backup is node  with switch node number 4
TBSworm_bfs_phase1: Switch Phase1 network Initialization Started!
syncFifoPh1: Cleaning up the Receive FIFO
```

In phase one of the switch initialization, the primary node visits all the switch chips indicated in the topology file and initializes them:

```
(i) TBSworm_bfs_phase1: Device ID = 100014
route to device 100014   = 00000000 00000000
route from device 100014 = 38000000 00000000
TBSworm_bfs_phase1: Device ID = 100014 has been Visited.
----------------------------------------------------------
Current Device ID: 100014 type: 101 level: 1
----------------------------------------------------------
route to device 100014   = 00000000 00000000
route from device 100014 = 38000000 00000000
TBSworm_bfs_phase1: handleSwDeviceInitResponse() successful
```

At each visited chip, all known ports connected to a switch chip are checked, and the expected device id is shown:

```
TBSworm_bfs_phase1: The current device port = 0
TBSworm_bfs_phase1: The attached device id = 13
```

Switch chips can be revisited:

```
TBSworm_bfs_phase1: The current device port = 4
TBSworm_bfs_phase1: The attached device id = 100010
handleSwSvcInitReVisitResponse: rc = 0
TBSworm_bfs_phase1: Device ID = 100014 has been ReVisited.
```

When a chip or a node is initialized during this phase, the route from the primary node to the device is shown, as well as the route from the device to the primary node. The latter is sent to the device in the initialization packet:

```
route to device 100010   = 8c000000 00000001
route from device 100010 = 43000000 00000001
```

When a node is initialized, its personality is noted. The primary node has the personality 1, the primary backup node is 2 and secondary nodes are 3.

```
personality =  3  db_cmd =  1  error_enable = 0000
```

During phase two of the switch initialization, the chips are reset and properly initialized for normal operation:

```
Phase-2 Switch Initialization Packet for device 100014
----------------------------------------------------------
route     = 00000000 00000000
Primary   = b0000000 00000000
Secondary = b0000000 00000000
bypass_enable =  0  central_queue_enable = ff  edc_frame_length = 1f
mode_bits =  2   receiver_enable = 1f   sender_enable = 1f
receiver_error_enable = 1f  sender_error_enable = 1f
```

After phase two of the initialization terminates successfully, the Worm generates the node-to-node routes, and synchronizes the TOD counter on all components. Finally, the routes are downloaded to the adapter.

```
DisableNode: rc = 0
TBSworm_bfs_phase2: Switch Phase2 network Initialization Ended!
TBSworm_bfs_phase2: rc = 0
(i) print_the_time_worm: The date and time  = Mon Aug  3 19:10:09 1998
TBSswitchInit: Worm Phase2 Initialization Successful.
(i) TBSswitchInit: Processor routes generated.
SetSystemTOD: Started.
(i) print_the_time_worm: The date and time  = Mon Aug  3 19:10:09 1998
Num_devices = 11
isTODvalid: 2510-930 Time-Of-Day is now: 80001dda12240d48.
SetSystemTOD: 2510-929 Time-Of-Day synchronization completed successfully.
TBSswitchInit: Switch network Time-Of-Day initialization completed
successfully!
(i) TBSswitchInit: Processor routes down loaded successfully.
TBSswitchInit: Switch network Initialization Ended!
```

9.2.7 The fs_daemon_print.file File

The file is written by the fault service daemon and exists on every node. It contains the commands the daemon responded to. Whenever a service packet or node command is received from the primary node, a time stamp is logged in the file and the event is noted. In the following case the node received a service packet from the primary node:

```
print_the_time: Time = Wed Jul 29 17:36:00 1998
fs_daemon_fsm_main: got request, type = service-msg-received
fs_daemon_fsm_main: packet Service Command (SC) = f9
```

The initialization information is also written to the file:

```
route to device 100010   = 8c000000 00000001
route from device 100010 = 43000000 00000001
Initial Personality:  0x0003, Enable error reporting: 0x0000
Device DB Command:    0x0001
Topology File path:   /etc/SP/expected.top.annotated.3
```

All node commands sent by the primary fault service daemon are also logged in this file. For instance, the following node initialization command is logged:

```
displayPacket Node Cmd = NODE_INIT:
fs_daemon_fsm_main: packet Node Command (NC) = 1
```

The command to load the node-to-node routes to the adapter is also logged:

```
displayPacket: Node Cmd = KLOAD_ROUTES:
```

```
fs_daemon_fsm_main: packet Node Command (NC) = 6
```

Personality change commands are also noted:

```
displayPacket Node Cmd = CHANGE_PERSONALITY:
fs_daemon_fsm_main: packet Node Command (NC) = 9
fs_daemon_fsm_main: old daemon personality is 3
fs_daemon_fsm_main: new daemon personality is 2
```

When the fault service daemon is killed, the following messages are logged in the file and the switch_responds is turned off in the SDR:

```
(i) handler_sigTerm: 2510-195 The fault service daemon got a SIGTERM
signal.
fs_daemon_exit: Turning off this nodes switchResponds bits in the SDR
```

9.2.8 The dtbx.trace File

The dtbx.trace file is found in the /var/adm/SPlogs/css directory on every node in the SP system. The file contains trace information for the last run of the css0 adapter diagnostics. When you notice a file creation time on this file of "Midnight Dec 31 1969" it indicates that this trace was created during the Power On Self Test (POST), when the node time had not been set yet.

The diagnostics goes through the following phases, which are then traced:

1. *Diagnostics setup* consists of making sure that ODM is configured properly, that the device css0 is configured and that diagnostics can get exclusive use of the device. At that point diagnostics is run.
2. *Clock selection* can be done from its own internal clock and the external clock. To complete diagnostics successfully, one of the external clock sources must be available for test purposes. If these clocks are not available, diagnostics will still be attempted on the internal clock. However, even if the diagnostics pass on this internal clock, a failure code is returned. This is due to the fact that, even though the adapter is okay, without an external clock source the card can not communicate with the switch.

   Clock selection is as follows:

   - First test whether the internal clock is operational. If it is not, diagnostics assumes the adapter is bad and no further testing is attempted.
   - Select the data cable clock. If it is available for testing, proceed with the test.

- If the data cable clock is not available, select the internal clock and proceed with the tests. Once tests have completed, mark the diagnostics as failed because of no external clock available.

3. The *Vital Product Data (VPD*) is read from the adapter EPROM and written to the /var/adm/SPlogs/css/dtbx.trace file. The VPD includes the following: Part Number, EC Level, Serial Number and FRU name, Manufacturer's code and the device description.
4. POS testing consists of reading and writing test data to the adapters' *Programmable Option Select (POS)* registers. It tests both the functionality of specific register bits, as well as doing pattern testing where applicable.
5. *TBIC FIFO Testing* is just as the name implies, functionally testing the FIFOs found on the TBIC chip.
6. *Testing SRAM* on the SP Switch adapter is done from the node because of its small size (512K). There are 10 passes of data patterns written to the SRAM.
7. *TBIC self-test* is a resident function of the TBIC chip. To execute the test, known seed data is scanned into the chip, then self-test is started for a determined number of passes. When it is complete, the information in the TBIC scan rings are then scanned out into the scan_out.log file in the /var/adm/SPlogs/css directory. The RPG and MISR register information is then extracted from the log and compared to the expected results.
8. The *Time-of-Day register* on the TBIC chip is tested by setting up the TOD register to a series of known values, then reading values back from the register to insure that every bit in the counter increments properly.
9. Interrupts allow the node and the adapter to interrupt each other, as well as for the adapter to interrupt itself. During the *interrupt test* each of the possible interrupts is forced and then checked.
10. Diagnostics are provided to test the DMA functions. The DMA test requires a DMA kernel extension to be loaded on the node.

Quite certainly, the file of greatest interest is /var/adm/SPlogs/css/dtbx_failed.trace. This file may or may not exist on a node. The file contains trace information for the last failed run of the css0 adapter diagnostics. When diagnostics fails, it renames the dtbx.trace to dtbx_failed.trace.

You should start examining the dtbx_failed.trace file by looking for the completion status at the bottom of the file. The SRN number at the bottom of the file should help you locate where in the file to start looking. The 3-digit code returned from the diagnostics can be broken down into the following failure codes:

n?? Broken *Component* Failure Code

?n? Test *Phase* Failure Code

??n Test *Unit* Failure Code

You can use Table 9 to decode the failure code given the indicator *n.* This table pertains to TB3 only. The table for MX or PCI is different from that of TB3.

Table 9. SP Switch Adapter TB3 Diagnostics Failure Codes

Indicator	Component	Phase	Unit
0			POST test
1	Software	POS test	Shared test
2	Clock	TBIC test	Full test
3	POS Register (R/W)	TBIC FIFO test	Card Wrap test
4	TBIC	DMA engine test	Cable Wrap test
5	SRAM	PPC601 test	Error Log analysis
6	PPC601	SRAM test	
7	Interrupt	Interrupt test	
8	TBIC FIFO	External clock test	
9	DMA engine	Packet send test	
A	Switch network	Internal clock	
B		Diagnostic setup (TED)	

9.2.9 The Snapshot Log css.snap.log

The css.snap.log file is found in the /var/adm/SPlogs/css directory and exists on any SP Switch node. The file is created every time `css.snap` is run, either manually or by the fault service daemon. It contains information about what happened during the snap operation. The file contains the following :

- Date and time information at the time of the snap
- Which node it was executed on

```
Wed Aug 12 17:45:24 EDT 1998
css.snap running on sp5n01
```

- A list of the contents of the /var/adm/SPlogs/css directory prior to `css.snap`:

```
Contents of /var/adm/SPlogs/css before css.snap:total 7189
-rw-r--r--   1 root     system      5363 Aug 12 14:18 flt
-rw-r--r--   1 root     system      1037 Aug 12 14:40
rc.switch.log.previous
-rw-rw-rw-   1 root     system       302 Aug 12 15:13 router.log.old
-rw-rw-rw-   1 root     system       302 Aug 12 15:13 router.log
-rw-rw-rw-   1 root     system      6361 Aug 12 15:13 out.top
-rw-rw-rw-   1 root     system      8155 Aug 12 15:13 core
-rw-r--r--   1 root     system      1044 Aug 12 17:43 rc.switch.log
-rw-r--r--   1 root     system       250 Aug 12 17:44 daemon.stdout
-rw-rw-rw-   1 root     system      2530 Aug 12 17:44 Ecommands.log
-rw-r--r--   1 root     system    175693 Aug 12 17:45 worm.trace
```

- Information on the tar and compress operation performed by `css.snap`:

```
tar: cable_miswire*: A file or directory in the path name does not
exist.
tar: dtbx.trace: A file or directory in the path name does not exist.
tar: cssadm.*: A file or directory in the path name does not exist.
a core 16 blocks.
a daemon.stdout 1 blocks.
a daemon.stderr 1 blocks.
a Ecommands.log 5 blocks.
a flt 11 blocks.
a fs_daemon_print.file 2427 blocks.
a rc.switch.log 3 blocks.
a router.log 1 blocks.
```

- Information on all running processes on the system at the time the snap.dump was taken:

```
"ps -ef" says:
     UID   PID  PPID   C    STIME    TTY  TIME CMD
    root     1     0  14 17:39:59      -  0:01 /etc/init
    root  4224     1   0 17:40:51      -  0:00 /usr/sbin/getty \
/dev/console
   root  4396    1   0 17:40:10     -  0:00 /usr/lib/methods/ssa_daemon
```

- Information about the ssp.css software product and updates to it:

```
"lslpp -h ssp.css" says:
  Fileset         Level     Action       Status       Date         Time
-----------------------------------------------------------------------
Path: /usr/lib/objrepos  ssp.css                3.1.0.0   COMMIT
COMPLETE     07/24/98     20:06:59
Path: /etc/objrepos  ssp.css                3.1.0.0   COMMIT
COMPLETE     07/24/98     20:07:23
```

- Information about the microcode of the switch adapter:

```
communications adapter information:
serial number -----
xilinx checksum --- 18515
ucode version --- 0x98062304
```

9.3 SP System Monitoring

Several tools are available on the SP system for system monitoring. This section gives a brief overview of what is available to monitor the switch. Refer to the system manuals for further information.

9.3.1 Monitoring the Switch Connection

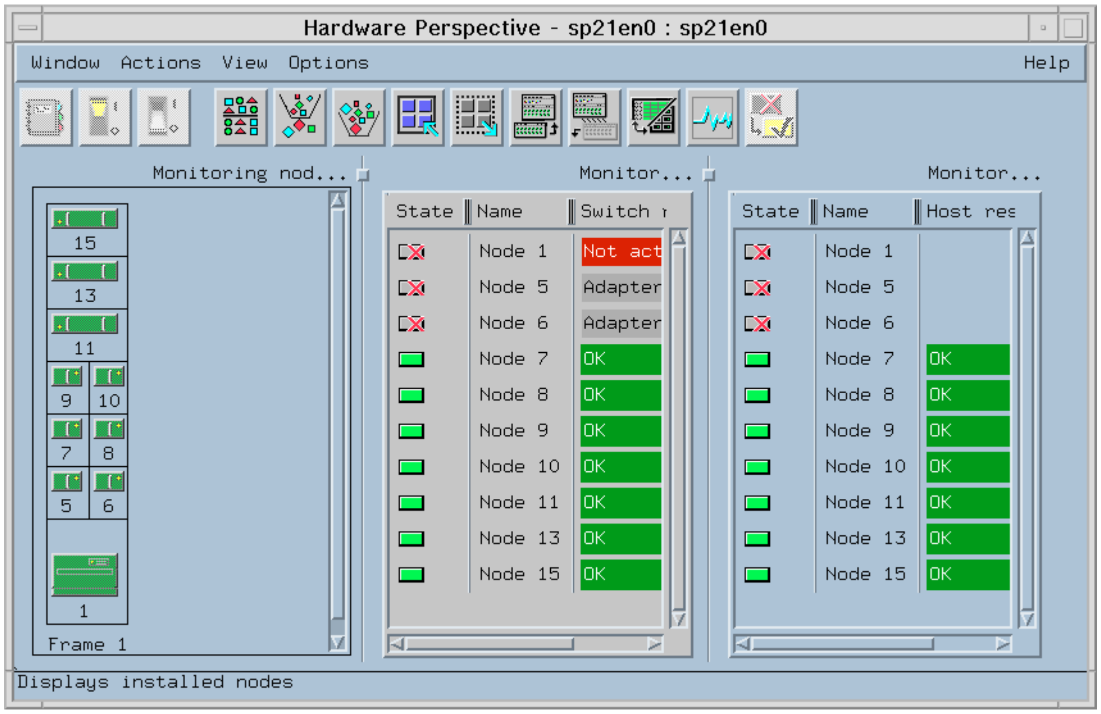
You can run the `SDRGetObjects` command on the Control Workstation to verify switch_responds:

```
# SDRGetObjects switch_responds
node_number  switch_responds autojoin     isolated     adapter_config_status
          1          0          0          1 not_configured
          5          1          1          0 css_ready
          9          1          1          0 css_ready
         13          0          1          1 css_ready
```

In the example, nodes 1 and 13 are not connected to the SP Switch but fenced (isolated). Nodes 5 and 9 are connected to the SP Switch.

You can also use SP Perspectives to monitor the switch. Perspectives is better described in the *PSSP: Administration Guide*, SA22-7348.

One monitoring option with Perspectives is to use Hardware Perspective. You may enable the monitoring of the switchResponds condition for nodes, partitions, or systems. Check on the switchResponds indicator on the monitor, it should be green for a running switch connection. In the following figure the middle frame displays the switchResponds indicator in the Perspective window.



State	Name	Switch r
✗	Node 1	Not act
✗	Node 5	Adapter
✗	Node 6	Adapter
🟩	Node 7	OK
🟩	Node 8	OK
🟩	Node 9	OK
🟩	Node 10	OK
🟩	Node 11	OK
🟩	Node 13	OK
🟩	Node 15	OK

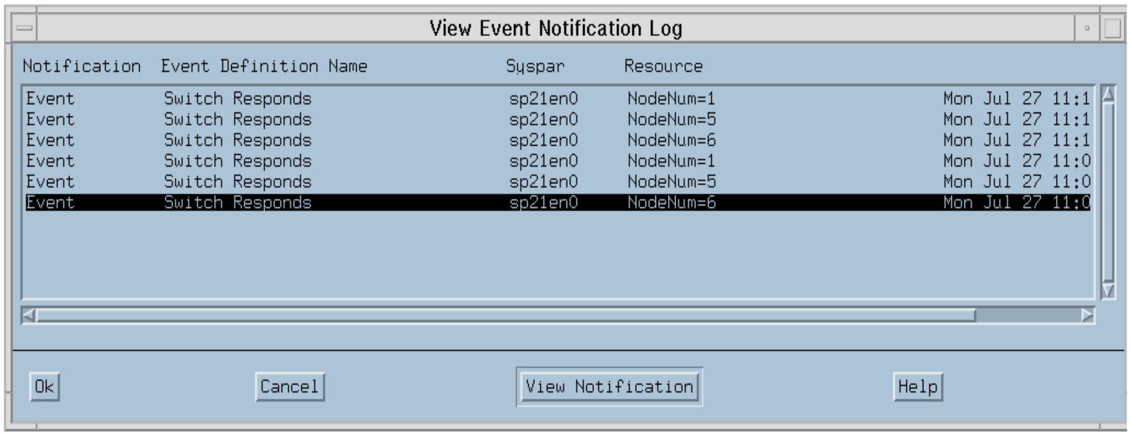
State	Name	Host res
✗	Node 1	
✗	Node 5	
✗	Node 6	
🟩	Node 7	OK
🟩	Node 8	OK
🟩	Node 9	OK
🟩	Node 10	OK
🟩	Node 11	OK
🟩	Node 13	OK
🟩	Node 15	OK

Displays installed nodes

Figure 61. Using Hardware Perspective

Equivalent information can be obtained by running `spmon -d -G`, as described in 6.6.1, “Verification Commands” on page 106.

You may also use Event Perspective to monitor resources and query resource variables, including switchResponds. Event Perspective uses the Event Management EMAPI interface. For instance, you may define an event "Switch Responds" that monitors the condition SwitchResponds. By requesting to view the Event Notification Log you can obtain a historic listing of the events as shown in Figure 62 on page 171.



```
Notification  Event Definition Name           Syspar      Resource
Event         Switch Responds                 sp21en0     NodeNum=1                          Mon Jul 27 11:1
Event         Switch Responds                 sp21en0     NodeNum=5                          Mon Jul 27 11:1
Event         Switch Responds                 sp21en0     NodeNum=6                          Mon Jul 27 11:1
Event         Switch Responds                 sp21en0     NodeNum=1                          Mon Jul 27 11:0
Event         Switch Responds                 sp21en0     NodeNum=5                          Mon Jul 27 11:0
Event         Switch Responds                 sp21en0     NodeNum=6                          Mon Jul 27 11:0
```

Figure 62. Using Event Perspective

9.3.2 Using the Problem Management Subsystem

This section gives two examples of how to use the HA Problem Management subsystem (pman) to be notified of problems. To find out more about how to use the pman subsystem, refer to *RS/6000 SP Monitoring: Keeping It Alive*, SG24-4873 and to the *PSSP: Diagnosis Guide*, GA22-7350.

The fault service daemon is essential to keep the SP Switch running. If the system administrator can be made aware of a daemon failure early, the availability of the switch may be improved. We show next how to monitor the fault service daemon (fault_service_Worm_RTG_SP) by running the `pmandef` command, as follows:

```
pmandef -s Monitor_Worm_daemon \
      -e 'IBM.PSSP.Prog.xpcount:NodeNum=*; \
      ProgName=fault_service_Worm_RTG_SP;UserName=root:X@0==0' \
      -r 'X@0>0' \
      -c /usr/lpp/ssp/bin/notify_event \
      -C "/usr/lpp/ssp/bin/notify_event -r"  \
      -n 0 -U root
```

When the daemon becomes inoperative on one of the nodes (X@0==0), `/usr/lpp/ssp/bin/notify_event` is executed (specified by the `-c` option). After that, when it becomes operative again (X@0>0), `notify_event -r` is executed (specified by the `-C` option). The following mail is sent to root at the Control Workstation by `notify_event`:

```
From root Tue Feb 11 15:15:05 1997
Date: Tue, 11 Feb 1997 15:15:04 -0500 (EST)
```

```
From: root
To: root
Subject: Monitor_Worm_daemon
Monitored situation exists on node 12.
Event Monitor_Worm_daemon reported at Tue Feb 11 15:15:03 1997.

Event Definition
----------------
Resource:   IBM.PSSP.Prog.xpcount
Instance:   UserName=root;ProgName=fault_service_Worm_RTG_SP;NodeNum=12
Condition:  X@0==0

Resource Value
--------------
Type:       sbs
Field0:     CurPIDCount=0
Field1:     PrevPIDCount=1
Field2:     CurPIDList=
```

The preceding command is part of the `/usr/lpp/ssp/install/bin/pmandefaults` script, which registers several useful events. Note that to use Problem Management you have to set up the /etc/sysctl.pman.acl file as described in the *PSSP: Administration Guide*, SA22-7348.

You can also use Problem Management to be notified of changes in switch_responds. You should use the IBM.PSSP.Response.Switch.state resource variable, which represents switch_responds. You may use the following script:

```
/usr/lpp/ssp/bin/pmandef -s Switch_Responds_Status \
      -e 'IBM.PSSP.Response.Switch.state:NodeNum=*:X==0' \
      -r 'X>0'  \
      -c /usr/lpp/ssp/bin/notify_event \
      -C "/usr/lpp/ssp/bin/notify_event -r"  \
      -n 0 -U root
```

Table 10 lists additional Event Management resource variables for the Switch:

Table 10. Event Management Resource Variables for the SP Switch

Event Variable	Description
IBM.PSSP.CSS.bcast_rx_ok	Number of broadcast packets received
IBM.PSSP.CSS.bcast_tx_ok	Number of broadcast packets sent
BM.PSSP.CSS.ibadpackets	Number of bad packets received from adapter
IBM.PSSP.CSS.ibytes_dlt	Total number of octets received (delta)

Event Variable	Description
IBM.PSSP.CSS.ibytes_lsw	Total number of octets received (lsw bits 0-30)
IBM.PSSP.CSS.ibytes_msw	Total number of octets received (msw bits 31-61)
IBM.PSSP.CSS.ierrors	Input errors on interface
IBM.PSSP.CSS.ipackets_dlt	Packets received on interface (delta)
IBM.PSSP.CSS.ipackets_drop	Number of packets not passed up
IBM.PSSP.CSS.ipackets_lsw	Packets received on interface (lsw bits 0-30)
IBM.PSSP.CSS.nobufs	No buffers available
IBM.PSSP.CSS.obytes_dlt	Total number of octets sent (delta)
IBM.PSSP.CSS.obytes_lsw	Total number of octets sent (lsw bits 0-30)
IBM.PSSP.CSS.obytes_msw	Total number of octets sent (msw bits 31-61)
IBM.PSSP.CSS.oerrors	Output errors on interface
IBM.PSSP.CSS.opackets_dlt	Packets sent on interface (delta)
IBM.PSSP.CSS.opackets_drop	Number of packets not transmitted
IBM.PSSP.CSS.opackets_lsw	Packets sent on interface (lsw bits 0-30)
IBM.PSSP.CSS.opackets_msw	Packets sent on interface (msw bits 31-61)
IBM.PSSP.CSS.recvintr_dlt	Number of receive interrupts (delta)
IBM.PSSP.CSS.recvintr_lsw	Number of receive interrupts (lsw bits 0-30)
IBM.PSSP.CSS.recvintr_msw	Number of receive interrupts (msw bits 31-61)
IBM.PSSP.CSS.xmitintr_dlt	Number of transmit interrupts (delta)
IBM.PSSP.CSS.xmitintr_lsw	Number of transmit interrupts (lsw bits 0-30)
IBM.PSSP.CSS.xmitintr_msw	Number of transmit interrupts (msw bits 31-61)
IBM.PSSP.CSS.xmitque_cur	Sum of driver+adapter xmit queues
IBM.PSSP.CSS.xmitque_max	Max transmits ever queued
IBM.PSSP.CSS.xmitque_ovf	Number of transmit queue overflows
IBM.PSSP.SP_HW.Node.nodePower	Switch powers up or down
IBM.PSSP.SP_HW.Switch.ps1PowerGood	Master oscillator failure

Event Variable	Description
IBM.PSSP.SP_HW.Switch.ps2PowerGood	Master oscillator failure
IBM.PSSP.SP_HW.Frame.controllerResponds	Controller responds goes up or down

Network status monitoring commands, such as `netstat -I`, cannot get such information. These resource variables are helpful in determining the cause of network problems. Most of them, however, are too detailed to be handled as events.

Monitoring those resource variables from Perspectives GUI or Performance Toolbox is described in *RS/6000 SP Monitoring: Keeping It Alive*, SG24-4873.

9.3.3 SNMP Traps on SP Switch Failures

PSSP provides a Management Information Base (MIB), named *ibmSP,* to be used by SNMP-based network management software, like NetView for AIX. A MIB defines a set of variables or objects that represent the physical and logical resources of the managed system or agent. The SNMP subagent responsible for the ibmSP MIB is implemented by the sp_configd daemon. The ibmSP variables cannot be modified by an SNMP manager.

The ibmSP MIB consists of three groups:

1. ibmSPConfig (1.3.6.1.4.1.2.6.117.1)

   This group defines objects containing SP system configuration information.

2. ibmSPErrlogVars (1.3.6.1.4.1.2.6.117.2)

   This group consists of a sequence of objects containing information about the last error log write that generated an SNMP trap.

3. ibmSPEMVariables (1.3.6.1.4.2.6.117.3)

   This group consists of a sequence of objects containing information about Event Management resource variables.

You can use the AIX error log notification facility to generate SNMP traps when specific log entries are cut. In addition, you may use the Problem Management subsystem to generate traps when conditions of interest are met. Refer to *RS/6000 SP Monitoring: Keeping It Alive*, SG24-4873 for a detailed description of setting SNMP traps.

9.4 SP Switch Utilities

PSSP provides several utilities for the SP Switch, which are located in the /usr/lpp/ssp/css directory:

- `rc.switch` restarts the fault-service daemon.

  `rc.switch` is automatically run during boot, but can also be run manually. You should run it if you detect that the fault service daemon is not up on the node. You should always check the rc.switch.log file and make sure that the daemon stays up.

- `css.snap` collects log and trace information into a compressed tar file, /var/adm/SPlogs/css/hostname.yymmddHHMMSS.css.snap.tar.Z

  This script is called whenever a serious error is detected by the switch support code (device driver, fault service daemon). It collects all SP Switch log files into a single package.

  The script can be called manually. Its syntax follows:

  ```
  css.snap [ -c | -n | -s ]
  ```

  -c Flushes the adapter cache and prints the result (default)

  -n Assumes that the device driver or daemon has flushed the cache

  -s Takes a soft snap, which does not dump the adapter state. Used for temporary errors, when the integrity of the adapter is in doubt, or when it is not desirable to corrupt the adapter state by the use of diagnostic routines.

  The following table shows the error log entries that automatically take a snap, as well as the type of snap performed:

Table 11. Error Log Entries that Call css.snap

Error log entries	soft/full snap
SP_SW_FIFOOVRFLW_RE	soft
SP_SW_RECV_STATE_RE	soft
SP_SW_INVALID_RTE_RE	soft
SP_SW_NCLL_UNINT_RE	soft
SP_SW_PE_INBFIFO_RE	soft
SP_SW_PE_ON_DATA_RE	soft
SP_SW_PE_ON_NCLL_RE	soft
SP_SW_PE_RTE_TBL_RE	soft

Error log entries	soft/full snap
SP_SW_RECV_STATE_RE	soft
SP_SW_SNDLOSTEOP_RE	soft
SP_SW_PE_ON_NCLL_RE	soft
TB3_CONFIG1_ER	full
TB3_LINK_ER	full
TB3_PIO_ER	soft
TB3_SVC_QUE_FULL_ER	full
TB3_THRESHOLD_RE	full

- `fs_dump` dumps the fault service kernel extension trace buffer.

  The `fs_dump` utility writes the css fault service kernel extension debug buffer to standard output. It should be called with the `-r` flag, otherwise subsequent updates to the debug buffer would be written to standard output until `fs_dump` is interrupted with Ctrl-C. The utility dumps status and error information for events such as user-space system calls to the fault service kernel extension.

- `css_dump` dumps css the device driver trace buffer.

  The `css_dump` utility writes the css device driver debug buffer to standard output. Unless the `-r` flag is specified, subsequent updates to the debug buffer are also written to standard output until `css_dump` is interrupted by Ctrl-C. The utility dumps status and error information for events such as ioctl calls to the device css0 and adapter error interrupts.

- `read_regs` prints the switch adapter's Programmable Options Selection (POS) registers and Trail Blazer Interface Chip (TBIC) registers in addition to the adapter clock status.

  The `read_regs` utility is used to read all registers on the local SP Switch adapter. Additionally, it returns information about the kernel extension and the clock source being used by the adapter. The exact output is different for TB3, MX, or PCI switch adapters. An example output for a TB3 is as follows:

```
Kernel extension /usr/lpp/ssp/css/fault_service_SP is loaded.

POS REGS :  0  1  2  3  4  5  6  7 31 32 33 34 35 36 37
CONTENTS : 69 8F 9B FE FF 8F 00 FF BF 07 00 FD 0E 71 FE

Clock signal present  --- DATA_CABLE clock selected

TBIC Registers:
TIME_OF_DAY                 : 800009f8 768be320
XMIT_FREE_SPACE Data & Hdr  : 00000100 00000040
RECV_FULL_SPACE Data & Hdr  : 00000000 00000000
SVC_FULL_SPACE              : 00000000
XMIT_FIFO_THRESH_DATA & Hdr : 00000000 00000000
RECV_FIFO_THRESH_DATA & Hdr : 00000000 00000000
INT_ERR error status flags  : 00001000 00000000
                                  STI_RETIMING
INT_MASK                    : ffffe8c7 ffffffe0
TBIC_CTRL                   : 0000ec7f ba1fff04
TBIC_STATUS                 : 78000000
TBIC_DIAG                   : 00000000
```

Check the TBIC_STATUS register. If bits 3 and 4 are both set, the switch clock is driving the adapter (bits are numbered from left to right, starting at 0). The value 0x78 for the first byte indicates that the node is part of the switch.

You can also execute `/usr/lpp/ssp/css/diags/read_tbic -s` to get the TBIC status.

Chapter 10. SP Switch Problem Diagnosis

In this chapter we go over some steps that may help you with problem determination of the SP Switch hardware and software. Refer to the *PSSP: Diagnosis Guide*, GA22-7350, for further information.

The list of recovery examples included in this section is not complete. The recovery actions suggested might not be successful with the problem you are experiencing. Nevertheless, this chapter is meant to provide first-aid information.

10.1 Verification Procedures

The SP Switch software depends on other PSSP components to work correctly. You should be aware of this dependence when diagnosing switch problems. You should routinely verify that the following subsystems are working correctly:

1. Topology Services, that is, the host_responds indicator, which is used by the E-commands to identify whether nodes are up or down.
2. SDR, which is accessed during all phases of the switch initialization, as well as during the execution of E-commands and whenever switch_responds needs to be updated.
3. Kerberos, which is needed to execute the E-commands and distribute the topology file.
4. Remote commands, which are used by the E-commands and to distribute the topology file.

Refer to *PSSP: Diagnosis Guide*, GA22-7350 for information on how to diagnose those subsystems. Needless to mention, those subsystems rely heavily on IP connectivity over the administrative Ethernet, which should always be monitored.

In addition, you should always double-check that your installation procedures were successfully carried out and that they are still valid. Namely, you should verify the following:

1. CSS lpp installation, through the `CSS_test` script, described in 6.6.1, “Verification Commands” on page 106. This script is an interesting tool since it not only checks the software installation but also IP connectivity over the Ethernet and over the switch, as well as Kerberos and remote commands.

2. The switch topology file, which could not correspond any longer to the actual switch topology. You should use the `Etopology -read` command to read the topology file stored in the SDR. See 6.3, “Specifying the SP Switch Topology File” on page 93.
3. The clock topology file, as described in 6.4, “Specifying the SP Switch Clock Distribution Tree” on page 100 and 8.2, “Establishing the SP Switch Clock” on page 134.

10.2 Diagnosing Procedures

As a general procedure, you should use the following steps when diagnosing a switch-related problem:

1. View the summary log /var/adm/SPlogs/css/summlog on the Control Workstation. Carefully examine the entries from the newest to the oldest until you find the entry reporting the error that seems to be the cause of all subsequent errors.
2. Note the node that reported the error. It might be the primary node. If not, you may need to also check the logs on the primary node.
3. Examine the AIX Error Log on the node that reported the error to get more details. Also check the explanations for some error log entries found in *PSSP: Diagnosis Guide*, GA22-7350.
4. If the information in the error logs is not helpful enough, check the /var/adm/SPlogs/css/flt file on the primary node. Use the timestamp in the AIX Error Log entry as a starting point to locate the relevant entries. Note that the format of the timestamps changes from log to log. You may find additional information on the error messages in *PSSP: Messages Reference*, GA22-7352.
5. If you have not pinned down the problem yet, you might try to investigate other log files, specially fs_daemon_print.file. Otherwise you should contact the IBM Support Center.

You should not only check the summlog file when you discover a switch problem, but use that file as a monitoring tool. Many switch faults can leave no trace other than an error message on the log files. For example, the failure of a switch-to-switch cable can go undetected since its absence may not necessarily take some nodes down, but an error log entry will be generated by the primary node when failing to initialize that link during an under-the-covers switch initialization.

It is also a good idea to run `css.snap` on the node that reported the error (and possibly on the primary node if it is a different node) as soon as possible after the error, so you do not risk losing the log files at the time of the error, or to have them growing with information irrelevant to the problem in question. Remember that many errors automatically take a snap. You should view the summlog file to check whether a snap was already taken or not.

10.2.1 Estart Fails

In this section we go over some basic steps to diagnose problems with `Estart`. To solve uncommon or hard problems, a good understanding of the steps taken by the script is extremely helpful. Refer to 7.3, “Starting the SP Switch” on page 119 for detailed information on `Estart`.

`Estart` may exit with an error right away if some of the initial checks fail. A common problem occurs when the oncoming primary node is not reachable or is fenced. If you need to start the switch right away, you may try using the `Eprimary` command to select another oncoming primary node and then issuing another `Estart`. You may also try to solve the problem in your oncoming primary node, since it could be a problem that is affecting all nodes. Refer to 10.2.2, “Node Is off the Switch” on page 182 on how to diagnose a node problem.

The fault service daemon may fail or time out during the switch initialization. If this happens, you should check the end of the /var/adm/SPlogs/css/fs_daemon_print.file file on the oncoming primary node for the cause of the error. In *PSSP: Diagnosis Guide*, GA22-7350, there is a table with possible causes and actions that could be taken. If the information there does not help you in solving the problem, you may try to look at the /var/adm/SPlogs/css/worm.trace file on the oncoming primary node to trace the activities of the fault service daemon. In C.1, “SP Switch Worm Return Codes” on page 253 you may find the list of error codes returned by the fault service daemon during the switch initialization.

`Estart` may finish normally but with some nodes or links not initialized. You should initially check the /var/adm/SPlogs/css/out.top file on the primary node for a more detailed explanation of why the nodes or links were not initialized. If the problem is in a single node, you should try to diagnose the node as described in 10.2.2, “Node Is off the Switch” on page 182. If the problem is on a single switch-to-switch connection, you may have a cable problem. If several nodes or links were not initialized, you should try to find a pattern that could indicate a common problem. For instance, if all links of two switch chips do not initialize, you may have an intermittent cable problem between those two chips. You may also have a cable miswire, in which case the /var/adm/SPlogs/css/cable_miswire file is created with additional information. Refer to *PSSP: Diagnosis Guide*, GA22-7350 for additional information on the analysis of the out.top log file.

Another possible reason for a node failing to join the switch is a corrupted or absent topology file. If there is a failure in the distribution of the topology file, you should check the /var/adm/SPlogs/css/dist_topology.log file on the primary node for detailed information on the failure. You may have a Kerberos problem, for example. A more subtle problem would occur if the topology file on a node gets corrupted or becomes lost. As described in 7.3.1, “Distributing the Topology File” on page 120, an `Estart` does not always distribute the topology file, causing the node with the lost or corrupted topology file to fail to initialize because of an error during the Route Table generation. The flt file on the node would contain the reason for the failure (see C.2, “Return Codes from Route Table Generation” on page 255).

10.2.2 Node Is off the Switch

There are many reasons for a node to be off the switch. Arguably, the first check to be made is whether the fault service daemon is running on the node or not. In case it is not running, you may try to restart it by calling the `rc.switch` script. But, of course, that may not solve the problem that caused the daemon to exit in the first place.

Whenever the fault service daemon is not running on a node, you should first check the AIX error log. Almost all error situations will cause the fault service daemon to make an error log entry. Refer to *PSSP: Diagnosis Guide*, GA22-7350, for the table with an explanation and recovery actions for most of the daemon’s error log entries.

An error log entry is not generated when there is a failure in the `rc.switch` script that bars the fault service daemon from running. You should check the /var/adm/SPlogs/css/rc.switch.log file to get more details. One possible reason for a failure in `rc.switch` is an error during the configuration of the SP Switch adapter as described in 7.1, “Configuration Method of the SP Switch Adapter” on page 111. You may check the adapter_config_status attribute of the switch_responds SDR object for the error status returned by the configuration method. An explanation and some actions that could be taken are found in *PSSP: Diagnosis Guide*, GA22-7350.

When you are having problems on a node, you should also check whether the adapter is being driven by the switch’s synchronous clock or not. You may use the `read_tbic -s` utility, which should return a value with bits 3 and 4 set. Typical values for TB3 are 1C000000, 1E000000, and 78000000. If not, you should initially try to restart the fault service daemon. If that does not work, your next step would be to `Eclock` the frame or the system, as described in 8.2, “Establishing the SP Switch Clock” on page 134. If running `Eclock` is not effective, you may be experiencing a cable problem.

When permanent errors on a node are detected by the local fault service daemon or by the primary fault service daemon, the node is fenced. Thus, simply unfencing a fenced node may not solve the problem that caused the node to be fenced in the first place. You should view the AIX error log on the failed node as well as on the primary node for the reason of the failure. If you decide to unfence the node you may use the `Eunfence` command, or use the autounfence feature of PSSP 3.1 and just reset the isolated attribute of the switch_responds object:

```
SDRChangeAttrValues switch_responds node_number==<node> isolated=0
```

The latter command also works even when the switch is down, especially when you want to unfence the oncoming primary node.

10.2.3 Eunfence Fails

The `Eunfence` command can fail for several reasons. Before the actual unfence, the topology file is distributed to all nodes being unfenced, and that could be a source of problems. The fault service daemon has to be running on the nodes for the unfence to work, otherwise the unfence will finish with a timeout error.

You may also get a timeout, even with the fault service daemon running, if there is an adapter problem that prevents the primary node from talking to the fault service daemon on the node being unfenced. That may happen after a cable was changed, for example. In this case you should restart the daemon through `rc.switch`. If it does not work, you might try unconfiguring and reconfiguring the adapter as described in 7.1, “Configuration Method of the SP Switch Adapter” on page 111, checking for any errors during the reconfiguration, and then restarting the daemon. Your last resort would be to reboot the node.

10.2.4 Other E-command Failures

The general error determination procedure for any E-command that fails follows:

- Note the error message returned by the command.
- Cross reference the message number with the information in *PSSP: Messages Reference*, GA22-7352.
- View details on the execution of the E-command in the /var/adm/SPlogs/css/flt file on the primary node.

10.3 Examples of Recovery Procedures

In this section we show some switch-related problems and how to proceed to solve them.

10.3.1 Estart Problem One

When you run the `Estart` command on the Control Workstation, the following problem occurs:

```
# Estart
Estart:  0028-029 Fault service worm not up on oncoming primary backup node, cannot \
Estart : sp5n01.
```

Check the AIX error log on the failing node for possible fault service daemon failures:

```
LABEL:          HPS_FAULT6_ER
IDENTIFIER:     89F34F83

Date/Time:       Wed Aug 12 14:18:52
Sequence Number: 877
Machine Id:      00010011A400
Node Id:         sp5n01
Class:           S
Type:            PERM
Resource Name:   Worm

Description
Switch Fault Service Daemon Terminated
...........
---------------------------------------------------------------------------
LABEL:          SP_SW_SIGTERM_ER
IDENTIFIER:     5A38E3B0

Date/Time:       Wed Aug 12 14:18:52
Sequence Number: 876
Machine Id:      00010011A400
Node Id:         sp5n01
Class:           S
Type:            PERM
Resource Name:   Worm

Description
Switch daemon received SIGTERM

Probable Causes
Another process sent a SIGTERM

User Causes
Operator ran Eclock
Operator ran rc.switch on node and switch daemon was restarted
User program sent SIGTERM

        Recommended Actions
        Run rc.switch to restart switch daemon

Detail Data
Software ID String
LPP=PSSP,Fn=fs_daemon_init.c,SID=1.33,L#=819,
PID of process sending SIGTERM
       23120
Name of process sending SIGTERM
---------------------------------------------------------------------------
```

The first (last in time) AIX error log, with label `HPS_FAULT6_ER`, indicates that the daemon has terminated. The previous switch-related error message describes the possible cause of the fault service daemon's termination. It indicates that the daemon got a SIGTERM signal that killed it. The scripts `Eclock` and `rc.switch` kill the daemon with that signal, but automatically restart the daemon. Since there are no later log entries that indicate a problem in one of those scripts, the daemon may have been killed inadvertently by the system operator or another privileged user.

The log entry suggests to restart the fault service daemon through `rc.switch`:

```
# /usr/lpp/ssp/css/rc.switch
"adapter/mca/tb3"
# ps -ef | grep Worm
    root 14014 14550   4 10:54:15  pts/0  0:00 grep Worm
    root 15914     1   0 10:53:35      -  0:00 \
/usr/lpp/ssp/css/fault_service_Worm_RTG_SP -r 8 -b 1 -s 4 -p 3 -a TB3 -t 22
```

Running `Estart` again should initialize the switch.

10.3.2 Estart Problem Two

After an `Estart`, not all nodes are initialized. Check the out.top file on the primary node to decode any error messages for nodes you expect to be connected to the switch. In this example we expected nodes 7 and 8 to be connected and found the following errors:

```
s 16 2  tb3 6 0           E01-S17-BH-J24 to E01-N7   -8 R: device has been removed \
from network because of a miswire (link has been removed from network - probable \
miswire)
s 16 3  tb3 7 0           E01-S17-BH-J23 to E01-N8   -8 R: device has been removed \
from network because of a miswire (link has been removed from network - probable \
miswire)
```

Check the /var/adm/SPlogs/css/cable_miswire file on the primary node to confirm the information:

```
# pg /var/adm/SPlogs/css/cable_miswire
ProcessNodeMiswire: 2510-920 Node mis-wire detected, Please check node cabling.
_________________________________________________________________________________
(i) print_the_time_miswire: The date and time  = Thu Aug 13 11:24:29 1998
ProcessNodeMiswire: 2510-745 Node Mis-wire detected for Device id - Expected = 6 \
Actual = 7.
ProcessNodeMiswire: 2510-794 Node Mis-wire detected for switch number - Expected = 1 \
Actual = 1.
ProcessNodeMiswire: 2510-795 Node Mis-wire detected for switch chip - Expected = 6 \
Actual = 6.
ProcessNodeMiswire: 2510-796 Node Mis-wire detected for switch port - Expected = 2 \
Actual = 2.
ProcessNodeMiswire: Connection line:  s 16 2  tb3 6 0           E01-S17-BH-J24 to \
E01-N7

ProcessNodeMiswire: 2510-745 Node Mis-wire detected for Device id - Expected = 7 \
Actual = 6.
ProcessNodeMiswire: 2510-794 Node Mis-wire detected for switch number - Expected = 1 \
Actual = 1.
ProcessNodeMiswire: 2510-795 Node Mis-wire detected for switch chip - Expected = 6 \
Actual = 6.
ProcessNodeMiswire: 2510-796 Node Mis-wire detected for switch port - Expected = 3 \
Actual = 3.
ProcessNodeMiswire: Connection line:  s 16 3  tb3 7 0           E01-S17-BH-J23 to \
E01-N8
```

You should now check the cables to nodes 7 and 8 (jacks 24 and 23, respectively), which could be switched. This problem could also have occurred because the nodes were placed in the wrong slots during maintenance.

10.3.3 Eunfence Problem

Node 8 is off the switch and you try to unfence it:

```
# Eunfence 8
dist_to_bootservers: 0028-178 Received errors in distribution of topology file from \
bootserver to at least one node.
See /var/adm/SPlogs/css/dist_topology.log on primary node for details.
Unable to unfence the following nodes:
sp3n08.msc.itso.ibm.com  No topology
```

Now we check the dist_topology.log file on the primary node:

```
# cat dist_topology.log
Eunfence: Tue Nov  3 12:58:26 EST 1998

sp3en0: sp3en0: krshd: Kerberos Authentication Failed.
sp3en0: sp3en0: /usr/lpp/ssp/rcmd/bin/rcp: 0041-004 Kerberos rcmd failed: rcmd \
protocol failure.
sp3en0: sp3en0: rshd: 0826-813 Permission is denied.
sp3en0: pexscr:  5025-509 sp3en0 rsh had exit code 1
sp3en0: sp3n08.msc.itso.ibm.com: krshd: Kerberos Authentication Failed.
sp3en0: sp3n08.msc.itso.ibm.com: spk4rsh: 0041-004 Kerberos rcmd failed: rcmd \
protocol failure.
sp3en0: sp3n08.msc.itso.ibm.com: rshd: 0826-813 Permission is denied.
sp3en0: dsh:  5025-509 sp3n08.msc.itso.ibm.com rsh had exit code 1
```

There is a Kerberos problem on the node. You should correct it using the procedures described in *PSSP: Diagnosis Guide*, GA22-7350. For example, you could start checking the Kerberos configuration files on the node:

```
# ls /etc/krb*
/etc/krb.conf    /etc/krb.realms
```

The file /etc/krb-srvtab is missing. To recreate it you should use the `setup_server`'s `create_krb_files` wrapper:

```
# spbootins -r customize -s no -l 8
# create_krb_files
create_krb_files: tftpaccess.ctl file and client srvtab files created/updated
on server node 0.
# ls /tftpboot/*srvtab
/tftpboot/sp3n08-new-srvtab
# spbootins -r disk -s no -l 8
```

The file was created on the Control Workstation with the name sp3n08-new-srvtab on the /tftpboot directory. You must copy it to the node as /etc/krb-srvtab. You should use FTP, since remote commands are not functioning.

10.3.4 Node Off the Switch

When you run `Estart` on the Control Workstation, not all of the 13 nodes are initialized:

```
# Estart
Estart:  0028-061 Estart is being issued to the primary node: sp3n01.
Switch initialization started on sp3n01.
Initialized 12 node(s).
Switch initialization completed.
```

Through the SDR we can see that node 8 was not initialized:

```
# SDRGetObjects switch_responds node_number==8
node_number  switch_responds autojoin     isolated     adapter_config_status
          8            0            1            1 css_ready
```

Look for the entry in the /var/adm/SPlogs/css/summlog file on the Control Workstation to find the error entry for the failing node:

```
# tail /var/adm/SPlogs/css/summlog
110311211998 sp3n15 N sp3en0 49 SP_SW_RSGN_BKUP_RE
110311211998 sp3n08 N sp3en0 51 HPS_FAULT6_ER
110311221998 sp3n01 N sp3en0 51 SP_SW_UNINI_NODE_RE
110311221998 sp3n01 N sp3en0 52 SP_SW_UNINI_LINK_RE
```

Being the only relevant error log entry, label `HPS_FAULT6_ER` only indicates that the fault service daemon on the node terminated with an error. There are also no helpful log entries on the primary node. We now check the flt file on the primary node:

```
(i) 11/03/98 11:21:29 : 2510-744 Estart initiated.
(i) 11/03/98 11:21:29 : The date and time  = Tue Nov  3 11:21:29 1998
(n) 11/03/98 11:21:29 : Switch and Adapter Error bits found during switch \
initialization.
(i) 11/03/98 11:21:29 : Device ID = 3
(i) 11/03/98 11:21:29 : 2510-918 Interrupt Error Register = 0000010000000000.
(n) 11/03/98 11:21:30 : 2510-826 Device ID 15 un-initialized during switch \
initialization. Disabling the device.
(n) 11/03/98 11:21:30 : 2510-826 Device ID 2 un-initialized during switch \
initialization. Disabling the device.
(n) 11/03/98 11:21:30 : 2510-826 Device ID 1 un-initialized during switch \
initialization. Disabling the device.
(n) 11/03/98 11:21:47 : 2510-606 A switch Error/Status was service packet received \
during a broadcast operation.
(n) 11/03/98 11:21:47 : 2510-606 A switch Error/Status was service packet received \
during a broadcast operation.
(n) 11/03/98 11:21:47 : 2510-606 A switch Error/Status was service packet received \
during a broadcast operation.
(n) 11/03/98 11:21:47 : 2510-606 A switch Error/Status was service packet received
during a broadcast operation.
(i) 11/03/98 11:21:53 : 2510-824 Switch initialization will be executed.
(i) 11/03/98 11:21:53 : The date and time  = Tue Nov  3 11:21:53 1998
(n) 11/03/98 11:21:53 : Switch and Adapter Error bits found during switch \
initialization.
(i) 11/03/98 11:21:53 : Device ID = 100016
(i) 11/03/98 11:21:53 : 2510-793 First  Error Capture Register  = 000008.
(i) 11/03/98 11:21:53 : 2510-741 Second Error Capture Registers = 00000000 00000010 \
00000000 00000000 00000010 000000
(i) 11/03/98 11:21:53 : Device ID = 3
(i) 11/03/98 11:21:53 : 2510-918 Interrupt Error Register = 0000010000000000.
(n) 11/03/98 11:21:54 : 2510-826 Device ID 15 un-initialized during switch \
initialization. Disabling the device.
(n) 11/03/98 11:21:54 : 2510-826 Device ID 7 un-initialized during switch \
initialization. Disabling the device.
(n) 11/03/98 11:21:54 : 2510-826 Device ID 2 un-initialized during switch \
initialization. Disabling the device.
(n) 11/03/98 11:21:54 : 2510-826 Device ID 1 un-initialized during switch \
initialization. Disabling the device.
(i) 11/03/98 11:21:58 : The Primary backup is node sp3n15.msc.itso.ibm.com
(i) 11/03/98 11:22:00 : Switch initialization completed successfully!
```

As marked on the screen, the switch initialization was retried, quite certainly because of the Error/Status packets received during a broadcast. We also notice that device 7 (node 8) was removed from the switch during the retry. To get more detail on the broadcast error we check the fs_daemon_print.file:

```
(n) fs_daemon_bcast_DBupdates: 2510-606 A switch Error/Status was service packet \
received during a broadcast operation.

displayPacket: Packet type = TBS_SVC_CMD_ERROR_STATUS:
        Command:  0xfa  Flag: 0x00      Chip ID: 0x0010 Seq no: 0xa5
        First Error Capture Reg: 0x000008
        Second Error Capture Reg: 0x00000000 00000010 00000000 00000000 00000010 \
000000

fs_daemon_bcast_DBupdates:11/03/98 11:21:53 : No ACK received from switch node \
number 7
fs_daemon_bcast_DBupdates:11/03/98 11:21:53 : Returning -2
init_on_startup_msg: Timed out on ACKs in fs_d_bcast_DBupdates()
```

As expected, the error occurred on node 8. It happened while it was building its version of out.top. We now check the /var/adm/SPlogs/css/daemon.stdout file on the failing node:

```
# tail /var/adm/SPlogs/css/daemon.stdout
ERROR: deviceConnect: id 100015 port 7 connected more than once

ERROR: buildDeviceDatabase: device-connect error, line 75
```

This is most likely an indication for a corrupted topology file on the node. One possible solution is to force the redistribution of the topology file on the next `Estart`, accomplished by:

```
# SDRChangeAttrValues Switch_partition num_nodes_success=0
```

Now run `rc.switch` to restart the fault service daemon on the failing node. Then run `Estart` on the Control Workstation to distribute the correct topology file to all nodes and start the switch.

Part 4. Application and Server Tuning for the SP Switch

Chapter 11. SP Switch-Specific Application and Server Tuning

In this chapter we describe the challenge of tuning the SP system and its applications for maximum possible performance. Applications running on the SP system with radically different network traffic characteristics can cause tuning problems. Where an application inherits its network tunables is key to optimal performance.

11.1 General Tuning Recommendations

When tuning or monitoring the performance of the SP system you need to make sure that you know what you change, when you change it, when you collected any data, and what the data you collect is for. Not using change control, or having time stamps on performance data, makes tuning the SP system and its switch a very difficult task. We highly recommend that a change control system be used to track and monitor any changes to the tunables on any part of the SP system, including changes to the Control Workstation and the nodes.

You need to take into account the total system picture when you are tuning large configurations. For example, if you are tuning a partitioned SP system, subsystems that span several system partitions (such as the SP system Ethernet) can cause changes within more than the partition being tuned.

When setting tunables, take into account the total system, not necessarily only those nodes being used for the application or subsystem being tuned.

The approach to tuning the SP system is, in some situations, the opposite of how you would tune an AIX workstation. In tuning an AIX workstation or server, the most common approach is to tune the machine to handle the amount of traffic or services requested of it. In the case of a file server, the server is tuned to the maximum amount of traffic it can receive. In general, the bottleneck in a high-end server is the capacity of the network through which services are requested.

In the SP system, the SP Switch is faster than any other network available. With the non-blocking nature of the switch, the number of requests and volume of data that may be requested of a node can far exceed the node’s capacity. To properly handle this situation, the volume of services requested of a server or destination node must be managed. Instead of trying to tune a node for the maximum amount of services requested, each of the nodes requesting services needs to manage the volume of requests made of another node. In other words, you should consider reducing the volume of requests at the client, rather than increase the capacity of the server. It is very easy on large SP system configurations to require more services than the most powerful node can currently deliver.

11.1.1 Scheduling Administrative Tasks

Some administrative tasks, such as collecting and clearing log files, cause network traffic that could affect system performance. You should consider scheduling such tasks for off-shift hours.

11.2 Tuning Considerations

Tuning the network interfaces is critical to maintaining peak throughput for network traffic. When tuning an SP switch adapter, there are two things to tune: the SP switch buffer pools and the ARP cache. Additional tuning is done using the communication protocols tunables.

The SP usually requires that tunable settings be changed from the default values in order the achieve optimal performance of the entire system. Network options can be changed with the `no` command. However, where to set these tunable values is very important. If they are not set in the correct places, subsequent rebooting of the nodes, or other changes, can cause tunable settings to change or be lost.

All dynamically tunable values (those that take affect immediately) setting should be set in the `/tftpboot/tuning.cust` script on each node. There is also a copy of the file in this same directory on the Control Workstation. Tunables changed using the `no`, `nfso`, or `vmtune` command can be included in this file.

There are several reasons why the `/tftpboot/tuning.cust` file should be used rather than /etc/rc.net for dynamically tunable settings. If you cause errors in /etc/rc.net, you can render the node unusable, requiring a reinstall of the node. If you partially cause errors in /etc/rc.net, getting to the node through the console connection from the Control Workstation can take several minutes or even more than one hour. This is because parts of the initialization of the node try to access remote nodes or the Control Workstation. Because /etc/rc.net is defective, each attempt to get remote data takes 9 minutes to timeout and fail.

If you cause errors in `tuning.cust`, at least the console connection will work, enabling you to log in through the Control Workstation and fix the bad tunable settings.

If you decide to create your own /etc/inittab entry to call a file with the tuning settings, future releases of PSSP will require a `/tftpboot/tuning.cust` set of tunables to be run to override your local modified settings. `/tftpboot/tuning.cust` is run from /etc/rc.sp, so it will always be run on a reboot. `/tftpboot/tuning.cust` includes a stop and start of inetd, which is required for all daemons spawned by inetd to inherit the SP-specific tuning settings. Using the sample `/tftpboot/tuning.cust` settings selected as a part of the install, the SP nodes will at least function well enough to get up and running for the type of environment selected. See 11.5, “SP Environment Tuning for Performance” on page 217 for specific environment tuning suggestions.

If the system has nodes that require different tuning settings, we recommend that a copy of the each settings be saved on the Control Workstation. Then, when nodes with a specific tuning setting are installed, that version of `tuning.cust` is moved to the /tftpboot directory on the node from the Control Workstation.

11.2.1 SP Switch Options

When you tune the switch buffer pools, you need to take several factors into consideration in deciding what the optimal settings should be. Along with performance problems, insufficient buffer space could also cause TCP connections to stop transferring data and then time out after a set amount of time because nothing comes ready on the sockets. The switch buffer pools are used to stage the data portion of IP packets for the switch. However, the allocation and sizes utilized from the pools can cause buffer starvation problems.

As discussed in 4.6.2, “Send Data Flow” on page 61, the send pool and receive pool are separate buffer pools, one for outgoing data (send pool) and one for incoming data (receive pool). The amount of data that will fit in an mbuf is up to 228 bytes, depending on what type of TCP options are set. When an IP packet is passed to the switch interface, if the data can fit in an mbuf (that is, the data is at most 228 bytes long), an mbuf is used from the mbuf pool, and no send buffer pool space is allocated for the packet. If, however, the data is too large to fit in an mbuf, buffer pool space will be allocated.

Use the `lsattr` command to view the current settings for the SP Switch pools:

```
lsattr -El css0
```

```
# lsattr -El css0
bus_mem_addr   0x04000000 Bus memory address        False
int_level      0xb        Bus interrupt level       False
int_priority   3          Interrupt priority        False
dma_lvl        9          DMA arbitration level     False
spoolsize      524288     Size of IP send buffer    True
rpoolsize      524288     Size of IP receive buffer True
adapter_status css_ready  Configuration status      False
```

The switch buffer pools are allocated in the kernel at initialization of the switch adapter. They are not dynamic for the SP Switch. This may change for future technology. Make the changes for the switch buffer pool size to the ODM database only. Then reboot the node for the changes to become effective.

Use the `chgcss` command to apply configuration changes to the SP Switch adapter in the ODM database. The command is located in the /usr/lpp/ssp/css directory. Configuration changes are later applied to the device when it is configured at system reboot. You must have root privilege to run this command.

```
chgcss -l css0 -a rpoolsize=1048576 -a spoolsize=1048576
```

Important

When allocating the receive pool and send pool, realize that this space is pinned kernel space in physical memory. This takes away space from user applications and is particularly important in small memory nodes.

The sizes of the switch buffer pools allocated by the device driver start at 4096 bytes and increase to 65536 bytes. All allocation sizes in between are a power of 2 value. This means that the buffer sizes allocated from the pools are:

Bytes	Size in K
40964	
81928	
1638416	
3267832	
6553664	

If the size of the data being sent is just slightly larger than one of these sizes, the buffer allocated from the pool is the next size up. This can cause a 50% efficiency in use of the buffer pools. More than half of the pool can go unused in bad circumstances. When assembling a TCP/IP packet, one mbuf from the

IP mbuf pool is always used to assemble the packet header information in addition to any data buffers from the send pool. If the mbuf pool size is too small, and the system runs out of mbufs, the packet is dropped. The mbuf pool is used globally for all IP traffic, and the size is set using the thewall tunable in the `no` command.

When sending 4 KB of data over the switch, an mbuf from the mbuf pool is used as well as one 4 KB send pool buffer for the data. If the amount of data being sent is less than 228 or so bytes, no buffer from the send pool is allocated because there is space in the mbuf used for assembling the headers to stage the data. However, if sending 256 bytes of data, you will end up using one mbuf for the IP headers, and one 4 KB send pool buffer for the data. This is the worst case, in which you are wasting 15/16 of the buffer space in the send pool. These same scenarios apply to the receive pool when a packet is received on a node.

The key for peak efficiency of the send pool and receive pool buffers is to send messages that are at or just below the buffer sizes allocated from the pools, or less than 228 bytes.

The appropriate values for the tunables are unique for each installation. Therefore, a sizing exercise like the one described here is necessary. When trying to determine the appropriate receive pool and send pool sizes, you need to get a profile of the message sizes that are being sent by all applications on a node. This will help determine how the receive pool and send pool will be allocated, in total number of buffers. At a given pool size, you will get 16 times as many buffers allocated out of the pool for 4 KB messages than for 64 KB messages. However, the total pool size in both cases is identical.

Once you have a profile of the packet or buffer sizes used by all applications using the switch on a node, you can then determine roughly how many of each size send pool or receive pool buffers will be needed. This then determines your initial receive pool and send pool settings. A node that stages a mix of packet sizes, of which 25% are smaller than 228 bytes, 50% are 5 KB and 25% are 32 KB, if the number of packets projected to be staged at any one time is 1024, then the send pool initial setting should be 12582912 or 12 megabytes. None of the small packets need any send pool space, the 5 KB packets each use 8 KB out of the send pool and need about 4 MB of space, and the 32 KB packets need about 8 MB of send pool space. The total estimated pool size needed is 12 MB.

Another way to determine the maximum pool size is to determine the maximum number of active socket connections at any one time. If you then multiply by the smaller of the tcp_sendspace or tcp_recvspace size for the connections, that gives you the maximum amount of pool space in bytes. However, if the buffer sizes that the connections use are not aligned against pool allocation sizes, the actual requirements will be higher.

The allocations from the pools are transient. The actual number of buffers used at any time is very dynamic due to the large volume of packets the switch can handle. The above calculation is a conservative estimate in that it assumes all packets will be staged at once. In reality, as packets are being staged into the pool, other packets are being drained out, so the effective number of active buffers should be less.

The sizes allocated from the pool are not fixed. At any point in time, the device driver will divide the pool into the sizes needed for the switch traffic. If there is free space in the send pool, and smaller buffers than the current allocation has available are needed, then the device driver will carve out the small buffer needed for the current packet. As soon as the buffer is released, it is joined back with the rest of the 64 KB buffer it came from. The buffer pool manager tries to return to 64 KB block allocations as often as possible to maintain high bandwidth at all times. If the pool were fragmented, and a large buffer needed 64 KB, then there may not be 64 KB of contiguous space available in the pool. Such circumstances would degrade performance for the large packets.

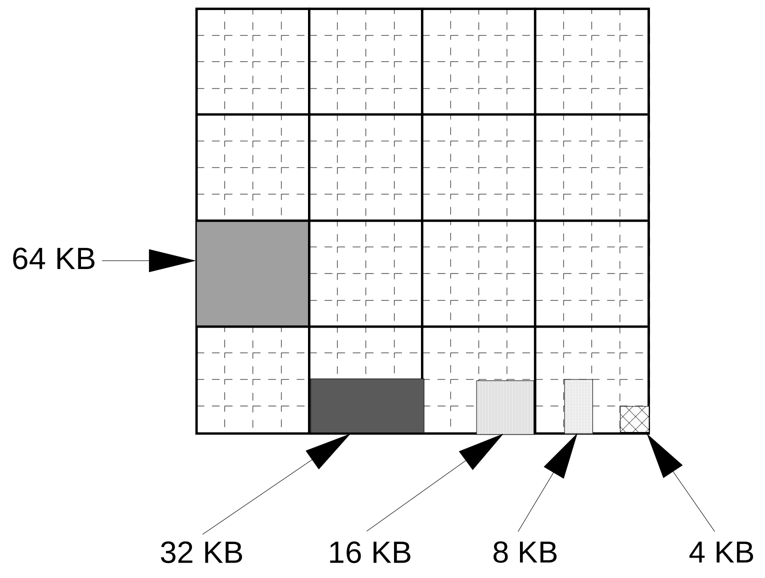


Figure 63. Switch Buffer Pool Allocation

If all buffer space is used, then the current packet is dropped, and TCP/IP or the application will need to resend it, expecting that some of the buffer space was freed up in the mean time. This is the same way that the transmit queues are managed for Ethernet, Token Ring and FDDI adapters. If these adapters are sent more packets than their queues can handle, the adapter drops the packet.

Currently, the upper limit for the send pool and receive pool is 16 MB for each. This means you can get a maximum of 4096 4 KB or 256 64 KB buffers each for sending and receiving data.

To verify that the switch buffer pools are exceeding the threshold, run the `vdidl3 -i` command to check the send pool:

```
# vdidl3 -i|pg
get ifbp info...

send pool: anchor@=0x0a265200 start@=0x0a300000 tags@=0x0552bc00
bkt   allocd     free  success     fail    split     comb    freed
 12        0        0  2981828        0  8019979        0        0
 13        0        0   617083        0  1604613        0        0
 14        0        0  7022485        0 14810213        0        0
 15        0        0   234483        0   493880        0        0
 16        0       32    10448        0        0        0        0
```

Where:

bkt	Lists the pool allocation in powers of 2 for the line it is on. The line starting with 12 means 2 to the 12th or 4 KB allocations and the line starting with 16 means 2 to the 16th or 64 KB allocations.
allocd	Lists the current allocations at the time the command was run, for each of the size allocations in the first column. This is an instantaneous value and, therefore, can increase and decrease between successive executions of the command.
free	Lists the number of buffers of each allocation that are allocated and unused at the time the command was run. In the above example, there are 32 64 KB allocations free for use. This is a snapshot value and, therefore, can increase and decrease between successive executions of the command.
success	Increments every time an allocation of the given size succeeded. This counter accumulates and, therefore, shows the number of successes since the adapter was last initialized.
fail	Increments every time an allocation is not available for the size requested. This counter accumulates and, therefore, shows the number of fails since the adapter was last initialized.
split	Indicates how many times the allocation size was extracted from the pool by carving the size needed from a larger allocation in the pool. This counter accumulates and, therefore, shows the number of splits since the adapter was last initialized.
comb	Currently not used.
freed	Currently not used.

For the receive pool, check the AIX error log for "mbuf pool threshold exceeded" entries by running `errpt -a|grep ENOBUF`. If you experience slow network traffic or ENOBUF errors in the error log, it indicates the system is running out of receive buffer pool space.

Use `netstat -m` to check for "request for memory denied" errors for small (less than 228 bytes) packets. Increase the size of thewall in the `no` options:

```
/usr/sbin/no -o thewall=131072
```

For large packets (greater than 228 bytes) use the `vdidl3 -i` command to check if the failed count is non-zero. Increase the switch buffer pools using the `chgcss` command and reboot the system:

```
/usr/lpp/ssp/css/chgcss -l css0 -a rpoolsize=1048576 -a spoolsize=1048576
```

When tuning the receive and send pools, it is important to understand the network traffic expected. As shown, if the size of the buffers for the applications is not ideal, much of the send and receive pool will be wasted. Such circumstances can cause the need for a larger receive and send pool because of inefficient usage.

If there are a small number of active sockets, then there is usually enough receive and send pool space that can be allocated. In systems where a node has a large number of sockets opened across the switch, it is very easy to run out of send pool space when all sockets transmit at once. For example, 300 sockets each sending 33 KB of data will far exceed the 16 MB limit for the send pool. Or, 1100 sockets each sending four 1 KB packets will also exceed the maximum limit.

On the receive side of a parallel or client/server implementation, where one node acts as a collector for several other nodes, the receive pool runs into the same problem. Four nodes each with 600 sockets, each sending two 1 KB packets to one node, will exceed the receive pool limit, but those same sockets each sending twice as much data, 4 KB in one 4 KB packet, will succeed!

Another situation where a node combination can cause buffer exhaustion is when a faster node sends data to a slower node. A P2SC node can generate traffic several times faster than an older POWER2 node. So, if the P2SC node sends packets as fast as it can, the slower POWER2 node can run out of receive pool space. The receive pool space can fill because the slower processor cannot drain the receive pool as fast as the P2SC node can fill it from across the switch. One way to prevent overrunning the receiving node is to make sure that the aggregate TCP window for all active sockets is less than the receive pool size. See additional information regarding the TCP window mechanism in *IBM AIX Versions 3.2 and 4 Performance Tuning Guide*, SC23-2365.

The other situation that aggravates exhaustion of the pools is the use of SMP nodes. Only one CPU is used to manage the send or receive data streams over the switch. However, each of the other CPUs in the SMP node is capable of generating switch traffic. As the number of CPUs increases, so does the aggregate volume of TCP/IP traffic that can be generated. We suggest that for SMP nodes, the send pool size be scaled to the number of processors when compared to a single CPU node setting.

For applications, changing the size of messages or buffers sent is key to prevent exhausting the switch buffer pools. Most applications tuned for peak efficiency for slower networks, such as Ethernet, will try and send 1 KB messages, causing problems on the switch. Increasing the buffer size for the SP not only reduces the requirements on the receive and send pool, but it can also improve performance. TCP/IP transfers data faster over the switch when you use larger packets.

The key to tuning the buffer pools on the SP is to understand the number of packets using the switch pools at any one time and the size of the data being sent. The receive and send pool sizes need to be set so there are enough buffers allocated to handle the size and number of packets being sent or received. Applications that do not efficiently use full buffers from the pool aggravate the pool size needed or can cause exhaustion of the pools.

11.2.2 AIX Tuning Option

There are a small number of `no` options that are not dynamically tunable (that is, they are load-time attriutes) and need to be changed in the /etc/rc.net file. These tunables are for ARP cache tuning and setting the number of network interface structures per interface. They are:

arptab_nb
arptab_bsize
arpqsize
ifsize

The first level network protocol is the Address Resolution Protocol (ARP), which dynamically translates Internet addresses into the unique hardware MAC addresses on local area networks. These addresses are kept in a series of entries in buckets. If a MAC address is not in the ARP cache when connecting to a remote adapter, an ARP broadcast is sent to get the MAC address from the remote host and store the information in the local ARP cache. If the cache is too small, then the ARP cache thrashes, loading up MAC addresses, causing network performance to suffer.

SP systems with more than 128 nodes can suffer from ARP thrashing. Though the SP Switch does not include a MAC address, the system uses the switch number as MAC address and stores it in the ARP cache:

```
sp2a03.gsi.de (140.181.80.3) at 0:2:0:0:0:0
sp2a04.gsi.de (140.181.80.4) at 0:3:0:0:0:0
sp2a07.gsi.de (140.181.80.7) at 0:6:0:0:0:0
sp2b05.gsi.de (140.181.80.21) at 0:14:0:0:0:0
sp2b07.gsi.de (140.181.80.23) at 0:16:0:0:0:0
```

The ARP cache is controlled by the `no` command. The default tunables allocate 175 ARP cache entries. The tunable `no` options are:

arpqsize	Specifies the maximum number of packets to queue while waiting for ARP responses. The default value is 1. This attribute is supported by Ethernet, 802.3, Token Ring and FDDI interfaces.
arptab_bsiz	Specifies Address Resolution Protocol (ARP) table bucket size. The default value is 7.
arptab_nb	Specifies the number of ARP table buckets. The default value is 25.
arpt_killc	Specifies the time in minutes before a complete ARP entry will be deleted. The default value is 20 minutes.

The total available ARP entries are calculated using:

arptab_nb * arptab_bsiz

For fast lookups, a large number of small buckets is ideal. For memory efficiency, a small number of medium buckets is best. Having too many buckets wastes memory. The following is the recommended way to calculate the values for the ARP cache sizing. For systems with more than 64 nodes, round the number of nodes down to the next power of 2, and use that for arptab_nb.

Table 12 shows the values for systems with from 1 to 512 nodes.

Table 12. arptab_nb Tuning

Number of Nodes	arptab_nb value
1-64	25 (default)
65-128	64
129-256	128
257-512	256

To set the number of entries for each ARP bucket, set the arptab_bsize to double the number of adapter interfaces on a node.

Table 13. arptab_bsize Tuning

Number of adapter interfaces	arptab_bsize value
1-3	7 (default)
4 and more	8 (2 times the number of interfaces)

Display the number of entries in the ARP cache and compare the result with the maximum number you set with the `no` options:

```
# arp -a|wc -l
      23
```

You can see if any of your buckets are full by pinging an IP address on a local subnet that is not in the ARP cache and is not being used. See how long the ARP entry stays in the cache. If it lasts for a few minutes, that particular bucket is not a problem. If it disappears quickly, that bucket is doing some thrashing. Carefully choosing the IP addresses to ping will let you monitor different buckets. Make sure the ping actually made the round trip before timing the ARP cache entry.

11.2.3 IP Tuning Parameters

A problem that often occurs on the SP system is that an application runs very slowly when using the SP switch, while performance is significantly better on an Ethernet or FDDI network interface. This problem is sometimes caused by the Nagle Algorithm (used by TCP/IP) interacting with the delayed ACK (acknowledgment) timer.

A 200 msec delay in sending data over a switch can be caused by either the Nagle Algorithm being invoked or by the application writing buffers so that only one packet is sent over the interface and the delayed ACK timer causing a 200 msec delay on the acknowledgment.

The effect of the Nagle Algorithm or delayed ACK timer is easy to see if only one socket connection is running. If you check the packet rate over the switch, you should see an average of 5 packets per second. Typically, a throughput rate of 150 to 300 KB per second transfer rate is reported by an application. To check the packet rate over the switch and total IP packet rates per second, use the following command:

```
# netstat -I css0 1
    input    (css0)    output           input   (Total)    output
 packets  errs  packets  errs colls  packets  errs  packets  errs colls
171158001     8 141967087     0     0 181988911     8 152981357     0     0
       5     0        3     0     0        6     0        4     0     0
       3     0        2     0     0        4     0        3     0     0
       6     0        3     0     0        7     0        4     0     0
       3     0        2     0     0        4     0        4     0     0
       7     0        2     0     0        9     0        3     0     0
       4     0        2     0     0        5     0        3     0     0
```

This listing shows how many packets per second go through the switch interface. The following suggestions will help to avoid the Nagle Algorithm:

- If you are running an application and do not have access to the source code, use the `no` command to increase the TCP window. This may not always be effective because increasing the tcp_sendspace and tcp_recvspace size on the sending and receiving nodes may cause other negative effects on the SP system or to other applications running on the system. Make sure that you set rfc1323 to 1 if the window size exceeds 65536.
- Changing the MTU of the switch will move the window size and buffer size combination. When writing 32 KB buffers to a TCP connection, if the TCP/IP window is 65536, only 5 packets per second are transmitted. If you change the MTU of the switch interface to 32768, there is no delay in transmitting a 32768 buffer because it is the same size as the segment size of the switch. However, reducing the MTU of the switch to 32768 degrades the peak TCP/IP throughput slightly. Reducing the MTU even further degrades the peak throughput even more.
- From within an application, you can change the TCP window size on the socket by setting a different size using the SO_SNDBUF and SO_RCVBUF settings on the socket. For good performance across a switch, we suggest that both SO_SNDBUF and SO_RCVBUF be set to at least 327680 on both the client and server nodes. You need to set both sides since TCP uses the lowest common size to determine the actual TCP window.

  If you set the SO_SNDBUF and SO_RCVBUF sizes above 65536, you also need to set TCP_RFC1323 on the socket unless the `no` options already have it set. Setting TCP_RFC1323 to 1 will take advantage of window sizes greater than 65536. You also want to ensure that the system setting for sb_max is at least 655360 or sb_max will reduce the effective values for SO_SNDBUF and SO_RCVBUF. You cannot change the sb_max setting from within an application. You must use the `no` command.

- Set TCP_NODELAY on the socket of the sending side to turn off the Nagle Algorithm. All data sent will go immediately, no matter what the data size. However, if the application sends very small buffers, you will significantly increase the total number of packets on the network.

One common problem that causes unexpected Nagle behavior is setting tcp_sendspace and tcp_recvspace to a high number, and forgetting to set rfc1323 to 1 on all nodes. In addition, setting tcp_sendspace and tcp_recvspace large on one node and not the other will cause Nagle to occur. The TCP window size is negotiated as the least common denominator of the sending node's tcp_sendspace and receiving node's tcp_recvspace.

On systems where a node is talking to several other nodes, it is harder to see the Nagle effect. In this case, the only way to detect it is to examine `iptrace` output to extract a single socket's traffic. What can happen is that if one node is talking to two nodes, each connection can be seeing the Nagle effect, and the packet rate over the switch is 10 packets per second. If one node is talking to five nodes, the packet rate can be 25 packets per second but the aggregate switch traffic is 1.5 MB/s. This rate exceeds the throughput on a slow Ethernet network, but is well below what the switch can handle.

Important

Changes to the network options do not affect existing socket connections. Child processes inherit default socket settings from parent processes.

11.2.4 MPI Tuning

In the Message Passing Interface (MPI) there are two tunables that affect the SP Switch performance. They are the message size MP_EAGER_LIMIT and the maximum amount of memory staging per message per task.

Set the MP_EAGER_LIMIT, which the rendezvous methodology will use. Size it accordingly to ensure that at least 32 messages can be outstanding between two tasks in user space. Table 14 shows the *eager_limit* values relative to the number of tasks:

Table 14. MP_EAGER_LIMIT

Tasks	MP_EAGER_LIMIT
1-16	4096
17-32	2048

Tasks	MP_EAGER_LIMIT
33-64	1024
65-128	512

The MP_BUFFER_MEM sets the maximum amount of memory for staging per task (see Table 15):

Table 15. MP_BUFFER_MEM

Protocol	Default size
User Space	64 MB
MPI over IP	2.67 MB

Important

Using MPI over IP gives you a much smaller default memory buffer size. It can be increased by setting a larger value through your shell variables.

11.3 Files Used on the SP for Tuning

There is no one way to tune an SP system that will work in every type of application configuration or environment. The following sections describe the characteristics of several of the more common environments found on the SP and how initial tuning settings are determined. They also describe the way that the system utilizes these optimal tunables to benefit the types of network traffic generated.

One difficulty in tuning networks is that different mixes of traffic, and the way applications utilize the network, can work against each other. A set of tunables that benefits a single stream point-to-point large data transfer can cause severe problems for a parallel application. In many cases you cannot optimize for two types of traffic using the `no` tunables. In addition, if your configuration has several types of network media (Token ring, Ethernet, FDDI and ATM), it is not always possible to optimize for all network interfaces.

The sections that follow address settings for a single application environment. Other options to optimize tuning can be handled by setting groups of nodes into different settings, or using the applications themselves to set network tuning for their own socket connection.

11.3.1 Select an IBM-Supplied Alternate Tuning File

When a node is installed, migrated, or customized (set to customize and rebooted), and that node's boot/install server does not have a `/tftpboot/tuning.cust` file, a default file of system performance tuning variable settings in /usr/lpp/ssp/install/config/tuning.default is copied to `/tftpboot/tuning.cust` on that node. You can override these values by following one of the methods described in the following list.

IBM supplies three alternate tuning files which contain initial performance tuning parameters for three different SP environments:

/usr/lpp/ssp/install/config/tuning.commercial contains initial performance tuning parameters for a typical commercial environment.

/usr/lpp/ssp/install/config/tuning.development contains initial performance tuning parameters for a typical interactive and or development environment.

/usr/lpp/ssp/install/config/tuning.scientific contains initial performance tuning parameters for a typical engineering and or scientific environment.

Use SMIT or issue the `cptune` command. When you select one of these files, it is copied to `/tftpboot/tuning.cust` on the Control Workstation and is propagated from there to each node in the system when it is installed, migrated, or customized. Each node inherits its tuning file from its boot/install server. Nodes that have as their boot/install server another node (other than the Control Workstation) obtain their `tuning.cust` file from that server node, so it is necessary to propagate the file to the server node before attempting to propagate it to the client node. The settings in the `/tftpboot/tuning.cust` file are maintained across a boot of the node.

11.3.2 Create and Select Your Own Alternate Tuning File

The following steps enable you to create your own customized set of network tunable values and have them propagated throughout the nodes in your system. These values are propagated to each node's `/tftpboot/tuning.cust` file from the node's boot/install server when the node is installed, migrated, or customized and are maintained across the boot of the node.

- On the Control Workstation, create the file `/tftpboot/tuning.cust`. You can choose to begin with a copy of the file located in `/usr/lpp/ssp/samples/tuning.cust` that contains a template of performance tuning settings have been commented out. Or you may prefer to begin with a copy of one of the IBM-supplied alternate tuning files.
- Select the tunable values that are best for your system. Refer to Table 16 for more information on tuning values.

- Edit the `/tftpboot/tuning.cust` file by ensuring that the appropriate lines are uncommented and that the tunable values have been properly set.

Using SMIT:

Select SP System Management
Select SP Cluster Management
Select The desired tuning file

Table 16. Initial Values for SP Switch Performance at the Expense of Ethernet

Tunable	Ethernet	Ethernet no SPS	Ethernet SP Thin node with SPS	Ethernet SP Wide, High and SMP nodes with SPS
sb_max	163840	442368	475236	655360
rfc1323	1	1	1	1
thewall	16384	16384	16384	16384
subnetsarelocal	1	1	1	1
tcp_sendspace	65536	221184	237568	262144
tcp_recvspace	65536	221184	237568	262144
udp_sendspace	32768	65536	65536	65536
udp_recvspace	65536	221184	237568	655360
TCP_mssdflt	1448	1448	1448	varies

If you have additional network adapters in your nodes and you want to optimize the network performance for these adapters, use the initial suggested settings shown in Table 17:

Table 17. Initial Values for Other Adapter Types

Tunable	Token-Ring	FDDI	ATM
sb_max	655360	1302528	655360
rfc1323	1	1	1
thewall	16384	16384	16384
subnetsarelocal	1	1	1
tcp_sendspace	262144	196608	262144
tcp_recvspace	262144	196608	262144

Tunable	Token-Ring	FDDI	ATM
udp_sendspace	65536	65536	65536
udp_recvspace	655360	655360	655360

As you set up the network tunables for your environment, it is important to tune for the type of data transfer your site has. If you transfer mainly large blocks of data or buffer sizes of 4 KB or greater, the tunable settings can be different than when you send files or buffers of less than 228 bytes.

Once you have updated `tuning.cust`, continue installing the nodes. After the nodes are installed or customized, on all subsequent boots, the tunable values in `tuning.cust` will be automatically set on the nodes. Note that each of the supplied network tuning parameter files, including the default tuning parameter file, contains the command `/usr/sbin/no -o ipforwarding=1`. IBM suggests that on non-gateway nodes, you change this command to read `/usr/sbin/no -o ipforwarding=0`. After a non-gateway node has been installed, migrated, or customized, you can make this change in the `/tftpboot/tuning.cust` file on that node.

For the latest performance and tuning information, refer to the RS/6000 Web site at

`http://www.rs6000.ibm.com/support/sp/perf`

11.4 Common SP Application Tuning for Performance

This section provides information for some common applications in an SP environment.

11.4.1 Server Tuning

The server environment usually is a node serving a lot of data to one or many other nodes on an SP. It can also be serving data to machines outside the SP through gateway nodes. If a server node in an SP potentially serves hundreds of requests or connections, tcp_sendspace and tcp_recvspace need to be small. This prevents a large number of large data requests from consuming the entire switch and the TCP/IP buffer pools.

In systems where there is one server, and the rest of the nodes run an application that needs larger tcp_sendspace and tcp_recvspace, it is acceptable to use different settings on the appropriate nodes. In this situation, the nodes talking to each other use large TCP windows for peak performance, and when talking to the server use small windows. The TCP window is set using the least common denominator of the tcp_sendspace and tcp_recvspace values.

The following list provides network tunable settings designed as initial values for server environments. You need to change these values on each of the installed nodes. To temporarily change these values, use the `no` command. If you want them to be preserved across booting the nodes, we suggest that you change the `tuning.cust` script on each node. Details on how to change the script are found in 11.2, “Tuning Considerations” on page 196.

thewall =16384
sb_max =131072
subnetsarelocal =1
ipforwarding =1
tcp_sendspace =65536
tcp_recvspace =65536
udp_sendspace =65536
udp_recvspace =655360
rfc1323 =1
tcp_mssdflt =1448
tcp_pmtu_discover (new in AIX 4.2.1) =1
udp_pmtu_discover (new in AIX 4.2.1) =1

These settings are only initial suggestions. Start with them and realize that you may need to change them. These initial values are derived by expecting network traffic from lots of connections. To prevent exhaustion of the TCP/IP buffer pools or the switch buffer pools, tcp_sendspace, tcp_recvspace and sb_max are reduced. If the total number of active requests from the server is small, these values may be increased to allow more buffer area per connection. This increase will help improve performance for small numbers of connections as long as the aggregate TCP window space does not exceed other buffer areas.

11.4.2 Tuning for FTP

When tuning the SP Switch network for FTP performance, keep in mind that in most cases the bottleneck is not the network but the disk access rate. Tune AIX for memory file caching. You can use the suggestions in the /usr/lpp/ssp/install/config/tuning.scientific script as initial performance tuning parameter for FTP service. FTP opens a single connection to the server and can require a large amount of data. The initial settings in the tuning.scientific script set up the network tunables so that a single connection or a few connections can get the full SP Switch bandwidth. Refer to 11.5.2, “Tuning for Scientific and Technical Environments” on page 218.

11.4.3 Tuning for NFS

Three aspects of NFS that are particularly challenging to tune for large SP system configurations are:

- The number of concurrent NFS requests to an NFS server
- The number of NFS daemons (nfsd and biod) running
- The NFS socket size

List the current NFS settings on a node with the `nfso -a` command. The parameters vary depending on the NFS version you are running on the AIX system. The example was taken from NFS V3:

```
# nfso -a
portcheck= 0
udpchecksum= 1
nfs_socketsize= 1300000
nfs_tcp_socketsize= 60000
nfs_setattr_error= 0
nfs_gather_threshold= 4096
nfs_repeat_messages= 0
nfs_udp_duplicate_cache_size= 1000
nfs_tcp_duplicate_cache_size= 1000
nfs_server_base_priority= 0
nfs_dynamic_retrans= 1
nfs_iopace_pages= 32
nfs_max_connections= 0
nfs_max_threads= 96
nfs_use_reserved_ports= 0
nfs_device_specific_bufs= 1
nfs_server_clread= 1
nfs_max_write_size= 0
nfs_max_read_size= 0
```

The first problem related to NFS that we generally see in large SP system configurations occurs when a single node is acting as the NFS server for a large number of nodes. In this scenario, the aggregate number of NFS requests can overwhelm the NFS socket or nfsd daemons on the server, or enough daemons cannot be configured.

When NFS is configured on a node, AIX, by default, configures 8 nfsd and 6 biod daemons. The nfsd handle NFS requests on the server. The biod daemons handle block I/O, primarily in NFS writes from the client node.

For the server configuration, nfsd daemons are the primary concern. If you have a 64-node configuration and configured one NFS server for the other 63 nodes, and if all 63 client nodes made an NFS request at the same time, you will need at least 63 nfsd daemons on the server. If you had only 8 nfsd daemons, as in the default configuration, then 55 NFS requests would have to wait and they could time out. Obviously, average NFS response time will be very slow.

However, there is a limit on how many nfsd daemons you can configure before the amount of processing on behalf of the NFS traffic overwhelms the server. Generally, when you configure more than 100 nfsd daemons you start to see NFS performance degradation, depending on the characteristics of the NFS traffic. The size of the requests, the mix of NFS operations, and the number of processes on each node that generates NFS requests influence the amount of NFS performance degradation. Examine `nfsstat` command output to check indications that you are generating too much NFS traffic to a server.

```
Server nfs:
calls      badcalls   public_v2  public_v3
1944188    0          0          0
Version 2: (1939153 calls)
null       getattr    setattr    root       lookup     readlink   read
271345 13% 164811 8%  3021 0%    0 0%       723431 37% 269332 13% 399288 20%
wrcache    write      create     remove     rename     link       symlink
0 0%       30071 1%   1924 0%    1725 0%    871 0%     23 0%      4 0%
mkdir      rmdir      readdir    statfs
10 0%      3 0%       63445 3%   9849 0%
Version 3: (5035 calls)
null       getattr    setattr    lookup     access     readlink   read
57 1%      424 8%     0 0%       1175 23%   74 1%      8 0%       19 0%
write      create     mkdir      symlink    mknod      remove     rmdir
3180 63%   0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
0 0%       0 0%       12 0%      17 0%      27 0%      34 0%      0 0%
commit
8 0%
```

If you see RPC timeouts or RPC retransmits when checking on the client nodes, then you probably are overwhelming the current number of nfsd daemons or nfs_socketsize size on the server.

```
Client rpc:
calls      badcalls   retrans    badxid     timeout    wait       newcred
3595281    695        0          665        690        0          0
```

Determining the number of biod daemons to configure on the client node is a bit complicated. The biod daemons are used when writing to an NFS server from a client node. Typically you would configure enough biod daemons on a client node for the NFS write traffic that you expect. However, NFS has a limit of 6 biod daemons on a client node per NFS mounted file system. If you have only one process on a client node writing to a single file system on the server, there is no reason to increase the number of biod daemons. And, if you have only one remote mounted NFS file system on each client, you can use only 6 biod daemons no matter how many processes are writing to the file system.

The limitation to 6 biod’s on a client node per NFS mounted file system is imposed by NFS. In NFS Version 2.0, all NFS writes use a fully synchronous 4 KB write model. When you write to an NFS-mounted file system, you can send only 4 KB at a time from the client to the server. If you have only one I/O outstanding at a time, this is very slow. In a fully synchronous model the data must be written on the disk at the server before the client can send the next 4 KB block. To speed this up, NFS allows staging 6 requests at the client per file system, but no more. Given the time it takes to write a block on disk in AIX, one block at a time is very slow.

If you are using AIX 4.2.1 or higher, on the server and client, you have the ability of running NFS Version 3.0. This version provides improved performance by using larger I/O request sizes. In NFS V2.0, the read request size was 8 KB and the write size 4 KB. In NFS V3.0, you can increase the read and write request size to 32 KB each. This larger I/O request size improves performance due to the larger amount of data per request. In addition, NFS V3.0 has the ability to use asynchronous write. This improves write performance to close to read performance. However, there is a very slight risk that data can be lost if the server crashes.

If there is more than one NFS-mounted file system on your client nodes, you can configure up to 6 biod daemons per file system. If the number of biod’s gets too high, performance will suffer in the same way as when too many nfsd daemons are configured on the server. Again, a general rule is that more than 100 biod’s will start to degrade performance.

In general, the same rules discussed in the previous section apply for nfsds on the client node when the client is also used as NFS server. However, we do not suggest that you use the same number of nfsd daemons on the client nodes acting as servers when a large number of nodes all use the same server (16 clients to 1 server, for example). If all nodes have 16 biod daemons, each nfsd daemon on the server nodes can potentially be sent 16 NFS requests. In the situation where you use nodes as server and client, monitor `nfsstat` on both the client and server nodes to look for RPC timeouts and retransmits.

If you already have a lot of nfsd daemons configured on the server, the best solution is to split the NFS server duties across several nodes, and keep the potential number of concurrent NFS requests below 100. As a general guideline, configuring up to 100 nfsd’s on a server usually works well on the SP system.

The other parameters that need addressing are the nfs_socketsize and nfs_tcp_socketsize parameters. These settings can be found using the `nfso` command. The current default values of 60000 are much too small for use on an SP system. These parameters specify the amount of space available in the NFS queue of pending requests. This queue is eventually read, and the requests handed off to an nfsd. In the queue, write requests include the data to be written so that enough space must be allocated to handle concurrent write requests to a server. If this size is too small, then the client node will record NFS timeouts and retries as requests are dropped at the server.

In NFS V2.0 you only have the nfs_socketsize. In NFS V3.0, since it now includes the ability to use TCP as the transport mechanism between client and server, you must make sure that both these parameters are set to the correct size. To determine the appropriate size on the server, you need to calculate the maximum number of servers that will be writing to the server at the same time, times the number of file systems they will be writing to, times 6, which is the maximum number of biod's per remote file system. By taking this number, and multiplying it by 4K for NFS V2.0 or 32K for NFS V3.0, you can estimate the queue space needed.

If the sizes needed for nfs_socketsize and nfs_tcp_socketsize are very large, you might want to reduce the number of biod's per mounted file system from the client nodes. If you do, then you can reduce the size previously determined by using the smaller number of biod's per file system rather than the value of 6 .

For more detailed information regarding tuning NFS, refer to the *IBM AIX Versions 3.2 and 4 Performance Tuning Guide*, SC23-2365, and the *AIX System Management Guide: Communications and Networks*, GC23-2487.

11.5 SP Environment Tuning for Performance

This section provides some tuning considerations for typical SP environments.

11.5.1 Tuning for Development Environments

The typical development environment on the SP consists of several users all developing an application or running small tests on a system. Tuning such an environment is usually based on having reasonable TCP/IP performance to prevent exhaustion of the buffer pools. It is important that aggregate requirements from several developers do not exhaust system resources.

The following table provides network tunable settings designed as initial values for development user environments. To temporarily change these values on each node, use the command:

```
dsh -w node "/usr/sbin/no -o option=value"
```

To preserve these values across booting of the nodes, we suggest that you change the `tuning.cust` script on each node. These settings are only initial suggestions. Start with them and realize you may need to change them.

thewall	=16384
sb_max	=1310720
subnetsarelocal	=1
ipforwarding	=1
tcp_sendspace	=65536
tcp_recvspace	=65536
udp_sendspace	=32768
udp_recvspace	=65536
rfc1323	=1
tcp_mssdflt	=1448
tcp_pmtu_discover (new in AIX 4.2.1)	=1
udp_pmtu_discover (new in AIX 4.2.1)	=1

These initial values are derived by expecting the network traffic to be small packets with lots of socket connections. tcp_sendspace and tcp_recvspace are kept small so that a single socket connection cannot use up lots of network buffer space, causing buffer space starvation. It is also set so that high performance for an individual socket connection is not expected. However, in aggregate, if lots of sockets are active at any one time, the overall resources will enable high aggregate throughput over the switch.

11.5.2 Tuning for Scientific and Technical Environments

The typical scientific and technical environment usually has only a few network sockets active at any one time, but can require large amounts of data. The information below addresses setting up the network tunables so that a single socket connection or a few connections can get the full SP Switch bandwidth. In doing this, however, you can cause problems on small packet networks like Ethernet and Token Ring. This is the trade-off that has to be made to get peak performance out of the SP system.

To achieve peak data transfer across a switch using TCP/IP, you need to increase the `no` tunables that affect buffer sizes, queue sizes, and the TCP/IP window. The tunables to adjust are:

thewall

```
sb_max
tcp_sendspace
tcp_recvspace
rfc1323
```

To get the best TCP/IP transfer rate, you need to size the TCP/IP window large enough to keep data streaming across the SP Switch without stopping the IP stream. The switch has a Maximum Transmission Unit (MTU) of 65520 bytes. This is the largest buffer of data that it can send. When using TCP/IP, TCP will send as many buffers as it can until the total data sent without acknowledgment from the receiver reaches the tcp_sendspace value. We found that having at least 4 buffers as the size of the window, allows TCP/IP to reach high transfer rates. Faster nodes can require a greater number of buffers. However, if you set tcp_sendspace and tcp_recvspace to 655360, it can hurt the performance of the other network adapters connected to the node. This can cause adapter queue overflows. The settings below are only initial suggestions. Start with them and realize you may need to change them.

```
thewall =16384
sb_max =1310720
subnetsarelocal =1
ipforwarding =1
tcp_sendspace =655360
tcp_recvspace =655360
udp_sendspace =65536
udp_recvspace =655360
rfc1323 =1
tcp_mssdflt =(Varies depending on other network types.)
tcp_pmtu_discover (new in AIX 4.2.1) =1
udp_pmtu_discover (new in AIX 4.2.1) =1
```

11.5.3 Tuning for Commercial and Database Environments

The following table provides network tunable settings designed as initial values for commercial and database user environments. These initial settings are derived from the fact that commercial and database type applications generally have lots of network connections between nodes. For environments with lots of active connections, tcp_sendspace and tcp_recvspace need to be adjusted so that the aggregate demand across all connections does not exceed the available buffer space for the SP switch. Commercial and database environments where only a few connections are active may be able to increase the tcp_sendspace and tcp_recvspace sizes to get better per connection performance over the switch.

You need to change these values on each of the installed nodes. To temporarily change these values, use the `no` command. If you want these values to be preserved across booting the nodes, we suggest that you change the `tuning.cust` script on each node. Details on how to change the script are found in 11.3, “Files Used on the SP for Tuning” on page 209. These settings are only initial suggestions. Start with them and realize that you may need to change them.

thewall =16384
sb_max =1310720
subnetsarelocal =1
ipforwarding =1
tcp_sendspace =221184
tcp_recvspace =221184
udp_sendspace =65536
udp_recvspace =655360
rfc1323 =1
tcp_mssdflt =1448
tcp_pmtu_discover (new in AIX 4.2.1) =1
udp_pmtu_discover (new in AIX 4.2.1) =1

These initial values are derived by expecting the network traffic to be small packets with lots of socket connections. However, when running a parallel database product, you want to be able to get as much SP switch throughput to a single connection as you can without causing problems on other network adapters. The settings are also designed to enable a single socket to be able to send to an Ethernet adapter without causing adapter queue overruns. In addition, tcp_sendspace and tcp_recvspace are large enough to get most of the switch bandwidth at database size packets.

If other applications are run on the same node that have vastly different network characteristics, such as ADSM, or are a data mining type application that tends to use few sockets, these settings will not provide peak performance. In these cases, the TCP window settings may have to be increased. Conflicts with the settings needed by ADSM can be resolved by having ADSM do its own socket-level tuning. See 11.6.1, “Tuning for the ADSTAR Distributed Storage Manager (ADSM)” on page 221 for more information.

11.6 Application-Specific Tuning

To get peak performance over the SP Switch for various applications is to tune application sockets using tunable files. Set reasonable defaults based on generic application network traffic characteristics and provide a socket tuning input file so the users can select their own optimized settings. In general, however, the best place to look for tuning information is the product documentation itself.

11.6.1 Tuning for the ADSTAR Distributed Storage Manager (ADSM)

When running ADSM on the SP system, the performance of a backup or restore is affected by several tunable settings. However, the tunables and their settings differ depending on the backup destination or restore source. The following section lists the settings for initially trying to run the ADSM server on an SP system node. Because ADSM is not always the only application running on a client node or server, some of these values may not be reasonable in your environment.

The following are the no tunable values that will achieve the best ADSM performance on the SP ADSM server node:

```
thewall =16384
sb_max =1310270
rfc1323 =1
tcp_mssdflt =32768
```

If you already have a larger value for thewall, you need to keep it.

The following are the entries in the /usr/lpp/adsm/bin/dsmserv.opt file on the SP system ADSM server node:

```
TCPWindowsize =256
TCPBuffsize =32
Txnbyte =25600
tcpnodelay =Y
```

11.6.2 Tuning for Virtual Shared Disk (VSD) Servers

The following information is excerpted from VSD tuning information in the VSD manuals. It serves as a quick reference to tuning and performance of VSD, but does not address everything about configuration, setup etc. If you need more information, refer to *PSSP: Managing Shared Disks*, SA22-7349.

The processing capability required for VSD service is a function of the application I/O access patterns, the node model, the type of disk, and the disk connection. If you plan to run time-critical applications on a VSD server, remember that servicing disk requests from other nodes might conflict with the demands of these applications.

Make sure that you have sufficient resources to run the VSD program efficiently. This includes enough buddy buffers of sufficient size to match your file system block size, as well as sufficient rpoolsize and spoolsize blocks in the communications subsystem.

Buddy Buffers:

The VSD server node uses buddy buffers to temporarily store data for I/O operations originating at a non-server node. Buddy buffers are used only when a shortage in the switch buffer pool occurs, or on certain networks. In contrast to the data in the cache buffer, the data in a buddy buffer is purged immediately after the I/O operation completes.

Buddy buffer space is allocated in powers of 2. If an I/O request size is not a power of 2, the smallest power of two that is larger than the request is allocated. For example, for a request size of 24 KB, 32 KB is allocated on the server.

If you do not plan to create any GPFS file system with a block size larger than 16KB, two or three buddy buffers of 16 KB each on VSD server nodes should suffice. Create one buddy buffer of 16 KB on each non-server node.

If you do plan to create a GPFS file system with a block size larger than 16 KB, one buddy buffer equal to the largest block size used will do for non-server nodes, but the number of buddy buffers on VSD server nodes should be sufficient to handle the maximum number of disk I/Os expected at any given time. Start with two buffers per physical disk attached to a server. If you monitor the output of the *statvsd* command for queued buddy buffer requests, you can determine whether this setting is correct.

Tuning Virtual Shared Disks:

Each VSD requires parameters in the System Data Repository (SDR). These parameters must be set to sufficient values in order to operate efficiently:

IP Packet Size
Cache Buffers
Request Count
Buddy Buffers

The purpose of these parameters is outside the scope of this document. Consult *PSSP: Managing Shared Disks*, SA22-7349 for detailed descriptions. The following is an example of the `vsdnode` command that could be used to initially set up VSD on an 8-node SP:

```
vsdnode 1 2 3 4 5 6 7 8 css0 64 256 256 48 4096 262144 33 61440
```

This is only an example; the actual settings you may need will vary. On a system where VSD is already set up, these parameters could be changed using the `updatevsdnode` command instead.

To verify these values on the SP nodes, enter

```
vsdatalst -n
```

Output similar to the following is returned:

```
VSD Node Information
                                              Initial Maximum     VSD      rw      Buddy Buffer
   node                    VSD      IP packet   cache   cache request request minimum maximum size: #
  number host_name         adapter    size    buffers buffers   count   count    size    size maxbufs
  ------ --------------- -------- --------- ------- ------- ------- ------- ------- ------- -------
       1 k13n01.ppd.pok. css0         61440      64     256     256      48    4096  262144      33
       2 k13n02.ppd.pok. css0         61440      64     256     256      48    4096  262144      33
       3 k13n03.ppd.pok. css0         61440      64     256     256      48    4096  262144      33
       4 k13n04.ppd.pok. css0         61440      64     256     256      48    4096  262144      33
       5 k13n05.ppd.pok. css0         61440      64     256     256      48    4096  262144      33
       6 k13n06.ppd.pok. css0         61440      64     256     256      48    4096  262144      33
       7 k13n07.ppd.pok. css0         61440      64     256     256      48    4096  262144      33
       8 k13n08.ppd.pok. css0         61440      64     256     256      48    4096  262144      33
```

For more information on buddy buffers, refer to *PSSP: Managing Shared Disks*, SA22-7349.

11.6.3 Tuning for GPFS

Two SP Switch tunable parameters affect GPFS performance: rpoolsize and spoolsize. Insufficient memory allocation to these switch pools can result in major performance degradation. Unless your nodes are very short of memory, we recommend that both of these pools be set to 16 MB (the maximum allowed).

You can use SMIT or the `mmchconfig` command to change the following GPFS configuration attributes after the initial configuration has been set:

1. pagepool
2. mallocsize
3. priority
4. autoload
5. client_ports

6. server_port_number
7. server_kprocs

Attributes 1 through 3 take effect the next time GPFS is started. Attributes 4 through 7 require that the nodes be rebooted before new values take effect.

Part 5. Appendices

Appendix A. SP Switch Service Interface

In this appendix we describe in detail the service packets defined in the SP Switch. We also describe the First Error Capture Register and the Second Error Capture Register.

A.1 Service Packets

There are five service packets that the primary node can send to a switch chip:

- Initialization Packet
- Read Status Packet
- Reset Error Packet
- Set TOD Packet
- Send TOD Packet

A switch chip can generate a single type of packet, either as an acknowledgment or to report an error:

- Error/Status Packet

They are all described in the following sections.

Attention

In a byte, bit 0 is the most significant bit. For instance, if a byte has the hexadecimal value 0x40, only bit 1 is set on the byte.

When a byte describes the status of the switch chip ports, bit 0 represents port 0, bit 1 represents port 1, and so on.

A.1.1 Initialization Packet

The initialization service packet is used to configure all the parameters of the switch chip. After having updated all its internal registers and variables, the chip returns an Error/ Status service packet with the actual configuration as an acknowledgment to the initialization command.

The packet contains the information described in Table 18

Table 18. Initialization Service Packet

Byte	Description
0	Packet Length/Service Command
1	Reserved
2:9	Primary Route Table
10:17	Secondary Route Table
18:19	Chip Identification
20	Link Round Trip Time-Out Threshold
21	Receiver End-of-Packet Time-Out Threshold
22	Receiver EDC Error Count Threshold
23	Sender Token Error Count Threshold
24	Receiver Bypass Path Enables
25	Receiver Central Queue Path Enables
26	Receiver Link Enable Bits
27	Sender Link Enable Bits
28	EDC Frame Length
29	Mode Bits
30	EDC Error Enables
31	Parity Error on Route Enables
32	Undefined Control Character Error Enables
33	Unsolicited Data Error Enables
34	Receiver Lost EOP Error Enables
35	Reserved
36	STI Data Re-Time Request reporting Enables
37	Receiver Link Synchronization Error Enables
38	FIFO Overflow Error Enables
39	Token Count Miscompare Error Enables

Byte	Description
40	EDC Error Threshold Reporting Enables
41	Receiver State Machine Error Enables
42	Parity Error on Data Enables
43	Token Sequence Error Enables
44	Invalid Route Error Enables
45	Reserved
46	STI-Re-Time Request Reporting Enables
47	Token Count Overflow Error Enables
48	Token Error Threshold Reporting Enables
49	Sender Link Synchronization Error Enables
50	Sender State Machine Error Enables
51	Central Queue Error Enables
52	Service Error Enables
53	Reserved
54-63	Padded Out by Adapter to 8-Byte Boundary

The two Route Tables, primary and secondary, are used when the switch chip is either reporting errors or generating acknowledgments. The three least significant bits of the last byte will indicate how many of the previous seven route bytes were valid. If the destination node is directly attached to the switch chip, the length field will be equal to zero. The destination port through which the service logic will transmit the Error/Status packet must be in the most significant nibble of the first byte of the route table specification, even for a route of length equal to zero.

The Chip Identification is used as part of the template for generating Error/Status packets which are sent to the service processor. The service software uses this chip address to identify each unique chip in the system. The service software can define the chip address any way it sees fit; the switch does not care about the meaning of the chip address.

The Link Round Trip Time-out Threshold is used by each receiver port to control link synchronization.

The Receiver End-of-Packet Time-out Threshold tells each receiving port how long they must wait for the end of the packet. If the timer expires, the receiving port inserts a Packet Fail character and starts waiting for a Beginning of Packet.

When the EDC Errors on a switch chip link reach the EDC Threshold Error, an Error/Status packet is sent and the link starts the initialization process.

When a sender’s Token Error counters reach the Sender Token Error Count Threshold, an Error/Service packet is sent and the link starts the initialization process.

The Receiver Bypass Path Enable Bits are used by each receiver state machine to determine if packets are allowed to take a bypass path to the destination port. If a receiver has its bypass path disabled, it is forced to buffer all packets into the central queue buffer.

The Receiver Central Queue Path Enable Bits are used by each receiver state machine to determine if packets are allowed to use the central queue buffer. If a receiver has its central queue path disabled, it is forced to transmit all packets across the bypass path.

The Receiver Link Enable Bits are used by each receiver state machine to determine if they should accept packets or not. One cannot disable *all* the links of the chip since this makes it impossible for the chip to receive a new configuration packet.

The Sender Link Enable Bits are used by each sender state machine to determine if it should send packets or not. One cannot disable *all* the links of the chip since this makes it impossible for the chip to send error reports.

The EDC Frame Length is used by each of the senders to define the number of bytes per EDC Frame. The sender notifies its corresponding receiver of the change in the buffer.

The Mode Bits are used to control the Central Queue data allocation.

The EDC Error Enable Bits are used by each receiver logic to determine if single EDC errors should be reported when detected. The EDC Error Threshold Counter, however, is increased even if this error is disabled.

Bytes 31 through 50 are used to control which errors should be reported when they occur.

Central Queue Error Enables controls the reporting of errors in the management of the Central Queue.

Service Error Enables are used to enable a set of reports on the consistency of service packets received:

- Incorrect CRC on a Service Packet
- Incorrect Service Packet Length
- Parity Error on Inbound FIFO
- Parity Error on Route Table
- Invalid Link Enable
- Send TOD Error
- State Machine Error

Bytes 53 through 63 are reserved for future use.

A.1.2 Read Status Packet

This service packet is used to request the switch chip to send an Error/ Status service packet with the actual configuration.

The packet contains the information described in Table 19:

Table 19. Read Status Service Packet

Byte #	Description
0	Service Command
1:7	Padded Out by the Adapter to 8-Byte Boundary

Only the command is needed, since the real action to be taken is to generate the Error/Status packet. The remainder of the bytes are reserved for future use.

A.1.3 Reset Error Packet

The Reset Error service packet requests that a switch chip reset sections of logic and/or error registers. The switch chip will send only one Error/Status packet when it encounters one of the error scenarios defined in Table 20, and will not send another until another Reset Error packet is received. This behavior ensures that the switch network is not flooded with service packets and the primary node does not miss any error. The packet data is defined in Table 20.

Table 20. Reset Error Service Packet

Byte #	Description
0	Service Command
1	Reserved
2:3	Receiver Error Resets
4	Receiver EDC Error Counter Resets
5	Receiver Port Logic Resets
6:7	Sender Error Resets
8	Sender Token Error Counter Resets
9	Sender Port Logic Resets
10	Central Queue Resets (Error and Logic)
11	Service Resets (Error and Logic)
12:15	Padded Out by Adapter to 8-Byte Boundary

The two Receiver Error Resets bytes are used to reset the second error capture registers defined as receiver-type errors. Their meaning is the following:

- High Byte (byte #2)
  - Bits 0:3 are undefined
  - Bit 4 resets the EDC Error Register
  - Bit 5 resets the Parity Error on Route Register
  - Bit 6 resets the Undefined Control Character Error Register
  - Bit 7 resets the Unsolicited Data Error Register
- Low Byte (byte #3)
  - Bit 0 resets the Lost EOP Error Register
  - Bit 1 is reserved
  - Bit 2 resets the STI Data Re-Time Reporting Register
  - Bit 3 resets the Link Synchronization Error Register
  - Bit 4 resets the FIFO Overflow Error Register

    - Bit 5 resets the Token Count Miscompare Error Register
    - Bit 6 resets the EDC Error Count Threshold Error Register
    - Bit 7 resets the State Machine Error Register

The Receiver EDC Error Threshold Counter Reset byte is used to reset the EDC Error Threshold Counter of one or more receiver modules in the switch chip. Each bit corresponds to one receiver. The counters can be used as an instrumentation device, where the software monitors may come in and read the counter values for each receiver (Read Status packets), and reset them as they please to monitor link error activity. The Receiver EDC Error Threshold counter is automatically reset when its threshold value has been reached and the error is enabled.

The Receiver Port Reset Bits are used by the service logic to determine if the receiver logic should be reset to a known (Disabled) state.

The two Sender Error Resets bytes are used to reset the second error capture registers defined as receiver type errors. Their meaning is the following:

- High Byte (byte #6)
    - Bits 0:6 are undefined
    - Bit 7 resets the Parity Error on Data Register
- Low Byte (byte #7)
    - Bit 0 resets the Token Sequence Error Register
    - Bit 1 resets the Invalid Route Error Registers
    - Bit 2 is reserved
    - Bit 3 resets the STI Token Re-Time Reporting Register
    - Bit 4 resets the Token Count Overflow Error Register
    - Bit 5 resets the Token Error Threshold Register
    - Bit 6 resets the Link Synchronization Error Register
    - Bit 7 resets the State Machine Error Register

The Sender Token Error Threshold Counter Reset Bits are used to reset the Token Error Counter of one or more sender modules in a switch chip. Each bit represents a sender port. These counters can be used as an instrumentation device, where the software monitors may come in and read the counter values for each sender (Read Status packets), and reset them as they please to monitor link error activity. The Sender Token Error Threshold counter is automatically reset when its threshold value has been reached and the error is enabled.

The Sender Port Reset Bits are used by the switch chip’s sender module to determine if the sender logic should be reset to a known (Disabled) state.

The Central Queue Error Resets and Control Logic Reset bytes are used to reset the second error capture registers defined as central queue type errors, and to reset the central queue logic. The bits are used in the following way:

- Bits 0:3 are undefined
- Bit 4 resets the Next Chunk Linked List Initialization Error Latch
- Bit 5 resets the Next Message Linked List Error Latch
- Bit 6 resets the Next Chunk Linked List Error Latch
- Bit 7 resets the Central Queue logic

The Service Error Enables and Control Logic Reset bytes reset the second error capture registers defined as service-type errors, and also reset the service logic. In the latter case, the bits are used in the following way:

- Bit 0 resets the CRC Error Latch
- Bit 1 resets the Length Error Latch
- Bit 2 resets the Parity Error on Inbound FIFO Latch
- Bit 3 resets the Parity Error on Route Table Latch
- Bit 4 resets the Invalid Link Enable Error Latch
- Bit 5 resets the Send TOD Error Latch
- Bit 6 resets the State Machine Error Latch
- Bit 7 resets the Service logic

The last action of the service state machine when it finishes handling this service message is to reset the First Error Capture Register, thereby enabling the reporting of new errors. This will be done for any Reset Packet received, even if there are no reset bits specified in the packet.

A.1.4 Set Time-of-Day Packet

The service Set TOD packet is a short packet that sets the local time-of-day counter on the switch chip. This packet may be sent by the primary node or by a switch chip or adapter when a Send time-of-day packet is received. The packet data is defined in Table 21.

Table 21. Set Time-of-Day Packet

Byte #	Description
0	Service Command
1	Reserved
2:7	Reserved
8:14	Time-of-Day
15	Time-of-Day Delay

The time-of-day field is the value of the TOD counter inserted by the transmitting adapter or switch.

The time-of-day Delay field is the calculated delay between the launch of the Set TOD packet from the adapter or the switch to the loading of the TOD clock at the destination adapter or switch. This value is calculated by the software.

The time-of-day Delay is concatenated with the time of day and loaded into the destination chips TOD counter. The switch inherits this new time of day, and starts incrementing the TOD counter from there.

A.1.5 Send Time-of-Day Packet

The service Send TOD packet is used by the primary node to make a switch send the time-of-day counter value to an adjacent device, adapter or switch, using a Set time-of-day packet. The packet data is defined in Table 22.

Table 22. Send Time-of-Day Packet

Byte #	Description
0	Service Command
1	Reserved
2:6	Reserved
7	Transmit Port
8:14	Reserved
15	Time-of-Day Delay

The Transmit Port contains in the three least significant bits from which of the eight send ports of the switch chip the resulting Set TOD packet is to be transmitted through. This value is stored in an 8-bit register which is used to make a connection request to the proper send port when the Set TOD packet is ready for transmission.

The time-of-day Delay is the calculated delay between the launch of the Set TOD packet from the adapter or the switch to the loading of the TOD clock at the destination adapter or switch. This value is calculated by the software.

A.1.6 Error/Status Packet

A switch chip sends a packet to the primary node using the two routes defined during initialization in response to one error or as an acknowledge to the Read Status Service Packet, an Initialization Service Packet, a Reset Service Packet, or a Set TOD Service Packet. The Error/Status packets are transmitted sequentially and not in parallel across the two routes. The Error/Status packet contains the information in Table 23:

Table 23. Error/Status Packet

Byte #	Definition
0	Service Command
1	Reserved
2:3	Chip Identification Number
4	Sequence Number
5	Reserved
6:8	First Error Capture Register
Second Error Capture Registers	
9	EDC Error Register
10	Parity Error on Route Register
11	Undefined Control Character Error Register
12	Unsolicited Data Error Register
13	Receiver Lost EOP Error Register
14	Reserved
15	STI Data Re-Time Request reporting Register
16	Receiver Link Synchronization Error Register
17	FIFO Overflow Error Register

Byte #	Definition
18	Token Count Miscompare Error Register
19	EDC Error Threshold reporting Register
20	Receiver State Machine Error Register
21	Parity Error on Data Register
22	Token Sequence Error Register
23	Invalid Route Error Register
24	Reserved
25	STI-Re-Time Request Reporting Register
26	Token Count Overflow Error Register
27	Token Error Threshold Reporting Register
28	Sender Link Synchronization Error Register
29	Sender State Machine Error Register
30	Central Queue Error Register
31	Service Error Register
Chip Status	
32:33	Receiver0 States
34:35	Receiver1 States
36:37	Receiver2 States
38:39	Receiver3 States
40:41	Receiver4 States
42:43	Receiver5 States
44:45	Receiver6 States
46:47	Receiver7 States
48	Sender0 States
49	Sender1 States
50	Sender2 States
51	Sender3 States

Byte #	Definition
52	Sender4 States
53	Sender5 States
54	Sender6 States
55	Sender7 States
56	Receiver0 EDC Error Count Value
57	Receiver1 EDC Error Count Value
58	Receiver2 EDC Error Count Value
59	Receiver3 EDC Error Count Value
60	Receiver4 EDC Error Count Value
61	Receiver5 EDC Error Count Value
62	Receiver6 EDC Error Count Value
63	Receiver7 EDC Error Count Value
64	Number of Free Addresses in Receiver0 FIFO
65	Number of Free Addresses in Receiver1 FIFO
66	Number of Free Addresses in Receiver2 FIFO
67	Number of Free Addresses in Receiver3 FIFO
68	Number of Free Addresses in Receiver4 FIFO
69	Number of Free Addresses in Receiver5 FIFO
70	Number of Free Addresses in Receiver6 FIFO
71	Number of Free Addresses in Receiver7 FIFO
72	Sender0 Token Error Count Value
73	Sender1 Token Error Count Value
74	Sender2 Token Error Count Value
75	Sender3 Token Error Count Value
76	Sender4 Token Error Count Value
77	Sender5 Token Error Count Value
78	Sender6 Token Error Count Value

Byte #	Definition
79	Sender7 Token Error Count Value
80	Number of Tokens Queues in Sender0
81	Number of Tokens Queues in Sender1
82	Number of Tokens Queues in Sender2
83	Number of Tokens Queues in Sender3
84	Number of Tokens Queues in Sender4
85	Number of Tokens Queues in Sender5
86	Number of Tokens Queues in Sender6
87	Number of Tokens Queues in Sender7
88	Number of Free Addresses in the Central Queue (High)
89	Number of Free Addresses in the Central Queue (Low)
90	Number of Messages in the Central Queue Queued for Sender0
91	Number of Messages in the Central Queue Queued for Sender1
92	Number of Messages in the Central Queue Queued for Sender2
93	Number of Messages in the Central Queue Queued for Sender3
94	Number of Messages in the Central Queue Queued for Sender4
95	Number of Messages in the Central Queue Queued for Sender5
96	Number of Messages in the Central Queue Queued for Sender6
97	Number of Messages in the Central Queue Queued for Sender7
Initialization Packet Data	
98:105	Primary Route Table
106:113	Secondary Route Table
114	Link Round Trip Time-Out Threshold
115	Receiver End-of-Packet Time-Out Threshold
116	Receiver EDC Error Count Threshold
117	Sender Token Error Count Threshold
118	Receiver Bypass Path Enables

Byte #	Definition
119	Receiver Central Queue Path Enables
120	Receiver Link Enable Bits
121	Sender Link Enable Bits
122	EDC Frame Length
123	Mode Bits
124	EDC Error Enables
125	Parity Error on Route Enables
126	Undefined Control Character Error Enables
127	Unsolicited Data Error Enables
128	Receiver Lost EOP Error Enables
129	Reserved
130	STI Data Re-Time Request Reporting Enables
131	Receiver Link Synchronization Error Enables
132	FIFO Overflow Error Enables
133	Token Count Miscompare Error Enables
134	EDC Error Threshold reporting Enables
135	Receiver State Machine Error Enables
136	Parity Error on Data Enables
137	Token Sequence Error Enables
138	Invalid Route Error Enables
139	Reserved
140	STI-Re-Time Request Reporting Enables
141	Token Count Overflow Error Enables
142	Token Error Threshold Reporting Enables
143	Sender Link Synchronization Error Enables
144	Sender State Machine Error Enables
145	Central Queue Error Enables

Byte #	Definition
146	Service Error Enables
147	Reserved
148:155	TOD Counter
156:251	Reserved
252:255	Signature (MISR Data)

The two-byte Chip Identification uniquely identifies each switch chip to the service processor. It is defined to the chip during initialization through the initialization service packet.

The Sequence Number is an eight-bit number which uniquely identifies each pair of Error/Status packets transmitted from a given switch chip to their respective primary and secondary nodes. Both the first and second packets have the same sequence number. The service logic increments the sequence number at the end of the Error/Status packet transmission.

The First Error Capture Register and the Second Error Capture Register collect all events on the switch chip, separating the first event from the following ones. See A.2, “Error Registers” on page 242 for more details.

The Chip Status bytes give information relative to the status of the switch chip. This information can be used for debugging, instrumentation, and monitoring the switch chip. Several parameters are collected from each sender and receiver module and from the central queue which provide a snapshot of the chip status. Network congestion, link and protocol problems can be monitored and isolated using this data.

The Initialization Register Values are set up during chip initialization. They are read out to verify that an initialization packet has been received properly.

The time-of-day Counter Value is included for two reasons. First, it puts a time stamp on the Error/Status packet. Secondly, the most significant bit of the TOD clock indicates whether or not the TOD clock has ever been initialized. A value of logic 1 says that the TOD clock has been initialized. If the Error/Status packet is generated as a result of an error, the TOD value contains the time that the error occurred.

The signature/MISR is a 32-bit number stored at the completion of the Built-In Self Test. It is included as a means for software to get at the results of the chips' built-in self test. The signature for all switch chips will be identical if the chips pass the self test.

A.2 Error Registers

To isolate errors, the first error that occurs is captured in the First Error Capture Register and no other errors are added to that register until it has been explicitly reset by a Reset packet. Subsequent errors are added to the Second Error Capture Register

Attention

As previously mentioned, in a byte, bit 0 is the most significant bit. For instance, if a byte has the hexadecimal value 0x40, only bit 1 is set on the

A.2.1 First Error Capture Register

The First Error Capture Register should have at most one bit set, indicating whether or not an error has occurred and which error it was. To discover on which module the first error occurred, look at the Second Error Capture Register. The contents of the register are described in Table 24, making reference to the Error/Status packet's bytes.

Table 24. First Error Register

Byte of Error/Status Packet	Bit	Meaning
6	0	Reserved
	1	EDC Error
	2	Parity Error on Route
	3	Undefined Control Character
	4	Unsolicited Data
	5	Receiver Lost EOP
	6	Reserved
	7	STI Data Re-Time Request

Byte of Error/Status Packet	Bit	Meaning
7	0	Receiver Link Synchronization Fail
	1	FIFO Overflow
	2	Token Count Miscompare
	3	EDC Error Threshold
	4	Receiver State Machine Error
	5	Parity Error on Data
	6	Token Sequence Error
	7	Sender Invalid Route
8	0	Reserved
	1	STI Token Re-Time Request
	2	Token Count Overflow
	3	Token Error Threshold
	4	Sender Link Synchronization Fail
	5	Sender State Machine Error
	6	Central Queue Error
	7	Service Error

A.2.2 Second Error Capture Register

The Second Error Capture Register details exactly where the error indicated in the First Error Capture Register occurred, as well as any subsequent errors. The Second Error Capture Register continues to accumulate errors until it is read and put into an Error/Status packet.

A second set of registers (the Pending Error Capture Register described in Table 25) is used to collect errors that occur between the read of the register and the reset of the register. Once the reset has occurred, the errors that were flagged between the read and the reset are loaded into the Second Error Capture Register. This prevents some errors from being missed.

Table 25 shows the Second Error Capture Register, as represented in the Error/Status packet:

Table 25. Second Error Capture Register

Byte	Meaning	Bit meaning
9	EDC Error	Receivers
10	Parity Error on Route	Receivers
11	Undefined Control Character Error	Receivers
12	Unsolicited Data Error	Receivers
13	Receiver Lost EOP	Receivers
14	Reserved	
15	STI Data Re-Time Reporting	Receivers
16	Receiver Link Synchronization Error	Receivers
17	FIFO Overflow Error	Receivers
18	Token Count Miscompare Error	Receivers
19	EDC Error Threshold Reporting	Receivers
20	Receiver State Machine Error	Receivers
21	Parity Error on Data	Senders
22	Token Sequence Error	Senders
23	Sender Invalid Route Error	Senders
24	Reserved	
25	STI Token Re-time Request Reporting	Senders
26	Token Count Overflow Error	Senders
27	Token Error Threshold Reporting	Senders
28	Sender Link Synchronization Error	Senders
29	Sender State Machine Error	Senders
30	Central Queue Error	0:4 = undefined 5 = NCLL Initialization Error 6 = NMLL Error 7 = NCLL Error

Byte	Meaning	Bit meaning
31	Service Error	0 = undefined 1 = CRC Error 2 = Length Error 3 = PE on Inbound FIFO Error 4 = PE on Route Table Error 5 = invalid Link Enable 6 = send TOD error 7 = State Machine Error

Appendix B. Example Configuration Files

In this appendix we list an example of a switch topology file and an example of a clock topoloy file.

B.1 Example of a Switch Topology File

This is the annotated version of the expected.top.2nsb.0isb.0 topology file.

```
#pragma comment (copyright, "@(#) expected.top.2nsb.0isb.0 1.1 1   7/11/94
14:37
:51 \0" )
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog.
#
#
#
# Licensed Materials - Property of IBM
#
# (C) COPYRIGHT International Business Machines Corp. 1993,1997
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# IBM_PROLOG_END_TAG
#
# FUNCTIONS:  This file describes the wiring configuration for the High
#             Performance Switch.  It is used during switch initialization
#             ("Estart" command).  It should not be changed unless the
#             node-to-switch or switch-to-switch cabling differs from the
#             prescribed pattern.
#
# FORMAT:  A node-to-switch connection looks like:  s 36 3   tb2 15 0
#                                                   | || |    |   |
#                                          Switch...| || |    |   |
#                                     in switch 3.....|| |    |   |
#                                          chip 6......| |    |   |
#                                          port 3........|    |   |
#               is connected to the TB0 or TB2 adapter......|   |
#                        in switch node number .................|
#
#      Switch-to switch connections just use the first four components
twice.
#
# ORIGINS: 27
```

```
#
#CPRY
# 5765-296 (C) Copyright IBM Corporation 1993,1994
# Licensed Materials - Property of IBM
# All rights reserved.
# US Government Users Restricted Rights -
# Use, duplication or disclosure restricted by
# GSA ADP Schedule Contract with IBM Corp.
#CPRY
#
####################################################################
#
#   Initial version - 3 Feb 94
#
####################################################################
#
format 1
32 28
# Node connections in frame L01 to switch 1 in L01
s 15 3  tb3 0 0           E01-S17-BH-J7 to E01-N1
s 15 2  tb3 1 0           E01-S17-BH-J8 to E01-N2 # Dependent Node
s 16 0  tb3 2 0           E01-S17-BH-J26 to Exx-Nxx
s 16 1  tb3 3 0           E01-S17-BH-J25 to Exx-Nxx
s 15 1  tb3 4 0           E01-S17-BH-J9 to E01-N5
s 15 0  tb3 5 0           E01-S17-BH-J10 to E01-N6
s 16 2  tb3 6 0           E01-S17-BH-J24 to E01-N7
s 16 3  tb3 7 0           E01-S17-BH-J23 to E01-N8
s 14 3  tb3 8 0           E01-S17-BH-J31 to E01-N9
s 14 2  tb3 9 0           E01-S17-BH-J32 to E01-N10
s 17 0  tb3 10 0          E01-S17-BH-J18 to E01-N11
s 17 1  tb3 11 0          E01-S17-BH-J17 to E01-N12
s 14 1  tb3 12 0          E01-S17-BH-J33 to E01-N13
s 14 0  tb3 13 0          E01-S17-BH-J34 to E01-N14
s 17 2  tb3 14 0          E01-S17-BH-J16 to E01-N15
s 17 3  tb3 15 0          E01-S17-BH-J15 to Exx-Nxx
# On board connections between switch chips on switch 1 in Frame L01
s 14 7    s 13 4          E01-S17-SC
s 14 6    s 12 4          E01-S17-SC
s 14 5    s 11 4          E01-S17-SC
s 14 4    s 10 4          E01-S17-SC
s 15 7    s 13 5          E01-S17-SC
s 15 6    s 12 5          E01-S17-SC
s 15 5    s 11 5          E01-S17-SC
s 15 4    s 10 5          E01-S17-SC
s 16 7    s 13 6          E01-S17-SC
s 16 6    s 12 6          E01-S17-SC
s 16 5    s 11 6          E01-S17-SC
```

```
s 16 4    s 10 6           E01-S17-SC
s 17 7    s 13 7           E01-S17-SC
s 17 6    s 12 7           E01-S17-SC
s 17 5    s 11 7           E01-S17-SC
s 17 4    s 10 7           E01-S17-SC
# Node connections in frame L02 to switch 2 in L02
s 25 3  tb3 16 0           E02-S17-BH-J7 to E02-N1
s 25 2  tb3 17 0           E02-S17-BH-J8 to Exx-Nxx
s 26 0  tb3 18 0           E02-S17-BH-J26 to Exx-Nxx
s 26 1  tb3 19 0           E02-S17-BH-J25 to Exx-Nxx
s 25 1  tb3 20 0           E02-S17-BH-J9 to E02-N5
s 25 0  tb3 21 0           E02-S17-BH-J10 to E02-N6
s 26 2  tb3 22 0           E02-S17-BH-J24 to E02-N7
s 26 3  tb3 23 0           E02-S17-BH-J23 to E02-N8
s 24 3  tb3 24 0           E02-S17-BH-J31 to E02-N9
s 24 2  tb3 25 0           E02-S17-BH-J32 to E02-N10
s 27 0  tb3 26 0           E02-S17-BH-J18 to E02-N11
s 27 1  tb3 27 0           E02-S17-BH-J17 to Exx-Nxx
s 24 1  tb3 28 0           E02-S17-BH-J33 to E02-N13
s 24 0  tb3 29 0           E02-S17-BH-J34 to Exx-Nxx
s 27 2  tb3 30 0           E02-S17-BH-J16 to E02-N15
s 27 3  tb3 31 0           E02-S17-BH-J15 to Exx-Nxx
# On board connections between switch chips on switch 2 in Frame L02
s 24 7    s 23 4           E02-S17-SC
s 24 6    s 22 4           E02-S17-SC
s 24 5    s 21 4           E02-S17-SC
s 24 4    s 20 4           E02-S17-SC
s 25 7    s 23 5           E02-S17-SC
s 25 6    s 22 5           E02-S17-SC
s 25 5    s 21 5           E02-S17-SC
s 25 4    s 20 5           E02-S17-SC
s 26 7    s 23 6           E02-S17-SC
s 26 6    s 22 6           E02-S17-SC
s 26 5    s 21 6           E02-S17-SC
s 26 4    s 20 6           E02-S17-SC
s 27 7    s 23 7           E02-S17-SC
s 27 6    s 22 7           E02-S17-SC
s 27 5    s 21 7           E02-S17-SC
s 27 4    s 20 7           E02-S17-SC
# switch 1 to switch 2
s 13 3    s 23 3           E01-S17-BH-J3 to E02-S17-BH-J3
s 13 2    s 23 2           E01-S17-BH-J4 to E02-S17-BH-J4
s 13 1    s 23 1           E01-S17-BH-J5 to E02-S17-BH-J5
s 13 0    s 23 0           E01-S17-BH-J6 to E02-S17-BH-J6
s 12 3    s 22 3           E01-S17-BH-J27 to E02-S17-BH-J27
s 12 2    s 22 2           E01-S17-BH-J28 to E02-S17-BH-J28
s 12 1    s 22 1           E01-S17-BH-J29 to E02-S17-BH-J29
```

```
s 12 0    s 22 0          E01-S17-BH-J30 to E02-S17-BH-J30
s 11 3    s 21 3          E01-S17-BH-J11 to E02-S17-BH-J11
s 11 2    s 21 2          E01-S17-BH-J12 to E02-S17-BH-J12
s 11 1    s 21 1          E01-S17-BH-J13 to E02-S17-BH-J13
s 11 0    s 21 0          E01-S17-BH-J14 to E02-S17-BH-J14
s 10 3    s 20 3          E01-S17-BH-J19 to E02-S17-BH-J19
s 10 2    s 20 2          E01-S17-BH-J20 to E02-S17-BH-J20
s 10 1    s 20 1          E01-S17-BH-J21 to E02-S17-BH-J21
s 10 0    s 20 0          E01-S17-BH-J22 to E02-S17-BH-J22
```

B.2 Example of a Clock Topology File

This is the clock file Eclock.top.7nsb.4isb.0.

```
# NAME:       Eclock.top.7nsb.4isb.0
#
# FUNCTIONS:  This file describes the clock configuration for an 112-way
#             with seven (7) Node Switch Boards (NSBs) and four (4)
#             Intermediate Switch Boards (ISBs).
#             It is used during clock source selection process
#             ("Eclock" command).  It should not be changed unless the
#             switch-to-switch cabling differs from the prescribed pattern.
# FORMAT:  See below.
# NOTES:
#       Switch Numbers: This is the switch number of the target switch
board.
#                     Node Switch Boards (NSBs) numbers are less than 1000.
#                     Intermediate Switch Boards (ISBs) numbers are greater
#                       than 1000.
#
#       Multiplexor values:
#               0 - Use the internal oscillator (make this switch board the
#                   master frame).
#               1 - Use input 1 (Clock input from Jack J3 -- Both Switches)
#                2 - Use input 2 (Clock input from Jack J5 -- High Perf.
Switch)
#                               (Clock input from Jack J4 -- SP Switch)
#                3 - Use input 3 (Clock input from Jack J7 -- High Perf.
Switch)
#                               (Clock input from Jack J5 -- SP Switch)
#                4 - Use input from Jack J4  (SP Switch)
#                5 - Use input from Jack J5  (SP Switch)
#                6 - Use input from Jack J6  (SP Switch)
#                7 - Use input from Jack J7  (SP Switch w/ ISBs)
#                8 - Use input from Jack J8  (SP Switch w/ ISBs)
#                9 - Use input from Jack J9  (SP Switch w/ ISBs)
#               10 - Use input from Jack J10 (SP Switch w/ ISBs)
#               27 - Use input from Jack J27 (SP Switch)
#               28 - Use input from Jack J28 (SP Switch)
#               29 - Use input from Jack J29 (SP Switch)
#               30 - Use input from Jack J30 (SP Switch)
#               31 - Use input from Jack J31 (SP Switch w/ ISBs)
#               32 - Use input from Jack J32 (SP Switch w/ ISBs)
#               33 - Use input from Jack J33 (SP Switch w/ ISBs)
#               34 - Use input from Jack J34 (SP Switch w/ ISBs)
#
#       Clock source switch numbers will equal zero (0) when the associated
#       clock multiplexor (mux) value is set to internal (0).
#       Jack numbers will equal xx when the 'receiving' switch is using an
#       internal clock, in which case 'receiving' and source jack numbers
#       are meaningless.
#
#  NOTE: Programs use the Switch number, Clock multiplexor (mux) value,
#        and the Clock source switch number. Those values must be the first,
#        second, and fourth values on the non-comment lines in this file.
#        The values are separated by spaces.
#        Given the Clock Multiplexor (mux) value and the Type of Switch,
#        you can determine the Clock receiver jack number.  If the mux value
#        is 1,then the 'Clock receiver jack number' is J3 for both switches.
#        If the mux value is 2, then the 'Clock receiver jack number' is J5
#        for the High Performance Switch and J4 for the SP Switch.
#        If the mux value is 3, then the 'Clock receiver jack number' is J7
#        for the High Performance Switch and J5 for the SP Switch.
#
#  Switch number
#  |  Clock multiplexor (mux) value
#  |  |    Clock receiver jack number (High Performance Switch) / (SP
Switch)
#  |  |    |     Clock source switch number
#  |  |    |     |    Clock source jack number (High Performance Switch)
#  |  |    |     |    |   Clock source jack number (SP Switch)
#  |  |    |     |    |   |
   1  1  J3/J3   1001 J3  J3
   2  1  J3/J3   1001 J5  J4
   3  1  J3/J3   1001 J7  J5
   4  1  J3/J3   1001 J9  J6
   5  1  J3/J3   1001 J4  J34
   6  1  J3/J3   1001 J6  J33
   7  1  J3/J3   1001 J8  J32
1001  0  xx/xx      0 xx  xx
1002  1  J3/J3      1 J5  J4
1003  1  J3/J3      1 J7  J5
```

```
1004  2  J5/J4      2 J9  J6
#
# The following is the alternate clock source selections
#
#  Switch number
#  |  Alternate clock multiplexor (mux) value
#  |  |    Alternate Clock receiver jack number (High Perf. Switch)/(SP
Switch)
#  |  |    |     Alternate Clock source switch number
#  |  |    |     |    Alternate Clock source jack number (High Perf. Switch)
#  |  |    |     |    |   Alternate Clock source jack number (SP Switch)
#  |  |    |     |    |   |
  alternate 1
   1  2  J5/J4   1002 J3  J3
   2  2  J5/J4   1002 J5  J4
   3  2  J5/J4   1002 J7  J5
   4  2  J5/J4   1002 J9  J6
   5  2  J5/J4   1002 J4  J34
   6  2  J5/J4   1002 J6  J33
   7  2  J5/J4   1002 J8  J32
1001  2  J5/J4      2 J3  J3
1002  0  xx/xx      0 xx  xx
1003  2  J5/J4      2 J7  J5
1004  3  J7/J5      3 J9  J6
```

Appendix C. SP Switch Error Messages

In this appendix we list the return codes from the Worm code and the Route Table Generation code.

C.1 SP Switch Worm Return Codes

This table provides the Worm return codes.

Table 26. Worm Return Codes

Code Reference	Return Code	Description
_TBS_RC_ES_ON_FIFO	6	E/S packet on the receive FIFO
_TBS_RC_DEV_DISABLED	5	Device is disabled
_TBS_RC_DB_ALTERED	4	Device database was altered
_TBS_RC_NEW_BACKUP	3	New backup has been selected
_TBS_RC_PORT_WRAPPED	2	Port is wrapped
_TBS_RC_FIFO_EMPTY	1	Receive FIFO is empty
_TBS_RC_SUCCESS	0	No error recorded
_TBS_RC_RCV_TIMEOUT	-1	Received a time-out
_TBS_RC_TBOP_NO_OP	-2	Local adapter sender not enabled
_TBS_RC_TBIP_NO_OP	-3	Local adapter receiver not enabled
_TBS_RC_NO_RTG	-4	Unable to generate routes for the network
_TBS_RC_RS_SEND_FAILED	-5	Send packet from local node failed
_TBS_RC_SW_MISWIRE	-6	Switch miswire detected
_TBS_RC_NODE_MISWIRE	-7	Node miswire detected
_TBS_RC_FIFO_FULL	-8	Receive FIFO is full
_TBS_RC_BFS_FIFO_ERR	-9	Unable to initialize FIFOs
_TBS_RC_SWT_DEV_ON_FIFO	-10	Found TBIC device on the Sw device FIFO
_TBS_RC_NODE_VISITED	-11	Node has been visited
_TBS_RC_NO_NEW_BK	-12	No new backup has been selected
_TBS_RC_NOT_A_SWT	-13	The device is not a switch

Code Reference	Return Code	Description
_TBS_RC_NO_MATCH	-14	Unable to locate entry in device DB
_TBS_RC_BAD_SIGNATURE	-15	Bad signature in E/S packet
_TBS_RC_WRONG_DEV	-16	Wrong device responded packet
_TBS_RC_NOT_SW_ES_PKT	-17	E/S packet received was not from a switch device
_TBS_RC_WRONG_BK_RT	-18	Wrong backup (Secondary) route
_TBS_RC_UNEXP_PORT_STATE	-19	Unexpected port state
_TBS_RC_NOT_NODE_ES_PKT	-20	E/S packet received was not from a node device
_TBS_RC_NODE_FAULTY	-21	Node is faulty
_TBS_RC_BK_ID_NOT_TBIC	-22	The selected backup is not a TBIC device
_TBS_RC_NO_RESPONSE	-23	No response received
_TBS_RC_SEND_FAILED	-24	Send phase1 Sw init packet failed
_TBS_RC_NOT_TBIC	-25	Device ID not a TBIC
_TBS_RC_NOT_SWT	-26	Device ID not a Sw
_TBS_RC_TBIC_NOT_INIT	-27	The TBIC was not initialized
_TBS_RC_UNEXP_ES_PKT	-28	Unexpected E/S packet received
_TBS_RC_FENCE_FAILED	-29	The fence operation failed
_TBS_RC_ECR_BIT_ON	-30	Error Capture Register has a bit on
_TBS_RC_RCV_FIFO_NOT_EMPTY	-31	Receive FIFO is not empty
_TBS_RC_UNFENCE_FAILED	-32	The Unfence operation failed
_TBS_RC_ALREADY_FENCED	-33	The node was already fenced
_TBS_RC_NOT_FENCED	-34	The node is not fenced
_TBS_RC_INVALID_ROUTE_BIT	-35	Invalid route
_TBS_RC_RESIGN_PRIMARYSHIP	-36	Give up Primary node operations on this node
_TBS_RC_ES_LIMIT_HIT	-37	Number of errors discovered exceeded threshold
_TBS_RC_LINK_LIMIT_HIT	-38	Number of disabled Links exceeded threshold

Code Reference	Return Code	Description
_TBS_RC_NODE_TIMEOUT_HIT	-39	Node timed out on Scan
_TBS_RC_SWT_TIMEOUT_HIT	-40	Number of switch time-outs exceeded threshold
_TBS_RC_PHASE2_LIMIT_HIT	-41	Worm Phase 2 error threshold reached
_TBS_RC_NOT_PH1_PKT	-42	Packet received was not generated during Phase1 of the worm
_TBS_RC_TBIC_RW_FAILED	-43	Read/Write operation to TB3 adapter failed
_TBS_RC_TOD_ERROR	-44	Time Of Day (TOD) error detected
_TBS_RC_INTERMED_LINK_FAILURE	-45	Intermediate Link Failure
_TBS_RC_DUPLICATE_MISSING	-46	Duplicate response was not detected
_TBS_RC_PH2_FATAL	-47	Phase 2 of the worm failed
_TBS_RC_NOT_PH2_PKT	-48	Packet received was not generated during Phase2 of the worm
_TBS_RC_TOD_NOT_VALID	-49	TOD synchronization failed
_TBS_RC_LINK_DISABLED	-50	Link has been disabled

C.2 Return Codes from Route Table Generation

This table provides the RTG return codes.

Table 27. RTG Return Codes

Code Reference	Return Code	Description
ERR_TOPOLOGY	101	Problems with the input topology file
ERR_NOT_SWITCH	102	The device specified is not a switch
ERR_NOT_PROC	103	The device specified is not a processor
ERR_ROUTING_RULE	104	Error in Routing Rules Table
ERR_CALLOC_BFS_FIFO	107	Error allocating memory for Breadth First Search FIFO
ERR_ILLEGAL_PROC_ID	112	The processor ID is illegal

Code Reference	Return Code	Description
ERR_CALLOC_ROUTE_ARRAY	113	Error allocating memory for route array
ERR_CALLOC_DEVLIST_ARRAY	114	Error allocating memory for Device List
ERR_CALLOC_CSS_CREATE	115	Error allocating memory for create
ERR_SPECIFY_SOURCE_PROC	116	The source processor specified in invalid
ERR_BAD_ROUTE	117	Bad route generated
ERR_UNDEFINED_ALGORITHM	118	The routing algorithm specified does not exist
ERR_BF_SEARCH	119	Breadth First Search failed
ERR_TOP_FILE_FORMAT	120	Format error in Topology file
ERR_DEVICE_NOT_REACHABLE	122	Cannot generate routes for the device
ERR_NODELIST_EMPTY	123	No nodes in the node list
ERR_SWITCHLIST_EMPTY	124	No switches in the switch list
ERR_DISABLED_CHIP_CNT	125	Too many switch chips disabled

Appendix D. Special Notices

This publication will help both RS/6000 SP specialists and general users who want in-depth knowledge about the SP Switch. The information in this publication is not intended as the specification of any programming interfaces that are provided by the SP Switch Communication Subsystem (CSS). See the PUBLICATIONS section of the IBM Programming Announcement for Parallel System Support Programs (PSSP) for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have

ADSTAR	AIX
AS/400	AT
CT	IBM
LoadLeveler	NetView
PowerPC 601	PowerPC 603
PowerPC 603e	RS/6000
SP	SP1
System/390	

Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of some of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 263.

- *RS/6000 SP Monitoring: Keeping It Alive*, SG24-4873
- *Inside the RS/6000 SP*, SG24-5145
- *PSSP 2.4 Technical Presentation*, SG24-5173
- *PSSP 3.1 Announcement*, SG24-5332
- *RS/6000 SP Problem Determination Guide*, SG24-4778
- *GPFS: A Parallel File System*, SG24-5165

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

E.3 Other Publications

These publications are also relevant as further information sources:

• *SP: Planning, Volume 1, Hardware and Physical Environment*, GA22-7280
• *SP: Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281
• *PSSP: Installation and Migration Guide*, GA22-7347
• *PSSP: Command and Technical Reference*, SA22-7351
• *PSSP: Administration Guide*, SA22-7348
• *PSSP: Diagnosis Guide*, GA22-7350
• *PSSP: Messages Reference*, GA22-7352
• *PSSP: Managing Shared Disks*, SA22-7349
• *SP Switch Router Adapter Guide*, GA22-7310
• *RSCT: Event Management Programming Guide and Reference*, SA22-7354
• *IBM AIX Versions 3.2 and 4 Performance Tuning Guide*, SC23-2365
• *AIX System Management Guide: Communications and Networks*, GC23-2487

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** `http://www.redbooks.ibm.com/`

  Search for, view, download or order hardcopy/CD-ROM redbooks from the redbooks web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders via e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

This information was current at the time of publication, but is continually subject to change. The latest information for customer may be found at `http://www.redbooks.ibm.com/` and for IBM employees at `http://w3.itso.ibm.com/`.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at `http://w3.itso.ibm.com/` and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may also view redbook. residency, and workshop announcements at `http://inews.ibm.com/`.

IBM Redbook Fax Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
APA	All Points Addressable
CSS	Communication Subsystem
DMA	Direct Memory Access
EDC	Error Detection Code
IBM	International Business Machines Corporation
ISB	Intermediate Switch Board
ISC	Intermediate Board Switch Chip[
ITSO	International Technical Support Organization
LAPI	Low Level Application Interface
LSC	Link Switch Chip
MPI	Message Passing Interface
MPL	Message Passing Library
NSB	Node Switch Board
NSC	Node Switch Chip
PE	Parallel Environment for AIX
PLL	Phase Locked Loop
STI	Self-Timed Interface
TBIC	Trail Blazer Interface Chip
TOD	Time-Of-Day

Index

ITSO Redbook Evaluation

Understanding and Using the SP Switch
SG24-5161-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ **Customer** _ **Business Partner** _ **Solution Developer** _ **IBM employee**
_ **None of the above**

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction __________

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

____

____

____

____

What other redbooks would you like to see published?

____

____

____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

____

____

____

____

SG24-5161-00
Printed in the U.S.A.

Understanding and Using the SP Switch

SG24-5161-00