IRIX® GSN™ Administrator's Guide

CONTRIBUTORS

Written by Carlin Otto
Illustrated by Carlin Otto
Production by Amy Swenson
Engineering contributions by Jeffrey Chung, Jim Pinkerton, and Jean-Michel Pittet

IRIX® GSN™ Administrator's Guide
Document Number 007-3719-001

Contents

List of Figures

List of Tables

About This Guide

This document, the *IRIX GSN Administrator's Guide*, provides instructions for configuring, verifying, monitoring, and troubleshooting an SGI Gigabyte System Network (GSN) connection. This guide does not provide information about system administration of a GSN fabric. This guide documents IRIX GSN version 1.0 operating over IRIX 6.5.5 or later.

Audience

The audience for this document is network system administrators who are already familiar with general IRIX networking and system administration. This document assumes that the reader already knows the information documented in the online, IRIS InSight-viewable guide *IRIX Admin: Networking and Mail*, which is shipped with each copy of IRIX.

Where to Find More Information

Figure i illustrates sources of additional information for the IRIX GSN product and IRIX networking. Subsequent sections describe each of the illustrated sources.
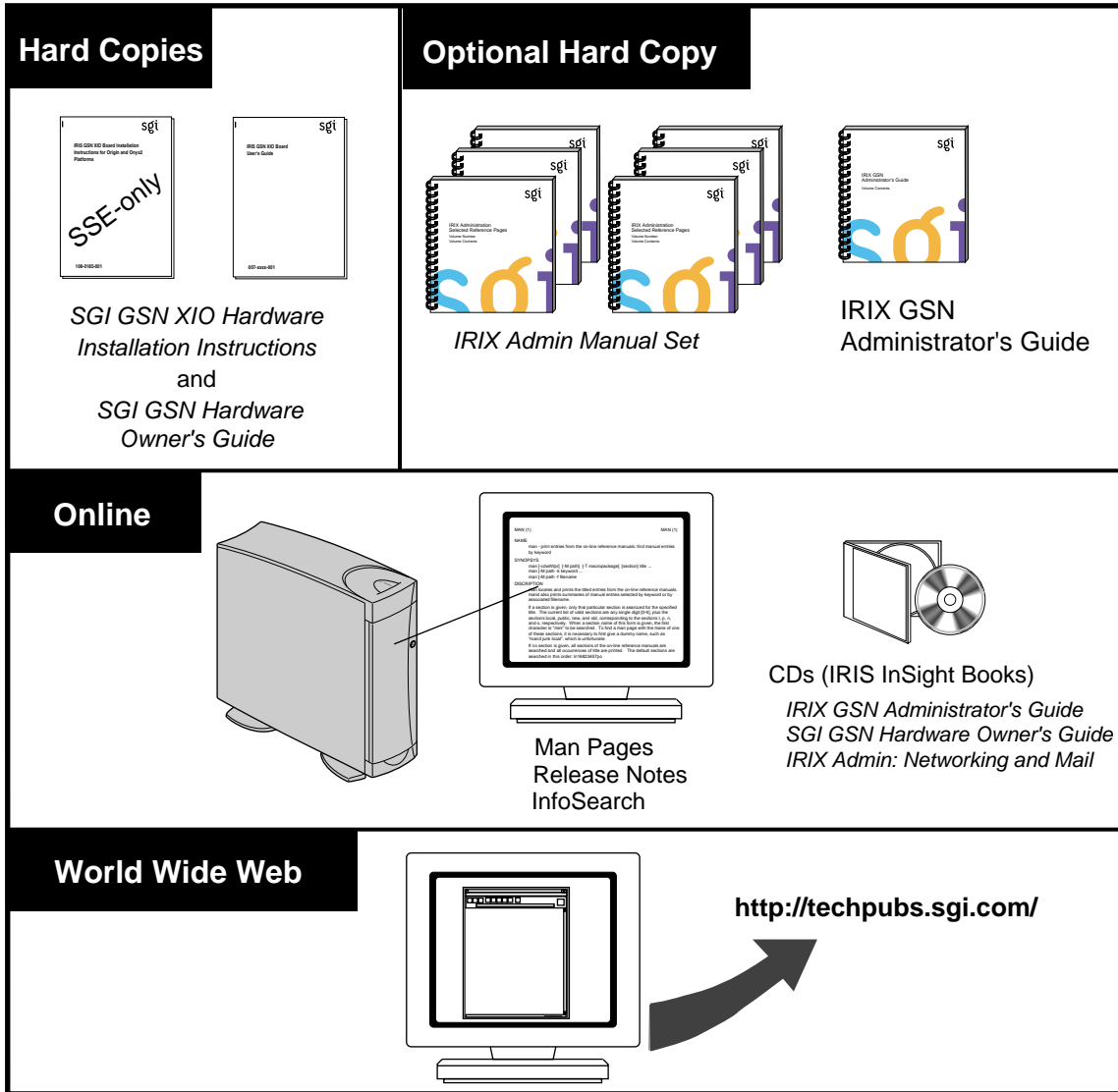
Figure i   Sources of Additional Information

Installation Instructions

The *SGI GSN XIO Hardware Installation Instructions for SGI Origin Family, SGI-2000-series, and Silicon Graphics Onyx2 Platforms* provides instructions for System Service Engineers (SSEs) who have been trained by SGI. This document is for the SSE (only).

Hardware Owner's Guide

The *SGI GSN Hardware Owner's Guide* describes the customer interface to the SGI GSN hardware: the LED behavior, the connectors on the panel plate, and the cable specifications. A printed copy of this guide is shipped with the SGI GSN hardware /IRIX GSN software product.

Administrator's Guide

This document, the *IRIX GSN Administrator's Guide*, describes the IRIX GSN software. Refer to the *IRIX GSN Administrator's Guide* to set up, configure, verify, monitor, and troubleshoot the IRIX GSN connection, including the network interface (for IP and ST), and suspected hardware problems. IRIX GSN is designed so that you can maintain most of the components of the system without the help of a trained technician. Hardware installation/removal is the only part of the IRIX GSN product that requires an SSE.

The administrator's guide can be viewed by the following methods:

- Use IRIS InSight. The digital format for this document (that is, the installable image) is included with the IRIX GSN software.

- Use a Web browser to view the HTML version at http://techpubs.sgi.com.

- Order a printed manual from http://techpubs.sgi.com or from the local SGI sales representative.

- Use a Web browser to download PostScript or PDF files from http://techpubs.sgi.com.

IRIX Administration Documents

The *Networking and Mail* volume of the *IRIX Admin* manual set is recommended as a reference for system administration tasks of logical network interfaces (for example, IP).

This document can be viewed by the following methods:

- Use IRIS InSight. The image is included with IRIX, either preinstalled on the system disk or available on CD-ROM. You can display this book from a graphics workstation using the IRIS InSight viewer.

- Use a Web browser to view the HTML version at the Silicon Graphics Technical Publications Library on the World Wide Web: http://techpubs.sgi.com.

- Order a printed copy from the local SGI sales representative.

- Use a Web browser to download PostScript or PDF files from http://techpubs.sgi.com/library.

Online Man Pages

The IRIX GSN software includes a set of IRIX reference (manual) pages, formatted in the standard UNIX "man page" style. These are installed on the main system disk along with the IRIX GSN software, and are displayed using the *man* command. For example, to display the man page for the *gsncntl* command, enter the following command at a shell prompt:

```
man gsncntl
```

IRIX GSN includes man pages for the following items: gsn(7m), gsncntl(1m), gsntest(1m), and gsnarp(1m).

Citations in the documentation to these man pages include the name of the command and the section number in which the command is found. For example, "gsncntl(1)" refers to the *gsncntl* command and indicates that it is found in section 1 of the IRIX reference.

For additional information about displaying man pages using the *man* command, see man(1).

In addition, the *apropos* command locates man pages based on keywords. For example, to display a list of man pages that contain information about GSN, enter the following command at a shell prompt:

apropos gsn

For information about setting up and using *apropos*, see apropos(1) and makewhatis(1M).

World Wide Web-Accessible Documentation

SGI maintains a World Wide Web (WWW) page from which you can retrieve manuals in a variety of formats. For example, you can retrieve the latest versions of many of the company's documents, or you can order printed (paper-copy) versions of online documents.

To view or retrieve the latest version of a document, use your Web browser to open the following URL:

http://techpubs.sgi.com/library/

To locate the latest versions of IRIS GSN documents (including this one), make the following selections:

1. In the "Keyword search" field, enter **gsn**

2. In the "Search In" area, select "Title only".

3. Click the "Search" button.

4. From the list of documents, click on the document that you want to view, download and print, or purchase in bound printed format.

To order a printed (paper-copy) version of a document, use your Web browser to open the following URL:

http://techpubs.engr.sgi.com/library/tpl/cgi-bin/order.cgi

Release Notes

You can view the release notes for a variety of SGI products and software subsystems using one of two utilities:

relnotes   Text-based viewer for online release notes

grelnotes   Graphical viewer for online release notes

To see a list of available Release Notes, type the following at a shell prompt:

```
relnotes
```

For more information, see the relnotes(1) and grelnotes(1) man pages.

InfoSearch

InfoSearch is a unified system for retrieving and viewing online information, providing you with quick and easy access to online information available on SGI computers. With InfoSearch, you can search or browse through release notes, man pages, application help cards, online books, and other forms of online information.

There are two ways to use InfoSearch:

- The *infosearch* utility, which runs on IRIX workstations.

- A World Wide Web interface, *infosrch.cgi*, that you can access through any Web browser.

An SGI system can be configured as an InfoSearch server so that other workstations on your network can use it to retrieve information. However, you must have a graphical workstation in order to use InfoSearch, either with the *infosearch* command or via a Web browser.

For basic information about getting started with InfoSearch, see the infosearch(1), sgindexAdmin(1), booksAdmin(1), and infosrch.cgi(1) man pages.

Obtaining Updated or Printed Versions of This Document

To order printed (paper-copy) manuals or to retrieve a newer version of a manual, use the SGI Web page, as described in "World Wide Web-Accessible Documentation" on page xv, or contact your sales representative.

SGI Product Support

Silicon Graphics, Inc. (SGI), provides a comprehensive product support and maintenance program for its products. If you are in North America and would like support for your SGI-supported products, contact the Technical Assistance Center at 1-800-800-4SGI. If you are outside North America, contact the SGI subsidiary or authorized distributor in your country.

Conventions Used in This Guide

Throughout this guide, the following stylistic conventions are used:

Italicized text	represents commands, variables, document titles, and filenames.
Initial capitalization	identifies proper names for protocol entities. The initial capital letter distinguishes the word from generic use of the term. For example, GSN Message, refers to the item described and labeled in the ANSI standard, as opposed to message, which can refer to any communication that has a clear beginning and an ending.
`Courier font`	represents text that appears on a terminal.
Glossary term	indicates that the term is explained in the Glossary.

Overview of IRIX GSN

This chapter provides an overview of IRIX GSN version 1.0.

Gigabyte System Network (GSN) is a full-duplex, error-free, flow-controlled communications protocol that simultaneously provides a full gigabyte (8 gigabits) of data transfer in each direction. Table 1-1 compares theoretical GSN data rates to the theoretical rates of other communications protocols.

Table 1-1   GSN Compared to Other Communication Technologies

Protocol	BAUD Rate	Peak User Payload Rate[a]	Sustained User Payload Rate Using TCP/IP[b]
GSN (copper)	500 MBaud on 20 lines	6.4 gigabits/sec.[c]	6.365 gigabits/sec.
Gigabit Ethernet	1256 MBaud on one line	1.0 gigabit/sec.	0.924 gigabits/sec.
HIPPI-800-Parallel HIPPI-800-Serial	25 MBaud on 32 lines 1200 MBaud on one line	0.8 gigabits/sec.	0.789 gigabits/sec.
ATM OC12c over SONET	622 MBaud on line	0.622 gigabits/sec.	0.541 gigabits/sec.

a. Peak rate is the rate required for hardware's direct-memory-access (DMA) when hardware has small input queue.

b. Assumes TCP/IP, v4 packets with 49152 bytes of user data.

c. All rates are decimal not digital (that is, they are base-ten, not base-two); for example, giga is 1,000,000,000.

SGI GSN Products

Components of Products

The GSN products offered by SGI consist of multiple components that implement the following protocols:

- SGI GSN hardware: copper-based Gigabyte System Network (GSN, also known as HIPPI-6400 or SuperHIPPI) hardware for use in XIO slots.

- IP-over-GSN driver (*gsn#*) included in IRIX GSN. This component is the interface between the GSN hardware and the Internet Protocol (IP) with its associated transport-layer protocols: TCP, UDP, ICMP, and so on. Requires IRIX 6.5.5 or later.

- ST-over-GSN driver (*gsn#*) included in IRIX GSN. This component interfaces the GSN hardware to the Scheduled Transfer Protocol (ST). Requires IRIX 6.5.6 or later.

- Address resolution protocol server (daemon) and client functionality shipped with IRIX GSN. The dynamic HARP component handles both HIPPI-6400 and HIPPI-800 clients. IRIX HARP also supports static table lookup for handling HIPPI systems that do not support HARP.

- IRIX sockets-based application programming interface (API) to the IP network stack (driver) for use by customers who want to develop or port applications to send/receive data through the IP-over-GSN subsystem. Available with IRIX 6.5.5 and subsequent versions.

- IRIX sockets-based application programming interface (API) to the ST network stack (driver) for use by customers who want to develop or port applications to send/receive data through the ST-over-GSN subsystem. Available with IRIX 6.5.6 and subsequent versions.

GSN Within IRIX Network Stacks

The SGI GSN hardware and IRIX GSN software support the following network stacks (illustrated in Figure 1-1):

- IP-over-GSN: applications that use the standard IRIX interface (BSD sockets) to send/receive data using the IP suite of protocols.

- ST-over-GSN: applications that use the IRIX GSN product's Scheduled Transfer (ST) programmatic interface to send/receive data over GSN. Applications that use this interface include the IRIX utilities shipped with the IRIX GSN product and customer-developed ST applications.

- ARP for HIPPI/GSN (HARP): automatically resolves physical-layer HIPPI-6400 ULA addresses to and from network-layer addresses (IP and ST).

   Note: Each *gsn#* network interface services two main protocols: ST and IP. The *INET address* that the customer assigns to an instance of *gsn#* is shared by the ST-over-GSN and IP-over-GSN stacks. Some of the upper-layer address processing (for example, routing) that is performed on the address applies to both IP and ST traffic.

KEY: **bold** type identifies GSN modules
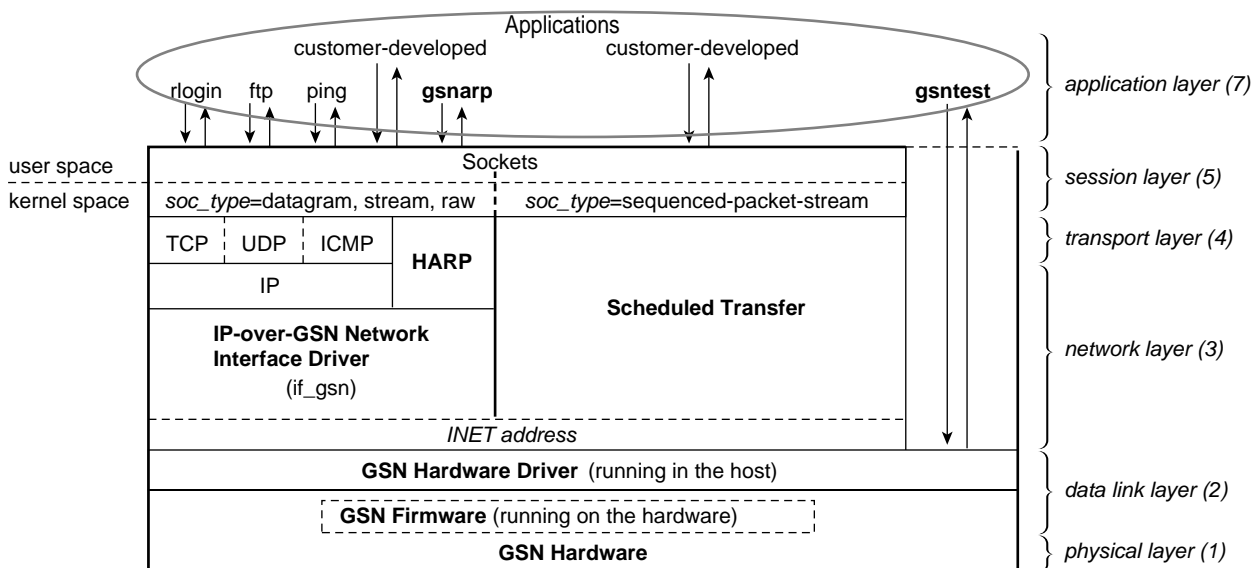
Figure 1-1   IRIX GSN Modules Within OSI-style Network Protocol Stack

Standards Compliance

IRIX GSN complies with the following industry standards:

- GSN (also called HIPPI-6400 or SuperHIPPI)

    - *Information Technology - High-performance Parallel Interface - 6400 Mbit/s Physical Layer (HIPPI-6400-PH)*, ISO/IEC 11518-10, NCITS (ANSI) standard.

    - *Information Technology - High-performance Parallel Interface - 6400 Mbit/s Switch Control (HIPPI-6400-SC)*, T11.1, Project 1231-D, Rev. 2.5, August 1998, working draft for NCITS (ANSI). Only those functions that apply to GSN endpoints.

- ST-over-GSN

    - *Information Technology - Scheduled Transfer Protocol (ST)*, T11.1, Project 1245-D, Rev. 2.6, December 1998, working draft for NCITS (ANSI).

- IP-over-GSN

    - RFC 2067, IP over HIPPI

    - Other standard internet protocols provided with IRIX (IP versions 4 and 6, NFS versions 2 and 3, TCP, UDP, ICMP, and so on.)

- IRIX HARP

    - RFC Internet Draft, *IP and ARP over HIPPI-6400*, December 1998

    - RFC Internet Draft, *ARP and IP Broadcast over HIPPI-800*, December 1998

    To obtain copies of the GSN and ST documents, see the Web site http://www.hippi.org, or contact the American National Standards Institute (ANSI) at 11 West 42nd Street, New York, New York 10036, telephone: 212-642-4900. For RFCs, see the Web site http://info.internet.isi.edu/in-notes/rfc.

GSN Product Names

The following strings are used to identify the GSN product:

- Name for software image: `gsn`
  (for example, **inst gsn** and **versions gsn**)

- Name for each hardware device/adapter: `gsn#` or `xio_gsn`
  (for example, **gsncntl gsn#** and **find /hw/module -name xio_gsn**)

- Hardware inventory name for each adapter:
  GSN 1-Port adapter and GSN 2-Port adapter

- Name for each logical IP or STnetwork interface: `gsn#`
  (for example, **`netstat -i gsn3`** and **`ifconfig gsn0 up`**)

Compatibility Issues

IRIX GSN 1.0 requires IRIX 6.5.5. Use this command to verify the version of IRIX that is currently running on the system. The version number (displayed by the -n option) must be equal to or greater than the version shown in this example:

```
% versions -n eoe
I eoe  1275719131    IRIX Execution Environment, 6.5
```

The SGI GSN hardware requires the system's HUB ASICs to be version 5. Use this command to verify the version of the HUB on each Node board:

```
% hinv -v | grep HUB
HUB in Module #/Slot 1: Revision 5 Speed 97.50 Mhz (enabled)
HUB in Module #/Slot 2: Revision 5 Speed 97.50 Mhz (enabled)
HUB in Module #/Slot 3: Revision 5 Speed 97.50 Mhz (enabled)
HUB in Module #/Slot 4: Revision 5 Speed 97.50 Mhz (enabled)
```

Overview of Protocols

These sections provide an overview of the protocols that make up and interoperate with IRIX GSN. Figure 1-1 illustrates the GSN protocol stacks.

- "What Is GSN?"

- "What is ST?"

- "Address Resolution for GSN"

- "How Address Resolution Works for ST-over-GSN"

5

What Is GSN?

Gigabyte System Network (GSN) is a set of ANSI standards (listed in "Standards Compliance" on page 4) that defines physical and data link layers for a very high-speed communications protocol. The GSN protocol is also known by two other names: HIPPI-6400 and SuperHIPPI. Throughout this document, the term GSN is used for this entire set of protocols, except when referring to an item from a specific ANSI standard, in which case the term from the ANSI document's title is used (for example, HIPPI-6400-PH micropacket).

GSN Terminology

The following terms have specific meanings when used within the context of GSN:

Physical link
> One section of HIPPI-6400-PH cable (copper or fiber-optic) that connects two HIPPI-6400-PH elements. Each element can be either a switch or an endpoint. Each physical link is a full-duplex link composed of two simplex links; each simplex link carries data in only one direction; the two streams of data in the full-duplex link flow in opposite directions. The path (virtual connection) between an original point of transmission (the originating source) and a final point for reception (the final destination) can involve numerous physical links.

Element
> Any component of a HIPPI-6400 fabric or system that is able to receive, process, and send HIPPI-6400 Admin micropackets in a manner that conforms with the HIPPI-6400 standard. Each HIPPI-6400 element contains both a source and a destination. For example, the SuMAC chip in an SGI GSN product is a GSN element.

Source
> The transmitting element located at one end of a physical link. An upper-layer entity (host, network-layer interface, or program) that uses the GSN subsystem is sometimes loosely referred to as the source. However, it is more correct to call these software entities upper-layer protocols (that is, source ULPs). An "originating source" refers to the element that first transmitted a micropacket; an element that is retransmitting the micropacket (for example, a switch) is simply a source.

Destination

> The receiving element located at the other end of a physical link. An upper-layer entity (host, network-layer interface, or program) that receives communications through the GSN subsystem is sometimes loosely referred to as the destination. However, it is more correct to call these software entities upper-layer protocols (that is, destination ULPs). A "final destination" refers to the element that is the ultimate receiver for a micropacket; an element that receives, then retransmits a micropacket (for example, a switch) is simply a destination.

Endpoint

> A final destination or an originating source of GSN traffic. An endpoint may have only one GSN port. A single system may have many endpoints (for example, an Origin module with two SGI GSN products has two endpoints).

Switch

> A node that is located along the route between two endpoints. GSN traffic passes through the switch on its way to a destination endpoint. A switch must have at least two, and usually has more, GSN ports.

Fabric

> All the HIPPI nodes (switches, endpoint devices, extenders) that are physically interconnected and communicate using the same physical-layer protocol. For example, a GSN fabric is a separate entity from a HIPPI-800 fabric.
>
> One GSN fabric can be logically divided into multiple upper-layer address spaces (that is, networks). For example, a single GSN fabric can support multiple IP networks. And, conversely, one logical network can include members from multiple HIPPI fabrics. For example, an IP network can include members from a GSN (HIPPI-6400) fabric as well as members from a HIPPI-800 fabric.

Hop count

> A number used in HIPPI-6400 Admin micropackets to specify the number of elements through which the micropacket should be forwarded. Each time a micropacket exits an element, the hop count is decremented by one. See "GSN Admin Micropackets" on page 18 for further details.

GSN Overview

The GSN protocol provides 6.4 gigabits of user data per second from source to destination (in each direction) over either copper-based or fiber-optics-based physical media.[1] The protocol is point-to-point, full-duplex, and flow-controlled. It uses small fixed-size micropackets (illustrated in Figure 1-4 and Figure 1-6) and up to four interleaved logical datastreams (channels) per point-to-point connection.

GSN Physical Layer

Each physical link is composed of two simplex links that connect two HIPPI-6400 elements; data flows in only one direction on each simplex link. Both simplex links are required for a connection because control information for each datastream travels in the reverse direction (that is, along the other simplex link of the connection). This design provides a full-duplex connection between two endpoints.

The GSN data rate is stated as 6.4 gigabits of user data per second on each simplex link; however, each link physically carries a total of 8 gigabits (1 gigabyte) of data (user and control) every second. The following items describe the GSN bandwidth:

- At the physical layer (that is, on the wire), GSN uses a dual-edged 250-million-cycle -per-second clock, which results in 500 million transmission events per second. Said another way, GSN operates at 500 MBaud.

- For each baud, GSN transmits 16 bits of user data and 4 bits of control data that is encoded with 4b/5b. This means that 20% of the total bandwidth is overhead for the encoding, and, of the remaining bandwidth, 20% is overhead for the HIPPI-6400 protocol. This results in user bandwidth of 6.4 gigabits or 6400 megabits per second.

- The available bandwidth for user data is 6400 megabits/second, which is 6.4 gigabits or 0.8 gigabytes of per second in each direction.

---

[1] For SGI GSN release 1.0, only the copper-based medium is supported.

Table 1-2 summarizes the mathematical calculations:

Table 1-2   GSN Bandwidth Calculations

Item	Bandwidth	Calculation Details
Total physical signal carrying capacity	10 GBaud	20 simultaneous signals multiplied by 500 MBaud, which is 10 billion signals per second in each direction.
Bandwidth available for protocols	8.0 Gbits/s	Rate in row above, minus bandwidth used by 4b/5b encoding.
Bandwidth available for users (that is, layers above the HIPPI-6400 layer)	6.4 Gbits/s	Rate in row above, minus amount used by GSN control information. GSN control = 4 of the 20 bits (20% of 8 Gbits).

GSN Virtual Channels

Each simplex link can carry up to four logical datastreams (virtual channels). These virtual channels are allocated for control traffic, low latency traffic, and bulk traffic to avoid the latency/blocking issues that occur when only a single channel is attempting to handle both bulk and interactive traffic.

Each virtual channel is commonly implemented as a queue; micropackets are selected alternately from the active queues and placed onto the physical link in an interleaved fashion, as illustrated in Figure 1-2. Not all four channels need to be active on every connection. All the micropackets belonging to a single GSN Message always travel through the same channel, even when the message traverses switches along its way to the final destination. The restrictions for the data that can be carried on each channel are described in Table 1-3.

Table 1-3   Data Restrictions for Each GSN Virtual Channel

Virtual Channel	Description
0	Carries GSN Messages that do not exceed 68 micropackets of TYPE data (about 2176 bytes of upper-layer data). For ST-over-GSN traffic, ST data channel 0 maps to this GSN channel; all ST control operations (for example, Request_To_Send and Clear_To_Send) travel on this virtual channel.
1	Carries GSN Messages that do not exceed 4100 micropackets of TYPE data (about 128 kbytes of upper-layer data) and Admin micropackets in which the COMMAND field specifies a request or a command (that is, not a response). IP-over-GSN traffic is carried on this VC. For ST-over-GSN traffic, ST data channel 1 maps to this GSN channel.
2	Carries GSN Messages that do not exceed 4100 micropackets of TYPE data (about 128 kbytes of upper-layer data) and Admin micropackets in which the COMMAND field specifies a response. For ST-over-GSN traffic, ST data channel 2 maps to this GSN channel.
3	Carries GSN Messages that do not exceed 134,217,728 micropackets that are of TYPE data (about 4Gbytes of upper-layer data). This channel requires that the final destination endpoint agree to accept this Message via a flow-controlled protocol such as Scheduled Transfer. For ST-over-GSN traffic, ST data channel 3 maps to this GSN channel.
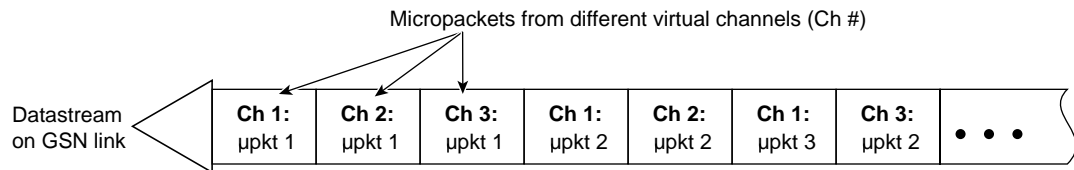
Figure 1-2   GSN Micropackets From Virtual Channels Interleaved In Datastream

GSN Micropacket

The micropacket is the basic protocol data unit for GSN. Each GSN micropacket is 32-bytes of data accompanied by 8 bytes (64 bits) of control information. The TYPE field within the control bits indicates the format and purpose of the micropacket's 32 bytes of data. The VC field determines which virtual channel carries the micropacket. Some of the control bits that accompany a 32-byte chunk of data refer to that chunk of data (for example, the VC and TYPE fields), and some bits refer to the datastream traveling in the opposite direction on the other physical link (for example, the credits in the CR field that allow the reader/receiver of the control bits to transmit more data for its own datastream). Figure 1-3 illustrates the control bits and Table 1-4 describes them. Table 1-5 summarizes the different TYPEs of GSN micropackets.
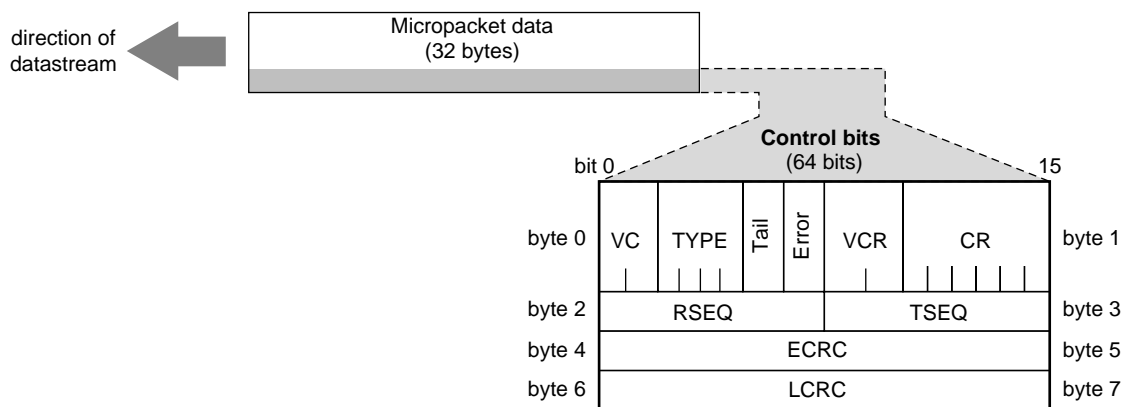
Figure 1-3   GSN Micropacket Control Bits

Table 1-4   GSN Micropacket Control Bits

Name of Field	Number of Bits in Field	Description	Applies to Data in Which Link
VC	2	Virtual channel selector for this micropacket (binary values): 00=channel_0; 01=channel_1; 10=channel_2; 11=channel_3	This one
TYPE	4	Type of micropacket: see Table 1-5	This one

Table 1-4 (continued)   GSN Micropacket Control Bits

Name of Field	Number of Bits in Field	Description	Applies to Data in Which Link
T	1	Tail: 0=more micropackets follow to complete this GSN Message; 1=this is the last micropacket for this Message.	This one
E	1	Error: 0=this GSN Message is OK so far; 1=an unrecoverable error was detected for this Message.	This one
VCR	2	Virtual channel for which the credits (in CR field) apply.	Other
CR	6	Credits: number of credits the source (that is, the receiver of these control bits) can add to the data transfer on the virtual channel indicated in the VCR field. (See "GSN Flow Control" for further explanation.)	Other
RSEQ	8	Reception sequence number: Acknowledgment for the highest-received sequence number (TSEQ) for data micropackets on the other link.	Other
TSEQ	8	Transmission sequence number: The sequence number associated with this micropacket.	This one
ECRC	16	End-to-end checksum. Checksum for all data bytes of the GSN Message, up to and including, the bytes in this micropacket. This checksum is verified by the final destination.	This one
LCRC	16	Link checksum. Checksum for the 32 bytes of data and the first 48 bits of control information in this micropacket. This checksum is verified by each GSN element at the end of a link.	This one

Most of the GSN micropacket TYPEs are related to control and management of the GSN link. Only three TYPEs of micropackets are passed to the upper layers: Admin, Header, and Data micropackets. The Admin micropacket (illustrated in Figure 1-4) is used by upper-layer GSN administrative programs to manage and configure a GSN fabric; hence, the Admin micropacket is defined by the Switch Control ANSI standard (HIPPI-6400-SC). The Header and Data micropackets are used to create GSN Messages (illustrated in Figure 1-6) that carry user-level data.

Table 1-5   Types of GSN Micropackets

Type (Name)	Type (Hexadecimal)	Description of the Micropacket	Supported by IRIX GSN Hardware?
Reset	2	Causes the receiving HIPPI-6400-PH device to reset the local link (that is, the physical link between this sender and the device at the other end of the physical link).	Y
Reset_Ack	3	Acknowledges that the Reset micropacket was received and that the HIPPI-6400-PH link reset was completed.	Y
Initialize	4	Causes the receiving HIPPI-6400 device to reinitialize.	Y
Initialize_Ack	5	Acknowledges that the Initialize micropacket was received and that the HIPPI-6400-PH initialization procedure was completed.	Y
Reserved	6	n.a.	n.a.
Null	7	Contains no data in the 32-byte data area; there may be valid information in the Control Bits. This type is transmitted only when there is nothing else to transmit; it keeps the physical link active/alive.	Y
Data	8	Contains data for a GSN (HIPPI-6400) Message (illustrated in Figure 1-6).	Y
Header	9	Contains the header information for a GSN (HIPPI-6400) Message (illustrated in Figure 1-6).	Y

Table 1-5 (continued)   Types of GSN Micropackets

Type (Name)	Type (Hexadecimal)	Description of the Micropacket	Supported by IRIX GSN Hardware?
Credit-only	A	Contains only valid credits (VCR and CR fields of Control Bits) that allow the transmitter to send more data. The micropacket contains no data in the 32-byte data area. This type is transmitted only when there are no Admin, Header, or Data micropackets awaiting transmission.	Y
Reserved	B-E	n.a.	n.a.
Admin	F	Used for administering GSN switches and endpoints. Format for Admin micropacket is defined by the HIPPI-6400-SC standard. A number of functions (commands) are supported, including: ping another GSN device, request ULA of a remote GSN device, and set up broadcast capability for a GSN fabric.	Y

One of the functions for the Admin micropacket is to allow each switch on a GSN fabric to discover the fabric's physical configuration and each endpoint to discover the universal LAN MAC address (ULA) that its switch has assigned to it. This functionality is not available on every GSN product; however, when it is implemented, this is how it works.

- For an endpoint, upon starting, it transmits an Admin micropacket that asks the device at the other end of the link to identify its function (for example, is it an endpoint or a switch). If the device is a switch, the endpoint asks for an assigned ULA; if the device is another endpoint, the local endpoint uses its locally assigned ULA (which might be stored in the hardware's PROM).

- For a switch, upon starting, it transmits Admin micropackets that ask for other devices' functions (for example, is it a switch or an endpoint). The switch sends one such request to each hop (successive hardware device) down each of its links until an endpoint is reached. Upon discovery of each endpoint or a switch, it uses Admin micropackets to exchange ULA information with that device. As it receives responses from these Admin requests, the switch constructs a map (spanning tree) of its fabric. Once this map has been constructed, a micropacket destined for a known endpoint (that is, any endpoint discovered within that fabric) can be delivered.

14

Note: This fabric discovery scheme does not solve the problem of how each endpoint comes to know the ULA for the other endpoints with which it wants to communicate. That problem can be solved by an upper-layer address resolution mechanism (for example, HARP or another network-layer address resolution mechanism). For details, see "Address Resolution for GSN."
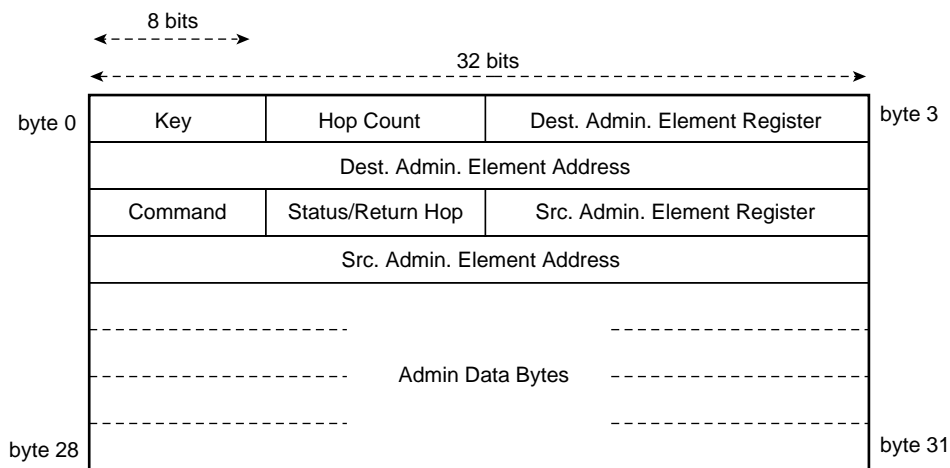
Commands include:
ping remote element (device)
discover remote element's function (is it a switch or an endpoint?)
clear (reset) remote element's HIPPI-6400-SC state information
request remote element's ULA
read remote element's administrative register
request a list of ULAs for all connected elements
set remote element's HIPPI-6400-SC address

Figure 1-4   GSN Admin Micropacket

GSN Flow Control

A GSN destination (receiving) endpoint controls the flow of micropackets by periodically releasing credits to the source.[1] Each credit represents memory at the destination for one GSN micropacket. Each credit gives the source permission to send one additional micropacket on a specific channel. The destination gives credits to the source in the control bits (CR and VCR bits) that accompany the destination's own micropackets. Note that the credits travel in the opposite direction from the data, as illustrated in Figure 1-5, and can accompany micropackets traveling on any of the GSN virtual channels for the connection.
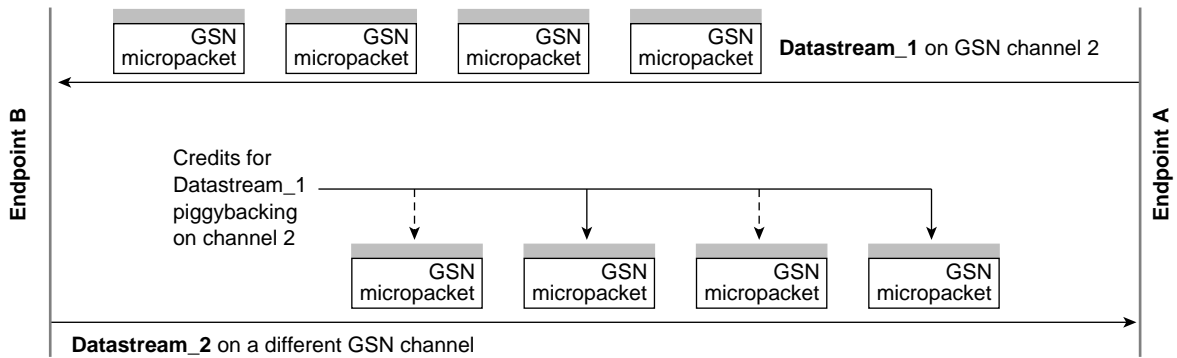
Figure 1-5   GSN Flow Control

---

[1] Flow control is a mechanism for preventing data loss that is caused by a source transmitting data faster than the destination can process it. Without flow control, the destination drops incoming data when it does not have memory available (free) in which to store the data.

GSN Message

The GSN Message is the basic data transfer unit between source and final destination endpoints. Each Message is composed of one initial Header micropacket followed by zero or more Data micropackets (illustrated in Figure 1-6). The micropackets of a Message are sequentially ordered and all travel over the same virtual channel using the same originating source (S_ULA value) and final destination (D_ULA value). The last micropacket in a Message has a bit set (the TAIL flag) to indicate that the Message is complete. Figure 1-6 illustrates a complete GSN Message.
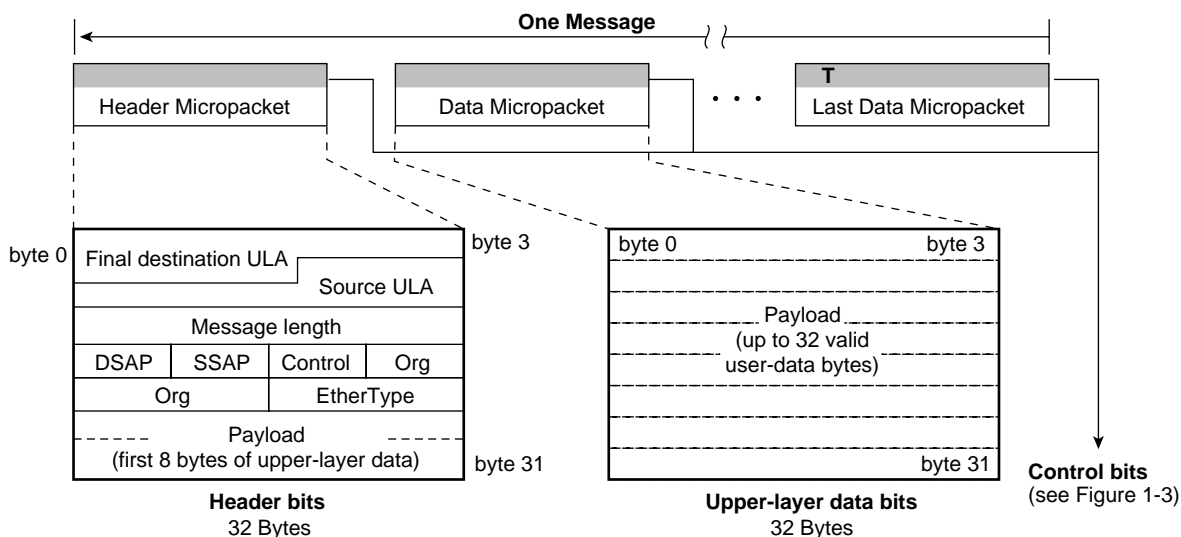
Figure 1-6   GSN Message Composed of Header and Data Micropackets

When the GSN Header micropacket is carrying an IP datagram (EtherType=2048), the 8 bytes of payload in the Header micropacket are the first 8 bytes of the IP header. (Note that the 8 bytes immediately preceding the Payload are an 802.2 SNAP header.) When the GSN Header micropacket is carrying an ST transfer (EtherType=8181), the payload bytes in the Header micropacket are the initial 8 bytes of the ST Header.

GSN Admin Micropackets

Every HIPPI element is capable of processing GSN (HIPPI-6400-SC) Admin micropackets. These micropackets configure elements, discover the fabric topology, and maintain the elements of a GSN fabric. The TYPE field of the control bits (illustrated in Figure 1-3) indicates that a micropacket is of the Admin type. Admin micropackets have the format illustrated in Figure 1-4.

Most HIPPI-6400 elements have two ports: one leading toward the fabric and the other leading toward the host/core. For example, a link end element (such as the SuMAC ASIC) has one port connected to a physical link/the fabric and the other port connected to additional GSN logic (which may be another local element) on an adapter board. Notice that a GSN system may contain more than one element; this fact is important in understanding the processing of Admin micropackets.

An Admin micropacket can enter an element through either port, as illustrated in Figure 1-7. Each Admin micropacket is either processed and responded to or forwarded to the next element through the element's other port, as illustrated in Figure 1-7. A response to an Admin micropacket always exits the element through the same port by which the original Admin micropacket arrived.

Admin arrives from "host" direction:
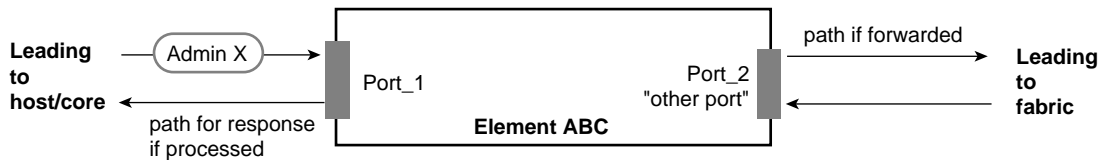
Admin arrives from "fabric" direction:

Figure 1-7   Dual-port HIPPI-6400-PH Elements

The *hop count* field in the Admin micropacket determines when the Admin packet is acted upon/processed. The count indicates the number of elements (hops) through which the Admin micropacket is propagated/forwarded before it is processed. As long as the hop count is greater than zero, the receiving element decrements the hop count by one and transmits the Admin micropacket out the element's other port (which leads to another element), as illustrated in Figure 1-8. When the count is zero, the receiving element processes the micropacket and responds, as illustrated in Figure 1-9. Figure 1-10 through Figure 1-12 show examples of various hop count values and the manner in which hop count determines which element acts on and responds to the micropacket.

Table 1-6 lists the administrative commands that are available with Admin micropackets.
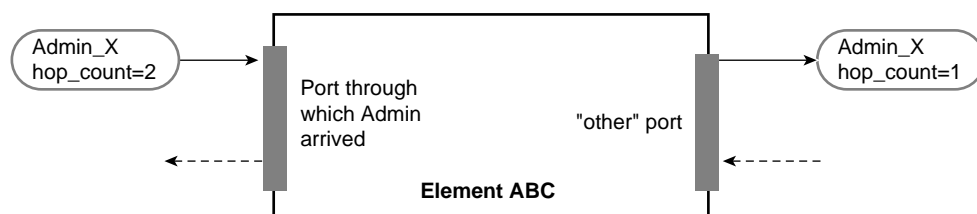
Figure 1-8   Hop Count >0 Indicates Forward Admin Micropacket
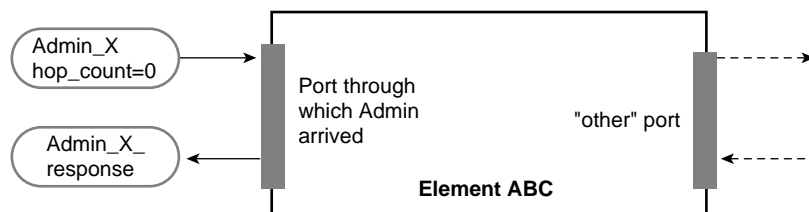
Figure 1-9   Hop Count =0 Indicates Process Admin Micropacket
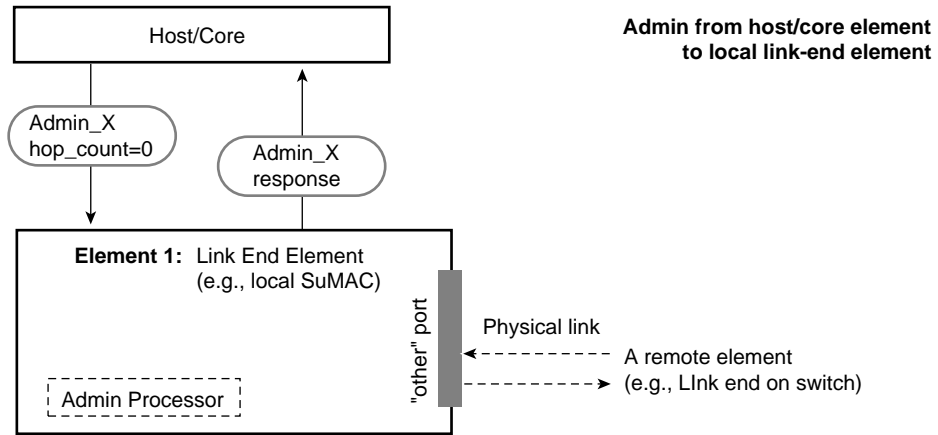
Figure 1-10   Hop Count Example: hop_count = 0
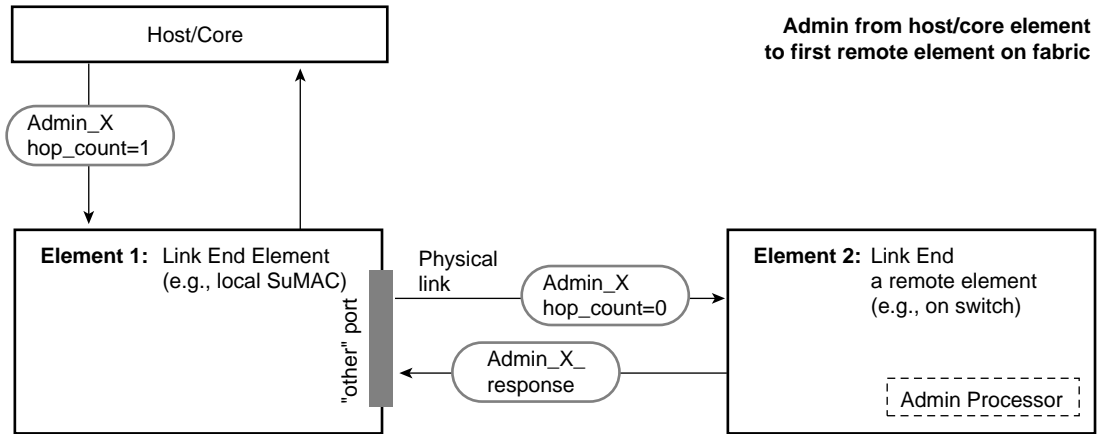
Figure 1-11   Hop Count Example: hop_count = 1
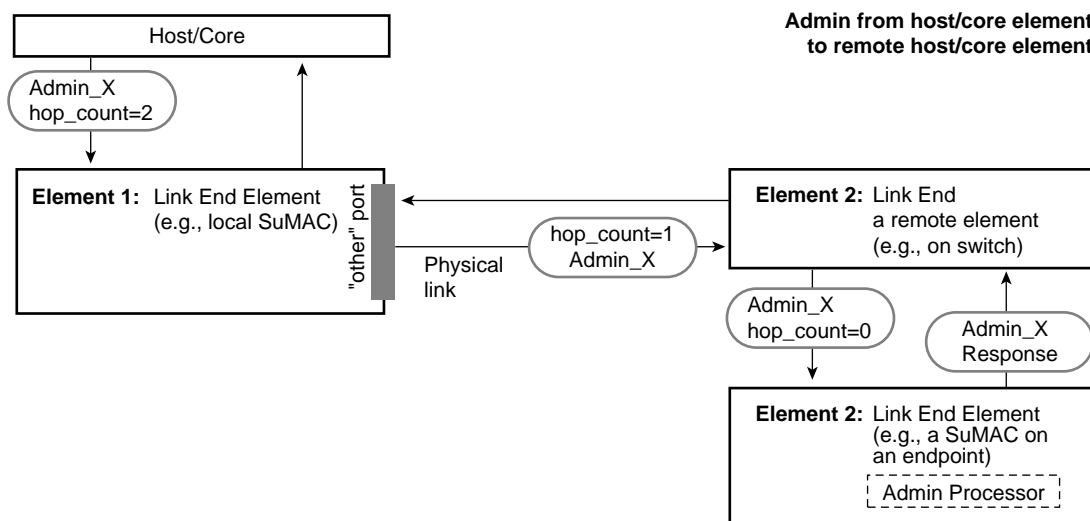
Figure 1-12   Hop Count Example: hop_count = 2

Table 1-6   GSN Admin Micropacket Commands

Admin Command	Description	Required (R) or Optional (O) for Switches and Endpoints
Ping	Are you there?	O
*_response	Yes I am here (and functioning).	R
Set_element_address	Here is your "element address".	O
*_response	Status (for example, I have started using the assigned address).	O
Reset	Initialize yourself	O
Exchange_element_ function	I am a <switch/endpoint> element. What are you?	R
*_response	I am a <switch/endpoint> element.	R

Table 1-6 (continued) GSN Admin Micropacket Commands

Admin Command	Description	Required (R) or Optional (O) for Switches and Endpoints
ULA_request	Assign me a ULA.	R
*_response	Here is your ULA.	R for switches
Read_register	Give me the data from this Admin register.	O
*_response	Here is the data you requested.	O
Write_register	Put this data into this Admin register.	O
*_response	Status (for example, the data has been written).	O
Invalid_command	I received an invalid/unrecognized/ unsupported Admin micropacket.	R
ULA_list_request	Give me a list of all the ULAs connected to you.	O
*_response	Here is the list.	R for switches
Port_remap	For all traffic containing the specified ULA, change the route (output port) to a specified (new) port ID.	O
*_response	Status.	R for switches
Port_map_request	Give me the port ID that I must use to contact the specified ULA.	O
*_response	Here is the port ID.	R for switches

What is ST?

Scheduled Transfer (ST) is an upper-layer protocol that can be implemented to operate over a number of physical-layer subsystems, including GSN, HIPPI-800, ATM, FDDI, and Ethernet. This section describes the main characteristics of the ST protocol. For the sake of introduction and ease of understanding, many of the less important functional details of ST are not covered in this description. Refer to the ANSI standard (listed in the section "Overview of Protocols") for complete details.

ST Overview

The most salient feature of ST is that it prepares both endpoints for the data movement before any data is transmitted. The first step in the preparation is to create a condition (state) called a virtual connection or VC (described in "ST Connection Setup Sequence"). The second step is a handshake that allocates memory for the data movement and exposes this memory to the other endpoint (described in "ST Data Movement Sequences Including Memory Allocation"). There are two kinds of the memory-allocation handshake: one provides memory that is used once (described in "Single-use Memory Data Movements"); the other provides memory that is used many times until released (described in "Persistent Memory Data Movements"). The two endpoints exchange ST control operations to accomplish these prearrangements. Only after these prearrangements are complete can the first data movement begin; the data movement is performed with ST data operations.

ST Terminology

The following terms have specific meanings within the context of ST:

operation   The ST protocol data unit. It is composed of a 40-byte header and variable-length data ranging from 0 bits to 4 gigabits (illustrated in Figure 1-13). Each ST operation is transmitted as one GSN Message, as illustrated in Figure 1-13.

sequence   A series of operations that occur in a specific order and accomplish an ST protocol task.

initiator   The ST endpoint that sends the first operation within an ST sequence. The endpoint that acts as initiator during one sequence (for example, the connection setup) can act as the responder in a subsequent sequence (for example, the data movement).

responder   The other (not the initiator) ST endpoint participating in an ST sequence.

slot   Memory at an ST destination that is reserved for holding one incoming ST Header.

ST Operations

The Operation is the basic protocol data unit for ST. Each ST Operation is carried within a single GSN Message, composed of two or more HIPPI-6400 micropackets, as illustrated in Figure 1-13.
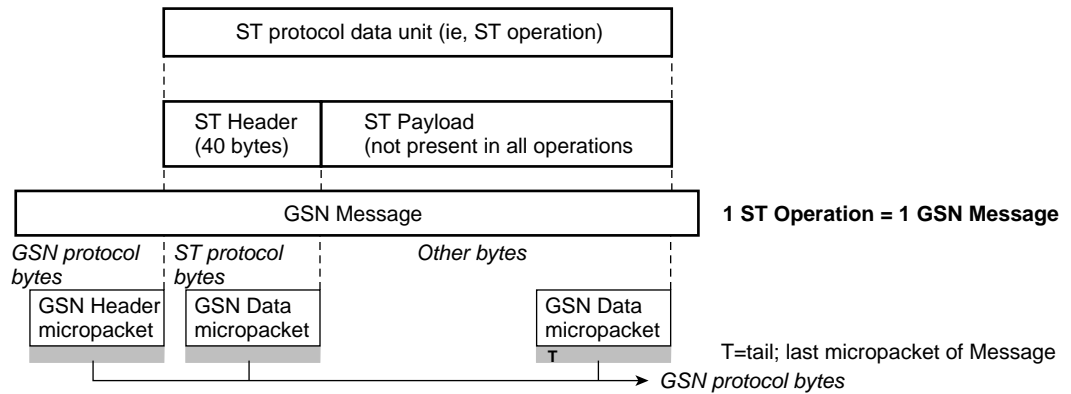
Figure 1-13   ST Operation

ST operations (listed in Table 1-7) are commonly grouped into the following categories:

- Connection management operations: used to set up and tear down a VC

- Control operations: used to manage a VC (for example, status or flow control)

- Data operation: used to transmit ST payload (upper-layer data) and/or data checksum during data movement sequences

24

Table 1-7   ST Operations

Name of Operation	Acronym	Category	Sequence in Which Operation is Used	Description
Request_Connection	RC	connection management	Setup	Requests that a VC be created. Issued by any endpoint. First operation of setup sequence.
Connection_Answer	CA	connection management	Setup	Response to RC. Accepts (creates VC) or rejects the RC. Second (and last) operation of setup sequence.
Request_Disconnect	RD	connection management	Teardown	Indicates that sender (initiator) is tearing down the VC. Issued by either endpoint of VC. First operation of teardown sequence.
Disconnect_Answer	DA	connection management	Teardown	Response to RD. Indicates that the sender (responder) is tearing down the VC. Second operation of teardown sequence.
Disconnect_Complete	DC	connection management	Teardown	Response to DA. Indicates sender (initiator) has finished tearing down VC. Third (and last) operation of teardown sequence.
Request_Memory_Region	RMR	control	Data Movement_ Persistent	Requests that responder expose memory. First operation of persistent memory sequence.
Memory_Region_Available	MRA	control	Data Movement_ Persistent	Response to RMR. Exposes responder's memory to initiator.
Get	GET	control	Data Movement_ Persistent	Issuer (initiator) is destination for the data movement. Exposes initiator's memory to receive the requested data. Data comes from source's exposed persistent memory region. RMR/MRA handshake must have occurred.
FetchOp	FETCHOP	control	Data Movement_ Persistent	Issuer (initiator) is destination for the data movement. Exposes initiator's memory to receive the requested data. Data comes from source's exposed persistent memory region. RMR/MRA handshake must have occurred.

Table 1-7 (continued)   ST Operations

Name of Operation	Acronym	Category	Sequence in Which Operation is Used	Description
FetchOp_Complete	FC	control	Data Movement_ Persistent	Response to FETCHOP.
Request_To_Send	RTS	control	Data Movement_ Single-use	Issued by source (=initiator for write or =responder for read). Indicates issuer is ready to transmit data; asks responder to expose single-use memory. First operation of write sequence.
Request_To_Receive	RTR	control	Data Movement_ Single-use	First operation for a read sequence. Indicates issuer is ready to receive data. Issuer becomes the initiator of the read sequence.
Clear_To_Send	CTS	control	Data Movement_ Single-use	Response to RTS. Gives source permission to transmit one block of data. Exposes single-use memory for that data.
Data	DATA	data	Data Movement	Carries ST payload and/or checksum; used in every data movement sequence. Sent by data source, which can be either initiator or responder within the data movement sequence.
Request_Answer	RA	control	Data Movement	Response to an RTS, RTR, RMR, GET, or FETCHOP. Accepts, rejects, or pauses the request to which it is responding.
Request_State	RS	control	Status	Requests VC status information. Issued by either endpoint.
Request_State_Response	RSR	control	Status	Communicates VC state information. Response to either an RC operation or a DATA operation in which the Send_state flag (within the ST Header) is set.

Table 1-7 (continued)   ST Operations

Name of Operation	Acronym	Category	Sequence in Which Operation is Used	Description
End	END	control	Abort Data Movement	Terminates an in-progress data movement (read/write transfer or a persistent memory region) by causing the allocated memory to be released; leaves VC open. Issued by either endpoint.
End_Ack	EA	control	Abort Data Movement	Response to END. Indicates responder has aborted the associated data movement.

ST Header

The ST Header (illustrated in Figure 1-14) carries the information that implements the ST protocol features. Some of the parameters that are communicated within the ST Header are:

- Type of operation (listed in Table 1-7)

- Data channel through which this operation travels, which, for ST-over-GSN, maps directly to GSN virtual channels (summarized in Table 1-3)

- Number of memory spaces (slots for holding ST Headers) that are currently available at each endpoint for this data channel (that is, VC)

- Port values for initiator and responder within each VC

- Key values for initiator and responder within each VC

- Length of the data to be moved from one endpoint to the other

- Block number for use in tracking progress, managing flow control and resource allocation, and performing striping within a data movement

- Memory address (buffer index and offset) to use for the data movement

- Checksum for the operation

- Identification numbers for tracking and sequencing operations: DATA operations, FETCHOP operations, GET operations, and REQUEST_STATE_RESPONSE operations within each VC

The following are some of the endpoint behaviors that can be controlled by the operation's ST Header:

- Whether or not the destination for a data movement supports reception of out-of-order Blocks

- Whether or not the operation's ST Header should be delivered to the destination's upper-layer protocol (ULP)

- Whether or not the destination ULP should be interrupted when this operation arrives

- Request status information from the endpoint receiving this ST Header

- Inform initiator that responder is rejecting a request

- Pause the transmission during a data movement

bit 0   31

| byte 0 | Op + Flags (below) | Param | byte 3 |
| | D_Port | S_Port | byte 7 |

carried in HIPPI-6400 Header micropacket

| D_Key |
| D_id | S_id |
| Bufx |
| Offset |
| Sync |
| B_num |
| I_Bufx |
| byte 36 | I_Offset | byte 39 |

carried in first HIPPI-6400 Data micropacket for GSN Message

Op and Flags Fields

bit 0   15

| Operation_code (see Table 1-7) | Function | T (ST Header not delivered to ULP) | Inerrupt ULP | Send_state | Out_of_order Blocks | Last STU of Block | Reject | Data channel |
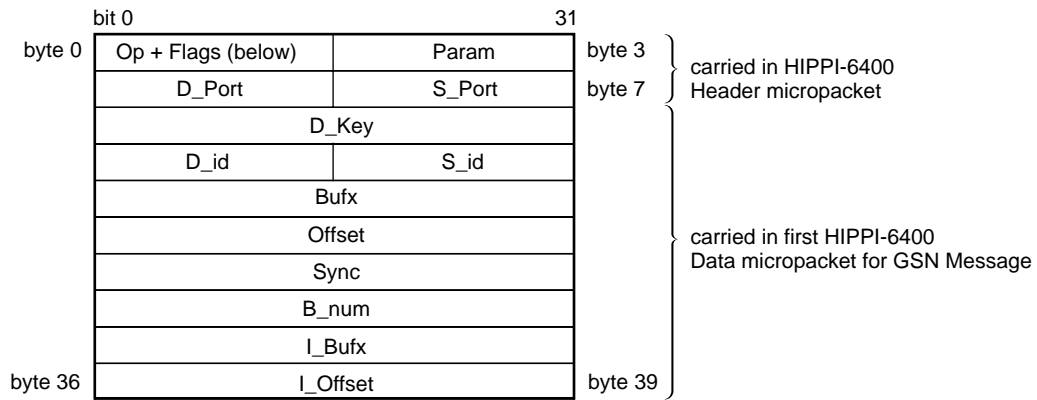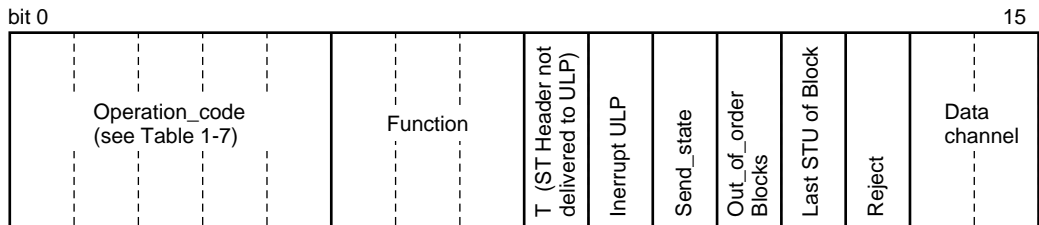
ULP = upper-layer protocol

Figure 1-14   ST Header

ST Sequences

ST defines sequences of operations for accomplishing various tasks, including the following:

- To open a connection between two endpoints and negotiate the parameters associated with the virtual connection. (See "ST Connection Setup Sequence.")

- To perform a data movement including the handshake that allocates memory at the destination. (See "ST Data Movement Sequences Including Memory Allocation.")

- To control the data flow during the data movement, thus enabling full-rate, non-congested data flow between the endpoints. (See "ST Flow Control Sequences.")

- To tear down a connection. (See "ST Connection Teardown Sequence.")

Each ST sequence allows the two endpoints to exchange a set of control parameters and information. The parameters are carried in the ST Header (illustrated in Figure 1-14). Each type of operation uses the Header fields differently and exchanges a different set of parameters.

ST Connection Setup Sequence

Before any ST data can be exchanged, a Virtual Connection (VC) must be set up between the initiator and the responder. Upon successful completion of this exchange, each endpoint will have stored a set of parameters associated with the VC and will have set aside some resources for exclusive use by this VC. Three of the stored parameters are used (as a tuplet) for identifying/validating operations that arrive to the VC. The verification tuplet consists of: the remote endpoint's ST port number, the local endpoint's ST port number, and the key value that the local endpoint has assigned to this VC. Figure 1-15 illustrates how these identification parameters are set up.

Note: The initiator for the connection setup sequence is the endpoint that sends the first control operation for the sequence (that is, the Request_Connection).

The connection setup sequence consists of two control operations: a Request_Connection sent by the initiator, followed by a Connection_Answer sent by the responder). Figure 1-15 and Figure 1-16 illustrate different subsets of the information exchanged in one successful connection setup sequence. Figure 1-17 illustrates a connection setup sequence in which the responder refuses to create the VC.

The ST connection setup sequence negotiates and sets the following parameters and resources that remain in effect for the duration of the VC:

- I_Port and R_Port
  ST port value on which endpoint (initiator and responder) wants to receive all communication associated with this VC.

- I_Key and R_Key
  Locally unique identification number (key) for use in verifying and identifying this VC. Each endpoint gives the other endpoint a key, which the other simply echoes back in each communication; the key means nothing to the remote end and is only "unique" at the endpoint where it was assigned.

- I_Bufsize and R_Bufsize
  Size of the buffers used by each endpoint for data it receives on this VC.

- I_Slots and R_Slots
  Initial number of "slots" available at each endpoint. Each slot indicates memory that has been set aside for storing ST headers that are received on this VC. Each slot normally consists of one 40-byte data structure.

- CTS_req
  Number of Clear_to_Sends that the source would like to have outstanding (available) at all times during the data movement.

- I_MaxSTU and R_MaxSTU
  Maximum size STU that each endpoint is willing to receive. The other endpoint must respect this size when transmitting on this VC.

- EtherType
  Identity of the protocol being encapsulated (carried) within the ST Messages on this VC. For example, for IP datagrams, the EtherType is 0x0800; when the ST Messages carry user data that is not enclosed in any additional protocol, the EtherType is 0x0000. The initiator specifies this parameter.
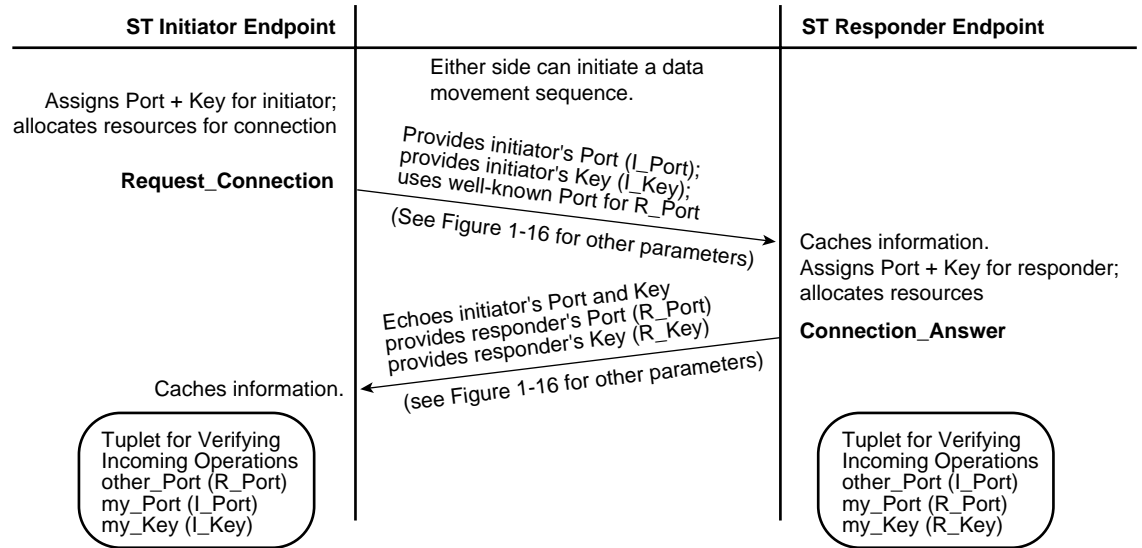
Figure 1-15　ST Connection Setup Sequence: Identification Parameters Only
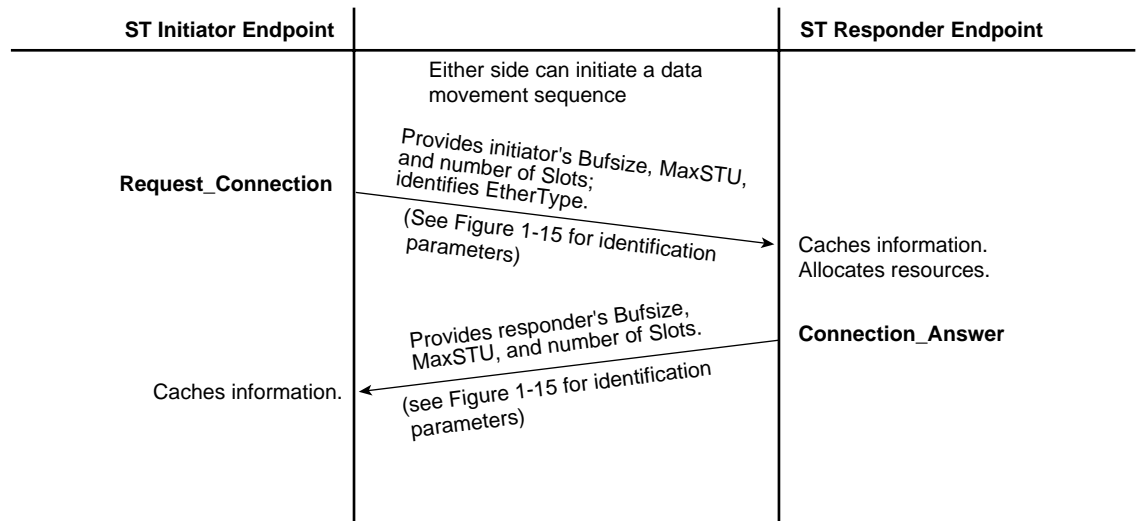
Figure 1-16　ST Connection Setup Sequence: VC Parameters Only
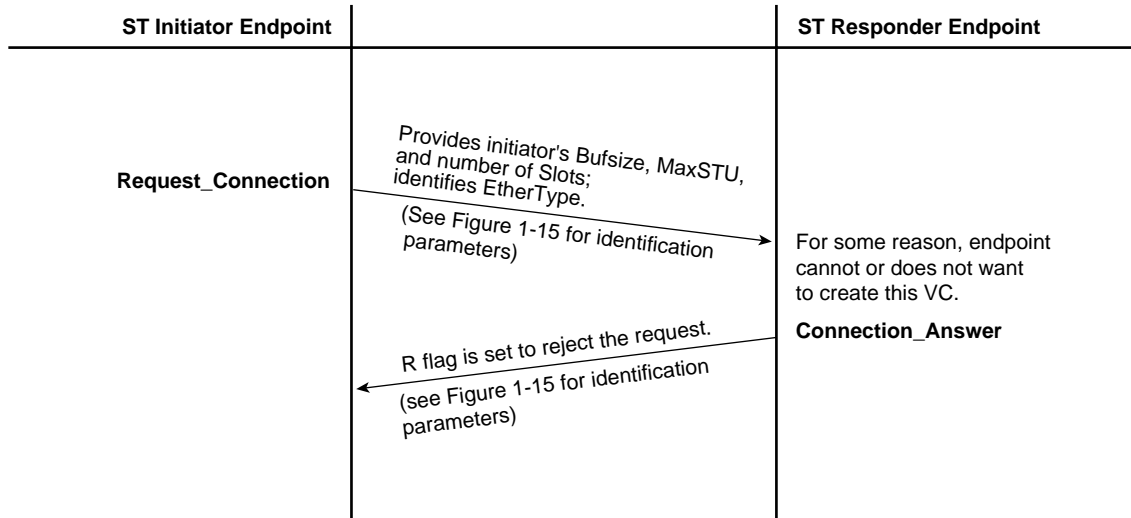
Figure 1-17   ST Connection Setup Sequence: Rejection

ST Connection Teardown Sequence

When an endpoint no longer wants a VC, it initiates the connection teardown sequence illustrated in Figure 1-18. This sequence is not used to terminate data movements. (See "ST Termination Sequence for a Data Movement".)
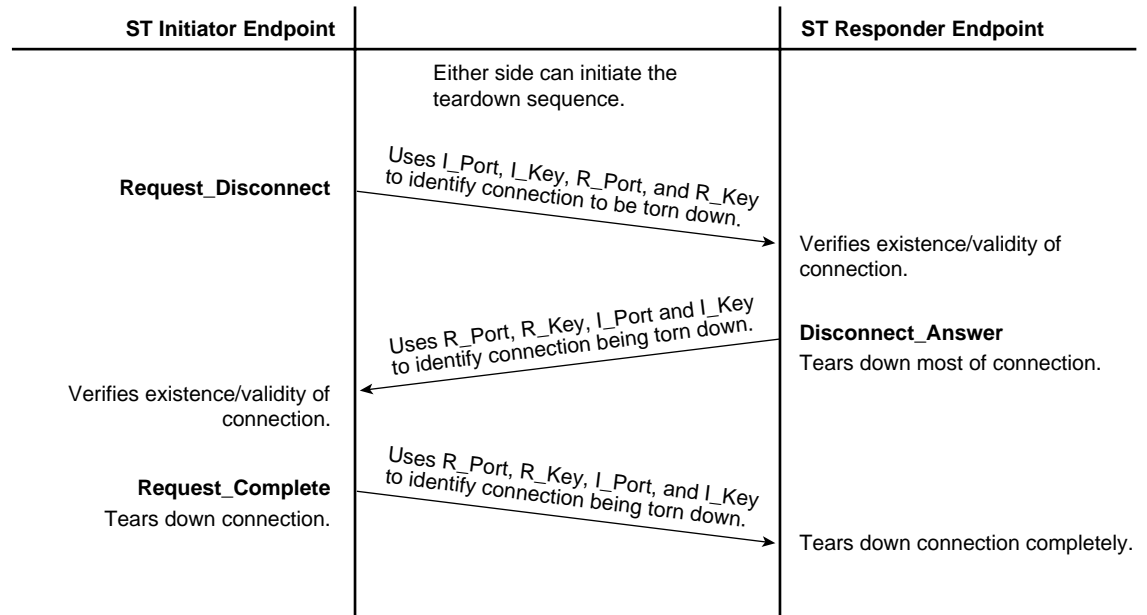
Figure 1-18   ST Connection Teardown Sequence

ST Data Movement Sequences Including Memory Allocation

This section describes ST data movement sequences. Each ST *data movement* sends upper-layer (user) data from one endpoint (the source) to one other endpoint (that is, one final destination). The entire data transfer is controlled by the VC parameters negotiated during one ST connection setup procedure (described in "ST Connection Setup Sequence") or renegotiated during the data movement. The setup sequence must be completed before any data movement sequence is initiated.

The data movement sequences consist of two to five operations, exchanged between the VC's two endpoints (the memory-allocation handshake), followed by one or more data operations. There are five different data movement sequences, as summarized in Table 1-8. The initiator controls which sequence is used, depending on the type of memory it wants to have allocated, the type of functionality it desires for the data movement, and the role it wants to assume in the transfer.

The memory-allocation handshakes allow either of the following types of memory to be allocated for receipt of the data:

- Persistent memory: a region of memory that is used over and over for the transfers that occur within that virtual connection, as described in "Persistent Memory Data Movements"

- Single-use memory: a region of memory that is written once, then released, as described in "Single-use Memory Data Movements"

Table 1-8 summarizes the five data movement sequences and indicates where each sequence is illustrated:

Table 1-8   Data Movement Sequences

	Persistent Memory	Single-use Memory
Initiator wants to be source	Figure 1-20	Figure 1-24
Initiator wants to be destination	Figure 1-21 and Figure 1-22	Figure 1-25

Note: Within a data movement sequence, the initiator is the endpoint that sends the first control operation for the sequence (for example, Request_to_Send or Request_Memory_Region), regardless of whether it operates as the data transmitter (source) or receiver (destination).

Table 1-9 summarizes the data size ranges for each type of data movement. As illustrated in Figure 1-19, the data is first chunked into one or more Blocks; the maximum size for a Block is negotiated during the memory allocation handshake. Each Block is divided into one or more scheduled transfer units (STU; the data for one data operation); the maximum size for the STU was negotiated during the connection setup sequence. Any ST data movement that is larger than the VC's maximum STU size requires multiple data operations. Each STU (that is, each data operation) is transmitted as one GSN Message.

The flow-control mechanism for user data (described in "ST Flow Control Sequences") operates at the Block level.

Table 1-9   Data Sizes Possible for Data Movements

Data Movement Type	Minimum Length Data Movement Sequence	Maximum Length for Data Movement Sequence
Single-use Memory: Write	1 byte	2^{64} minus 1 byte or unlimited
Single-use Memory: Read	1 byte	2^{64} minus 1 byte or unlimited
Persistent Memory: each Put	1 byte	2^{48} minus 1 byte or VC's max_STU (one Block)
Persistent Memory: each Get	1 byte	2^{16} minus 1 byte or VC's max_STU (one Block)
Persistent Memory: each FetchOp	8 bytes	8 bytes (one Block)

A set of user data destined for one endpoint (ie, ST Port)

Data for ST Movement

Chunk of data (one Block)

Another chunk

Another chunk

• • •

Final chunk

can be smaller than max_Block

can be smaller than max_Block

This is only Block (one STU) for PUT, GET, and FETCHOP operations.

data | data for STU | • • • | data

ST STU

ST Header (40 bytes) | ST Payload (≤ size of max_STU)

GSN Message

GSN protocol bytes | *ST protocol bytes* | *Upper layer bytes*

GSN Header micropacket | GSN Data micropacket | • • • | GSN Data micropacket

T

GSN protocol bytes

ST Blocks

All blocks are same size except, possibly, first and last. The max_Block size is negotiated by RTS/CTS handshake.
Absolute block size in bytes:
$256 \leq max\_Block \leq 2^{48}$
Within each Block, data is passed to recipient in correct order. When function is negotiated, Blocks can be out of order.

ST STUs

Max_STU size is established during VC setup sequence (RC/CA). Negotiated maximum STU size is:
$\leq$ recipient endpoint's Bufsize, and
$\leq$ maximum allowed on the ST data channel (see Table 1-3).

One ST Data Operation = one STU

Credits are not issued for ST header bytes because Header bytes are not placed in the memory allocated at the destination endpoint.

Each STU = 1 GSN Message

Figure 1-19   Data Handling for ST Data Movements

Persistent Memory Data Movements

The persistent memory sequences consist of a few control operations (the memory-allocation handshake) followed by any number of Put, Get, and/or FetchOp sequences. The persistent memory handshake allocates one or more memory regions at the responding endpoint. These regions are then used multiple times; each buffer within each region is used over and over during the life of the virtual connection. When properly used, this method provides permanent, low-latency delivery, in which an unlimited number of transfers can be performed with no intervening overhead. There is an important caveat: the low latency on this type of data transfer depends on the speed at which the memory can be made available for the next use. This type of transfer works best for small (or fixed-size) data and for applications for which the transmission rate is well understood, so that the memory can be sized in a manner that allows it to be recycled within an acceptable period of time. It is the responsibility of the upper-layer applications to manage flow control and prevent precipitous overwriting of the memory region.

Once a persistent memory region has been allocated at the responder endpoint, the initiator can move data in or out of it in three manners:

- Put sequence (illustrated in Figure 1-20)
  One data operation (STU) that writes any portion of or the entire persistent memory region at the responder. This sequence can be repeated over and over with no intervening operations.

- Get sequence (illustrated in Figure 1-21)
  A GET control operation to expose memory at the initiator for receiving the requested data, followed by any number of data operations. Each data operation moves a portion or all of the data from the responder's allocated memory into the initiator's memory. Multiple GETs can be outstanding (occurring) simultaneously to different or shared portions of the persistent memory region.

- FetchOp sequence (illustrated in Figure 1-22 and Figure 1-23)
  A FETCHOP control operation to expose memory at the initiator for receiving the retrieved data and to specify the desired function (increment, decrement, or clear). Then, a single data operation (one STU) that moves one 64-bit Block of data from the responder's memory into the initiator's memory. When the data arrives successfully at the initiator, the initiator issues a completion control message, at which point the responder performs the specified function on its own copy of the data. If the completion does not arrive within a timeout period, the responder retransmits the data. Note that, unlike PUT and GET, this data movement sequence is atomic.

A persistent memory region is terminated (released) with an End operation, as described in "ST Termination Sequence for a Data Movement."

ST Initiator Endpoint		ST Responder Endpoint
	ST Connection Setup Sequence has been completed.	
	Either side can initiate the handshake.	

Request_Memory_Region

waits for permission.

Verifies existence/validity of connection.

Pauses or rejects request.

Request_Answer or

Allocates memory for reception.

Exposes size and location of responder's persistent memory region.

Memory_Region_Available

Verifies existence/validity of connection. Caches memory information.

Memory Allocation Handshake

Data

One STU = Block

Verifies existence/validity of connection. Stores data in persistent memory region.

Tuplet for Verifying Incoming Operations
other_Port (R_Port)
my_Port (I_Port)
my_Key (I_Key)

Initiator can do Put, Get, FetchOp, and/or End Sequences.

Tuplet for Verifying Incoming Operations
other_Port (I_Port)
my_Port (R_Port)
my_Key (R_Key)

Figure 1-20   ST Data Movement Sequence: Persistent Memory—Put
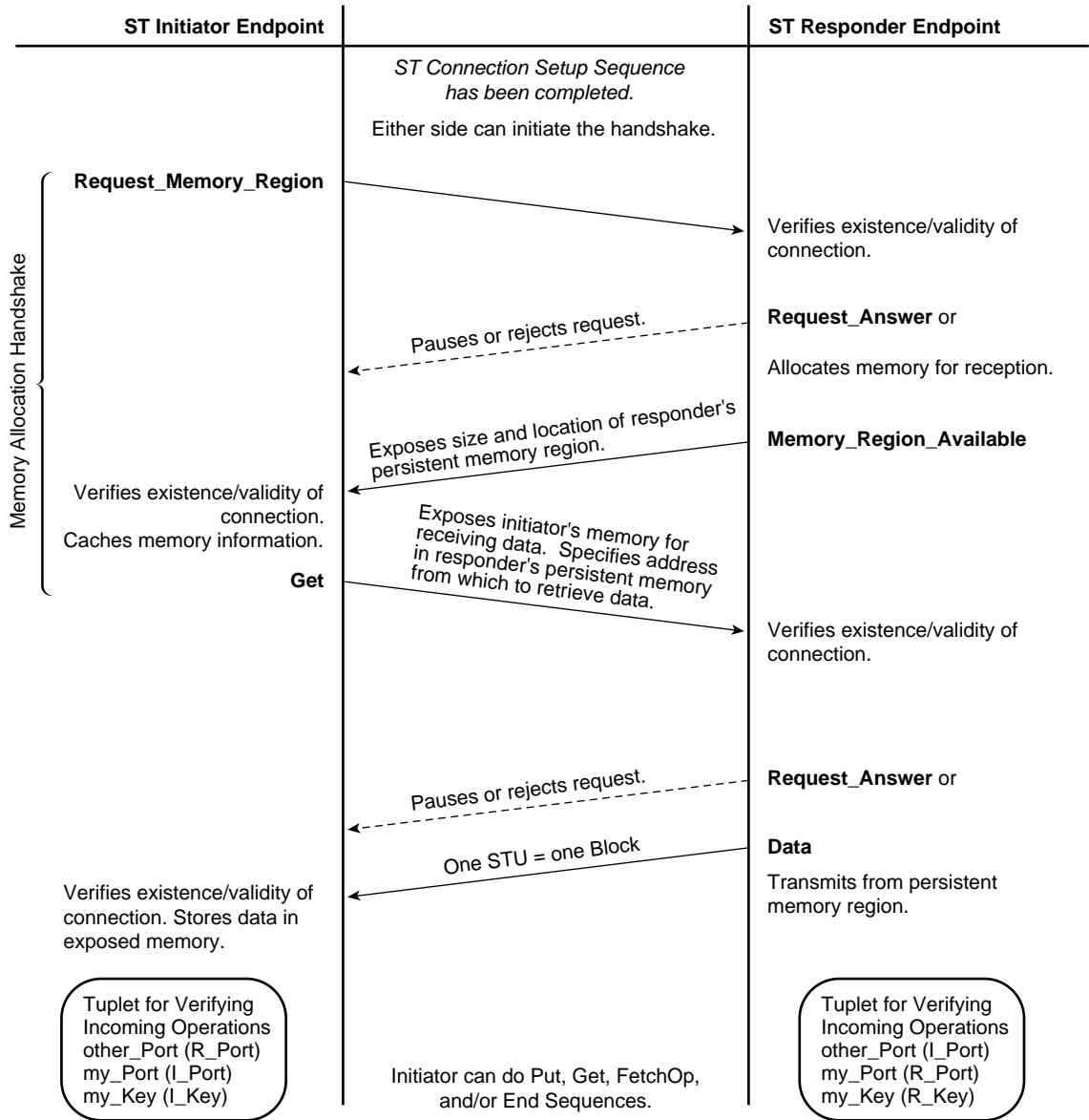
Figure 1-21   ST Data Movement Sequence: Persistent Memory—Get
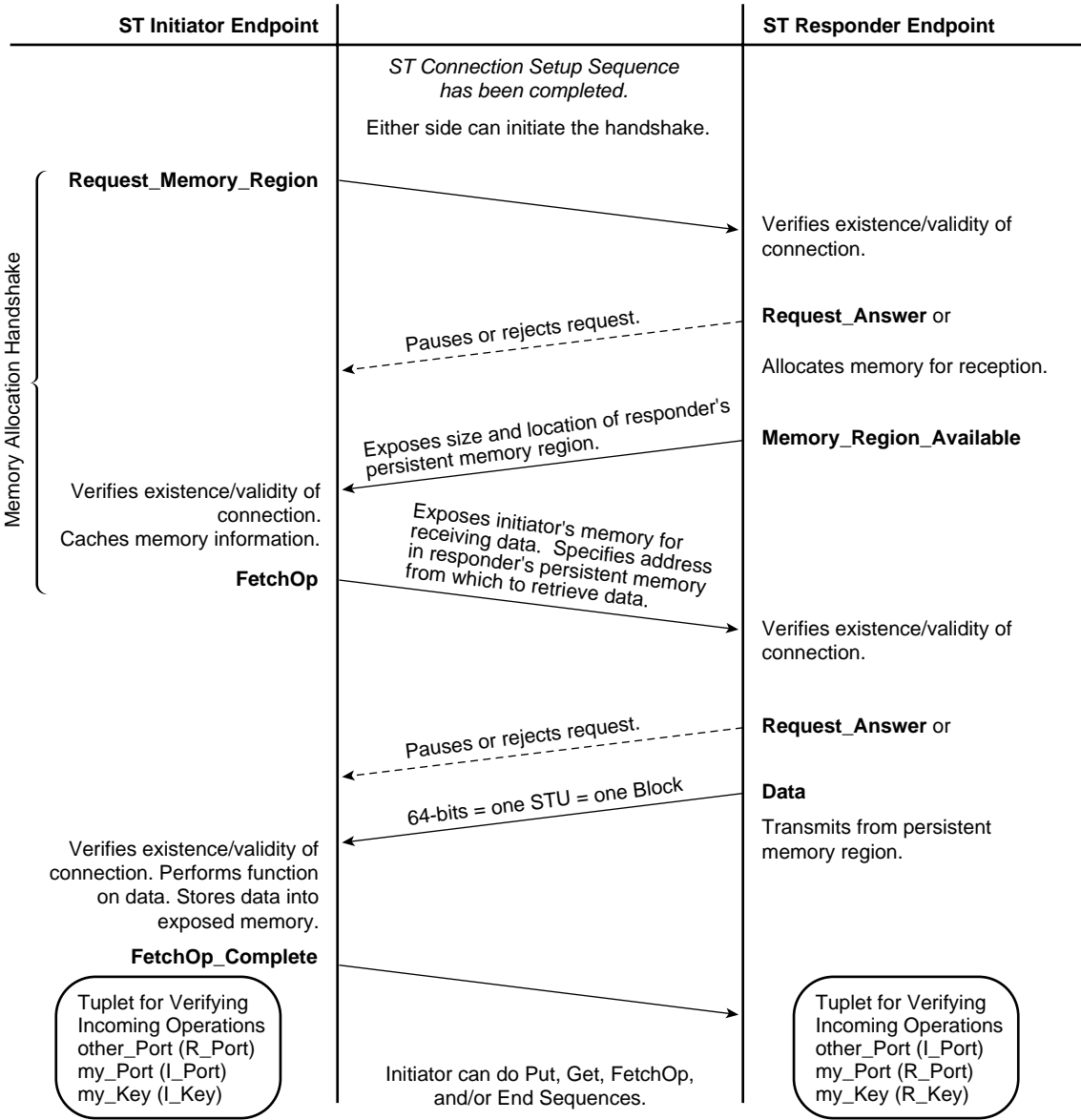
Figure 1-22   ST Data Movement Sequence: Persistent Memory—FetchOp
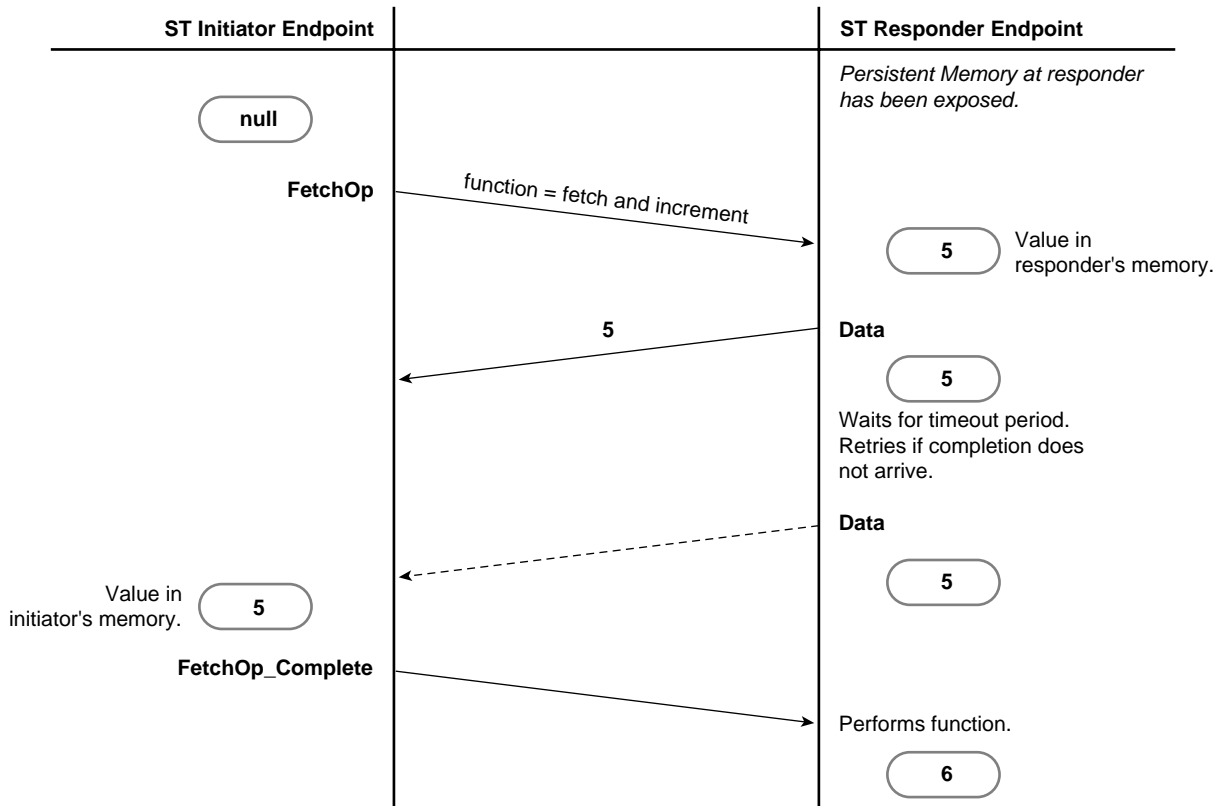
Figure 1-23   Example of FetchOp

Single-use Memory Data Movements

The single-use memory movement sequence consists of a few control operations (the memory-allocation handshake) that allocate memory at the destination endpoint, followed by one or more data operations for a specified amount of data. The data transfer uses the destination's allocated memory once; each buffer is used only once during the life of the transfer. This method allows high-bandwidth delivery after an initial delay for the allocation of resources: the transfer provides for a limited number of back-to-back writes or reads with no intervening overhead. This method is efficient for large, variable-length data.

A data transfer can be aborted (terminated before all the data has been transferred) with an End operation, as described in "ST Termination Sequence for a Data Movement."

Figure 1-24 illustrates the data transfer sequence used when the initiator is the data source. Figure 1-25 illustrates the sequence used when the initiator is the data destination. Each illustration includes the memory allocation handshake.
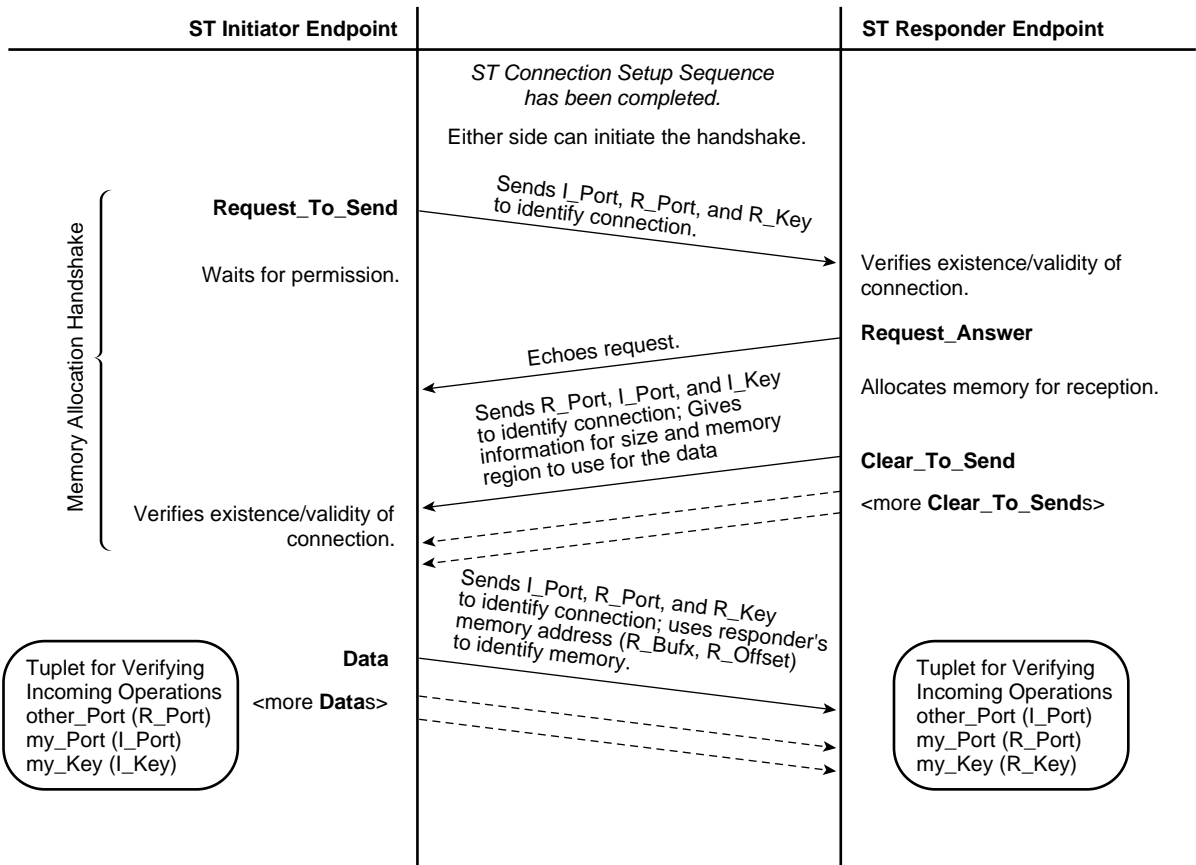
Figure 1-24   ST Data Movement Sequence: Single-use Memory with Initiator as Source

42

ST Initiator Endpoint		ST Responder Endpoint

*ST Connection Setup Sequence
has been completed.*

Either side can initiate the handshake.

Memory Allocation Handshake

Request_To_Receive

Uses I_Port, R_Port, and R_Key
to identify connection.

Waits for permission.

Verifies existence/validity of
connection.

Echoes request.

Request_Answer

Uses R_Port, I_Port, and I_Key
to identify connection;
Requests region of memory.

Request_To_Send

Verifies existence/validity of
connection. Allocates
memory for reception.

Waits for permission.

Sends I_Port, R_Port, and R_Key
to identify connection.
Gives information for size and memory
region to use for the data.

Clear_To_Send

Allocates memory for reception.

Verifies existence/validity of
connection.

<more **Clear_To_Send**s>

Uses R_Port, I_Port, and I_Key
to identify connection; uses responder's
memory address (R_Bufx, R_Offset)
to identify memory.

Data

<more **Data**s>

Tuplet for Verifying
Incoming Operations
other_Port (R_Port)
my_Port (I_Port)
my_Key (I_Key)

Tuplet for Verifying
Incoming Operations
other_Port (I_Port)
my_Port (R_Port)
my_Key (R_Key)

Figure 1-25   ST Data Movement Sequence: Single-use Memory with Initiator as Destination

ST Flow Control Sequences

Flow control operates differently for data transfers and ST operations. Each is explained below.

Data Transfer Flow Control

ST endpoints implement strict flow control for all data transfers done to single-use memory. For this purpose they use the Request_To_Send (RTS) and Clear_To_Send (CTS) control operations. There can be multiple CTSs generated in response to one RTS, as explained below and summarized in Table 1-10.

The ST flow control sequence regulates both the number of data transfer events that occur between the two endpoints and the size of these events. Before any data is transferred, the data transmitter (source) generates an RTS, in which it specifies the maximum size block of data and the number of blocks that it wants to send right now. The specified (requested) size and number do not oblige the receiver to give permission for that size or number; these are only suggestions that, if followed, could make the transfer more efficient.

The data receiver (destination) generates one or more CTSs in response to each RTS. In each CTS, the receiver gives the source permission to transmit one block of data; the number of CTSs issued by the receiver cannot exceed the number of "requested blocks" specified in the RTS. In the first CTS for the data movement, the receiver indicates the block size that it is willing to receive during this data movement; the block size must be no larger than the maximum block size specified in the associated RTS. Before issuing each CTS, the receiver must allocate the amount of memory specified by the block size in that CTS. See Figure 1-24 and Figure 1-25 for illustrations of the flow control sequence.

Note: ST does not use flow control for persistent memory data movements: Put, Get, and FetchOp.

Table 1-10   ST Flow Control Sequence

Transfer Event Parameter	Source Specifies	Destination Specifies
Number of events	In RTS: number of blocks the source would like to send at this time. Limitations: none.	In CTS: with each CTS, the destination gives the source permission to transmit 1 block of data. Limitations: Destination must not issue more CTSs than source has requested in its RTS. Destination must allocate memory for each CTS it generates.
Size of each event	In RTS: requested maximum block size for transfer events associated with this RTS. Limitations: none. **Note**: When the source does the actual data transfer, the size is not controlled by the RTS maximum block size; it is limited by the block size specified in the CTS.	In first CTS: block size that will be used for these transfer events. Limitations: Block size must not exceed maximum size specified in the associated RTS.

Operation Flow Control

Flow control for the ST Headers of ST operations is managed with a mechanism called slot allocation. Each *slot* represents memory that has been allocated at an endpoint to hold one incoming ST Header while it awaits processing. All incoming ST Headers use one slot, except Request_Connection operations and Data operations that have the Silent flag set.

Note: Data operations with the Silent flag set, do not occupy a slot because the ST Header for these operations is not passed to the receiving endpoint (and hence is not stored). The Request_Connection operation does not occupy a slot because the VC does not yet exist when this operation arrives. An implementation may have a queue of slots associated with Port 0 (the port to which the Request_Connection arrives), but the queue is not required because there are no consequences caused by the endpoint dropping the request other than the initiator trying again, until it succeeds.

During the setup sequence for a VC, each endpoint communicates to the other endpoint the number of slots it has allocated for that VC. Updates for slot availability are communicated during normal operation with Request_State_Response operations. (See "ST Status Sequences" for details.) Each source keeps track of the number of outstanding operations (that is, slot-consuming ST Headers that it sends) and makes sure that it does not send more operations than the destination can handle.

ST Status Sequences

During normal operation, the endpoints for a VC can use either of two status sequences (illustrated in Figure 1-26 and Figure 1-27) to obtain information from the other endpoint about its state and status.

ST Initiator Endpoint   **ST Responder Endpoint**

Request_State

Request_State_Response
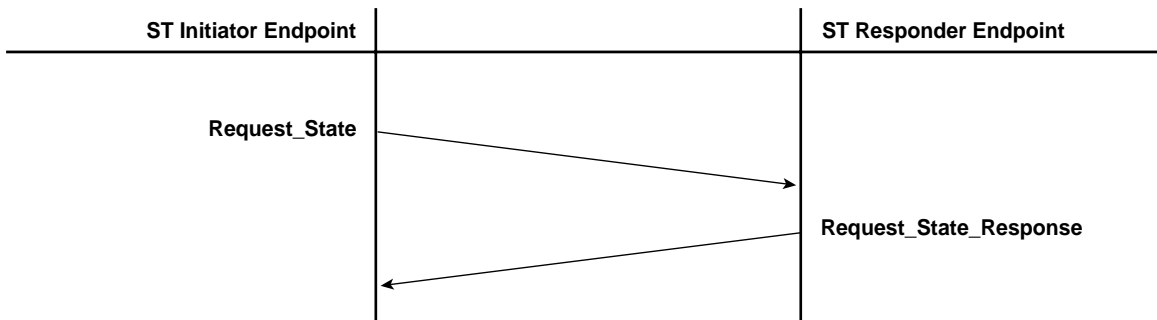
Figure 1-26   Status Sequence Using Request_State

ST Initiator Endpoint   **ST Responder Endpoint**

Data   *S flag = 1*

Request_State_Response
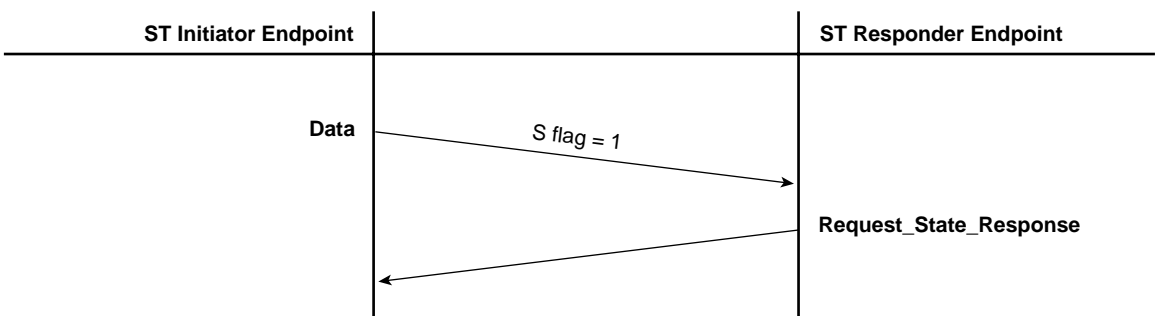
Figure 1-27   Status Sequence Using S Flag in ST Header

46

The information that can be exchanged with this mechanism includes:

- number of currently available slots for this VC

- highest Block received for a data movement

- reception status for a specific Block

ST Termination Sequence for a Data Movement

The following data movements do not have a natural ending:

- a persistent memory region

- a data transfer of unlimited size

To terminate either of the above data movements and release the associated resources, either endpoint initiates the termination sequence illustrated in Figure 1-28. In addition, this sequence can be used to abort a data transfer of specific length before all the data has been transferred.

ST Initiator Endpoint   **ST Responder Endpoint**

There is currently an active data movement. Either side can initiate the termination.

End

End_Ack
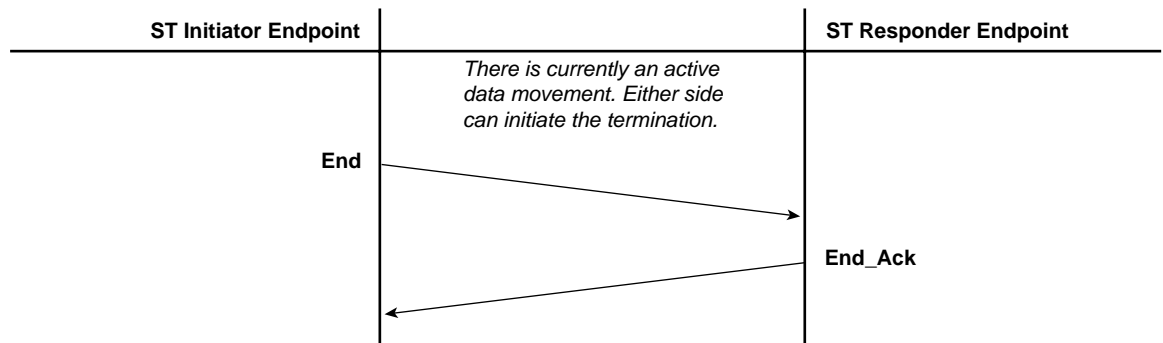
Figure 1-28   Termination Sequence

Example of ST Virtual Connections and GSN Channels

GSN virtual channels are designed to carry specific sizes of data (see Table 1-3). The various ST data channels (DCs) that exist within ST virtual connections (VCs) can take advantage of these sized GSN channels. The IRIX ST-over-GSN stack routes any ST operation with DC=0 to GSN channel 0, DC=1 to GSN channel 1, and so on. For example, each ST application (for example, ST Port), is required to have one data channel (DC_0) for its control operations and one or more other channels (DCs 1, 2, and/or 3) for its data operations. Note that each GSN channel is shared by many VCs; for example, DC_0 for all ST VCs share GSN channel 0. Figure 1-29 shows an example of ST VCs using their data channels (DC values) to effectively make use of the four GSN channels.
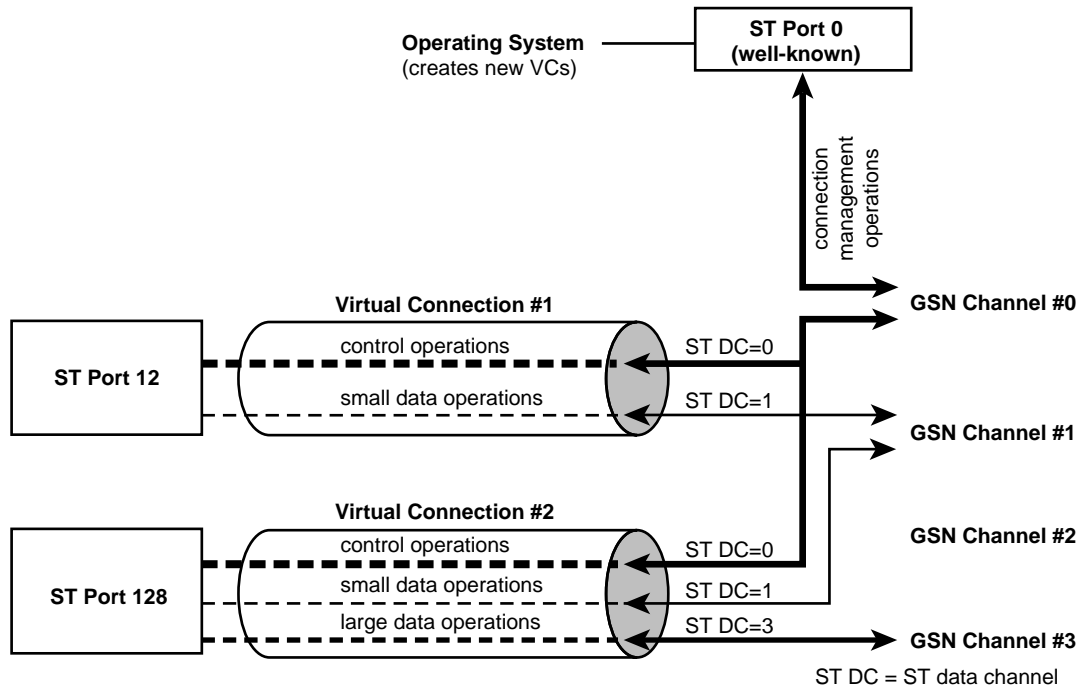
Figure 1-29   Example of ST Virtual Connections Using Multiple GSN Virtual Channels

48

GSN Fabrics and Logical Networks

This section explains how logical networks are created on GSN and HIPPI fabrics. The discussion assumes that you have a thorough understanding of the concept of a logical network, the format of *INET address*es, and the use of *subnet mask*s to divide a single INET network address space into smaller networks (called subnets or *LIS*es).

Note: For complete details on INET address subnetting and the netmask, see the comments in the */etc/config/ifconfig.options* file, the man page for inet(7F), the man page for ifconfig(1M), and the online IRIS InSight document *IRIX Admin:Networking and Mail*.

There are three basic concepts that underlie the discussion in this section. Each is discussed in more detail in subsequent sections:

Basic Concept #1

> The hosts connected to a GSN or HIPPI *fabric* do not have to function as one logical network whose addresses all come from one address space.

Basic Concept #2

> A logical (sub)network or LIS (one address space) can include hosts from physically different GSN and/or HIPPI fabrics, as long as there is a *bridging switch* between the fabrics.

Basic Concept #3

> Within a GSN or HIPPI fabric, direct communication (without use of an intermediate router) between INET hosts can occur only when (1) the network interfaces involved in the exchange have addresses that come from the same logical address space (for example, they are members of the same subnet/LIS), and (2) both hosts have access to an address resolution mechanism.

Basic Concept #1

The hosts connected to a GSN or HIPPI *fabric* do not have to function as one network address space. The hosts can be organized into smaller groupings (for example, based on function, project, or hardware manufacturer). Each grouping of hosts is a separate logical network or a logical subnet/LIS. Each (sub)network is assigned a sequence of network-layer addresses (that is, a unique address space). Figure 1-31 illustrates this concept.

A group's address space can be the complete range of addresses for an INET network address (192.0.2.0 to 192.0.2.255), or it can be a portion of the range (for example, subnet 192.0.2.0 to 192.0.2.31). Membership in a group is determined for each GSN network interface (for example, each *gsn#*) by the INET address associated with the interface (in the */etc/config/netif.options* file) and the netmask value (in the *ifconfig.options* or the *ifconfig-#.options* file). The netmask value defines the size of the address space for each group. For example, a netmask value of 0xFFFFFF00 creates an address range that provides 256 individual host addresses. However, netmask value 0xFFFFFFE0 (shown in Figure 1-30) creates eight subnets/LISs in which each subnet/LIS can have up to 32 "host" addresses.

Basic Concept #2

A logical network or a subnet/LIS can include hosts from physically different GSN and HIPPI fabrics, as long as there is a "bridging" communication path between the fabrics. Hosts that are members of the same INET address space (thus benefitting from the services provided by broadcast and routing) do not have to be physically attached to the same physical medium (fabric). Figure 1-32 illustrates this concept.

Basic Concept #3

Direct communication between INET hosts (without use of an intermediate router) can occur only when the network interfaces involved in the exchange are members of the same logical address space (network or subnet/LIS). Contact with members outside one's own (sub)network requires use of an INET address router.

This rule is true even when a shared hardware connection (for example, a switch) exists between the two hosts that belong to different (sub)networks/LISs. For example, for two hosts attached to the same switch, a message from host A in LIS 1, if sent to host B in LIS 2, must go through host C, an INET router. The benefit is that, no matter where a GSN network adapter is physically located or relocated, it continues to function as a member of the same LIS. Notice that no address or LIS-membership change is required when an endpoint is physically relocated.

In addition, all hosts must have access to both an address resolution mechanism (for example, HARP) and a network broadcast service. This duo enables mapping each INET-address to a GSN (ULA) address. The broadcast service can be implemented in hardware (for example, in the bridging switches) or in software (for example, a broadcast emulation server). For complete details, see "Address Resolution for GSN" on page 57.

The following facts explain why this concept exists:

- GSN switches do not resolve network-layer (INET) addresses.

- The local INET routing software (for example, IRIX' *routed*) does not maintain complete paths to destinations that are not members of the same subnet/LIS.

- Before transmission of an IP packet, a GSN hardware address (ULA) must be discovered for the destination. This step requires the services of a HARP server

- Each HARP server maintains mappings only for its own (sub)network.

Consequences and Examples

The basic concepts summarized in "GSN Fabrics and Logical Networks" on page 49, make the examples described in this section possible.

Figure 1-31 and Figure 1-32 show examples of subnetting within two different GSN fabric configurations. The subnet/LIS addressing used in these examples (summarized in Figure 1-30) is identical. The examples use network INET address 192.0.2, so that each host address is 192.0.2.*xxx*. Hosts in LIS_1 use addresses between 192.0.2.0 and 192.0.2.31; those in LIS_2 use addresses between 192.0.2.32 and 192.0.2.63, and so on.

32-bit INET address: nnnnnnnn_nnnnnnnn_nnnnnnnn_hhhhhhhh (binary)

32-bit INET address: subnetted: nnnnnnnn_nnnnnnnn_nnnnnnnn_sss hhhhh

                       24 network or netid bits   host or hostid bits

                                          additional bits to create subnet/LIS

subnet mask: 11111111_11111111_11111111_11100000 (binary)

                   FF:FF:FF:E0 (hexadecimal)

| **LIS addresses (showing only 3 subnet bits):** | sss = 000 for LIS 1
sss = 001 for LIS 2
sss = 010 for LIS 3
sss = 011 for LIS 4
and so on | **Host addresses (5 host bits):** | LIS 1: 192.0.2.0 to 192.0.2.31

LIS 2: 192.0.2.32 to 192.0.2.63 |

Figure 1-30   Subnet Mask for Examples

If you want a single-fabric site to have multiple address spaces, you can use multiple INET network addresses, or you can use a *netmask* to divide a single INET address space into smaller chunks (referred to as subnets or LISs). Likewise, in a multiple-fabric site (for example, an interconnected/bridged HIPPI-800 fabric and GSN fabric) you can group all the hosts into one logical address space, or into multiple subnets/LISs regardless of each host's location.

Figure 1-31 illustrates a GSN fabric that has one switch to which all the network interfaces are attached (that is, all endpoints in this fabric have a direct physical link to one another). The example shows two subnets/LISes. Communication from A in LIS_1 to C in LIS_2 passes through the router (network interfaces J and H). Messages do not go directly from endpoint A to C, because of the concept explained in "Basic Concept #3".
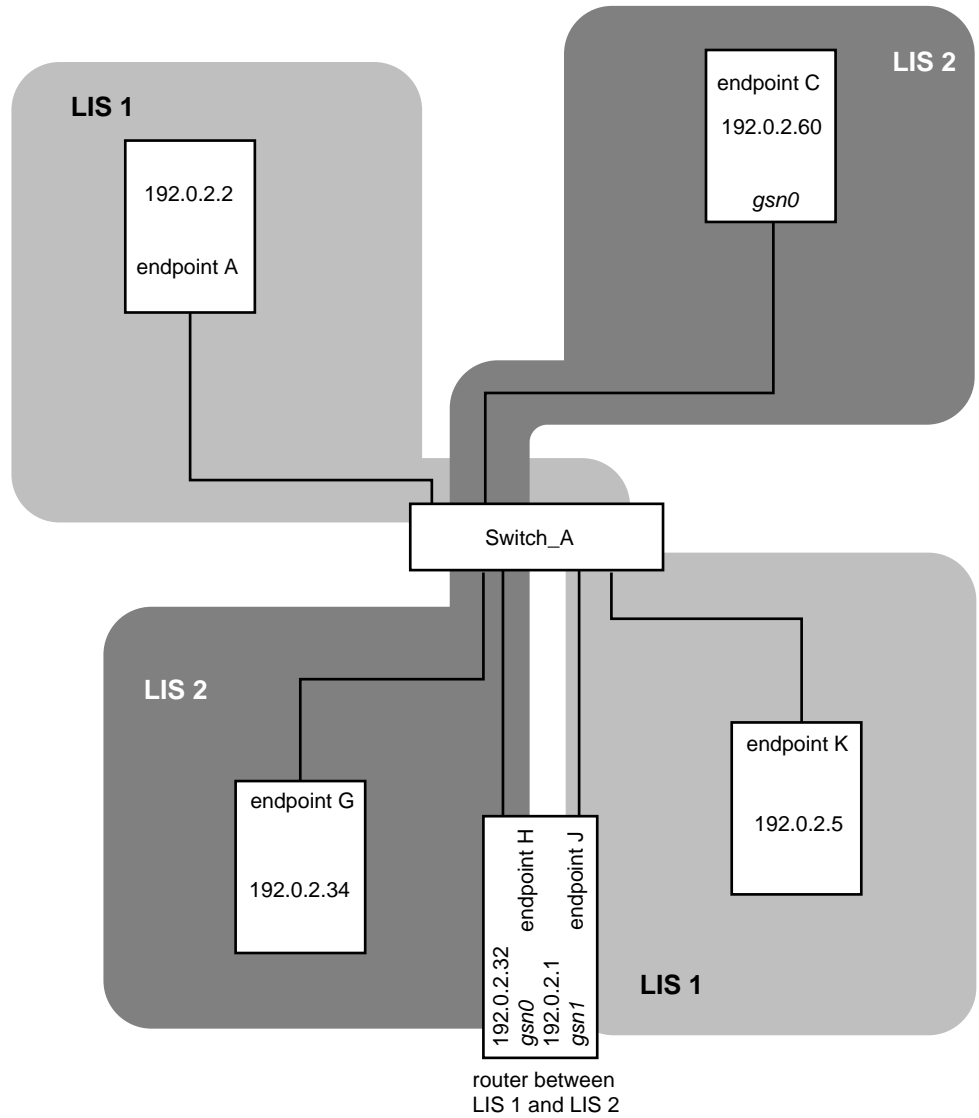
Figure 1-31   Single-switch GSN Fabric with Subnets/LISs

Figure 1-32 illustrates a different configuration for the same address space and network interfaces ("hosts") used in Figure 1-31. This configuration is a two-switch fabric. In this example, A, B, E, J, K, and L belong to LIS_1, while C, D, F, G, and H belong to LIS_2. The system with network interfaces H and J continues to perform as the router between the two LISs. Just as in the first example (Figure 1-31), communication directed to C in LIS_2 from A in LIS_1, goes first to the router (J/H), even though both A and C are physically attached to the same switch. But, most importantly, notice that the router has been moved to a different switch, and yet, the INET addressing is identical to that used in the first configuration. The hardware changes do not affect the addressing. Also note that a router for an LIS does not need to share a switch with the members of its LISs, as illustrated by router J in relation to hosts A and B and router H in relation to hosts C and D.
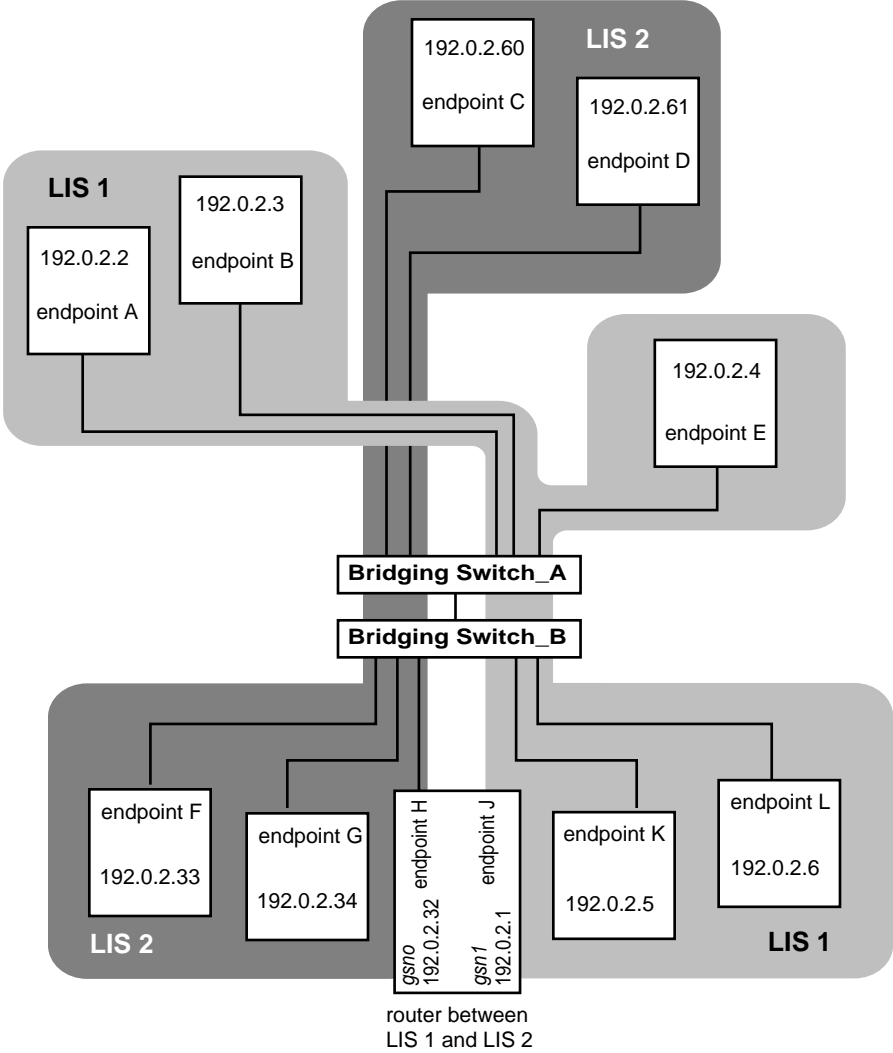
Figure 1-32   Multiple-switch GSN Fabric with Subnets/LISs
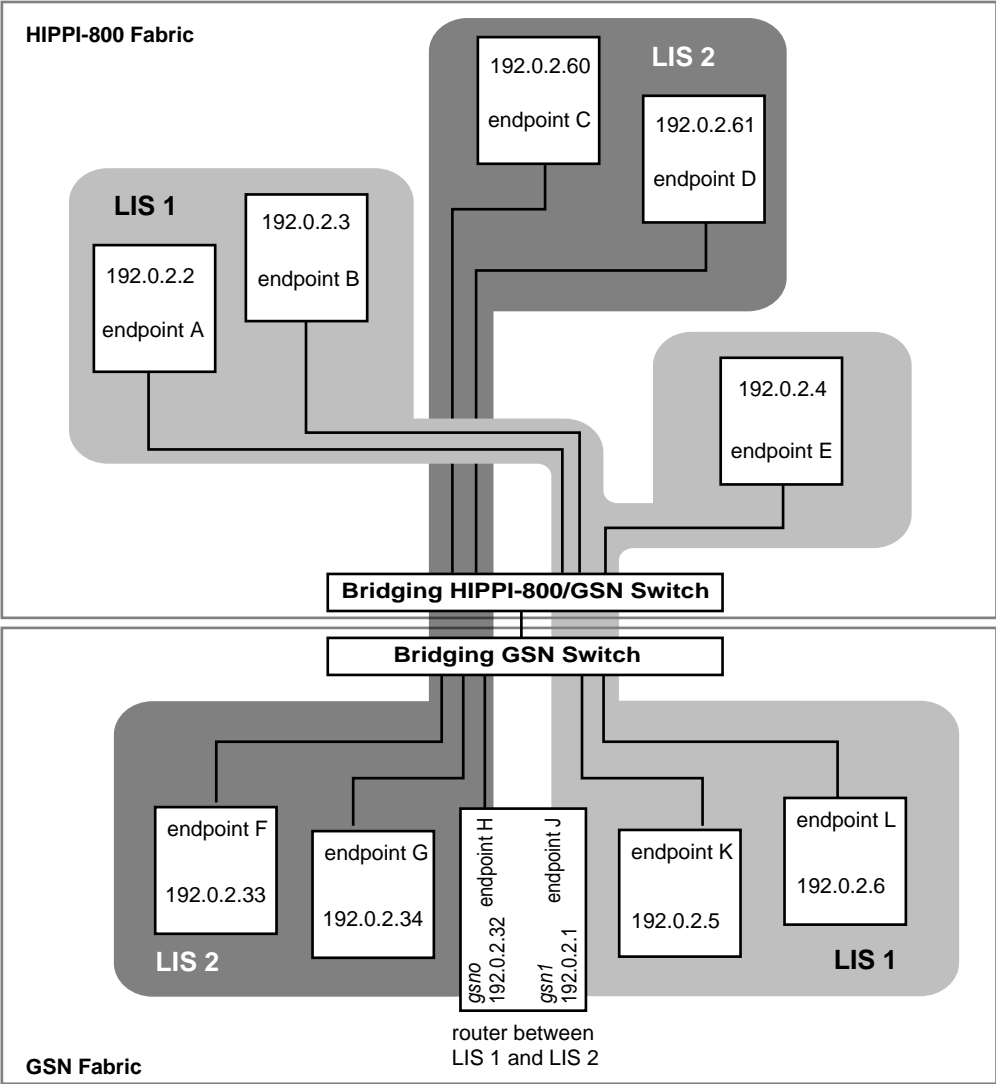
Figure 1-33   LIS Membership That Spans Fabrics

Address Resolution for GSN

This section describes how network (OSI layer three) addresses are mapped (resolved) to physical (OSI layer-one) addresses in a GSN fabric. This section assumes that you are familiar with standard Internet ARP (RFC 826, *Ethernet Address Resolution Protocol*) and Inverse ARP (RFC 1293, *Inverse Address Resolution Protocol*) protocols.

When a network-layer address is locally associated with (configured to) an IRIX GSN or IRIS HIPPI subsystem, address mapping is needed between network-layer addresses and physical-layer addresses so that communication can occur between the local network-layer entity and remote network-layer entities. The GSN/HIPPI physical address is known as the universal LAN MAC address or ULA. For IRIX, the default network protocol stack is the Internet Protocol and the network address is the *INET address*.[1] The address resolution scheme for IP/ST-over-GSN is defined by RFC Internet Draft, *IP and ARP over HIPPI-6400/GSN*, as described in the section "HARP Address Resolution."

Note: Each INET address (AF_INET) can support multiple protocols. For example, in IRIX 6.5, INET addresses support both the IP suite of protocols (PF_INET) and the ST protocol (PF_ST). For further details, see the man page for *inet(7)*.

To transmit data to another network-layer entity within the GSN fabric, each network-layer stack in the GSN fabric needs two addresses for each destination:

- The network-layer address for the destination host. In IRIX, this information is supplied by the static "hosts" database or the dynamic NIS server.

- The physical-layer address for the destination endpoint. This information is supplied by the static HARP table or the dynamic HARP server. See "HARP Address Resolution" on page 60 for details.

---

[1] For IRIX GSN, the Scheduled Transfer Protocol is an additional default stack; ST shares the INET address used by IP.

HARP and Broadcast Support

A GSN fabric is said to support broadcasting when all the switches of that fabric provide broadcasting. The behavior for HARP clients and HARP servers is slightly different depending on whether or not the underlying GSN/HIPPI fabric supports broadcasting:

When the fabric does not support broadcasting

Only one host within the address space (LIS) provides *HARP service* by behaving as the HARP server. All hosts behave as HARP clients. The HARP service is centralized—provided by a single HARP server. All hosts (that is, HARP clients in the address space) contact the server, both to register their own address mappings and to obtain information about the other members of the address space. There may be a list of hosts that can perform the HARP service; however, only one is used by a host at any point in time. The HARP standard requires that the active HARP server also provide broadcast emulation services for the address space when the fabric does not support broadcasting. Each time a client registers or re-registers itself with the HARP server, the server broadcasts the information to all the hosts in its database. Each host maintains a local dynamic HARP table. Each local host gathers and updates the HARP table by reading every broadcast HARP packet that it sees. The default address for contacting the HARP server is ULA 00:10:3b:ff:ff:e0 and the server must be located (for example, attached to the switch port) at that address.

When broadcast is supported by all switches in fabric

Each host within the address space (LIS) behaves as both a HARP client and a server. The HARP service is distributed. Each HARP client transmits (using the broadcast address) HARP packets that register it with every host (HARP server) in the address space. The switches broadcast these packets to all endpoints in the fabric. Whenever any host sees a HARP packet with its own INET address as the target, its local HARP server application generates a reply that provides the requested ULA. Each host maintains a local dynamic HARP table. It gathers and updates the HARP table by reading every broadcast HARP packet that it sees. The default address for contacting this distributed HARP service is the broadcast address (ULA FF:FF:FF:FF:FF:FF). No system is assigned ULA FF:FF:FF:FF:FF:FF.

To function properly, each HARP client and server must figure out whether its underlying fabric supports broadcasting or not. Each system discovers whether or not the attached fabric supports broadcasting:

Clients   When a client does not see its own initial registration packet (that is, its inverse ARP request), it knows the fabric does not support broadcasting. It then tries to register itself with the remote HARP server at either the standard (default) address (00:10:3b:ff:ff:e0) or at the ULA that was manually configured for contacting the server.

When a client sees its own inverse ARP request (which has been broadcast by the fabric's switches), it knows that there is fabric support for broadcasting, so it continues to register at the distributed dynamic HARP address: ULA FF:FF:FF:FF:FF:FF.

Servers   When a system's own ULA matches the "ULA for contacting the HARP service[1]", the system knows that it is the only HARP server and that the fabric does not support broadcasting. The address recommended by the standard (and IRIX HARP's default) for the server is ULA 00:10:3b:ff:ff:e0. In this scenario, the server responds to HARP requests from all clients in the LIS.

When a system's own ULA does not match the "ULA for contacting the HARP service", the system knows that it is not the only HARP server within the LIS, and that there is a server on every host. In addition, when a system sees its own inverse ARP request, it knows that the fabric supports broadcasting. In this scenario, the "ULA for contacting the HARP service" is FF:FF:FF:FF:FF:FF, but no system should have this as its own ULA. The fabric broadcasts packets that have FF:FF:FF:FF:FF:FF as their destination address. Each server responds only to those broadcast HARP packets that request information about itself.

---

[1] In IRIX HARP, the "ULA for contacting the HARP server" can be either the default (which matches the value required by the HARP standard) or it can be a customer-configured value.

HARP Address Resolution

The address resolution protocol for HIPPI networks (both HIPPI-6400/GSN and HIPPI-800) is specified in the HARP RFCs. The protocol works with fabrics that provide broadcasting and with those that do not. One of the first tasks of each HARP client is to determine if its underlying fabric supports broadcasting, as described in "Determining Fabric Support for Broadcast" on page 60.

HARP provides a dynamic, client/server-based address resolution service. The protocol makes it possible for each IP/ST-over-HIPPI endpoint (client) within a (sub)network to register or communicate its own INET address and physical address, and to discover the physical addresses for hosts with whom it wants to communicate. (The INET address must be obtained, for example, from the site's NIS or *etc/hosts* database.) The HARP server maintains a kernel-resident lookup table (database) that maps INET addresses to physical addresses. HARP occurs in two phases: a registration phase (summarized in the section "HARP Registration Phase") and a normal operation phase (summarized in "HARP Normal Operation Phase").

When a (sub)network/LIS includes one or more endpoints that do not support dynamic HARP, static mappings for those endpoints must be added to the address resolution table at the HARP server (as described in the section "Static Address Resolution" on page 65).

Determining Fabric Support for Broadcast

Every host that supports HARP transmits its first HARP packet to the broadcast ULA: FF:FF:FF:FF:FF:FF (hexadecimal notation). If the host subsequently sees this (its own) packet on the fabric, it knows that the fabric supports broadcasting. If it does not see the packet, it knows that the fabric does not support broadcasting, in which case it sends its next HARP packet to either ULA 00:10:3B:FF:FF:E0 or the manually-configured address for contacting the HARP server. For a more complete description of this client and server behavior, see "HARP and Broadcast Support" on page 58.

Note: The HARP protocol is designed so that this initial HARP packet serves the double purpose of both registering the client with the *HARP service* (as explained in "HARP Registration Phase" on page 61), and discovering the fabric's support for broadcasting.

HARP Registration Phase

During bringup, each dynamic HARP client registers its address pair (INET and physical address[1]) with the HARP service for its (sub)network.[2] The client does this by sending an inverse address resolution request (InARP_Request) to the physical address for the HARP service. The InARP_Request contains the client's INET address and physical address, and it asks for the server's INET address. To do this first step, the client *must* know the physical address for the HARP service.[3] The HARP standard recommends that the ULA FF:FF:FF:FF:FF:FF be used for this purpose. Complying with this recommendation allows clients to locate their service without any special configuration. If the recommended ULA is not used for the HARP service, the system administrator must manually configure this information on each client within the (sub)network.

If no reply (InARP_Reply) comes back to the client, the client waits five seconds and tries again; it continues this cycle until it succeeds. If the system has been configured with backup HARP servers, the client sends each request to all the addresses until it determines which addresses are active and selects one as primary. When the server receives an InARP_Request, it enters the new INET-to-ULA mapping into its table and replies with an InARP_Reply message that provides its own INET address to the client. If the fabric does not support broadcasting, the server broadcasts its reply to all the HARP clients in its database so that they all know about this new client. Once this is done, the client is registered.

Note: The format for HARP packets is slightly different for HIPPI-800 and HIPPI-6400 clients. IRIX HARP responds to requests in each format with a reply in the appropriate format.

---

[1] For HIPPI-6400, the physical address is the ULA. For HIPPI-800, it is the Switch Address (a portion of the I-field) and the ULA.

[2] The HARP service may be either of the models described in "Determining Fabric Support for Broadcast" on page 60.

[3] Note that the ULA for the HARP service does not need to be the same as the ULA for the HARP server that responds to the request. In a fabric with broadcasting, the HARP service ULA is always FF:FF:FF:FF:FF:FF, while the ULA for the HARP server is different in each reply because it is each endpoint's own ULA.

If a client wants to verify that its registration has been successful, it may send an ARP_Request to the service asking for its own physical address. (See "HARP Normal Operation Phase" for details about the ARP_Request.) When the server receives this standard ARP_Request, it looks in its table for the client's INET address to discover the physical address that is mapped to that INET address. It sends the results of this lookup back to the client in an ARP_Reply message.

Note: On a fabric with broadcast support, the HARP server that responds to this request is the client's own local HARP server.

Figure 1-34 illustrates the exchange that occurs between client and server during registration.

The server keeps each registered entry in its database for a maximum of 20 minutes. Whenever an entry has aged beyond this maximum, the server removes the entry. To prevent its entry from disappearing, each functional client re-registers itself with the service at least once every 20 minutes.

Note: When an endpoint is engaged in a very large data transaction or an ST data transfer of indeterminate length, the HARP client at that endpoint cannot access the link to communicate with its HARP server. This may cause the client's entry in the HARP server's database to disappear.

IP Client on GSN Network		HARP Server on Same GSN Network

Provides client's ULA; provides client's IP; includes server's ULA asks for server's IP.

InARP_Request

If no reply occurs within timeout, client waits and tries again later.

Adds time-stamped entry to table.

InARP_Reply

Provides server's IP.

Caches information.

Provides client's own IP (as target); asks for client's own ULA.

ARP_Request
(optional, to verify the entry)

Performs lookup in table.

InARP_Reply

Provides client's ULA.

Verifies that information is correct.

Figure 1-34   HARP Registration

HARP Normal Operation Phase

During normal operation, each HARP client requests address resolution from the HARP service for each IP/ST destination. The client makes this request by sending an ARP_Request to the service; the request contains the desired destination's (that is, target's) INET or physical address. The server responds with an ARP_Reply that supplies the target's missing address. Figure 1-35 illustrates the message exchange that occurs between client and server during normal operation.

When the HIPPI fabric does not support broadcasting, reception of an ARP_Reply implies one of the following about the target:

- The targeted client supports dynamic HARP and within the last 20 minutes was functional enough to register itself with the server that sent the ARP_Reply.

- The targeted client does not support dynamic HARP and its entry at the server is a static entry. There is no way to know whether the client is functional.

63

When the HIPPI fabric supports broadcasting, reception of an ARP_Reply means that the targeted client supports dynamic HARP and is currently functional.

An ARP_NAK indicates that the IP/ST client (endpoint) has not registered itself with the server for at least 20 minutes. This failure to register can be caused by any of the following conditions:

- The endpoint is powered off.

- The link to the client has been unavailable for at least the last 20 minutes. For example, the link might be continuously occupied by a network transaction, such as a large or indeterminate data transfer. The cable might be loose.

- The HARP client software at the endpoint is not up and running.

- The endpoint does not support dynamic HARP and the HARP server's database does not contain a static entry for the client.

**IP Client
on GSN Network**

**HARP Server
on Same GSN Network**

ARP_Request

Provides one of target host's addresses.
Asks for target host's other address.

Performs lookup in table.

IF entry is available.

Provides target host's missing address.

ARP_Reply

Caches information.

OR if no entry is available.

ARP_NAK

Waits and tries again later.

OR

no response at all
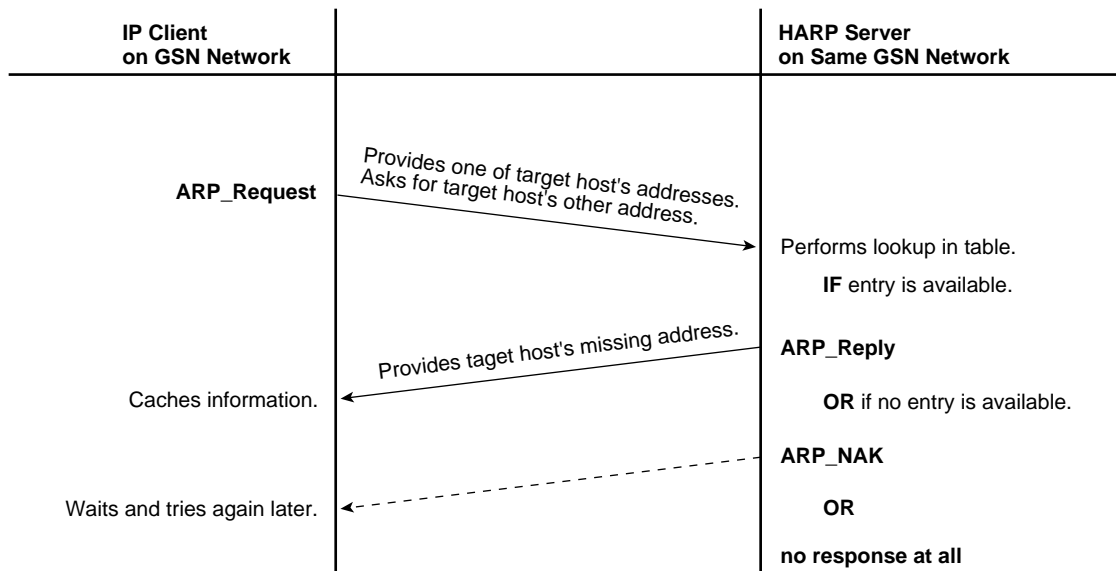
Figure 1-35   HARP Normal Operation

64

Static Address Resolution

When a host within a HIPPI/GSN (sub)network does not support dynamic HARP, the system administrator needs to add a static entry for that host to the HARP server's database. The entries can be placed into a file (for example, the */etc/config/gsnarp.options* file) that the HARP software automatically loads into the kernel-resident database when it starts, or the entry can be added directly (manually) to the database by using a utility (for example, *gsnarp*). Each entry in the database must map a ULA (IEEE or MAC address) to an INET address, and for HIPPI-800 hosts, an I-field must also be specified.

Guidelines for Selecting a HARP Server

These guidelines explain how to select a system to provide HARP services (that is, be the HARP server) when the HIPPI fabric does not support broadcasting. It is not necessary to identify a system for this purpose when the fabric supports broadcasting.

From among the members of the address space (network or subnet/LIS), one system must be selected to perform the HARP server function. Follow these guidelines when selecting this system:

- The system and its GSN connection to the fabric must be available (operational) whenever any member of the address space wants to access the GSN fabric. This does not necessarily mean that the HARP server must be up and running 7 days a week for 24 hours a day. It does mean that, if you select a system that is not always available, you need to understand the network-accessing habits/needs of all the clients. When the HARP server is not available, no HARP client within that address space can communicate with any other IP or ST host.

  Note: Members of the address space that use only local static table lookup address resolution are not affected by the unavailability of the HARP server for transmission. However these clients do not receive communications from members that do rely on the server for address resolution.

- The system must not be utilized for lengthy (that is, larger than 64-kbyte) data transfers over its GSN fabric connection. The link to the HARP server must become available frequently so that client requests can get through in a timely manner. In most cases, the amount of traffic on the system's GSN link does not affect performance; large numbers of small-sized datagrams or data movements do not present a problem; however, even one lengthy transfer could negatively affect many members of the address space.

- For ease of configuration and to enhance interoperability, the HARP server should be attached to the one port in the GSN fabric that uses Universal LAN MAC address (ULA) 00:10:3b:ff:ff:e0. This complies with the HARP standard.

Note: With IRIX GSN/IRIX HARP, one HARP server can service multiple address spaces (LISs).

How Address Resolution Works for ST-over-GSN

The IRIX GSN implementation of the ST protocol uses the same address resolution scheme as is used for IP-over-GSN. See "Address Resolution for GSN" for the details.

Note: Each *gsn#* network interface services two protocols: ST and IP. The INET address assigned to an instance of *gsn#* is shared by the ST-over-GSN and IP-over-GSN stacks. Some of the upper-layer address processing (for example, routing) that is performed on the address applies to both IP and ST traffic.

IRIX HARP Table

The HARP table is a list of address mappings. Each entry (mapping) consists of an IP address/name and a GSN ULA, and, possibly a HIPPI-800 I-field address. Each entry is either a dynamic entry or a static entry, as explained below.

Static Entries

IRIX HARP does not alter or remove static entries from its HARP table. These entries are loaded or removed from the table only with the *gsnarp* utility. With each invocation of *gsnarp -f filename*, static entries are loaded into the HARP table from the file. Entries are removed with *gsnarp -d*. A file of any name can be used to load entries; however, the command uses */etc/config/gsnarp.options* as the default filename. The text in the file must follow the format described in "Configure IRIX GSN Network Interfaces In Real Time" on page 79.

The *gsnarp* utility can be invoked manually at the command line. It is automatically called by the */etc/init.d/network* script during each system startup.

Dynamic Entries

IRIX HARP maintains the dynamic entries in its HARP table in conformance with the HARP standard. It adds entries as it learns about them, refreshes them as they are reregistered by their owners (the clients), and ages and deletes entries as they go stale.

Assignment of Unit Numbers and Network Interfaces to GSN Hardware

The description in this section applies to systems running IRIX 6.5.5 (or later) and to network interfaces for the Internet Protocol suite (INET address over GSN subsystem)and Scheduled Transfer (ST-over-GSN) protocol.

Assignment of Unit Numbers to Hardware

With each restart (for example, a power on, a *reboot* or *init 0* command), the startup routine probes for hardware on all the modules connected into the CrayLink interconnect fabric. All the slots and links in all the modules within the fabric are probed. The routine then creates a hierarchical filesystem, called the hardware graph, that lists all the located hardware. The top of the hardware graph is visible at */hw*. For complete details, see the man page for hwgraph(4). After the hardware graph is completed, the *ioconfig* program assigns a unit number to each located device that needs a number. Other programs (for example, *hinv* and each device's driver) read this assigned number and use it.

The XIO slots are searched (probed for a device) in the order shown below; this order is not the same sequence as the XIO slot numbering. For example, the device in XIO slot 4 is located before the device in slot 2 and, because of this, may have a lower unit number than the device in slot 2. After the first power on, you can edit the */etc/ioconfig.conf* file to assign unit numbers that are convenient for you. Your changes are used during each subsequent power on. See the ioconfig(1M) man page for further details.

1. slot 8
2. slot 11
3. slot 10
4. slot 7
5. slot 12
6. slot 9

7. slot 4

8. slot 2

9. slot 6

10. slot 5

11. slot 3

On an initial system startup, *ioconfig* groups devices into classes/types and assigns hardware unit numbers sequentially within each class. It records these assignments in the */etc/ioconfig.conf* file; for example, if two SGI GSN products are found, they are numbered unit 0 (*gsn0*) for the first one found and unit 1 (*gsn1*) for the second one. When an SGI GSN product is a two-board solution, both boards are associated with a single unit number. On subsequent startups, *ioconfig* distinguishes between hardware that it has seen before and new items. To previously seen items, it assigns the same hardware unit numbers (those that are recorded in the *ioconfig.conf* file). To new hardware, it assigns new sequential numbers and records them. It never reassigns a number, even if the device that had the number is removed and leaves a gap in the numbering. For example, in a system with two instances of some class of devices, if the *unit0* is removed, the next restart results in the system listing only *unit1*; if a new board is installed in a new location, it is listed as *unit2*.

New items are differentiated from previously seen items through the hardware graph listing (that is, the path under */hw/module/#/slot/io#/...*). The database of previously seen devices is kept in the file */etc/ioconfig.conf*. A replacement board (with the exact same *hardware device name*) that is installed into the location of an old board (so that it has the same hardware graph listing) is assigned the old board's unit number, but a board that is moved from one location to another is assigned a new number. For example, in a two-device system with *ioconfig.conf* entries illustrated below, if *unit0* is moved to a different slot, the next restart results in a new item in the *ioconfig.conf* file. The *hinv* command lists *unit1* (an original board in its original slot) and *unit2* (the board that has been moved to a new slot), but not *unit0*. For more information about the hardware graph and *ioconfig*, see the man pages for hwgraph(4) and ioconfig(1M).

```
Initial entries for two devices:
0 /hw/module/1/slot/io8/xio_gsn/device
1 /hw/module/1/slot/io4/xio_gsn/device
0 /hw/gsn/0
1 /hw/gsn/1

Entries after unit0 is moved:
0 /hw/module/1/slot/io8/xio_gsn/device
1 /hw/module/1/slot/io4/xio_gsn/device
2 /hw/module/1/slot/io5/xio_gsn/device
1 /hw/gsn/1
2 /hw/gsn/2
```

The two-board SGI GSN product occupies two XIO slots that are logically associated with a single device (one unit number). The device has two XIO slots and two hardware graph entries. All links (for example, the short or convenience path, */hw/gsn/#*) point to the XIO slot for the main SGI GSN board. All located SGI GSN hardware devices can be displayed with the */sbin/hinv* or *find* command.

Assignment of Network Interface to Hardware Device

As the startup process continues, it calls the network drivers and protocol software modules so that they can create their network and programmatic interfaces. For GSN, this step works in the following manner:

- For each located SGI GSN device (port), the startup process creates short (*/hw/gsn/#*) and long (*/hw/module/#/slot/io#/xio_gsn*) entries in the hardware graph. Then, the initialization scripts create a symbolic link in */dev* that points to the device's entry in the hardware graph.

- For each located GSN hardware device, the startup routine creates an entry in the hardware inventory database that can be displayed by *hinv.*

- For each located hardware device, the IRIX GSN driver creates a logical network interface and assigns it a number that matches the hardware. For example, if the only hardware device is */hw/gsn/2*, then the only network interface created is *gsn2*.

- The *ifconfig* command searches the *netif.options* file for IP-over-GSN *network interface name*s (for example, *gsn0*, *gsn1*, *gsn2*), associates each network interface with the hardware that is specified, then configures and enables each interface.

69

Installing and Configuring IRIX GSN

This chapter provides instructions and information about configuring the IRIX GSN software, as summarized in this list.

- The entire installation and configuration procedure is covered in "Complete GSN Installation Process" on page 72.

- The following sections provide quick-reference summaries of the configuration tasks required for specific functionality:

    - "Configure IP-over-GSN" on page 74

    - "Configure the Address Resolution Service" on page 75

- Each specific configuration task is described separately in detail in the reference section "Individual Configuration Tasks" on page 77.

- Table 2-2 provides a listing of all the individual parameters in IRIX GSN that can be configured.

See "Assignment of Unit Numbers and Network Interfaces to GSN Hardware" on page 67 for a description of how the physical network devices (*gsn0*, *gsn1*, *gsn2*, and *gsn3*) are numbered and assigned to IP/ST-over-GSN logical network interfaces (*gsn0*, *gsn1*, *gsn2*, and *gsn3*).

Complete GSN Installation Process

This section lists the steps required for configuring your IRIX GSN network connection. The procedures listed below are those that must be performed before an IRIX GSN connection is functional:

1. Use *inst* or the System:SoftwareManager to install the IRIX GSN software from CD-ROM, as explained in the *IRIX GSN Release Notes*. The *inst* command is described in the online man page and the InSight document *IRIX Admin: Software Installation and Licensing* that came with the system.

2. Collect the information you need for the configuration, as described in "Collect Information Before Starting" on page 73.

3. Follow the steps in the section "Configure IP-over-GSN" on page 74 to configure the IP-over-GSN and ST-over-GSN stacks.

4. Optional: change the default settings for optional configuration tasks, as summarized in Table 2-2 on page 94.

   Note: If the system has more than one GSN network interface, the netmask values are not optional; they must be set as described in "Summary of Network Interface Operational Parameters and Default Settings" on page 89.

5. Use the decision tree provided in "Configure the Address Resolution Service" on page 75 to determine which set of HARP configuration instructions you need to follow. Then follow those instructions.

6. If there are any members of the LIS that do not support dynamic HARP, follow the instructions in "Adding Static HARP Table Entries" on page 85.

7. Arrange to have the hardware installed and its functionality verified as described in *SGI GSN XIO Hardware Installation Instructions for SGI Origin Family, SGI-2000-series, and Silicon Graphics Onyx2 Platforms*.

   It is most efficient to install the IRIX GSN hardware after installing and configuring the software. This avoids at least one reboot of the system.

   Note: If the hardware is already installed, follow the instructions in the section "Building a New Driver Into the Operating System" on page 93. Then, follow the instructions in "Verify That the Board Has Been Located by the Software" on page 130 and "Verify the SGI GSN Hardware" on page 131 to verify the functionality of the GSN hardware.

Collect Information Before Starting

To configure the IRIX GSN software, you need the following information. SGI recommends collecting this information before you start to configure the software.

❑ An INET (also known as, IP) address and, optionally, a subnet mask for each IP/ST-over-GSN network interface.

If there are multiple GSN network interfaces on this system, each interface must belong to a different LIS. That is, the address and mask for each interface must identify all the interfaces on the system as members of different address spaces (LISs). See "GSN Fabrics and Logical Networks" on page 49 and Figure 1-30 for further explanation.

❑ A network connection name for each INET address. This name usually includes the system's hostname (as configured in the */etc/sys_id* file). An example is: gsn3-amazon.

❑ If the GSN port is connected to a GSN switch, answer these questions:

Do all the switches in the GSN fabric support broadcasting?

Does this system have multiple GSN network interfaces?

If the answer to either of these questions is yes, collect the information listed below. Otherwise, you are through collecting information; you can proceed to the next step in the installation procedure (step 3 on page 72).

❑ If broadcasting is not supported by the GSN fabric or if this system has multiple GSN network interfaces, determine the ULA of the HARP server for each interface. This information must be known even when this system is selected as the HARP server for one or more LISs.

❑ For each GSN interface, identify all members of its LIS that do not support dynamic HARP.

73

Configure IP-over-GSN

This section describes how to configure the IP-over-GSN and ST-over-GSN network interface. All these steps are standard IP configuration procedures, required for all IRIX network interfaces.

1. Enable the network layer stack:

    ```
    # chkconfig network on
    ```

2. Open the local */etc/hosts* file for editing. For each GSN physical port, add an entry (like the one below) that maps an *INET address* (for example, IP address) to a GSN *network connection name* ("hostname"). Each address is shared by both the IP and ST protocols. (See "Edit hosts File" for detailed instructions.)

    ```
    gsn-amazon  223.209.1.18
    ```

    Note: The local *hosts* file must be edited. In addition, if the site uses an NIS or DNS server, this information must be added to that server's database.

3. Open the */etc/config/netif.options* file for editing. For each GSN physical port, add a pair of entries (similar to the pair below) to map the GSN network interface (*gsn#*) of the port to one of the INET/IP addresses (or network connection names) from the *hosts* file. (See "Edit netif.options File" for detailed instructions.)

    ```
    if2name=gsn0
    if2addr=gsn-amazon
    ```

4. Edit the */etc/config/ifconfig-#.options* file for one or more of the GSN network interfaces to change the default settings for the operational parameters that are listed in Table 2-1. This step is required if there is a subnet mask (netmask) for this network interface; otherwise, this step is optional. (See "Edit ifconfig-#.options File" on page 87 for detailed instructions.)

5. Optional: edit the */var/sysgen/master.d/gsn* file to change default settings in the IP/ST-over-GSN driver. (See "Edit master.d/gsn File" on page 90 for detailed instructions.)

6. If this is the final configuration step, rebuild the operating system so that it includes the newly configured IRIX GSN driver:

```
% su
Password: thepassword
# /etc/init.d/autoconfig
Automatically reconfigure the operating system (y or n)? y
# /etc/shutdown
```

The system is now ready to have its IRIX GSN hardware installed, or if the hardware is already installed, you can restart the system. The IP/ST-over-GSN functionality automatically becomes operational during this restart.

7. To verify this configuration, follow the procedures described in "Verifying the IRIX GSN Subsystem" on page 130 once the hardware is installed .

Configure the Address Resolution Service

This section helps you determine which configuration procedures you need to follow to configure IRIX HARP when an LIS does not use HARP default parameters and behavior. If any of the following statements are true for your configuration, you must manually configure IRIX HARP:

• The SGI GSN port is directly connected to another host, not to a switch.

• One or more of the switches in the GSN fabric does not support broadcasting and the selected HARP server is not located at ULA 00:10:3b:ff:ff:e0.

• The IP/ST hosts attached to the GSN fabric belong to two or more LISs, and one (or more) of the switches in the GSN fabric do not support broadcasting. Said another way, the GSN fabric does not support broadcasting and some hosts have two or more GSN ports.

• One or more members of the LIS do not support dynamic HARP.

For each GSN connection/interface, use the decision tree shown in Figure 2-1 to discover which procedures are required for configuring the address resolution software.

What is the GSN port connected to?

- loopback device ——————————▶ Finished, no configuration required.
- point-to-point, direct connection to another member of LIS ——————————
- switch ——

Do all the switches in the GSN fabric support broadcasting?

Yes   No

Does the system havemultiple GSN interfaces?

No   Yes

For LIS #1   For all other LISs

Is LIS using standard ULA 00:10:3B:FF:FF:E0 for contacting HARP server?

Yes   No

Select one system to be HARP server. page 64

Discover ULA for the selected system. *hipcntl gsn# getmac*

Select one system to be HARP server. page 64

Configure each client with ULA for contacting selected HARP server. page 78

Attach selected server to switch port with ULA 00:10:3B:FF:FF:E0.

If there are any members of an LIS that do not support dynamic HARP, add these numbers to the HARP table. page 83

Finished, no configuration required. All clients use FF:FF:FF:FF:FF:FF to contact HARP service. Every system has an HARP server functioning.

Finished. All clients within this address space use 00:10:3B:FF:FF:E0 to contact HARP server. Only selected system runs HARP server program.

Finished. All clients within this address space use configured ULA to contact HARP server. Only selected system runs HARP server program.
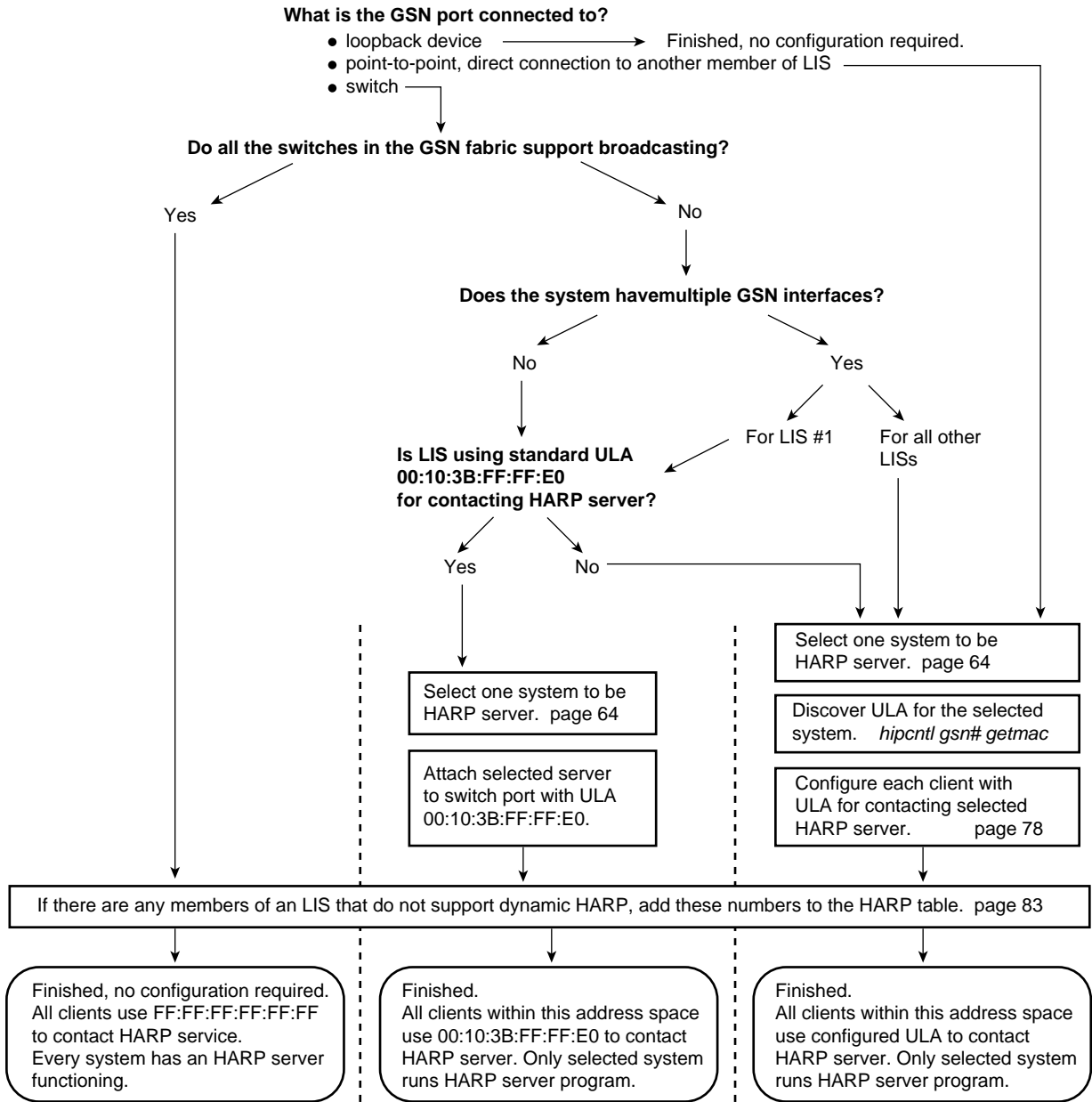
Figure 2-1   Decision Tree for HARP Configuration Procedures

Individual Configuration Tasks

This section provides detailed instructions for each individual configuration task. Most of these are optional. This is a reference section only; the tasks appear in alphabetical order, and are listed below. Most of these tasks are optional. See "Complete GSN Installation Process" on page 72 for the list of required configuration tasks.

- "Change HARP Lookup Table" on page 77
- "Configure IRIX GSN Network Interfaces In Real Time" on page 79
- "Configure IRIX HARP Client" on page 80
- "Configure IRIX HARP Server" on page 81
- "Edit gsnarp.options File" on page 83
- "Edit hosts File" on page 87
- "Edit ifconfig-#.options File" on page 87
- "Edit master.d/gsn File" on page 90
- "Edit netif.options File" on page 91
- "Enable Networking" on page 92

Note: For additional details about enabling IP networking software and configuring IP network interfaces, refer to *IRIX Admin:Networking and Mail*, which is available online through IRIS InSight.

Change HARP Lookup Table

The */usr/etc/gsnarp* command makes changes to the address resolution lookup table that is currently in memory for use by static HARP. This table maps 32-bit network-layer INET addresses (or network connection names) to 48-bit physical-layer Universal LAN Addresses (also known as MAC or Ethernet-style addresses). In addition, an entry may optionally specify a HIPPI-800 I-field address. Changes made with the *gsnarp* command are lost if the system or the GSN network interface is restarted; to make changes that survive restarts, follow the instructions in "Adding Static HARP Table Entries" on page 85.

- To add a static entry for a GSN host to the lookup table, use this command line:

  # **gsnarp** *name*   *ULA_value*

  where *name* is the network connection name or address for a logical network interface as listed in the */etc/hosts* file, and the *ULA_value* is a 6-byte (48-bit) value represented in hexadecimal characters separated by colons (for example, 7:8d:fe:8:13:5).

- To add a static entry for a HIPPI-800 host to the lookup table, use this command line:

  # **gsnarp** *name*   *ULA_value*   **0x***Ifield*

  where *name* is the network connection name or address for a logical network interface as listed in the */etc/hosts* file, the *ULA_value* is a 6-byte (48-bit) value represented in hexadecimal characters separated by colons (for example, 7:8d:fe:8:13:5), and the *Ifield* is a HIPPI-800 32-bit I-field value in hexadecimal notation (for example, 0x0700000A).

- To add a dynamic entry that will subsequently be treated like an entry that has been registered by the client, use this command line:

  # **gsnarp** *name*   *ULA_value*   [**0x***Ifield*] **temp**

  where *name* is the network connection name for a logical network interface as listed in the */etc/hosts* file, the *ULA_value* is a 6-byte (48-bit) value represented in hexadecimal characters separated by colons (for example, 7:8d:fe:8:13:5), and the optional *0xIfield* is a HIPPI-800 32-bit I-field value in hexadecimal notation (for example, 0x0700000A).

- To delete one entry from the table, use this command line:

  # **gsnarp -d** *name*

  where *name* is the network connection name or INET address for a logical network interface as listed in the */etc/hosts* file.

- To delete all the static entries from the lookup table, use this command line:

  # **gsnarp -d**

- To concatenate entries (load additional entries) from a file into the current lookup table, use this command line:

  # **gsnarp -f** *filename*

- To clear the current lookup table of all static entries, then load a new set of entries from the configuration file, use this command line:

  # **gsnarp -d -f**

  Note: This command line adds all the static entries currently in the *gsnarp.options* file. To add entries from a different file, use **gsnarp -d -f** *filename*.

Configure IRIX GSN Network Interfaces In Real Time

Dynamic configuration of the IRIX GSN network interfaces that use INET addresses is done with the */usr/etc/ifconfig* command, which is explained in detail in the ifconfig(1M) man page. The command lines listed below are appropriate with IRIX GSN:

```
# ifconfig  [gsn#] INET_addr
# ifconfig  [gsn#] netmask 0x########
# ifconfig  [gsn#] metric #
```

where *INET_address* is the 32-bit INET (IP) address, ######## is the 32-bit netmask value, and # is the routing metric. See the ifconfig(1M) man page for details about acceptable formats.

Note: Some of the standard *ifconfig* arguments are not supported for IRIX GSN (for example, broadcast and arp).

Configuration changes made in this manner do not persist across restarts of the system. To make configuration changes that persist, edit the configuration file as explained in "Edit ifconfig-#.options File" on page 87 and "Edit netif.options File" on page 91.

Configure IRIX HARP Client

This section describes how to configure each IRIX HARP client with one or both of the following:

Address for contacting client's primary HARP server
> This item must be configured when the GSN fabric does not support broadcasting and the client's HARP server is not located at the default ULA—00:10:3b:ff:ff:e0—for non-broadcasting fabrics. When the fabric supports broadcasting or when the default server ULA is used, no client configuration is required. For each GSN network connection, use the decision tree in Figure 2-1 to determine whether this configuration procedure is required.

Address for contacting alternate HARP server(s)
> Configuration of this item is optional in all configurations.

1. Open an */etc/config/gsnarp.options* file for editing.

2. For each IRIX GSN network interface on this client that requires configuration, add one line to configure the physical address that this client will use for contacting the primary HARP server. If the HARP server is an IRIX HARP system, this line must match the $server line placed in the server's own *gsnarp.options* file. This address must be the same on all the clients in the LIS. The entry must have one of these formats:

```
$server  gsn#  xx:xx:xx:xx:xx:xx          #server is a GSN system
$server  gsn#  xx:xx:xx:xx:xx:xx  0x########  #server is a HIPPI-800 system
```

where gsn# is the network interface, $xx:xx:xx:xx:xx:xx$ is the HARP server's 6-byte Universal LAN Address (also known as MAC address) in colon-separated hexadecimal notation, and 0x######## is a 32-bit I-field in hexadecimal notation. For example:

```
$server  gsn1  08:00:71:C5:AD:74
```

Note: Make entries in this file only for those specific network interfaces that need configuration. Use the decision tree in Figure 2-1 to determine whether this configuration is required.

3. Optional: add one or more alternate HARP servers by entering one line for each alternate server. Each line must be in one of these formats:

```
$alternate  gsn#  xx:xx:xx:xx:xx:xx             #a GSN system
$alternate  gsn#  xx:xx:xx:xx:xx:xx  0x########  #a HIPPI-800 system
```

For example, to add two backup servers for one network interface:

```
$alternate  gsn1  08:00:71:C5:AD:75
$alternate  gsn1  08:00:71:C5:3F:52
```

4. Optional: add static ("permanent") entries to the local HARP table. To do this, follow the procedure described in "Adding Static HARP Table Entries" on page 85.

5. To activate the changes, use the following command lines:

```
% su
Password: thepassword
# /usr/etc/ifconfig gsn# down
# /usr/etc/gsnarp -f
# /usr/etc/ifconfig gsn# up
```

where the # identifies the interface for which server address changes were made.

Configure IRIX HARP Server

This section describes how to configure one instance of IRIX HARP to function as an LIS' HARP server. This configuration task is only required in certain situations. Use the decision tree in Figure 2-1 to determine whether this task is required.

1. Verify that this system is appropriate for selection as the LIS' HARP server. See "Guidelines for Selecting a HARP Server" on page 65 for help in making this selection.

2. If the system already has its GSN hardware installed, use this command to discover the ULA that is assigned to this endpoint by its switch:

```
% gsncntl gsn# getmac
PROM ULA Address: 08:00:71:C5:AD:74
Device ULA Address: 00:01:3b:ff:00:0e
```

where gsn# identifies the GSN hardware that carries traffic for the server's LIS.

Note: Use the Device ULA value; this is the address assigned by the switch. The PROM ULA is the MAC address retrieved from the SGI GSN board.

Otherwise, discover the ULA that the switch will assign to this system, as explained in the switch manufacturer's documentation.

81

3. Open the */etc/config/gsnarp.options* file and add a line in this format. When IRIX HARP notices that the configured $server address matches its own address, it starts performing as the HARP server for its LIS. The line must have the following format:

```
$server gsn#  xx:xx:xx:xx:xx:xx
```

where gsn# is the network interface's name (for example, gsn0, gsn1, and so on) and *xx:xx:xx:xx:xx:xx* is this server's 6-byte ULA (in hexadecimal notation). Take note of this address; all HARP clients in the LIS must be configured to know this non-default address.

4. If the LIS has any members that do not support dynamic HARP, follow the instructions in the section "Adding Static HARP Table Entries" on page 85.

5. Use the following commands to shut down the GSN network interface, reconfigure IRIX HARP and the network interface, load any new entries into the current HARP table, then restart the network interface:

```
# ifconfig gsn# down
# gsnarp -d -f
# ifconfig gsn# up
```

where the # identifies the network interface being reconfigured.

Changing the HARP Database on the Fly

If IRIX HARP is already functioning and you want to make changes to the HARP database, invoke one of the following commands as superuser.

- Add one static entry

  ```
  # /usr/etc/gsnarp -s name  ULA_value  [Ifield]
  ```

  where *name* is the remote system's network connection name or INET address, *ULA_value* is the remote system's ULA address in colon-separated hexadecimal notation (for example, AA:12:CC:34:DD:56), and *Ifield* is the 32-bit I-field address in standard hexadecimal notation (for example, 0xAA12CC34). The *Ifield* is only required when the remote system is a HIPPI-800 system.

- Add one dynamic entry

  ```
  # /usr/etc/gsnarp -s name  ULA_value  [Ifield]  temp
  ```

  where *name* is the remote system's network connection name or INET address, *ULA_value* is the remote system's ULA address in colon-separated hexadecimal notation (for example, AA:12:CC:34:DD:56), and *Ifield* is the 32-bit I-field in standard hexadecimal notation (for example, 0xAA12CC34). The *Ifield* is required only when the remote system is a HIPPI-800 system.

- Add many static entries

  ```
  # /usr/etc/gsnarp -f /etc/config/filename
  ```

  where *filename* is a HARP configuration file. The *filename* can be the default file, *gsnarp.options,* or one of your choice.

- Load an entirely new table of static mappings into memory (and at the same time erase the old entries)

  ```
  # /usr/etc/gsnarp -d -f /etc/config/filename
  ```

Note: If a dynamic entry already exists for an INET address, the *gsnarp* command fails, the dynamic entry remains in the table, and the static entry is not added.

Edit *gsnarp.options* File

The */etc/config/gsnarp.options* file configures parameters for the IRIX HARP client and server. This file is optional; it does not have to exist for IRIX HARP to operate in a system with one GSN network interface. When this file is absent, or when the file does not contain entries for a particular network interface, IRIX HARP uses the HARP configurations recommended by the HARP standard. Use the decision tree in Figure 2-1 to determine if a network interface requires configuration in this file.

When present, the file can contain entries to accomplish one or more of the following configuration tasks for each IRIX GSN network interface:

- For an IRIX HARP client
  Configure the ULA for contacting the primary HARP service. When present, this entry replaces the IRIX HARP default address. This same configuration must be made on every IRIX HARP host in the LIS.

- For an IRIX HARP client
  Configure one or more ULAs for contacting alternate (backup) HARP servers.

- For an IRIX HARP server

  Add static entries to the HARP table to provide address resolution for hosts that do not support dynamic HARP. These entries must be added to the table on the LIS's HARP server and to the clients that do not support dynamic HARP; optionally, the entries can be added to HARP clients in the LIS.

Note: The *gsnarp.options* file is read automatically during system startup. The */etc/init.d/network* script calls */etc/init.d/network.gsn1* to process the HARP server configuration entries in this file and */etc/init.d/network.gsn2* to process the static HARP table entries in this file.

Adding Server Addresses for Client

Follow these instructions to configure each HARP client:

1. Open */etc/config/gsnarp.options*.

2. To configure the ULA for contacting the primary HARP server, enter a line similar to one of these. In addition to configuring the IRIX HARP client, this entry is also loaded into the local HARP table as a static entry.

   ```
   $server  gsn#  xx:xx:xx:xx:xx:xx            #server is a GSN system
   $server  gsn#  xx:xx:xx:xx:xx:xx  0x######## #server is a HIPPI-800 system
   ```

   where *gsn#* is a GSN network interface, $xx:xx:xx:xx:xx:xx$ is the server's 6-byte Universal LAN Address (ULA, also known as MAC address) in colon-separated hexadecimal notation, and 0x######## is the 32-bit I-field in hexadecimal notation that is needed for contacting a HIPPI-800 HARP server. For example:

   ```
   $server  gsn1  08:00:71:C5:AD:74
   ```

3. To add alternate HARP servers, enter one or more lines similar to one of these:

   ```
   $alternate  gsn#  xx:xx:xx:xx:xx:xx            #a GSN system
   $alternate  gsn#  xx:xx:xx:xx:xx:xx  0x######## #a HIPPI-800 system
   ```

   where *gsn#* is a GSN network interface, $xx:xx:xx:xx:xx:xx$ is the server's 6-byte Universal LAN Address (ULA, also known as MAC address) in colon-separated hexadecimal notation, and 0x######## is the 32-bit I-field in hexadecimal notation for contacting a HIPPI-800 HARP server.

   For example, to add two backup servers for one network interface:

   ```
   $alternate  gsn1  08:00:71:C5:AD:74
   $alternate  gsn1  08:00:71:C5:3F:52
   ```

Adding Static HARP Table Entries

The */etc/config/gsnarp.options* file may also contain static HARP entries. Each entry maps a network connection name (or INET address) to a Universal LAN MAC address (ULA) for a host that does not support dynamic HARP. The ULA is for any type of HIPPI-based system (GSN/HIPPI-6400 or HIPPI-800). For a HIPPI-800 host (reachable via a GSN/HIPPI-800 bridge), an I-field is required in addition to the ULA.

If the LIS has any members that do not support dynamic HARP, a static HARP table entry for each such host must exist in the HARP table configuration file of the LIS' HARP server and on each host that does not support dynamic HARP. These entries do not need to exist on the dynamic HARP clients.

Follow these instructions to configure static entries for the HARP table:

4. For each host that needs a static HARP entry, add a line similar to one of these:

    ```
    name xx:xx:xx:xx:xx:xx    #host is a GSN system
    name xx:xx:xx:xx:xx:xx 0x########  #host is a HIPPI-800 system
    ```

    where *name* is the remote host's network connection name or INET address as listed in the */etc/hosts* file, *xx*:*xx*:*xx*:*xx*:*xx*:*xx* is the remote endpoint's 6-byte Universal LAN Address (also known as MAC address) in colon-separated hexadecimal notation, and 0x######## is a 32-bit I-field in hexadecimal notation. For example:

    ```
    amazon-gsn0     08:00:56:78:9a:bc
    223.209.1.18    08:00:70:9B:FF:8E
    hip-nile        08:00:56:78:7c:92    0x07000001
    ```

    The IRIX HARP software does not check or verify these values. It is the system administrator's responsibility to ensure that each entry is both valid and correct. The ULA value must be the exact ULA (IEEE address) for use as the Destination ULA in HIPPI-6400-PH or HIPPI-800-LE headers and in the MAC Header of HARP packets (either HIPPI-800 format or HIPPI-6400 format). The I-field, if present, must be the exact value used as the Switch Address in HIPPI-800-LE Headers and HIPPI-800 formatted HARP packets.

    Note: Do not create an entry in this format for any system for which a `$server` or `$alternate` entry already exists in this file.

Loading the New Configuration

Follow these instructions to reconfigure IRIX HARP with the new parameters and data in the configuration file:

5. Save the file.

6. Use the *ifconfig* command to shut down each GSN network interface for which a change has occurred in a $server or $alternate entry:

   # **ifconfig** *gsn*# **down**

   where the *gsn*# matches the changed or new entries in the *gsnarp.options* file.

7. Use the following command to invoke *gsnarp*. This flushes the current HARP table, reconfigures IRIX HARP for any disabled network interfaces, and loads the entries from the *gsnarp.options* file into the new HARP table:

   # **gsnarp -d -f**

   Note: If a dynamic entry already exists for an INET address that was added as a static entry, the *gsnarp* command fails, the dynamic entry remains in the database, and the static entry is not added to the table. Wait for the dynamic entry to age and disappear from the table, then try again.

8. Use the *ifconfig* command to start each GSN network interface that was disabled:

   # **ifconfig** *gsn*# **up**

Edit *hosts* File

The */etc/hosts* file maps *network connection name*s[1] (commonly referred to as hosts) to *INET address*es. Each time the IP-over-GSN and/or ST-over-GSN driver starts (for example, *ifconfig gsn# up*), it uses information from this file to configure the interfaces. There must be one entry in the *hosts* file for each local IRIX GSN connection; this statement is true even in environments that are using an NIS or DNS server on an attached LAN. The local *hosts* file must exist so that the network interface(s) can be configured during system startup, before the NIS service is accessible. Each address in this file must also exist in the */etc/config/netif.options* file; the strings (names) in the two files must be identical.

The entries must be similar to the example below, which illustrates four IRIX GSN interfaces for a system whose hostname is *amazon*:

```
223.209.1.2     gsn0-amazon.brazil.com     gsn1-amazon
223.209.2.4     gsn1-amazon.brazil.com     gsn2-amazon
223.209.3.16    gsn2-amazon.brazil.com     gsn3-amazon
223.209.4.32    gsn3-amazon.brazil.com     gsn4-amazon
```

Note: For systems that have a primary interface served by an NIS or DNS server, this information must also be added to that server's database.

Edit *ifconfig-#.options* File

Each */etc/config/ifconfig-#.options* file configures one IP network interface. The # in the filename matches the numeral in the `if#name` entry in the *netif.options* file. Table 2-1 lists the operational parameters that can be controlled with this file. Each instance of this file is optional; when a file does not exist for a specific network interface, the default values are used (as listed in Table 2-1).

For IRIX GSN, the parameters that are most commonly configured are netmask and local buffer areas. For a complete description of the IP parameters that can be configured in this file, see the *ifconfig* man page.

---

[1] A network connection name is associated with an IP address (by an entry in the */etc/hosts* file) and with an IP network interface (by an entry in the */etc/config/netif.options* file). For example, for the entries "223.209.1.2 gsn1-amazon; if1name=gsn0 and if1addr=gsn-amazon," 223.209.1.2 is the IP address, *gsn1-amazon* is the network connection name, and *gsn0* is the IP network interface name. Due to UNIX convention, the hostname (in the */etc/sys_id* file) for this machine is probably *amazon*.

Configuring Netmask / LIS Address Space

If your site is using variable-length INET addressing or is dividing its INET network address space into subnets, place a line like this in the *ifconfig-#.options* file. The *netmask* entry determines the number of separate address spaces (LISs) possible within the INET address space and the number of hosts possible within each LIS.

```
netmask 0xyour_netmask
```

where *your_netmask* is a 32-bit value in hexadecimal notation in which each local network bit is a 1 and each local host bit is a 0. For example, FFFFFFE0 subdivides a Class-C INET network address into 8 subnets (LISs) with up to 30 hosts each by dividing the final byte (the 8 bits of host portion) into 3 bits of additional network address and 5 bits of host address (E0=1110 0000 binary).

Hint: From the 32 values possible with a 5-bit local host portion, 00000 and 11111 are not usable, leaving 30 local host addresses available.

Configuring TCP Local Buffer Areas

To obtain optimal TCP/IP performance on an IRIX GSN interface, the size of the local buffers for handling outstanding/in_transit TCP/IP data must match the sizes used on the other (remote) systems. If possible, the IRIX GSN default value (524288 bytes) should be configured on all the GSN interfaces within the GSN fabric. If this default cannot be used throughout the fabric, you must create an *ifconfig-#.options* file for each IRIX GSN interface and set the local buffering (sspace) value to the value selected for the other systems within the GSN fabric.

To configure the size of the TCP local buffer areas (in bytes), create an *ifconfig-#.options* file for the IRIX GSN interface and place these lines in the file:

```
sspace nnnn
rspace nnnn
```

where *nnnn* is any value, divisible by 1024, between 1024 and 524288.

Keep the following in mind:

- If the memory used by TCP/IP applications is an issue, you can obtain nearly full performance by using 262144 (256*1024), instead of the default 524288 (512*1024).

- If rspace is set to a value significantly smaller than the value used for sspace, TCP acknowledgments (ACKs) can be delayed. This can have a negative effect on performance.

- For large TCP windows to work, the tcp_winscale and tcp_tsecho variables in the */var/sysgen/master.d/bsd* file must be set to 1 (their defaults).

Summary of Network Interface Operational Parameters and Default Settings

Table 2-1 lists the operational parameters that can be controlled with each */etc/config/ifconfig-#.options* file and the default values that are used for each parameter when the file does not exist or when there is no entry in the file for that item.

Table 2-1 IP Network Interface Operational Parameters

Parameter	Default Setting for GSN Interfaces	Description
Netmask	The mask, appropriate for the INET address' Class, that does not extend the network portion or reduce the host portion of the address.	Value used by system to know which bits of the INET address are used locally to identify hosts and which bits identify local subnets (that is, LISs).
Broadcast address	off	Value used by system for broadcasting.
ARP	off	Enables/disables address resolution (ARP).
Routing metric	0	Number of hops added to the hop count for this interface. The higher the number, the less likely the interface will be selected as a route by the routing module.
sspace	524288	Value used by the transmitting TCP/IP module for size of buffering for transmitted but outstanding (unacknowledged) data for a specific GSN connection.

Table 2-1 (continued)   IP Network Interface Operational Parameters

Parameter	Default Setting for GSN Interfaces	Description
rspace	524288	Value used by the receiving TCP/IP module for size of buffering for incoming data that is not yet passed to the application for a specific GSN connection.
Automatic startup	up	When the parameter is set, each system restart configures and starts this interface.

Edit *master.d/gsn* File

The */var/sysgen/master.d/gsn* file configures the IRIX GSN hardware device driver and the hardware. This configuration is optional because all parameters have default settings that are considered optimal for most sites. The settings in this file affect all SGI GSN boards installed in the system.

The specific items that are configurable vary from release to release, so they are explained fully within the file. Here are a few of the more important items:

- Size of maximum transmission unit. `if_gsn_mtu`: valid values are 0=65,280 which is the GSN default, or 1 to (2^{32} minus 1) inclusive.

- Enable/disable onboard (hardware) IP checksumming for reception, for transmission, for both, or for neither. `if_gsn_cksum`: valid entries are 0=disabled, 1=receive_only, 2=transmit_only, and 3= both.

- Operate GSN board in onboard loopback mode. `gsn_use_loopback=1` enables internal loopback and `gsn_use_loopback=0` is for normal operational.

- Number of small buffers (2 KB each buffer) passed by the driver to *each* IP reception entity on the GSN board. Note that one IP reception entity is associated with each activated interrupt queue (QID). `if_gsn_small_mbuf_entries`: valid values are 0 to 8,192 inclusive.

- Number of large buffers (16 KB each buffer) passed by the driver to *each* IP reception entity on the GSN board. Note that one IP reception entity is associated with each activated interrupt queue (QID). `if_gsn_large_mbuf_entries`: valid values are 0 to 1024 inclusive.

- Number of 5 microsecond increments (ticks) for loading the interrupt holdoff timer. While this timer is counting, the hardware does not generate interrupts to the driver. A value of 40 means that interrupts are not generated any faster than one every 200 microseconds. Adjusting this parameter affects latency and throughput. The optimal value depends on site-specific network traffic patterns and application performance requirements. `gsn_intr_holdoff_ticks`: valid values are 0 to (2^{32} minus 1) inclusive.

- Size of queue for Admin micropackets awaiting transmission. `gsn_admin_desc_entries`: valid values are 2, 4, 8, 16, and 32.

- Number of unanswered HARP registration packets that results in the client making the alternate HARP server into the primary one. `has_hysteresis`: valid entries are 1 to (2^{32} minus 1) inclusive.

After editing this file, follow the instructions in "Building a New Driver Into the Operating System" on page 93 to start using the new configuration.

Edit *netif.options* File

The */etc/config/netif.options* file maps local *network connection name*s (or IP addresses) to IRIX GSN *network interface name*s (for example, *gsn0*, *gsn1*, and so on), and sets the maximum number of network interfaces for the system. Each time the IP-over-GSN and/or ST-over-GSN driver starts (for example, *ifconfig gsn# up*), it uses information from this file to configure its interfaces. There must be a two-line entry for each IRIX GSN network interface. Each address (if#addr entry) in this file must also exist in the */etc/hosts* file; the strings in the two files must be identical.

The first entry in the *netif.options* file (that is, the pair of lines: `if1name` and `if1addr`) defines the primary interface. In most situations, the primary interface should be Ethernet or FDDI; however, when the GSN fabric supports broadcasting and the LISs on the fabric are using dynamic client/server HARP, GSN can be configured as the primary interface. Any system that functions as a client or server for NFS, NIS, or other major client/server IP program should configure the network interface over which the client/server program functions occur as the primary network interface.

Note: Systems that function as a client or server for *bootp* must configure Ethernet as their primary network interface. The IRIX GSN driver is not included in the miniroot.

The example below illustrates a system with two IRIX GSN network interfaces (*gsn#*), an FDDI interface (*xpi0*), and a primary Ethernet interface (*ef0*). If this system's hostname is *amazon*, these IRIX GSN entries work with the examples of */etc/hosts* file entries used in the section "Edit hosts File." A line that starts with a colon (:) is a comment.

```
if1name=ef0
if1addr=$HOSTNAME

if2name=xpi0
if2addr=fddi-$HOSTNAME

: configuration associated with hardware device /hw/gsn/0
if3name=gsn0
if3addr=gsn0-$HOSTNAME

: configuration associated with hardware device /hw/gsn/1
if4name=gsn1
if4addr=gsn1-$HOSTNAME
```

Note: The use of the $HOSTNAME variable assumes that the system's hostname has been defined in the */etc/sys_id* file.

IRIX, by default, allows up to eight logical network interfaces of any type to be configured. To increase this maximum, edit the `if_num` line in the *netif.options* file as illustrated below. Change the default entry:

```
: if_num=8
```

to a numeral equal to the number (decimal format) of logical IP network interfaces that will be configured on this system. For example:

```
if_num=12
```

Enable Networking

To automatically enable the IP/ST network stacks each time the system is started, edit the */etc/config/network* file so that it contains the single word ON or on. If the file is missing, add the file, or invoke the command-line utility **chkconfig network on**.

Note: Enabling networking does not result in IP-over-GSN functionality; it only enables the IP/ST software within the operating system to operate over whatever drivers are available to service it. See "Edit netif.options File" for instructions that associate a network device (that is, its network interface) with the IP/ST stacks.

Building a New Driver Into the Operating System

This section describes how to rebuild the operating system to include a totally new driver or to include configuration changes to a driver that is already present. In either case, complete all the configuration steps listed in the "Complete GSN Installation Process" before rebuilding the operating system.

For the IRIX GSN subsystem to be functional, the IRIX operating system (kernel) that is currently running the system must be rebuilt (after the configuration) to include the new or reconfigured IRIX GSN driver. When changes are made to either of the following files, or when new IRIX GSN software is installed, it is necessary to rebuild the operating system:

- */var/sysgen/master.d/gsn*

- */var/sysgen/system/gsn.sm*

Note: After any configuration change, it is advisable to verify that the subsystem is functional. Follow the instructions in "Verifying the IRIX GSN Subsystem" on page 130 to verify the functionality of the GSN subsystem.

Each set of instructions below builds a new operating system and starts it running. It is not important which set of instructions you use. Note that in the second set, you shut the system down twice.

Instruction Set 1

```
% su
Password: thepassword
# /etc/init.d/autoconfig
Automatically reconfigure the operating system (y or n)? y
# /etc/reboot
.....<various messages are displayed on console>...
configuring gsn0 as hostname
configuring gsn1 as hostname
```

Instruction Set 2

```
% su
Password: thepassword
# /etc/shutdown
```

93

After the system shuts down, restart it by turning/pressing the reset/restart key or button. When the following question is displayed, answer with **yes** or **y**.

```
Automatically reconfigure the operating system (y or n)? y
<log on>

% su
Password: thepassword
# /etc/reboot
```

After the system shuts down, again restart it by turning/pressing its reset/restart key or button. When the above question is displayed, answer **no** or **n**.

List of All Configurable Parameters for IRIX GSN

Table 2-2 lists all the parameters that can be configured for the IRIX GSN product.

Table 2-2   Summary of Configurable Items for IRIX GSN

Parameter	Required (R)/ Optional (O)	Default Setting	Location of Configuration Instructions	How to Start Using the New Configuration
IP parameters:				
assign #s to GSN network interfaces	O	see page 67	page 69	*init 0* or *shutdown* or *reboot*
assign netmask	O	0 for each Class-defined host bit and 1 for each netid bit	page 87	*ifconfig gsn# down*, then *ifconfig gsn# up*
map name (alias) to IP address	R	none	page 87	same as above
assign IP/INET address	R	none	page 87	same as above
map address/name to *gsn#*	R	none	page 91	same as above
enable IP networking	R	disabled	page 92	same as above
enable broadcast	O	automatic, if needed	none	same as above
ARP (i.e., HARP)	O	see HARP entries in this table	page 77	
maximum MTU size	O	65,280 bytes	page 90	*autoconfig*, then *reboot*

Table 2-2 (continued)   Summary of Configurable Items for IRIX GSN

Parameter	Required (R)/ Optional (O)	Default Setting	Location of Configuration Instructions	How to Start Using the New Configuration
# of IP reception entities/queues	O	4	page 90	same as above
# buffers for IP reception	O	50 small and 50 large	page 90	same as above
Hardware parameters:				
ULA address	NA with switch[a]	switch provides addr	instructions for switch	each time link is activated
	NA for pt-to-pt	read from PROM	none	with each restart
set hold-off timer for interrupts to CPU	O		page 90	*autoconfig*, then *reboot*
# buffers for IP reception	O	50 small and 50 large	page 90	*autoconfig*, then *reboot*
assign unit #s to GSN boards	O	usually, hardware #s match network interface #s	page 67	*init 0* or *shutdown* or *reboot*
HARP parameters:				
HARP server	O for fabric with broadcasting	distributed dynamic HARP	page 81	
	O for non-broadcast fabric	assumes that server's ULA= 00:10:3b:ff:ff:e0	Figure 2-1 and page 81	*ifconfig gsn# down*, *gsnarp -f -S*, then *ifconfig gsn# up*
	R if LIS contains any hosts that do not support dynamic HARP	dynamic HARP	page 85	
	O for other server parameters	assumes defaults in */var/sysgen/master.d/gsn* file are used	page 90	*autoconfig*, then *reboot*

Table 2-2 (continued)   Summary of Configurable Items for IRIX GSN

Parameter	Required (R)/ Optional (O)	Default Setting	Location of Configuration Instructions	How to Start Using the New Configuration
HARP client	O	FF:FF:FF:FF:FF:FF to locate HARP service on broadcasting fabric; 00:10:3b:ff:ff:e0 to locate HARP server on non-broadcasting fabric	page 80	
	R if defaults not used	see "HARP client" cell (above)	page 80	*ifconfig gsn# down*, *gsnarp -f -S*, then *ifconfig gsn# up*
HARP table	O	dynamic HARP		
	R only when LIS contains clients that do not support dynamic HARP	none	page 85	*gsnarp -f*

a. NA stands for not applicable.

Maintaining, Monitoring, Verifying, and Troubleshooting IRIX GSN

This chapter describes how to maintain, monitor, verify, and troubleshoot the IRIX GSN subsystem.

Commands Available for IRIX GSN

IRIX GSN can be monitored and maintained with the commands summarized in Table 3-1.

Table 3-1   Utilities for Monitoring and Maintaining IRIX GSN

Command	Function	Page
/usr/etc/gsnarp	Configures ULA(s) for contacting interface's HARP server(s).	80
	Adds and deletes static entries to the HIPPI address resolution (HARP) lookup table (in kernel-resident memory). The HARP table maps ULAs to INET (IP/ST) addresses.	82
	Configures non-default address for contacting HARP server.	84
	Displays non-default HARP servers that are configured, and the content of the HARP table with status flags for entries.	127
/usr/etc/gsncntl	Provides control and status functions for the hardware and driver portions of the IRIX GSN subsystem, including display of GSN port's ULA.	100, 101
/usr/etc/gsntest	Verifies the functionality of the SGI GSN (HIPPI-6400) hardware, without using the IP or ST network interfaces.	131
/usr/etc/ttcp	Verifies the functionality of IP over any physical-layer subsystem.	133
/usr/etc/ping	Verifies the functionality of IRIX IP network interfaces. Can be used to verify that a *gsn#* IP network interface is functioning.	133

Table 3-1 (continued)   Utilities for Monitoring and Maintaining IRIX GSN

Command	Function	Page
/etc/init.d/gsn	Reconfigures all IRIX GSN network interfaces with HARP server information in *gsnarp.options* file. This script calls *ifconfig* and *gsnarp* commands.	126
/usr/etc/ifconfig	Configures standard IP stack options for IRIX GSN network interfaces (that is, *ifconfig gsn#*). Each *ifconfig up* of an IRIX GSN network interface, resets the SGI GSN hardware.	79
/usr/etc/netstat	Displays network parameters (including INET and ULA/MAC addresses), traffic statistics, and status information for IP-over-GSN and ST-over-GSN. When an IRIX GSN IP/ST network interface (*gsn#*) is not configured, the disabled interface is listed, but without an INET address.	133,136

Instructions for Common Procedures

This section describes some procedures that are commonly used to monitor and maintain the IRIX GSN subsystem. All of the IRIX GSN utilities (*gsnmap*, *gsncntl*, and *gsntest*) require the user to have superuser (root) privileges. Table 3-2 lists the procedures:

Table 3-2   Common Procedures for Monitoring and Maintaining IRIX GSN

Procedure	Utility	Page With Instructions
Hardware:		
Check status of hardware	gsncntl	101
Configure IRIX GSN to loopback through the board	none	100
Display ULA/MAC address for GSN hardware	gsncntl; netstat	100
Reset hardware (and network interface)	ifconfig	99
Verify that software has located GSN hardware	hinv	130
Verify/test GSN hardware	gsntest	131

Table 3-2 (continued)   Common Procedures for Monitoring and Maintaining IRIX GSN

Procedure	Utility	Page With Instructions
IP/ST Software:		
Enable/disable IP protocol stack	chkconfig	92
Enable/disable each IP/ST-over-GSN network interface	ifconfig	99
Check status of IP traffic/interface	netstat -p ip	125
Check status of ST traffic/interface	netstat -p stp	125
Verify/test IP-over-GSN interface	ping	133
HARP/ARP Software:		
Change address for contacting primary server	gsnarp -S	80
Change address for alternate (backup) HARP server(s)	gsnarp -A	80
Add/delete static entries to HARP table	gsnarp	85
Display current HARP server(s)	gsnarp -S	127
Check status of HARP table	gsnarp -a	127

Disable or Enable IP/ST Interface

To enable/disable the network interface to an IRIX GSN port that service the IP protocol stack and the ST protocol stack, use the standard */usr/etc/ifconfig* command:

```
# ifconfig  [gsn#] down
# ifconfig  [gsn#] up
```

Note: This sequence of commands also resets the XTALK interface(s) to the SGI GSN hardware, which results in a warm reset of the hardware.

Display ULA (MAC) Address

To display the ULA (MAC) address for the SGI GSN hardware, invoke this command:

```
% gsncntl [gsn#]  getmac
PROM ULA Address:    00:01:3b:ff:00:0e
Device ULA Address: 00:01:3b:ff:00:0e
```

This command displays both the ULA (MAC) address that is stored in (read from) the GSN board's PROM and the ULA that is currently being used for GSN communications, which is either the address assigned from the attached GSN switch or the address read from PROM. The IRIX GSN subsystem uses the "Switch ULA" for all communications.

You can also use the *netstat* command, which displays only the ULA that is currently being used by the GSN subsystem; *netstat* does not read from PROM:

```
% netstat -ina
gsn0   16256   <INET netid>    <complete INET address>  ...
                                <ULA/MAC address>
```

For example:

```
% netstat -ina
gsn0   16256   192.0.113    192.0.113.1          ...
                            00:01:3b:ff:00:0e
```

Configure the SGI GSN Board for On-board (Internal) Loopback Operation

To configure the SGI GSN board to loop outgoing traffic through the SHAC ASIC and back into the same system (host) that transmitted it, use the following procedure. Operating the board in this mode tests the software (that is, the host protocol stacks, the driver, and the firmware on the board) associated with the IRIX GSN subsystem as well as the SHAC ASIC on the board. This mode of operation does not utilize the GSN hardware (that is, the SuMAC ASIC and the panel plate receptacle).

1. Edit the SGI GSN board's configuration file, as described in "Edit master.d/gsn File" on page 90.

    Change this default entry: `gsn_use_loopback = 0`

    to this: `gsn_use_loopback = 1`

2. Use *ifconfig* to disable then enable (reset) the GSN interface:

    ```
    % ifconfig gsn# down
    % ifconfig gsn# up
    ```

To return the board to normal operation, change this variable back to its default setting, then disable and enable the interface.

Note: In this board-loopback mode, the functionality of the entire SGI GSN board cannot be verified. Use an external loopback connector for complete hardware verification.

Check Status of Hardware

There are utilities for checking the status of the following functions:

- verify that hardware was located during startup, page 102

- firmware version currently running on hardware, page 102

- ULA (MAC address) currently being used by network interface, page 100

- state of GSN physical link, page 102

- operational statistics for the GSN hardware, including SHAC ASIC, page 104

- XIO slot errors, page 105

- ST traffic errors, VC# receive and transmit status in Table 3-6 on page 106

- status for GSN element, page 113

- HIPPI-6400 ADMIN traffic, page 122

Verifying That GSN Hardware Was Found

To verify that the operating system located the SGI GSN hardware during startup, use this command:

```
# /bin/hinv -d gsn
<display for single-board product>
GSN 1-XIO adapter: unit #, in module # I/O slot #
<display for two-board product>
GSN 2-XIO adapter: unit #,
        XIO port 1 in module # I/O slot #
        XIO port 2 in module # I/O slot #
```

Displaying Firmware Version

To display the version of firmware that is currently running on SGI GSN hardware, use this command:

```
# gsncntl [gsn#] versions
```

where gsn# identifies the SGI GSN board for which you want information.

Displaying ULA (MAC) Address

See "Display ULA (MAC) Address" on page 100.

Displaying Status of Physical Link

To display the status of the SGI GSN board and the physical link attached to it, use the command below. The status information is described in Table 3-3.

```
# gsncntl gsn#  status  device
```

Table 3-3   Status Information for GSN Physical Link

Status Item	Normal Value	Description
State:	LNK_RDY	The GSN states are sequential. They are listed here in the order in which they occur during startup.
NULL		The driver has reset the GSN hardware but has not yet been able to configure it. This state is normal for a few seconds during startup.
CFG		The IRIX GSN driver has successfully configured the GSN hardware, but has not yet made contact with the SuMAC ASIC. This state is normal for a few seconds during startup.
SUMAC_POLL		The driver has started communication with the HIPPI-6400 (SuMAC) ASIC on the GSN board; HIPPI-6400 ADMIN micropackets are being exchanged. This state is normal for a few seconds during startup.
EX_ELEM		The local GSN subsystem has transmitted an EXCHANGE_ELEMENT_FUNCTION Admin micropacket and received a reply from the local SuMAC; the local GSN subsystem is attempting to contact the system at the other end of its physical link.This state is normal for a few seconds during startup; if it persists, there may be a problem with the remote GSN system or the physical link.
ULA_REQ		The local GSN subsystem has completed the EXCHANGE_ELEMENT_FUNCTION with its neighbor, has requested a ULA from the attached switch, and is waiting for a reply. This state is normal for a few seconds during startup; if it persists, there may be a problem with the attached switch.
LNK_RDY	LNK_RDY	The local GSN subsystem (driver and hardware) is ready to operate. This is the normal operational state.
Flags: LNK_SWITCH	LNK_SWITCH	The local GSN subsystem believes that its neighbor element is a GSN switch. That is, the physical link (cable) is attached to a GSN switch.

Table 3-3 (continued)   Status Information for GSN Physical Link

Status Item	Normal Value	Description
LNK_P-2-P		The local GSN subsystem believes that its neighbor element is another GSN endpoint (not a switch). That is, the physical link (cable) is attached to a HIPPI-6400 element, but is *not* attached to a HIPPI-6400 (GSN) switch.
LNK_EXT_LOOP		The SGI GSN board is operating with a loopback connector/cable installed.LNK_P_2_P indicates that the physical link (cable) attached to another GSN endpoint (not a switch).
LNK_INT_LOOP		The SGI GSN board is configured to operate in board-loopback mode.

Displaying Status of Local GSN Hardware

To display status information for an SGI GSN board, use either of these commands. The displayed information is described in Table 3-4. Most of the counted items are initialized to zero upon reset of the board and roll over to zero upon reaching 2^{32} (that is, at 4,294,967,295); exceptions are explained in the table.

```
# gsncntl [gsn#] status
```

or

```
# gsncntl -v [gsn#] status
<verbose mode expands general error counts into specific errors>
```

where gsn# identifies the SGI GSN board for which you want information

.

Table 3-4   GSN Status Information for Copper-Based XIO Hardware

Status Item	Description
Link state and flags	See Table 3-3.
ADMIN Packet Count and Errors	See Table 3-9.
SHAC Status	See Table 3-5 and Table 3-6. Display shows errors collected by the SHAC ASIC on the main SGI GSN board.
Hop Status	See Table 3-7. Display shows status for local HIPPI-6400-PH element (hop 0, SuMAC), as well as the remote (hop 1) link-end element.

Displaying Status of XIO and Network-layer Processing (SHAC ASIC)

To display status information for the SHAC ASIC, use this command. (The SHAC ASIC is the component on the main SGI GSN board that processes XIO [that is, XTALK] and network-layer traffic.) Table 3-5 describes the basic display; Table 3-6 describes the verbose display that occurs when the **–v** option is used.

```
# gsncntl [gsn#] status shac
```

or

```
# gsncntl -v [gsn#] status shac
<verbose mode expands general error counts into specific errors>
```

Table 3-5   SHAC ASIC Status: Basic Listing

SHAC Status Item	Normal Value	Description
Error interrupt status	OK	A count of the total number of error interrupts generated by the SHAC ASIC.
		Use the verbose option (**–v**) to list the specific errors.
QIDs with interrupts pending	None	A count of the total number of system interrupt queues (QIDs) that currently have interrupts (generated by the SHAC ASIC) awaiting processing by the operating system.
		Use the verbose option (**–v**) to list the specific QIDs that have interrupts pending.
VC# receive error status	OK	A count of the total number of errors detected by the SHAC ASIC's receive logic on the identified GSN virtual channel (VC#).
		Use the verbose option (**–v**) to list the specific errors.
Transmit error status	OK	A count of the total number of errors detected by the SHAC ASIC's transmit logic.
		Use the verbose option (**–v**) to list the specific errors.

Table 3-6   SHAC ASIC Status: Verbose Listing

SHAC Status Item	Normal Value	Description
Error interrupt status	OK	Error interrupts generated by the SHAC ASIC.
Xtalk0 error	0	An error occurred on the main GSN board's XIO slot (that is, XTALK interface 0).
Xtalk1 error	0	An error occurred on the additional GSN board's XIO slot (that is, XTALK interface 1 located on the XTOWN board).
MAC error	0	An error occurred on the SuMAC ASIC.
IC error	0	An error was detected by the SHAC port that connects to the SuMAC ASIC.
SSRAM parity error	0	An SSRAM parity error was detected.
Receive VC# error	0	An error was detected by the receive logic for the indicated VC (0, 1, 2, or 3).

Table 3-6 (continued)   SHAC ASIC Status: Verbose Listing

SHAC Status Item	Normal Value	Description
Transmit error	0	An error was detected by the transmit logic.
Synthetic port error	0	An error occurred during a read or write of the area in SSRAM where information is kept for IP processing and hardware interrupts to the driver.
Total I desc. oflow	0	SHAC cannot keep up with the interrupts being generated (by all types of traffic). Specifically, one or more QIDs overflowed.
ST I desc. oflow	0	SHAC cannot keep up with the ST traffic occurring on at least one of its QIDs. Specifically, the interrupts being generated by the ST traffic to a specific QID overflowed the queue.
Total I desc. uflow	0	The counts of unprocessed interrupts on one or more QIDs kept by the GSN driver and the SHAC ASIC are out of sync with each other. SHAC's count has fewer unprocessed interrupts than the driver's count.
ST I desc. uflow	0	The counts of unprocessed ST interrupts on one or more QIDs kept by the GSN driver and the SHAC ASIC are out of sync with each other. SHAC's count has fewer unprocessed ST interrupts than the driver's count.
DMA engine error	0	The on-board DMA logic has been halted because an error was detected on a XTALK interface or the SSRAM interface. This is a secondary error indication.
QIDs with interrupts pending	None	Identifies the QIDs that currently have interrupts awaiting processing. 1=one or more interrupts are pending on this QID; 0=no interrupts pending on this QID.
QID 0	0	
QID 1	0	
QID 2	0	
QID 3	0	
QID 4	0	
QID 5	0	
QID 6	0	

Table 3-6 (continued)   SHAC ASIC Status: Verbose Listing

SHAC Status Item	Normal Value	Description
QID 7	0	
VC# receive error status	OK	Errors detected by the SHAC ASIC's receive logic. VC# identifies the GSN virtual channel (0, 1, 2, and 3).
Sideband error	0	An incoming HIPPI-6400 micropacket had its ERROR bit set.
Underrun error	0	A TAIL bit was received unexpectedly. That is, an incoming HIPPI-6400 micropacket had its TAIL bit set before the receive logic received the number of micropackets implied by message's M_len field.
Overrun error	0	A TAIL bit was not received when expected. That is, the TAIL indicator was not set in the final HIPPI-6400 micropacket for the message, as calculated from the message's M_len field.
Fatal error	0	The SHAC ASIC's connection to the SuMAC ASIC detected a data or control parity error, a Data Valid error, or an external Link or Warm Reset. The RX logic cannot recover from these conditions. When one occurs, SHAC must be reset. All incoming data is discarded and all pending DMAs are aborted.
SSRAM error	0	The receive logic detected an SSRAM parity error.
STC invalid port error	0	The ST header on an incoming Control Operation contained an invalid entry: for example, an incorrect destination port or an invalid key.
STC invalid key error	0	The ST header on an incoming Control Operation contained an invalid key for the destination port.
STD invalid port error	0	The ST header on an incoming Data Operation contained an invalid destination port entry.
STD invalid BID error	0	The ST header on an incoming Data Operation contained an invalid Mx parameter. Either the Mx parameter itself was invalid (for example, greater than 4095), or the local information for the specified Mx parameter is not valid.
STD invalid key error	0	The ST header on an incoming Data Operation contained an invalid key for the destination Mx.

Table 3-6 (continued)   SHAC ASIC Status: Verbose Listing

SHAC Status Item	Normal Value	Description
STD invalid BUFX error	0	The ST header on an incoming Data Operation contained an invalid BUFX value. Either the BUFX value specified an unsupported striping feature or the local information for the specified BUFX is invalid.
STD BUFX range error	0	The *Bufx* in the Schedule Header of an ST Data Operation does not fall in the range allowed for the B_id as specified by the *Base_Bufx* and *Bufx_Range* fields in the B_ID table entry. Any of the following conditions result in this error: • ST Schedule Header's *Bufx*[31:29] does not equal *Base_Bufx_31_29* in the B_ID table. • ST Schedule Header *Bufx*[19:0] is less than *Base_Bufx_19_0* from the B_ID table entry. • ST Schedule Header *Bufx*[19:0] is greater than the sum of *Base_Bufx_19_0* and *Bufx_Range* from the B_ID table entry.
STD offset error	0	An ST Data Operation included incorrect offset of data length values. The *Offset* (from its Schedule Header) plus the length of the DMA portion of the STU is greater than the destination buffer size: $Offset + (M\_len - 0x30) > (2**Bufsize)$. This error also occurs when *Offset*[6:0] is non-zero when *Bufx*[31:29] is non-zero; this is a violation of the alignment restriction for memory striping.
STD STU num error	0	The STU sequence numbering for an ST Data Operation is not sequential with respect to the previous STU for the same B_id. This error can also occur when the Poisoned (P) bit in the B_ID table entry is set.
STD read timeout	0	During memory striping or dual-path operation, a readback timeout occurred on one of the XIO ports.
SRC port 0 oflow	0	SHAC dropped at least one ST Control Operation due to an overflow on its queue for Port 0.

Table 3-6 (continued)   SHAC ASIC Status: Verbose Listing

SHAC Status Item	Normal Value	Description
STD 6-way offset error	0	An error occurred during 6-way striping of data. This error occurs when, for a six-way striped ST Data Operation, the Offset (from the Schedule Header) plus the length of the DMA portion of the STU is greater than three-quarters the destination buffer size: $$Offset + (M\_len - 0x30) > .75*(2**Bufsize)$$
Transmit error status	OK	
SSRAM parity error	0	The transmit logic encountered an SSRAM parity error.
STD invalid BUFX error	0	The BUFX table entry is invalid for the specified Src_Bufx parameter in a transmit descriptor for a Data Operation. The transmission was not performed.
Xtalk read req. timeout	0	The transmit logic encountered a timeout error for a read request on the XTALK interface. This error is fatal to the GSN interface because the SHAC ASIC forces the SuMAC ASIC to drop all micropackets.
Xtalk read access error	0	The transmit logic encountered an error in a read request on the XTALK interface. This error is fatal to the GSN interface because the SHAC ASIC forces the SuMAC ASIC to drop all micropackets.
PULL desc. timeout	0	The transmit logic encountered a timeout error on the XTALK interface for a read request done in Pull Mode.
PULL desc. access error	0	The transmit logic encountered an error in a read response on the XTALK interface.
FIFO CR underrun error	0	A transmission was initiated by a user process when no credits had been allocated to the process. The transmission was not performed. This error cannot be produced by transmissions initiated by the kernel.
FIFO underrun	0	The pointers indicated that there was a descriptor on SHAC's transmission queue, but SHAC found none when it tried to read the queue. This error can occur only if software writes the queue's consumer pointer, which is illegal during normal operation.

Table 3-6 (continued)   SHAC ASIC Status: Verbose Listing

SHAC Status Item	Normal Value	Description
FIFO overrun	0	The pointers indicated that there were empty slots on SHAC's ST transmission queue, but the queue was full when a user process tried to initiate a transmission (that is, write to the queue). This error can occur only if software writes the queue's producer pointer, which is illegal during normal operation. This error cannot be produced by transmissions initiated by the kernel.
User virt. conn. error	0	A user process attempted to transmit an ST Control Operation. This is illegal. The transmission is not performed.
User RAW error	0	A user process attempted to transmit a raw ST Operation. This is illegal. The transmission is not performed.
User invalid port error	0	A user process initiated a transmission for an invalid PORT. The transmission is not performed. This error cannot occur when the initiator is the kernel.
Source offset error	0	A user process initiated a transmission using an invalid *Src_Offset* value; the value was greater than or equal to the source buffer size specified in *Src_Bufsize* in the PORT table. The transmission is not performed. This error cannot occur when the initiator is the kernel.
Buffer range error	0	A user process initiated a transmission and one of the following errors occurred: • The *Src_Bufx* field is less than *Base_Bufx* in the PORT table entry. • *Src_Bufx*[31:29] is not equal to *Base_Bufx*[31:29] in the PORT table entry. • The data to be transmitted extends past the last valid Bufx in the series of valid Bufxs for the PORT. The transmission was not performed. This error cannot be produced by a transmission initiated by the kernel.

Table 3-6 (continued)   SHAC ASIC Status: Verbose Listing

SHAC Status Item	Normal Value	Description
Stripe VC error	0	A user process initiated a transmission in which the *Src_Bufx* value indicated striping, but the transmission was to a VC0, VC1, or VC2, which is illegal; striping is legal only on VC3. The transmission was not performed. This error cannot be produced by a transmission initiated by the kernel.
Transmit Error Source		Identifies the source (the cause) of the first error recorded in the Transmit error status section. (This section consists of the rows immediately above this row in this table.) The item marked with a 1 is the source; only one item is marked.
VC0	0	When 1, indicates error was encountered on VC0.
VC1	0	When 1, indicates error was encountered on VC1.
VC2	0	When 1, indicates error was encountered on VC2.
VC3	0	When 1, indicates error was encountered on VC3.
IP VC1	0	When 1, indicates error was caused by IP traffic on VC1.
H2F FIFO	0	When 1, indicates error was caused by the queue that holds host-to-board (that is, operating system to firmware) interrupts/commands.
Unknown	0	When 1, indicates that the source for the captured error is unknown.
None	0	When 1, indicates that no error source is identified.

Displaying Status of Closest GSN Elements (Hops)

To display status information for the two closest HIPPI-6400-PH (GSN) elements (hop 0, hop1, and hop2), use this command:

gsncntl gsn# s**tatus elements**

The closest element (hop 0) is the SuMAC ASIC located on the local SGI GSN board; the next closest element (hop 1) is the link end located at the other side of the physical link. The identity of this element depends on the configuration of the GSN connection, as illustrated in Figure 3-1. When a loopback device is installed, the hop 1 element is the same local GSN element as hop 0; when a cable is connected, the hop 1 element is the element at the other end of the cable (for example, switch or endpoint). Table 3-7 describes the *gsncntl status elements* display. Figure 3-1 illustrates the physical position for each numbered GSN element; status messages refer to the items identified in this illustration.

Note: When the SGI GSN board is configured for internal loopback, the hop 0 element is the IRIX GSN driver. The local SuMAC ASIC is not available when the board is configured for internal loopback.

With External Loopback

Point-to-Point

With HIPPI-6400 Switch
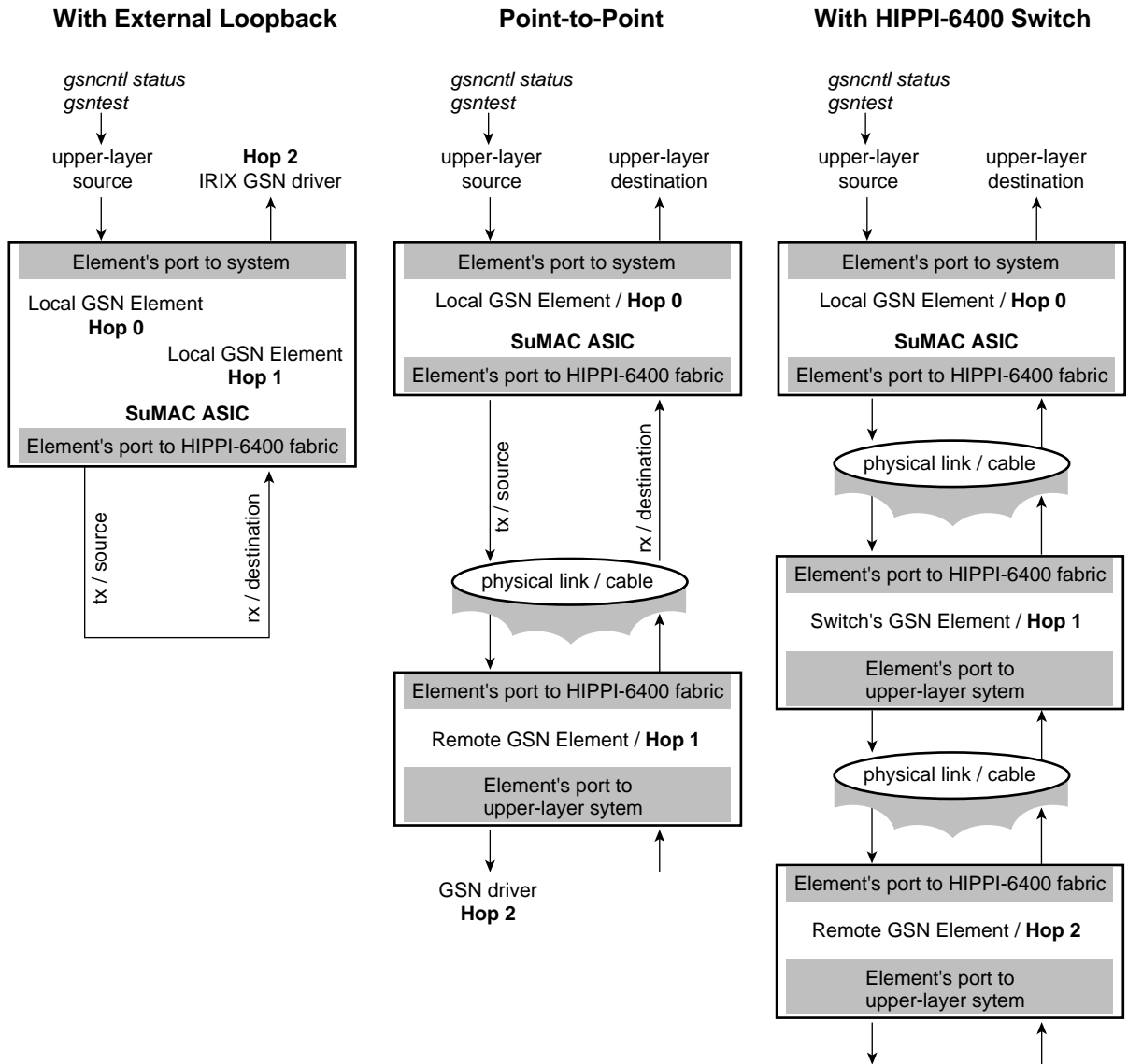
Figure 3-1   Physical Position of GSN Elements for Different Connection Scenarios

Table 3-7   HIPPI-6400-PH Element (Hop) Status

Status Item	Normal Values	Description
Hop # Status:		Status for the HIPPI-6400 physical element. When # = 0, status is for the SuMAC ASIC on the local SGI GSN board.
		When # = 1, status is for the first element on the other side of the physical link.
Port to endpoint's system	ready	Current state of element's port that connects to upper layers. States are sequential and are listed here in the order in which they occur. For hop 0, this information is for the port that connects to the SHAC ASIC on the GSN board.
		shutdown = the port is shut down.
		reset = the port is being reset.
		synchronizing ssr = the port is trying to synchronize itself with the port into the system.
		ready = the port is operational.
Port to HIPPI-6400 fabric	CLK2, TRAINED, LLP_OP, VC_OP	Status for element's port that connects to the HIPPI-6400 fabric. Multiple status items can be listed simultaneously.
		CLK2 = the element detects the activity-monitor signal (CLOCK2) coming from the remote element.
		TRAINED = the element has successfully completed its training sequence for dynamic adjustment of signal skew over physical link.
		LLP_OP = the port's link-level protocol is operational.
		VC_OP = the port's virtual channels are operating according to protocol.

Table 3-7 (continued)   HIPPI-6400-PH Element (Hop) Status

Status Item	Normal Values	Description
Port to endpoint's system Error Status	OK	Errors detected on the element's port that connects to the upper layers and the system. Errors are listed alphabetically. For hop 0 (local element), this information is for the SuMAC port that connects to the SHAC ASIC on the GSN board.
		`2nd link =` link errors (data/ctl parity or data invalid errors) have occurred on more than one clock cycle.
		`admin missing tail: VC` *hex_value* `=` an admin PDU (either GSN micropacket or SGI-LLP message) did not end with Tail bit set. Table 3-8 explains VC *hex_value*.
		`credit-only on VC` *hex_value* `=` an illegal credit-only (TYPE=0xA) micropacket was detected. Table 3-8 explains VC *hex_value*.
		`ctl parity =` control parity error was detected.
		`data invalid =` the signal that indicates the presence of invalid data was detected more than once in a single micropacket.
		`data parity in byte` *hex_value* `=` data parity error detected. Bit 7 covers most-significant byte of data, bit 0 covers least-significant byte. When a bit is set to 1, the corresponding data byte had an error.
		`ext link =` link error is asserted from system.
		`IC-SSR sample =` this port's SSR layer experienced an illegal clock transitions.
		`OK =` no errors were detected since last startup/reset of hardware.

Table 3-7 (continued)   HIPPI-6400-PH Element (Hop) Status

Status Item	Normal Values	Description
		`Missing end of msg on VC` *hex_value* = a GSN DATA micropacket without Tail bit was followed by a HEADER or ADMIN micropacket. Table 3-8 explains VC *hex_value*.
		`Missing strt of msg on VC` *hex_value* = a GSN micropacket with Tail bit was followed by DATA micropacket (HEADER was missing). Table 3-8 explains VC *hex_value*.
		`Rcvr tail timeout on VC` *hex_value* = receiver timed out before a GSN micropacket with Tail bit arrived. Table 3-8 explains VC *hex_value*.
		`RX VC buf overflow on VC` *hex_value* = receiver's buffer for the VC overflowed. Table 3-8 explains VC *hex_value*.
		`SGI-LLP admin extra tail on VC` *hex_value* = non-GSN protocol error in the SGI-LLP layer. Table 3-8 explains VC *hex_value*.
		`sndr credit=0 timeout on VC` *hex_value* = the credit=0 timeout expired due to internal VC full assertion from system's receiving port. Table 3-8 explains VC *hex_value*.
		`undefined pkt type on VC` *hex_value* = a GSN micropacket was detected whose type was not Header, Data, or Admin. Table 3-8 explains VC *hex_value*.
		`undefined val:` *hex_value* = TYPE value (in hexadecimal notation) read from last detected undefined micropacket.

117

Table 3-7 (continued)   HIPPI-6400-PH Element (Hop) Status

Status Item	Normal Values	Description
Port to HIPPI-6400 fabric Error Status	OK	Errors that have been detected on the element's port that connects to the HIPPI-6400 fabric. Errors are listed alphabetically.
		`activity mon lost` = activity monitor signal (CLOCK2) became inactive after port had been operational.
		`admin missing tail: VC` *hex_value* = an admin PDU (either GSN micropacket or SGI-LLP message) did not end with Tail bit set. Table 3-8 explains VC *hex_value*.
		`credit ovfl: VC` *hex_value* = more than 255 credits were received on the indicated virtual channel. Table 3-8 explains "VC *hex_value*".
		`ECRC rcv err cnt` = number of micropackets that were received with ECRC errors in which the micropacket's Error bit was not set.
		`ECRC snd err cnt` = number of micropackets that were transmitted in which the ECRC received from the system was erroneous and the micropacket ERROR bit was not set.
		`LCRC err cnt` = number of micropackets that were received with LCRC errors.
		`missing end of msg: VC` *hex_value* = a micropacket of Type=data without Tail bit set was followed by Type=header or Type=admin. The micropacket with its Tail bit set was missing. Table 3-8 explains VC *hex_value*.
		`missing start of msg: VC` *hex_value* = a micropacket with its Tail bit set was followed by a Type=data micropacket. The Type=header micropacket was missing. Table 3-8 explains VC *hex_value*.

Table 3-7 (continued)   HIPPI-6400-PH Element (Hop) Status

Status Item	Normal Values	Description
		`OK =` no errors were detected since last startup/reset of hardware.
		`rcv tail timeout: VC` *hex_value* = receiving VC timed out before final micropacket (with its Tail bit set) for a message arrived. This error is called the "stall timeout error" in the HIPPI-6400-PH standard. Table 3-8 explains VC *hex_value*.
		`Retry cnt =` number of retransmissions that have occurred.
		`retry failed =` a retransmission failed, forcing port into shutdown.
		`RSEQ missing cnt =` number of times the ACK timeout expired before an expected RSEQ was received, resulting in a retransmission.
		`RSEQ val err cnt =` number of times an incoming RSEQ value fell outside the expected range, resulting in a retransmission.
		`rx vc buf overflow: VC` *hex_value* = receiving VC's buffer overflowed. See Table 3-8 for explanation of VC *hex_value*.
		`SGI-LLP admin extra tail: VC` *hex_value* = non-GSN protocol error in the SGI-LLP layer. Table 3-8 explains VC *hex_value*.
		`skew retrain failed =` skew compensation retraining sequence failed (after it had been healthy).
		`skew retrain rst err =` skew compensation has not been successful since last reset.

119

Table 3-7 (continued)   HIPPI-6400-PH Element (Hop) Status

Status Item	Normal Values	Description
		`sndr credit=0 timeout:` VC *hex_value* = local source's credit=0 timeout expired due to VC full assertion from the element's port to the system. This error is called the "credit timeout error" in the HIPPI-6400-PH standard. Table 3-8 explains VC *hex_value*.
		`TSEQ val err cnt =` number of times a TSEQ value error was detected. TSEQ errors are not counted if a micropacket has not been accepted since the last TSEQ error.
		`undef pkt type:` VC *hex_value* = a micropacket of an undefined type was detected on the indicated virtual channel. Table 3-8 explains VC *hex_value*.
		`undef val:` *hex_value* = value of last detected undefined type of micropacket.

Table 3-8   Meaning of "VC *hex_value*" in Status Screen Displays

VC *hex_value*	Indicates Errors on These VCs	VC3 VC2 VC1 VC0 — each bit of *hex_value* identifies one VC
0x1	VC0	
0x2	VC1	
0x3	VC0, VC1	
0x4	VC2	
0x5	VC0, VC2	
0x6	VC1, VC2	
0x7	VC0, VC1, VC2	
0x8	VC3	
0x9	VC0, VC3	
0xA	VC1, VC3	
0xB	VC0, VC1, VC3	
0xC	VC2, VC3	
0xD	VC0, VC2, VC3	
0xE	VC1, VC2, VC3	
0xF	VC0, VC1, VC2, VC3	

Displaying Status of HIPPI-6400 ADMIN Traffic

See "Checking Status of ADMIN Traffic" on page 122.

Check Status of GSN Traffic

This section describes methods for monitoring various types of traffic that can move through an IRIX GSN subsystem: HIPPI-6400 Admin micropackets on page 122, ST messages on page 125, and IP datagrams on page 125.

Checking Status of ADMIN Traffic

To display status information about HIPPI-6400 ADMIN micropackets for a SGI GSN connection, use the *gsncntl status admin* command. This command displays counts for the various types of HIPPI-6400 ADMIN micropackets, as described in Table 3-9; each count includes both those transmitted and those received. The counts for pairs (for example, Ping and Ping_Response) should be very close.

```
# gsncntl [gsn#] status  admin
```

Table 3-9 describes the information that is displayed about ADMIN micropackets:

Table 3-9   Local Element's HIPPI-6400 ADMIN Traffic Status

Status Item	Description
ADMIN Packet Count (by command/type):	For each type of ADMIN micropacket, the display shows a count of request/command (Cmd) micropackets and a count of responses (Rsp).
Ping	Simple "HELLO, I'm alive." Response does same for remote endpoint.
Set Element Address	Commands the element to use the supplied element address. (This address is different from the globally-unique ULA.) Response is an acknowledgment only.
Exchange Element Function	Describes element's function: link-end, endpoint, or switch. Response does same.

Table 3-9 (continued)   Local Element's HIPPI-6400 ADMIN Traffic Status

Status Item	Description
ULA Request	Requests assignment of a Universal LAN MAC address (ULA) to the requestor from the receiver; the receiver must be a switch. Response assigns a ULA.
Read Register	Asks for current contents of element's ADMIN register. Response provides the contents. The local IRIX GSN driver is the main reader for local element's ADMIN register.
Write Register	Asks element to update (write) one or more fields in the ADMIN register. The local IRIX GSN driver is the main writer for local element's ADMIN register.
ULA List Request	Asks for a list of all connected ULAs. Response provides the list.
Port Remap	Changes the port-to-ULA mapping.
Port Map Request	Asks switch to provide the physical port that is necessary to open a connection to a particular ULA. Response is the port identification.
Reset	Commands element to reset (reinitialize) itself. There is no response to this.
Invalid Command	The ADMIN micropacket's command (that is, the type of ADMIN packet) was not recognized as one of those listed in this table. There is no response to this.
ADMIN Packet Errors:	
Admin cmd/rsp on vc0 or 3	An ADMIN micropacket was illegally detected on VC0 or VC3.
Admin cmd on vc != 1	A request/command ADMIN micropacket was illegally detected on a VC other than VC1.
Admin rsp on vc != 2	A response ADMIN micropacket was illegally detected on a VC other than VC2.

Table 3-9 (continued)   Local Element's HIPPI-6400 ADMIN Traffic Status

Status Item	Description
Admin rsp w/bad status	A response ADMIN micropacket was received with one of its status flags set. All status flags indicate that some problem occurred with the request/command that was generated by the source on the local element: undefined operation, invalid key, parameter out of range, invalid address for ADMIN register, command failed, etcetera.
Rec'd pkt when shut down	When the physical link was shut down, an ADMIN micropacket arrived. The packet has been discarded.
Rec'd unexp rd reg	An illegal Read Register request/command arrived. For example, the hop count asked that the micropacket be forwarded even though the local element's function is endpoint (not switch).
Rec'd unexp rd reg rsp	A Read Register response arrived when no request/command has been sent to initiate the response.
Rec'd unexp set elem addr rsp	A Set Element Address response arrived when no request/command has been sent to initiate the response.
Rec'd unexp xchange elem func	An illegal Exchange Element Function arrived. For example, the hop count asked that the micropacket be forwarded, indicating that the sender believes the local element is a switch.
Rec'd unexp xchange elem func rsp	An Element Function response arrived when no request/command has been sent to initiate the response.
Rec'd unknown elem type	An Element Function response arrived with an unknown function. The known functions are switch, link-end, and endpoint.
Rec'd unexp pkt in ULA_REQ state	While the local element was waiting for a response to its request for ULA assignment, an ADMIN micropacket arrived that seemed to require forwarding (for example, the hop count was not 0 or the destination element address was not 0xFFFFFFFF).

Table 3-9 (continued)   Local Element's HIPPI-6400 ADMIN Traffic Status

Status Item	Description
Rec'd unexp pkt in LINK_RDY state	After the local element (an endpoint) was functional, an ADMIN micropacket arrived that required forwarding (for example, the hop count was not 0 or the destination element address was not 0xFFFFFFFF).
Rec'd unexpected unknown admin pkt	An ADMIN micropacket was received that had an unknown command.

Checking Status of IP-over-GSN Traffic

To list the configuration information for the IRIX GSN network interface (*gsn*#), use this command:

```
% netstat -ina
```

To display statistics about the IP, TCP, UDP, or ICMP protocol stacks, use these commands:

```
% netstat -p ip
% netstat -p tcp
% netstat -p udp
% netstat -p icmp
```

Checking Status of ST-over-GSN Traffic

To list the configuration information for the IRIX GSN network interface (*gsn*#) that handles ST traffic, use this command:

```
% netstat -ina
```

To display protocol statistics for the ST protocol stack, use this command:

```
% netstat -p stp
```

Checking Status for All Protocols Simultaneously

To simultaneously display statistics for the IRIX GSN network interface, and traffic for all the protocol stacks, use this command:

```
% netstat -C
```

Use the numbers (highlighted in the menu area on the bottom of the display) to select the protocol or information you wish to view. Notice that there are two menus; press the **9** key on your keyboard to toggle between them. The first menu has Interfaces and IP; the second menus has ST (listed in the menu as STP), TCP, and UDP.

Use these letters to change the time period over which the displayed statistics have been counted:

r   displays totals collected (counted) since the last reset of the interface or operating system

z   resets all statistics to zero and starts counting

d   every second resets all statistics to zero and starts counting

Build New HARP Server Addresses Into Network Interfaces

Use either of command-line sequences below to build server address changes (from the /usr/etc/gsnarp.options file) into GSN network interfaces:

- To reconfigure all the GSN network interfaces:

  ```
  # /etc/init.d/gsn start
  ```

  Note: The *gsn* script marks each GSN network interface as down, reconfigures it, then brings it back online (that is, marks it up).

- To reconfigure one GSN network interface:

  ```
  # /usr/etc/ifconfig gsn# down
  <where gsn# identifies the network interface>
  # /usr/etc/gsnarp -f -S
  # /usr/etc/ifconfig gsn# up
  ```

  Note: The *gsnarp* command displays warning messages about any GSN network interfaces that are up. Just ignore these messages; the disabled interface is reconfigured and those that are up are left alone.

Display Client's Active HARP Server

Use the command line below to display the address that is currently being used by IRIX HARP to contact the HARP server (both the primary and the alternate) for the specified network interface.

```
% gsnarp -S gsn#
```

where *gsn#* identifies the IRIX GSN network interface for which you want information.

Display Address Resolution Lookup Table

Use the command line below to display the HARP table that is currently loaded into memory and being used by IRIX HARP. This table maps INET addresses to physical/hardware addresses (ULAs or ULAs+I-field):

```
# gsnarp -a
```

The following example of a *gsnarp* display illustrates one static (permanent) entry for a HIPPI-800 host and a dynamic entry for the HARP server:

```
HARP TABLE CONFIGURATION
Address     ULA                IFIELD        FLAGS
gsn-amazon  00:01:42:ff:b9:0a  0x07000005    COMPL, CONN, PERM
gsn-nile    FF:FF:FF:FF:FF:FF                 COMPL, CONN, SVR
```

The flags possible for each entry are explained in Table 3-10:

Table 3-10   HARP Flags

Flag	Description
COMPL	Registration or re-registration of the INET address for this entry is complete. The entry is valid and ready for use.
CONN	Registration or re-registration of the ULA for this entry is complete. The entry is valid and ready for use.
SVR	Entry is for the HARP server. Entry will not be removed until HARP is reconfigured.
INUSE	Entry is created and ready for use.

Table 3-10 (continued)   HARP Flags

Flag	Description
NAK	Address resolution request for this entry failed; HARP server did not have this entry. This flag occurs only in HARP client databases.
PEND	Registration with the HARP server is in progress; a reply from the HARP server is outstanding (pending).
PERM	Entry is permanent (static), not dynamic. Entry does not age,
SRCROUTE	802.2 SNAP heading is not used on traffic to this address.
VALIDATE	Entry is being validated because the age timeout period expired. As long as the COMPL flag is displayed, the entry continues to be used for address resolution.

Installing a Loopback Device

To run a loopback verification test on an SGI GSN board, use one of the procedures described below to set up the loopback:

1. Disable the IRIX GSN network interface:

    % **ifconfig gsn# down**

    where # identifies the network interface you are going to verify.

2. Use one of these methods for installing a loopback path:

    *   Attach a loopback device to the HIPPI-6400 port on the board's I/O panel plate. The connector is keyed with a long and a short side to ensure correct alignment, as illustrated in Figure 3-2.

Long/wide side of key

Short/narrow side of key
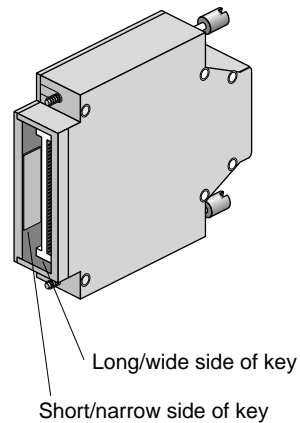
Figure 3-2   Installing a HIPPI-6400 Loopback Connector

- At the switch, configure the attached input port so that it loops back all traffic to the same port's output. For example, you might need to make all the entries in the port's routing (forwarding) table point to this port.

- Configure the board for internal loopback, as described in "Configure the SGI GSN Board for On-board (Internal) Loopback Operation" on page 100.

  Note: With internal board loopback, the HIPPI-6400 connector and the GSN component on the board are not verified during the verification procedures.

3. Enable the network interface:

    % **ifconfig gsn**# **up**

    where # is the interface you disabled.

Note: When you remove the loopback device or link and make a new connection (point-to-point or to a switch), you must invoke *ifconfig down* to disable, then *ifconfig up* to enable the network interface.

Verifying the IRIX GSN Subsystem

The most reliable method for verifying an IRIX GSN subsystem is to install a loopback device or looped back link (as described in "Installing a Loopback Device" on page 128), then run the *gsntest* hardware verification test, as described below. After the GSN hardware has been verified, further upper-layer verification and interconnectivity tests can be run (for example, the tests described under the headings "Verify an IP-over-GSN Interface") by communicating with other GSN systems.

Note: Unlike many IRIX drivers, the IRIX GSN driver does not automatically route self-addressed packets through the local loopback interface (*lo0*), so that even the IP stack can be verified with the loopback link in place.

Verify That the Board Has Been Located by the Software

To verify that a SGI GSN board has been located by the operating system during the last reboot, use any of the following commands:

```
% hinv -d gsn
GSN #-XIO adapter: unit #, in module # I/O slot #

% hinv -mvv -d gsn
. . .
Location: /hw/module/#/slot/io#/xio_gsn
  XIO_GSN Board: barcode ######    part 030-1361-00# rev #
  Group ff Capability ffffffff Variety ff Laser 0000002adfaa
. . .
GSN #-XIO adapter: unit #, in module # I/O slot #


% find /hw/module -name xio_gsn
/hw/module/#/slot/io#/xio_gsn
```

Each GSN solution may have multiple full-path entries in the IRIX hardware graph. For example, in the sample display shown below, the startup routine located two SGI GSN products installed into two different modules; one instance is the two-board product while the other is the single-board product. With the two-board product, the main GSN board is always identified as XIO port 1, and the additional board is XIO port 2.

```
% hinv -d gsn
GSN 1-XIO adapter: unit 0, in module 1 I/O slot 8
GSN 2-XIO adapter: unit 1,
     XIO port 1 in module 2 I/O slot 4
     XIO port 2 in module 2 I/O slot 12

% find /hw/module -name xio_gsn
/hw/module/1/slot/io8/xio_gsn
/hw/module/2/slot/io12/xio_gsn
/hw/module/2/slot/io4/xio_gsn
```

Verify the SGI GSN Hardware

To verify the main SGI GSN board (without going through the IP or ST stack), use the */usr/etc/gsntest* command. This test works only for a SGI GSN board that has a loopback device installed. (See "Installing a Loopback Device" for instructions.) The command requires the user to be superuser (root).

Note: Unlike many IRIX drivers, the IRIX GSN driver does not automatically route self-addressed IP/ST packets through the network stack's loopback interface (*lo0*); the GSN driver passes self-addressed packets to the hardware. This allows you to use standard IP utilities (such as *ping <myaddress>*) to test the IRIX GSN subsystem when it is physically looped back.

For a simple, quick verification test, use the commands below:

```
% cd /usr/etc
% su
Password: thepassword

# hinv -d gsn
<use the displayed unit number for # in the following command lines>

# /usr/etc/gsncntl gsn# status device
<verify that the STATE and FLAGS are correct
as described in Table 3-3 on page 103>

# /usr/etc/gsntest gsn#
gsntest: PING
GSN PING hop 0: Received ping cmd/response from element in 150.40 us
GSN PING hop 1: Received ping cmd/response from element in 117.60 us
GSN PING hop 2: Received ping cmd/response from element in 1707.20 us
<refer to Figure 3-1 to identify the physical element
associated with each hop #>
```

The *gsntest* utility sends HIPPI-6400 Admin "PING" micropackets. The command creates packets with the following nonconfigurable characteristics:

Key	0
Hop Count	0
Dest_Reg	0
Dest_Addr	0xFFFFFFFF
Command	HIPPI-6400-SC Ping
Status_Flags	0
Src_Reg	0
Src_Addr	Local interface's ULA obtained from switch, or if not available, as read from PROM.
Data_Reg	Bytes 0-3 contain the PID of the process. Bytes 4-15 contain randomly generated data.

Example:

This example illustrates the test when the local SGI GSN port is connected to a switch:

```
% /usr/etc/gsntest gsn#
gsntest: PING
GSN PING hop 0: Received ping cmd/response from element in 148.80 us
GSN PING hop 1: Received ping cmd/response from element in 131.20 us
GSN PING hop 2: Received ping cmd/response from element in 13844.80 us
```

Note: The hop 1 response came from the switch while the hop 2 response came from the remote endpoint.

This example illustrates the test when a loopback device is installed on the local SGI GSN port:

```
% /usr/etc/gsntest gsn#
gsntest: PING
GSN PING hop 0: Received ping cmd/response from element in 140.20 us
GSN PING hop 1: Received ping cmd/response from element in 148.80 us
GSN PING hop 2: Received ping cmd/response from element in xx us
```

Note: Hop 0 came from the local SuMAC's source logic. The hop 1 response came from the destination logic (on the "other side" of the external loopback device) within the local SuMAC ASIC, while the hop 2 response came from the local IRIX GSN driver.

The *gsntest* utility does not verify the data path through the second XIO slot of a two-board GSN solution.

Verify an IP-over-GSN Interface

To verify that each IP-over-GSN network interface is functional, follow the instructions in this section. This test assumes that the IRIX GSN subsystem has passed the *gsntest* verification, as described under the heading "Verify the SGI GSN Hardware."

Note: Unlike many network products, the IRIX GSN software does not loop IP packets through the system's local loopback interface (*lo0*). All IP-over-GSN packets are passed to the GSN hardware.

To accomplish this verification, use */usr/etc/ping -r* (lowercase -r, not -R) to make this station communicate with another GSN IP station (or itself) over the IRIX GSN subsystem.

133

1. Obtain the IP network addresses for all the IP-over-GSN interfaces (*gsn#*) on this system. This information can be displayed with the command shown below. The network address is listed in the column labeled `Network`, as illustrated in Figure 3-3.

   ```
   % /usr/etc/netstat -ina
   ```

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Coll
ef0	1500	192.74.28	192.74.28.64	873404	1248	316177	0	1576
			08:AC:15:B1:02:6F					
gsn0	65280	253.5.88	235.5.88.1	2578	2	28679	0	2148
			08:10:26:00:8A:EC					
gsn1	65280	none	none	0	0	0	0	0
lo0	8304	127	127.0.0.1	3609810	0	3609810	0	0

Network interface configuration for second IRIX GSN connection

Network interface configuration for first IRIX GSN connection

Ethernet connection

Figure 3-3   The */usr/etc/netstat -ina* Display

2. Obtain the name (or IP address) of at least one remote station on each of these GSN network addresses. Two methods for obtaining station names are described below.

   - For a system connected to a local area network that provides name lookup service (NIS), use the commands below to create a file for each GSN network connection. Each file will contain the names and addresses of stations that share a particular network address:

     ```
     % ypcat hosts | grep gsn0_networkaddress > gsn0.s
     % ypcat hosts | grep gsn1_networkaddress > gsn1.s
     <do this for each GSN IP network address>
     ```

     where each *gsn#_networkaddress* value is an address from the `Network` column of the *netstat* display (illustrated in Figure 3-3).

     Example:

     ```
     % ypcat hosts | grep 253.5.88 > gsn0.s
     ```

- For a system that does not have access to NIS, use these commands to create a file for each network connection. Each file will contain the locally-known names and addresses of stations that share a particular network address:

```
% grep gsn0_networkaddress /etc/hosts > gsn0.s
% grep gsn1_networkaddress /etc/hosts > gsn1.s
<do this for each IP-over-GSN network address>
```

Example:

```
% grep 253.5.88 /etc/hosts > gsn1.s
```

3. Communicate with one station on the GSN network used by the *gsn0* connection. For the variable *gsn0_station*, you can use any of the names or IP addresses from the *gsn0.s* file.

```
% ping -r gsn0_station
PING stationname (IPaddress): 56 data bytes
64 bytes from ... time=x ms...
<Ctrl><c>
----stationname PING Statistics----
# packets trans,# pckts rcvd, x% packet loss
```

Note: If a loopback link is in place, use the system's own IP address for the *gsn0_station* variable.

4. If *netstat* lists more than one IRIX GSN (*gsn#*) network interface, communicate with one station on each of those networks. For the variable *gsn#_station*, you can use any of the names from the *gsn#.s* file.

```
% ping -r gsn#_station
PING stationname (IPaddress): 56 data bytes
64 bytes from ... time=x ms

...
<Ctrl><c>
----stationname PING Statistics----
# packets trans, # pckts rcvd, x% packet loss
```

Note: If a loopback link is in place on any of the ports, use the system's own IP address for the *gsn#_station* variable.

5. If one *ping* on each network succeeds, you have completed the verification procedure. All the local network connections are functioning. Use the commands below to remove the files with the lists of stations:

```
% rm gsn0.s
% rm gsn1.s
<do this command line for each gsn#.s file created>
```

If the *ping* on a network fails, follow the instructions in the section "Troubleshoot IP-over-GSN Interfaces."

135

6. To verify the throughput for a functional network interface, open an IRIX shell window for the local station and antoher one for a remote station. Then, invoke the following commands:

```
<in the IRIX shell for station #1, the receiver>
% /usr/etc/ttcp -r -s -149152 -b1048576
```

```
<in the IRIX shell for station #2, the transmitter>
% /usr/etc/ttcp -t -s -149152 -b1048576 -n262144 station1_IPaddress
```

Troubleshooting

This section provides basic procedures for troubleshooting IRIX GSN.

Troubleshoot SGI GSN Hardware

See the *SGI GSN Hardware Owner's Guide*.

Troubleshoot IP-over-GSN Interfaces

This section describes separate sets of instructions for troubleshooting a system in which all the IP-over-GSN network interfaces are failing, and a system where only some of the network interfaces are failing.

All IP-over-GSN Network Interfaces Are Failing

If the *ping* verification tests fail for all the system's IP-over-GSN network connections, your system probably has been configured incorrectly. Verify the configuration by performing the steps below.

Note: This procedure assumes that the hardware has been varified by following the instructions in "Verify the SGI GSN Hardware" and has been found to function correctly.

1. Use this command to verify that the local GSN board is not in loopback mode and that the physical link is operational:

```
% gsncntl gsn# status  device
STATE: LNK_RDY
FLAGS: LNK_SWITCH
```

2. If the system is connected to a switch, use *gsncntl* to verify that the switch has assigned a ULA to the local system. The assigned address is the one labeled Device.

```
% gsncntl gsn# getmac
PROM ULA Address:   08:00:69:05:0d:2c4
Device ULA Address:  00:01:3b:ff:00:04
```

3. Verify that IP networking is enabled with the following command line:

```
% /sbin/chkconfig | grep network
network    on
```

4. Use */usr/etc/netstat -ina* to verify that the local IP-over-GSN network interfaces have been configured and enabled. The display should look similar to that shown in Figure 3-3.

   Refer to the online *IRIX Admin:Networking and Mail* guide for information about configuring and troubleshooting IP network interfaces.

5. Use *gsnarp -a* to verify that the address resolution mechanism is functioning. Check that a primary HARP server is listed for each local GSN network interface and verify that the HARP table has correct entries for the remote system(s) that you tried to *ping*.

```
# /usr/etc/gsnarp -a
HARP SERVER CONFIGURATION
Dev        Primary Server             Alternate Server
gsn0       ULA = 00:01:3B:FF:00:0E    ULA = 00:01:3B:FF:00:04

HARP TABLE CONFIGURATION
Address     ULA                IFIELD     FLAGS
nile-gsn0   00:01:3b:ff:00:0E   -          COMPL,CONN,INUSE,SVR
ob-gsn0     00:01:3b:ff:00:04   -          COMPL,CONN,INUSE
thames-gsn0 00:01:3b:ff:00:0D   -          COMPL,CONN,INUSE
```

137

6. Use *ttcp* to verify that the local IP protocol stack is functioning:

    ```
    % ttcp -r -s &
    % ttcp -t -s IPaddress
    ```

    where *IPaddress* is the INET address (as displayed by the *netstat* command) for the local IP-over-GSN network interface.

7. Verify that the remote system you tried to *ping* is operational.

Some IP-over-GSN Network Interfaces Are Failing

If the *ping* verification tests succeed for one GSN network connection, but others fail, the local IP stack is functioning, but one (or more) specific interface has a problem. To resolve the problem, follow the instructions below for each problematic network connection.

1. Make sure that you know which IRIX GSN port is associated with the GSN network interface (*gsn#*) that you are troubleshooting.

2. Use *gsnarp -a* to verify that a primary HARP server is listed for the problematic interface (Dev) and to verify that the HARP table has a correct entry for the remote system you tried to *ping*.

    ```
    # /usr/etc/gsnarp -a
    HARP SERVER CONFIGURATION
    Dev         Primary Server              Alternate Server
    gsn0        ULA = 00:01:3B:FF:00:04     ULA = 00:01:3B:FF:00:0E

    HARP TABLE CONFIGURATION
    Address     ULA                 IFIELD      FLAGS
    nile-gsn0   00:01:3b:ff:00:04   -           COMPL,CONN,INUSE,SVR
    ob-gsn0     00:01:3b:ff:00:0d   -           COMPL,CONN,INUSE
    ```

3. Verify that the other endpoint (IP host) is operational.

    Or, as an alternative, select a different station in this LIS, and use the *ping -r* command with the station's numerical address (instead of the name). If the *ping* works, the network connection is functional. If the *ping* fails, proceed to the next step.

4. Verify that the network portion (leftmost digits) of the addresses you are attempting to *ping* match the network address for the GSN interface you are troubleshooting. The network address for each GSN network interface can be displayed by the */usr/etc/netstat -in* command.

Check all the cables and nodes between this system and the system(s) you tried to *ping*. "Verify the SGI GSN Hardware" on page 131 describes how to do this for the local SGI GSN port and its link.

Glossary

bridging switch

A node that connects two different sections of a HIPPI/GSN network. The two sections can be two different fabrics (such as GSN and HIPPI-800) or two similar sections (such as two groups of GSN endpoints). The node complies with the following:

- provides broadcasting by using the IEEE 802.1d spanning tree algorithm and protocol

- uses the complete 6-byte destination ULA for path selection

- complies with the IEEE 802.1d algorithms for its automatic path-select configuration

destination

The receiving element located at the other end of a physical link. An upper-layer entity (host, network-layer interface, or program) that uses the GSN subsystem is sometimes loosely referred to as the destination; however, it is more correct to call these software entities upper-layer protocols (that is, destination ULPs). A "final destination" refers to the element that is the ultimate receiver for a micropacket; an element that receives then retransmits the micropacket (for example, a switch) is simply a destination.

element

Any component of a HIPPI-6400 fabric or system that is able to receive, process, and send HIPPI-6400 Admin micropackets in a manner that conforms with the HIPPI-6400 standard. Each HIPPI-6400 port on an element contains both a source and a destination. For example, the SuMAC chip in an SGI GSN product is a single-port HIPPI-6400 element.

endpoint

A final destination or an originating source of GSN traffic. An endpoint may have only one GSN port. A single system may have many endpoints (for example, an Origin module with two SGI GSN products has two endpoints).

fabric

All the HIPPI nodes (switches, endpoint devices, extenders) that are physically interconnected and speak the same physical-layer protocol. For example, a GSN fabric is a separate entity from a HIPPI-800 fabric.

One GSN fabric can be logically divided into multiple upper-layer address spaces (that is, networks or LISs). For example, a single GSN fabric can support multiple IP networks. Conversely, one logical network can include members from multiple HIPPI fabrics. For example, an IP network can include members from a GSN (HIPPI-6400) fabric as well as members from a HIPPI-800 fabric.

hardware device name

The string of characters and numerals used to identify a specific item of hardware. This string is assigned by the operating system, and is not configurable. The hardware names for the components/devices in a system can be displayed with the *hinv* command. For the SGI GSN hardware, this name has the format *gsn0*, *gsn1*, *gsn2*, and so on. See "Assignment of Unit Numbers to Hardware" in Chapter 1 for details on how the numbers are assigned to each hardware device (port).

HARP service

A logical IP subnet (LIS) service that provides address resolution for IP hosts operating over a HIPPI fabric. The service conforms with the proposed HARP RFC. When the HIPPI fabric supports broadcasting, every IP host provides the service by responding to address resolution requests that are addressed to it. When the HIPPI fabric does not support broadcasting, one IP host provides the service by maintaining a HARP table and answering address requests from the other hosts (clients) of the LIS; in this scenario the LIS has one HARP server.

hop count

A number used in HIPPI-6400 Admin micropackets to indicate through how many elements the micropacket should be forwarded. Each time a micropacket exits an element, the hop count is decremented by one. See "GSN Admin Micropackets" on page 18 for further details.

hostname

The string of characters and/or numerals used to identify a specific instance of an operating system (that is, a UNIX host). This string is completely customer-configurable; it is created by editing the */etc/sys_id* file.

INET address

Internet address, and frequently referred to as IP address or IP/ST address. A globally-unique 32-bit number used to identify a network-layer entity that speaks one of the DARPA Internet protocols (for example, TCP, UDP, ICMP) or another supported protocol, such as Scheduled Transfer (ST). This address is completely customer-configurable, but assigned in blocks by a globally-recognized address authority, such as the National Science Foundation's InterNIC. In IRIX, this address is mapped to a network interface name by editing the */etc/config/netif.options* file. See the *network connection name* glossary entry for a user-friendly name for this address.

initiator

The ST endpoint that sends the first operation within an ST sequence. The endpoint that acts as initiator during one sequence (for example, the connection setup) can act as the responder in a subsequent sequence (for example, the data movement).

LIS

See *logical IP subnet*.

logical IP subnet

Also known as LIS. A globally-known (public) logical address space that is defined by an INET network address and a *subnet mask*. The basic methodology of an LIS is similar to *subnet*, at the local (site) level; however, the consequences for routing are quite different for the two methodologies. Refer to "Classless Inter-domain Routing" RFC 1519. Also see *subnet* and *subnet mask*.

netmask

See *subnet mask*.

network connection name

Sometimes referred to as *alias for IP address* or (incorrectly) as *hostname*. The string of characters and/or numerals used as a user-friendly method for identifying a specific network-layer entity; the string is an alternate (alias) for an *INET address*. This string is completely customer-configurable; it is created and mapped to an INET address in the */etc/hosts* file; it can be used in the */etc/config/netif.options* file to configure a network interface. By convention, the network connection name includes the system's hostname. For example, a system with the hostname *granite* might have network connection names of *granite-ef0.rocks.com* and *granite-gsn0.rocks.com*.

network interface name

The string of characters and numerals used to identify a specific logical network-layer interface (sometimes known as an *if_net* instantiation). This string is assigned by the operating system and tied to a specific hardware subsystem; it is not customer configurable. For the IRIS GSN product, this name has the format *gsn0*, *gsn1*, *gsn2*, and so on. The network interface name is mapped to a configurable network-layer (for example, INET) address by the */etc/config/netif.options* file. See "Assignment of Network Interface to Hardware Device" in Chapter 1 for details on how the numbers are assigned to each GSN network interface.

operation

The ST protocol data unit. It is composed of a 40-byte header and variable-length data ranging from 0 bits to 4 gigabits (illustrated in Figure 1-13). Each ST operation is transmitted as one GSN Message, as illustrated in Figure 1-13.

physical link

One section of HIPPI-6400-PH transmission medium (copper or fiber-optic cable) that connects two HIPPI-6400-PH elements. Each element can be either a switch or an endpoint. Each physical link is a full-duplex link composed of two simplex links; each simplex link carries data in only one direction. The two streams of data in the full-duplex link flow in opposite directions. The path (virtual connection) between an original point of transmission (the originating source) and a final point for reception (the final destination) can involve numerous physical links.

responder

The other (not the initiator) ST endpoint participating in an ST sequence.

sequence

In the context of the ST protocol, a series of operations that occur in a specific order and accomplish an ST protocol task.

slot

Memory at an ST destination that is reserved for holding one incoming ST Header.

source

The transmitting element located at one end of a physical link. An upper-layer entity (host, network-layer interface, or program) that uses the GSN subsystem is sometimes loosely referred to as the source; however, it is more correct to call these software entities upper-layer protocols (that is, source ULPs). An "originating source" refers to the element that first transmitted a micropacket; an element that is retransmitting the micropacket (for example, a switch) is simply a source.

subnet

Also called subnetwork. A site-defined (private) address space that is carved from a globally-assigned (public) INET network address space. A subnet mask is used to divide the INET network address into many subnets. See *subnet mask*.

subnet mask

Also called netmask. A customer-configurable value for increasing the number of bits within an *INET address* that are used for network (netid) identification. The netid is increased by taking 1 or more bits from the host portion (hostid) of the INET address and using them as additional network bits. For example, a basic INET class C address consists of 24 bits of netid and 8 bits of hostid. A site can use a subnet mask to alter their addresses so that 28 bits are used to identify the network and only 4 bits are used to identify hosts.

Use of a subnet mask allows one INET network address to be subdivided into multiple networks (called *subnet*s). A basic class C address allows a site to have up to 254 hosts on one network. If a site increases its netid to 28 bits, it can have up to 15 hosts on each of 16 different networks (subnets).

Within a subnet mask, the bits set to 1 specify the portion of the address used to identify networks, while the bits set to 0 identify the hosts. For example, netmask 0xFFFFFF80 (hexadecimal) provides 25 bits of netid and 7 bits of hostid, while 0xFFFFFFF0 provides 28 bits of netid and 4 bits of hostid. The default netmask for IRIX is 0xFFFFFF00. This default can be changed for each network interface by editing the appropriate */etc/config/ifconfig-#.options* file.

switch

A node that is located along the route between two endpoints. GSN traffic passes through the switch on its way to a destination endpoint. A switch must have at least two, and usually has numerous, GSN ports.

unit number

The numeral portion of the *hardware device name*. See "Assignment of Unit Numbers to Hardware" on page 67 for a description of how these are assigned.

universal LAN MAC address (ULA)

Also known as a MAC (media access control) or ethernet address. A globally-unique 48-bit IEEE 802-style number used to identify a hardware device. The highest bits (that is, 47 to 24: the Organizationally Unique Identifier) are assigned by the globally-recognized address authority, IEEE. The lower 24 bits are assigned by the local organization that has been given (assigned) the OUI.

Index

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document

- Omission of material that you expected to find

- Technical errors

- Relevance of the material to the job you had to do

- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-3719-001.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:

  - On the Internet: techpubs@sgi.com

  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs

- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-932-0801

- To send your comments by **traditional mail**, use this address:

Technical Publications
Silicon Graphics, Inc.
1600 Amphitheatre Parkway, M/S 535
Mountain View, California 94043-1351