GRF 1.4 Addendum

Ascend Communications, Inc.
January, 1998
GRF software version 1.4

ECMP

Introduction to GRF ECMP

The Equal Cost Multi-path (ECMP) feature provides an ability to efficiently modulate traffic to destination networks. With ECMP enabled, multiple gateways for destination network or host prefixes (addresses) can be legally installed in the GRF route table. This release supports a maximum of eight gateways per ECMP group. Rather than use a single "best" route, ECMP routes packets toward a destination network by splitting the packet load between different, but similar, paths.

The diagram below shows a simplified ECMP group example. The GRF1 router is running ECMP. If ECMP is statically configured, ECMP routes are created by configuring entries in `/etc/grroute.conf`. If ECMP is turned on in the `/etc/gated.conf` file, GateD learns the routes and allows multiple gateways to be assigned to a single source address. The R300 router is the gateway for an available but unequal path.

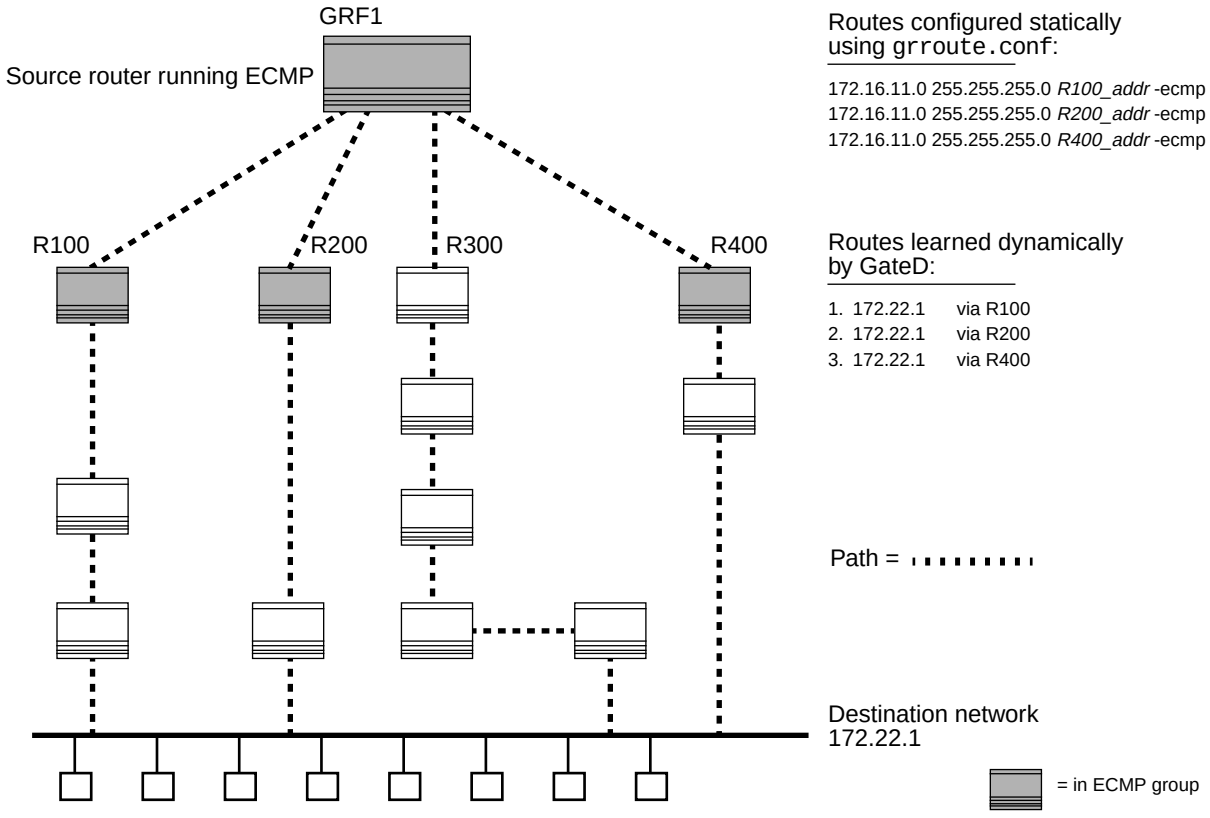


Figure 1. Example of alternate ECMP routes

For each packet, a determination mechanism selects the nexthop gateway from the ECMP group. The current determination mechanism selects from the group based on a source/destination hash. The hash is fixed, and, for most cases, provides a reasonable, equal distribution of traffic. The hash system uses a core application to provide the fastest processing possible on a per-packet basis. This method ensures packets from a given source arrive in order to a given destination.

GateD support

GateD provides these services for ECMP:

- a configuration option to enable/disable dynamic ECMP (default is disabled)
- the ability to insert multiple equal routes into the kernel
- internal OSPF protocol support

The current release supports equal cost multipath prefixes learned via the OSPF and OSPF_ASE protocols. Also, I/CBGP will resolve prefixes to multiple gateways if the nexthop resolving protocol is OSPF or OSPF_ASE.

The ECMP parameter is available in a Definition Statement:

multipath { **on** | **yes** | **off** | **no** } ;

This parameter enables/disables the installation of multiple gateways for network or host prefixes into the kernel route table. The default is off.

The **on** option is the same as **yes**, and enables GateD to install multiple routes for a single source with the same destination but different nexthops.

The **off** option is the same as **no**, and means that each route GateD installs in the kernel will have a unique destination and nexthop.

Dynamic ECMP configuration

Enable dynamic creation of ECMP routes in `/etc/gated.conf` by including either of these statements:

```
multipath on ;
```

or

```
multipath yes ;
```

Static ECMP configuration

Enter one line for each destination nexthop (gateway) in the `/etc/grroute.conf` file. The `-ecmp` parameter is optional only for the source router's first entry, it is required for the rest of the entries:

```
# source_addr  netmask  next_hop_addr -ecmp
172.16.11.0 255.255.255.0 R100_addr [-ecmp]
172.16.11.0 255.255.255.0 R200_addr -ecmp
172.16.11.0 255.255.255.0 R400_addr -ecmp
```

The source GRF router running ECMP is 172.16.11.0.

Checking ECMP routes

To verify that routes have been installed in the kernel as ECMP routes, use the **netstat -rn** command. The “E” in the flags field is the ECMP identifier:

```
# netstat -rn
Routing tables

Internet:
Destination        Gateway            Flags     Refs     Use  Interface
10.0.0.82          10.0.0.82          UH          0       0  lo0
10.0.0.110         10.8.1.110         UGHE        0       0  go0a0
10.0.0.110         10.8.2.110         UGHE        0       0  go0b0
                        Method:CRC16
10.0.0.176         10.8.1.110         UGHE      693     692  go0a0
10.0.0.176         10.8.2.110         UGHE        0       0  go0b0
10.0.0.176         10.8.3.177         UGHE        0       0  go0d0
                        Method:CRC16
10.0.0.177         10.8.1.110         UGHE        0       0  go0a0
10.0.0.177         10.8.2.110         UGHE        0       0  go0b0
10.0.0.177         10.8.3.177         UGHE        0       0  go0d0
                        Method:CRC16
10.1.5/24          10.8.1.110         UGE         0       0  go0a0
10.1.5             10.8.2.110         UGE         0       0  go0b0
10.1.5             10.8.3.177         UGE         0       0  go0d0
                        Method:CRC16
10.1.6/24          10.8.1.110         UGE         0       0  go0a0
10.1.6             10.8.2.110         UGE         0       0  go0b0
10.1.6             10.8.3.177         UGE         0       0  go0d0
```

Note that destination addresses have been assigned multiple gateways.

Frame Relay Configuration

This section describes how to configure Frame Relay on GRF HSSI and SONET OC-3c media cards.

Two options are available:

- UNI, routed Frame Relay
- NNI, switched Frame Relay

Configuring routed and switched Frame Relay circuits is the focus of this material. It is assumed that you have already configured the media card in the appropriate CLI profiles and card interfaces in `/etc/grifconfig.conf` and. Refer to the HSSI and SONET chapters in the *GRF Configuration Guide* for that information.

These topics are included:

Introduction to Frame Relay

In a Frame Relay network, each physical connection is called a link. A link is a point-to-point physical connection to another piece of Frame Relay equipment, such as a switch or router.

A virtual circuit is a path from an endpoint through one or more frame relay switches to another endpoint. A circuit goes across one or more links.

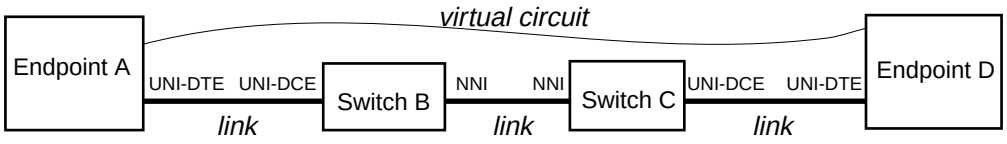


Figure 6-1. Frame Relay virtual circuit and links

Link types

You can specify a Frame Relay link to be UNI-DTE, UNI-DCE, or NNI.

UNI = User to Network Interface
NNI = Network to Network Interface
DTE = Data Terminal Equipment
DCE = Data Communications Equipment

UNI-DTE link

A UNI-DTE device is the device at the edge of a Frame Relay network. It connects to a UNI-DCE device inside the network. Only routing is performed on UNI-DTE links.

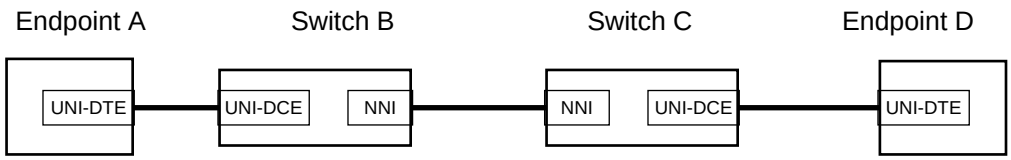
UNI-DCE link

A UNI-DCE device is the externally-connecting device at the edge of a Frame Relay network. It connects to a UNI-DTE device outside the network. Switching and routing can both be performed on UNI-DCE links.

NNI link

An NNI link is the link between two Frame Relay switches inside the network. Switching and routing can both be performed on NNI links.

In the example below, endpoint A views the link to B as a UNI-DTE link. Switch B views the link to A as a UNI-DCE link, and the link to C as NNI.



PVCs

A PVC (Permanent Virtual Circuit) is a logical path through a Frame Relay network from one endpoint to another endpoint. The path goes across segments that link network devices.

There is a segment for each link between the endpoints. Each segment of a circuit is identified with a Data Link Circuit Identifier (DLCI). The DLCI field is only 10 bits wide, for a maximum of 1024 circuits per link. It is important to note that DLCIs have local significance only. Otherwise, the entire network would be limited to 1024 circuits.

Each link on the path must have a unique DLCI number, as shown in Figure 6-2.

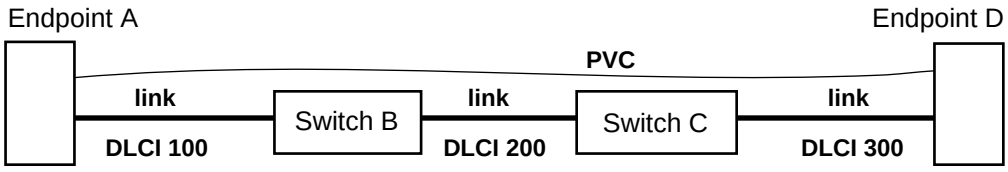


Figure 6-2. Components of a Frame Relay circuit

The circuit from endpoint A to endpoint D passes through two switches, B and C

- from A's point of view, the circuit to endpoint D uses DLCI 100
- from D's point of view, the circuit to endpoint A uses DLCI 300

For the switches, each circuit connects two link/DLCI pairs.

- for switch B, the circuit connects DLCI 100 on the A link to DLCI 200 on the C link
- for switch C, the circuit connects DLCI 200 on the B link to DLCI 300 on the D link

Each packet has a Frame Relay header. One of the fields in the header is the DLCI. The DLCI determines which circuit a packet is traveling on, and allows a switch to forward the packet to the appropriate next link.

The switches change the DLCI value in the header as the packet crosses the network.

At the ingress endpoint, a packet is encapsulated in a Frame Relay header and then placed on a link. Conversely, when the egress endpoint receives a packet from a link, it removes the Frame Relay header before processing. A router is a typical endpoint.

From the router's point of view, at the end of each circuit is another host with an IP address. From a switch's point of view, a circuit is merely a connection between two DLCIs on two links. In the example, switch C views the circuit as: "Link B-200 goes to Link D-300".

All the stations in a Frame Relay network (endpoints and switches) communicate with each other using Local In-Channel Signaling (LICS). A DLCI on each link is reserved for this purpose. LICS messages convey the status of circuits throughout the Frame Relay network. LICS is also referred to as LMI. On the GRF, the LICS messages are processed on the RMS.

GRF implementation features

A GRF router can be simultaneously configured as a Frame Relay router and as a Frame Relay switch. In addition to routing IP and IS-IS traffic over Frame Relay, the GRF provides switching and multicast service features.

The GRF implementation does not support forward explicit congestion notification (FECN) or backward explicit congestion notification (BECN), and does not provide an automated re-routing capability. Frame Relay MIB (RFC 1315) is not supported. Standard LMI (revision 1) is not supported.

Routing

Standard IP routing is supported across Frame Relay links via route circuits (PVCs).

IS-IS is also supported across Frame Relay PVCs.

Switching

The Frame Relay switching feature enables a GRF to function as a switch. When a GRF router functions as a Frame Relay switch, it performs layer-2 switching and forwards incoming data from incoming circuits to the appropriate out-going circuits without touching the payload of the data packets.

Frame Relay switching is supported on the HSSI and SONET media cards. An entire frame relay network can be built by connecting GRFs using high-speed links.

Multicast service

Multicast service enables a GRF to function as a Frame Relay multicast server. As a multicast server, the GRF receives multicast data messages from one incoming circuit and forwards the data messages to a group of outgoing circuits.

(Note that Frame Relay multicast is not the same as IP multicast.)

Link options

A link is a HSSI or SONET interface. Each link can be configured as:

- UNI-DTE (a router link)
- UNI-DCE (an access switch)
- NNI (a switch, internal to a Frame Relay network or between Frame Relay networks)

HSSI and SONET media cards support the configuration of both switch and route circuits on the same Frame Relay link. Switch and route circuits can both be configured on either a UNI-DCE or NNI link. Only route circuits can be configured on a UNI-DTE link.

Circuits

HSSI cards provide two physical interfaces, the SONET card provides a single physical interface. Each interface supports 975 Data Link Circuit Identifiers (DLCIs), numbered 16 through 991, which excludes those used for Local In-Channel Signaling (LICS).

Circuits are configured to switch, to route, or to multicast.

- A circuit is configured to switch where two segments of a Frame Relay circuit come together.
- A circuit is configured to route at the endpoint of a Frame Relay circuit.
- A multi cast circuit allows an inbound packet to be replicated to multiple destinations.

A circuit can be enabled or disabled, added or deleted, on-the-fly, via **grfr -c cxx** commands. Statistics are individually kept for each circuit and are also displayed using the **grfr -c dxx** commands. The **grfr** command is described later in his chapter.

Traffic shaping

Traffic shaping is assigned on a per-circuit basis. Three traffic shaping parameters can be configured:

- Committed Information Rate (CIR)
- Burst Excess (Be)
- Committed Burst (Bc)

Each circuit is guaranteed a certain bandwidth, the Committed Information Rate, or CIR.

Each circuit is allowed to consume bandwidth beyond the CIR to a second threshold, CIR+Be (Burst Excess), above which all packets are dropped.

Between the CIR and CIR+Be, packets are no longer guaranteed, and may be dropped by a congested network. These packets are considered Discard Eligible, and are marked as such with the DE bit set in the Frame Relay header.

The GRF supports asymmetric PVCs such that users can configure a different set of traffic shaping parameters on each direction of a circuit.

LICS protocols

The GRF implementation supports the following Local In-Channel Signaling (LICS) protocols:

- ANSI T1.617 Annex D
- CCITT Q.939 Annex A
- LICS disabled

Here are the supported Annex D and Annex A link parameters:

- N391
- N392
- N393

- T391
- T392

Interoperability

The GRF interoperates with any networking equipment that supports HSSI and SONET media interfaces. The connecting device can run Annex A, Annex D, or no LICS protocols.

Note: There is no standard mechanism defined for carrying Frame Relay over SONET.

Specifications

The Frame Relay Forum is the primary source of Frame Relay specifications.

These specifications are available at:
`http://www.frforum.com/5000/5000index.html.`

IS-IS protocol support

HSSI and SONET cards support IS-IS over Frame Relay.

IS-IS is a link state interior gateway protocol (IGP) originally developed for routing ISO/CLNP (International Organization for Standardization / Connectionless Network Protocol) packets. In ISO terminology, a router is referred to as an “intermediate system” (IS). IS-IS intra-domain routing is organized hierarchically so that a large domain may be administratively divided into smaller areas using level 1 intermediate systems within areas and level 2 intermediate systems between areas.

Refer to the *Introduction to IS-IS* chapter for more information.

Introduction to fred

The GRF Frame Relay daemon is known as **fred**. This program is responsible for configuring, administering, and monitoring Frame Relay interfaces and circuits on the media cards. **fred** is the source or destination of all Local In-Channel Signaling (LICS) packets.

The Frame Relay link and circuit configuration file is `/etc/grfr.conf`. . **fred** reads this file and other internal data structures to build PVC and link tables.

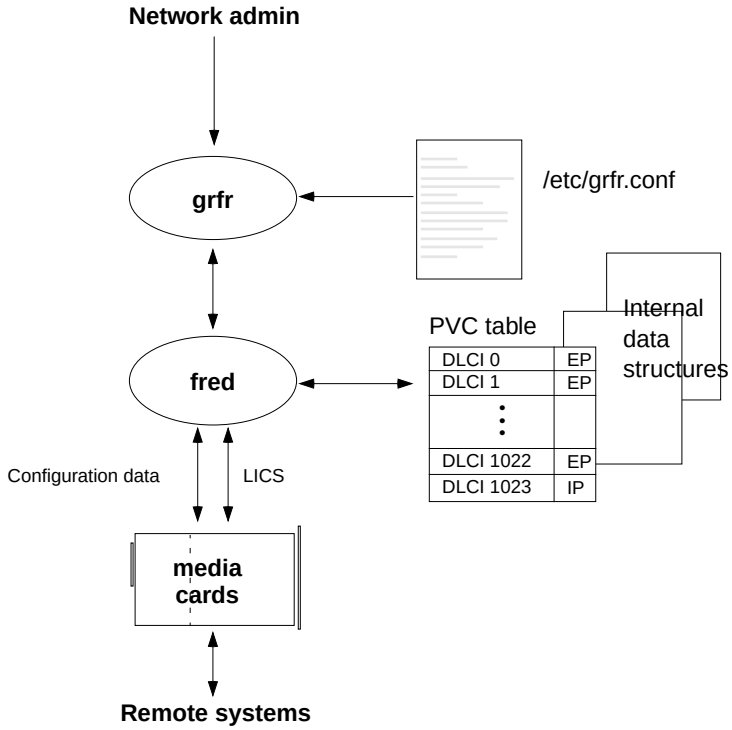
grfr is the program that changes Frame Relay configuration and also displays current Frame Relay status. Commands in the format **grfr -c cxx** add, enable, disable, or delete PVCs and links, or modify link configurations. Commands in the format **grfr -c dxx** display link and PVC status and statistics.

To attach route circuits to logical interfaces, **fred** reads the GRF IP address file, `/etc/grifconfig.conf`. Frame Relay error and event messages are collected by **fred** and sent to `var/log/fred.log`.

PVC and link tables

fred, via **grfr**, gets configuration data from the `/etc/grfr.conf` file to build the link table and a corresponding PVC table. Then **fred** downloads a copy of the PVC table to each Frame Relay media card. **fred** updates the tables from incoming LICS messages and from media card status (link up/down) messages, and updates the copies on the media cards.

The link table contains 32 entries, two ports for each of 16 possible slots. Each link entry includes data from the Link section of the `grfr.conf` file, and has a pointer to a corresponding PVC table.



Route circuits

Route circuits are defined using the `pvc` or `pvcr` keyword in `grfr.conf`. A route circuit may be configured on any link.

Switch circuits

A switch circuit is composed of two segments, each on a different link. Switch circuits are defined using the `pvcs` keyword in `grfr.conf`. A switch circuit may be configured on a UNI-DCE or NNI link.

Multicast circuits

A multicast circuit may be configured on a UNI-DCE or NNI link.

For One-Way multicast, a unicast circuit must already exist between the root and each of the multicast members. These multicast circuits are defined using the `pvcm1` keyword in `grfr.conf`.

For Two-Way multicast, a unicast circuit is added for each member of the multicast group. These multicast circuits are defined using the `pvcm2` keyword in `grfr.conf`.

N-Way multicast circuits are defined using the `pvcmn` keyword in `grfr.conf`.

LICS processing

Local In-Channel Signaling (LICS) processing is implemented in accordance with specifications from the Frame Relay Forum, and the Sprint Frame Relay Switch Specification (5404.03). CCITT Q.933 Annex A and ANSI T1.617 Annex D are implemented. LICS can also be disabled.

On each link, one circuit is reserved for LICS traffic. LICS procedures are performed on all link types, UNI-DTE, UNI-DCE, or NNI.

From a link configured as a UNI-DTE, **fred** sends a poll (status enquiry) to the UNI-DCE to which it is attached every *T391* seconds. Every *N391* polls, the status enquiry message is for a full status report. The event monitoring period is *N393* polls (a sliding window). If there is no response to *N392* polls during a monitoring period, the link is considered down. A full status message contains information about all the circuits on the link.

From a link configured as a UNI-DCE, **fred** responds to polls from the UNI-DTE to which it is attached. If a poll is not received within *T392* seconds of the last poll, or the sequence number is incorrect, an error is logged. The event monitoring period is *N393* polls received or missed (a sliding window). If *N392* errors are logged during an event monitoring period, the link is considered down.

From a link configured as an NNI link, **fred** both polls and answers polls from the switch to which it is attached.

fred keeps timers for each link to trigger a poll and to note missed polls.

Traffic shaping

Traffic shaping is performed by monitoring the amount of traffic being transmitted on each circuit.

Traffic shaping is assigned on a per-circuit basis. The GRF supports asymmetric PVCs such that users can configure a different set of traffic shaping parameters on each direction of a circuit. Three traffic shaping parameters can be configured:

- Committed Information Rate (CIR)
- Burst Excess (Be)
- Committed Burst (Bc)

grfr command functions

The **grfr** command provides a way to display configuration information, status and statistics of switch circuits, multicast group and modify configurations.

Functions include

- Configuration information to aid debugging includes link parameters, circuit endpoint parameters, a switch circuit, or a multicast group.
- Status information to aid debugging and provide data for analysis and reports includes statistics for a link, a switch circuit, and a multicast group.
- Temporary and minor configuration changes, such as enabling or disabling a circuit or endpoint, adding or deleting a switch circuit can be made using **grfr**. Permanent and major changes must be made via the `grfr.conf` file.

Examples of **grfr** display and configuration commands are found at the end of this chapter and in the *GRF Reference Guide*.

Debug and log levels

Four debug levels (1 to 4) manage event logging. Level 1 logs the lowest number of debug messages and level 4 provides the highest, level 1 is set by default. Log messages are written by default to the `/var/log/fred.log` file. You can set and change debug level on-the-fly using the **grfr** command **grfr -c csd -d *level***.

Level 1 - logs error and main transition events such as link active and inactive. Use this level for normal operations. You can change it on-the-fly.

Level 2 - logs all events related to the LMI protocols. These include sending, receiving, status enquiries, and status responses.

Level 3 - logs same events as in level 2, but provides more details and includes the contents (in hex) of all messages sent and received.

Level 4 - log messages include all activities to and from the media card.

Multicast service

Frame relay multicast service enables a GRF router to function as a multicast server. A multicast server is a system (a GRF) or switch that receives multicast data messages from one incoming circuit and forwards the data messages to a group of out-going circuits. Multicast services are supported on switch circuits only.

Frame Relay provides the three types of multicast service defined by the Frame Relay Forum in *Frame Relay PVC Multicast Service and Protocol Description*:

- One-Way
- Two-Way
- N-Way

In Frame Relay multicast, one switch (a node) within the network is designated as a "Multicast Server" and provides the multicast service. Messages to be multicast are first sent to the multicast server and then, at the multicast server, the messages are replicated and sent to members of the multicast group.

Frame Relay uses the term "upstream circuit" to refer to a circuit where a multicast server *receives* multicast messages. The term "downstream circuit" refers to a circuit where a multicast server *sends* multicast messages. These terms are also applicable to a switch circuit.

In both One-Way and Two-Way Multicast, one station acts as the root station.

One-way multicast

In One-way multicast, the root station sends traffic on a special circuit that delivers the data to all the other members of the multicast group.

This method requires that a unicast circuit also exist between the root station and each member of the group. Each member of the group receives its multicast packets on the unicast circuit, as if it had been sent by the root on that circuit. If the members of the group wish to communicate something back to the root, they send that traffic back on the unicast circuit. The root receives this traffic on the non-multicast circuits, not the unicast circuit.

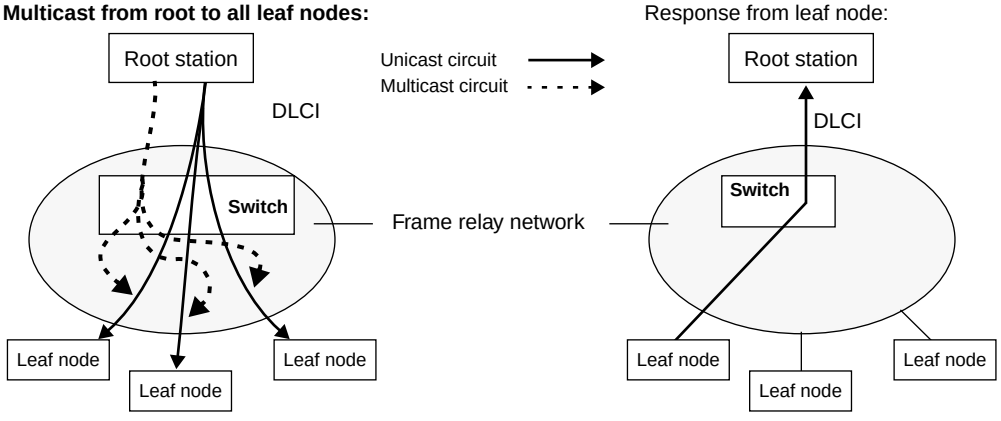


Figure 6-3. Diagram of one-way multicast circuits

Two-way multicast

In Two-way multicast, unicast circuits are not required between the root station and the members of the multicast group, but such circuits are permitted.

All members and the root use a special multicast circuit. Data transmitted by the root goes to all the members. Data transmitted by the members is sent only to the root using the multicast circuits.

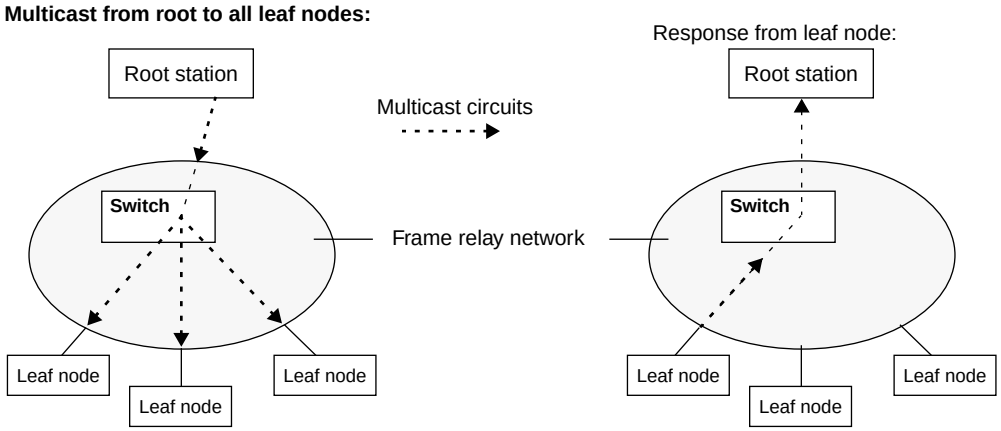


Figure 6-4. Diagram of two-way multicast circuits

N-way multicast

In N-Way multicast, all members of the group are peers. All members have special multicast circuits. Any data sent on these multicast circuits gets sent to all the other members of the group.

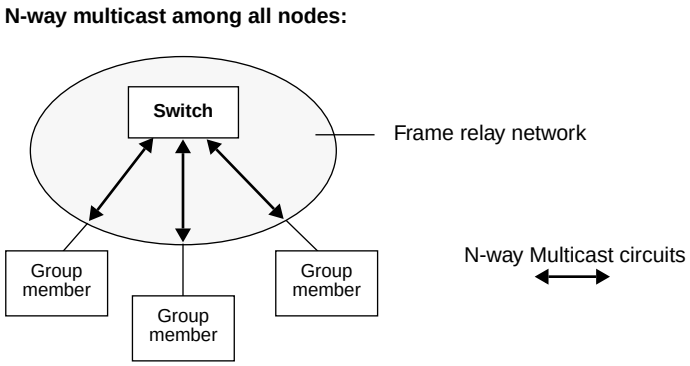


Figure 6-5. Diagram of N-way multicast circuits

Before you start...

Before you configure the Frame Relay protocol, be sure you configure the media cards themselves.

Card profile parameters

You must set SONET and/or HSSI parameters in the Card profile.

These parameters include framing protocol, CRC, and internal clock. Refer to the HSSI and SONET chapters in the *GRF Configuration Guide* for more information.

IP address assignment

Identify the endpoint router logical interfaces in `/etc/grifconfig.conf`.

Interface 0 configuration requirement

The following configuration step is required for Frame Relay to operate properly.

When running Frame Relay mode, the first logical interface on each HSSI and SONET physical interface must be configured. This enables the Frame Relay daemon (**fred**) to properly communicate LICS traffic via the card.

For example, if a HSSI card in slot 2 is configured for Frame Relay, then the following interfaces need to be configured:

`gs020` - slot 2, physical interface 0, logical interface 0
`gs0280` - slot 2, physical interface 1, logical interface 0

Only one entry is required for a SONET card. Use dashes in place of IP address, netmask and destination address. Specify `up` in the argument field. Here is an example:

```
# name address  netmask broad_dest argument
gs020 - - - up
gs0280 - - - up
```

Configure Frame Relay logging

You must start Frame Relay logging the first time you configure Frame Relay.

During site installation, system logging must be configured, it does not begin automatically. The *GRF 400/1600 Getting Started* and *GRF Configuration Guide* both describe how to configure logging to an external device.

These are the steps specifically required to configure Frame Relay logging.

1 Create the `fred.log` directory:

```
super> sh
# cd /var/log
# touch fred.log
```

2 Edit the `/etc/syslog.conf` file to have **syslogd** log to `fred.log`:

```
# cd /
# cd /etc
# vi syslog.conf
```

The entries should look like the following:

```
*.err;*.notice;kern.debug;lpr,auth.info;mail.crit      /var/log/messages
cron.info                                   /var/log/cron
local0.info                                 /var/log/gritd.packets
local1.info                                 /var/log/gr.console
local2.*                                    /var/log/gr.boot
local3.*                                    /var/log/grinchd.log
local4.*                                    /var/log/gr.conferrs
local5.*                                    /var/log/mib2d.log
```

Add the following line at the end of the file:

```
local6.*                                    /var/log/fred.log
```

Save the file and exit.

3 Modify `/etc/grclean.conf` to specify a size limit for `fred.log`:

```
# vi syslog.conf
```

The file entries should look like the following:

```
size=10000
logfile=/var/log/cron
size=10000
logfile=/var/log/aitmd.log
size=10000
logfile=var/log

**********************************************************************
```

Add a `fred.log` entry after the `var/log` entry. An example is shown below.
The file size (in K) you specify will depend upon the available memory resources.

```
size=1000
logfile=/var/log/fred.log
```

4 Save all changes and reboot:

```
# grwrite -v
# reboot -i
```

If you are upgrading software rather than doing an initial installation, you will have to signal (HUP) **syslogd** to re-read the `syslog.conf` file so the Frame Relay changes are incorporated.

The next several sections provide configuration examples for sections in the `grfr.conf` file.

Configuring link parameters

Configure link parameters in the `/etc/grfr.conf` configuration file. Please see the *GRF Reference Guide* for a template of this and all other GRF configuration files.

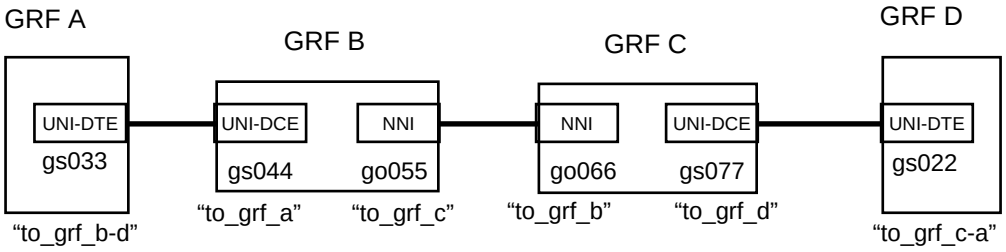
Link parameters are set in the *Link Section*. On each link you can configure the following:

- Link descriptors - *required*
  Specify link slot and port numbers in decimal.
- Link type - *required* for UNI-DCE and NNI links, default is UNI-DTE
  Specify type.
- LMI type - *required* for AnnexA and AnnexD, default is none
  Specify type.

- Link name - *optional*
  Each link can be named for convenience.
- Enabled Y|N - *optional*
  Enable link, default is Y.
- T391 - *optional*
  Heartbeat poll interval. Default is 10.
- N391 - *optional*
  Status poll intervals. Default is 6.
- T392 - *optional*
  Poll verification timer. Default is 15.
- N392 - *optional*
  Error reporting threshold. Default is 3.
- N393 - *optional*
  Measurement interval for Mn2. Default is 4.
- AutoAddGrif - *optional*
  Enables remote devices to assign a PVC to a GRF interface. Default is no.

Link configuration example

In this example, six links need to be configured:



The GRF A, B, C, and D examples below show the Link section entries in `/etc/grfr.conf` and the interface/logical interface 0 requirements in `/etc/grifconfig.conf`. Note that SONET cards have only one entry in `/etc/grifconfig.conf`.

GRF A

- `/etc/grfr.conf` Link entries:

```
#Slot Port Optional Parameters
#==== ==== ==================
link 3 0 name="to_grf_b-d" LMIType=AnnexD
```

- `/etc/grifconfig.conf` Interface entries:

```
# name address  netmask broad_dest argument
gs030 - - - up
gs0380 - - - up
gs033 192.168.0.1 255.255.255.0
```

GRF B

- `/etc/grfr.conf` Link entries:

```
#Slot Port Optional Parameters
#==== ==== ==================
link 4 0 name="to_grf_a" Linktype=UNI-DCE LMIType=AnnexD
link 5 0 name="to_grf_c" Linktype=NNI LMIType=AnnexD
```

- `/etc/grifconfig.conf` Interface entries:

```
# name address  netmask broad_dest argument
gs040 - - - up
gs0480 - - - up
go050 - - - up
gs044 192.168.10.1 255.255.255.0
go055 192.168.10.2 255.255.255.0
```

GRF C

- `/etc/grfr.conf` Link entries:

```
#Slot Port Optional Parameters
#==== ==== ==================
link 6 0 name="to_grf_b" Linktype=NNI LMIType=AnnexD
link 7 0 name="to_grf_d" Linktype=UNI-DCE LMIType=AnnexD
```

- `/etc/grifconfig.conf` Interface entries:

```
# name address  netmask broad_dest argument
go060 - - - up
gs070 - - - up
gs0780 - - - up
go066 192.168.20.1 255.255.255.0
gs077 192.168.20.2 255.255.255.0
```

GRF D

- `/etc/grfr.conf` Link entries:

```
#Slot Port Optional Parameters
#==== ==== ==================
link 2 0 name="to_grf_c-a" LMIType=AnnexD
```

- `/etc/grifconfig.conf` Interface entries:

```
# name address  netmask broad_dest argument
gs020 - - - up
gs0280 - - - up
gs022 192.168.30.1 255.255.255.0
```

Link type

Specify a link to be UNI-DTE, UNI-DCE, or NNI.

T391 - link integrity verification timer

T391 represents the Link Integrity Verification timer. This link option specifies how long (*T391 seconds*) a device waits before sending a poll (a status inquiry message).

- A UNI-DTE sends polls to the connected UNI-DCE.
- Two NNIs send polls to each other.
- A UNI-DCE does not send polls to a UNI-DTE.

N391 - full status polling cycle

N391 represents the Full Status Polling cycle. This link option specifies that every *N391* polls, a full status report is requested.

T392 - polling verification timer

T392 represents the Polling Verification timer. This link option specifies the number of seconds to wait for an expected poll. If a poll is not received within *T392* seconds of the previous poll, a missed poll error is logged.

N392 - error threshold

N392 represents the Error Threshold number. This link option specifies the number of missed poll errors in a single monitoring period before the link is taken down.

N393 - monitored events count

N393 represents the Monitored Events count. This link option determines the length of a monitoring period. Each period is actually a sliding window that is *N393* events long, where an event is a received poll, or a missed poll.

Link name

Each link can be named for administrative convenience, a name is optional.

LMI type

Specify a link to be AnnexA or AnnexD, default is none.

Configuring circuit parameters

You automatically specify circuit type by the section of `/etc/grfr.conf` file in which you configure the circuit:

- Route - PVC section
- Switch - PVCS section
- 1-way multicast - PVCM1 section
- 2-way multicast - PVCM2 section
- N-way multicast - PVCMN section
- Endpoint parameters - PVCEP section

You can also configure ATMP PVCs in the PVCATMP section of `/etc/grfr.conf`.

Route circuits - PVC/PVCR section

The keywords PVC and PVCR are interchangeable and are processed for configuration purposes in exactly the same way.

Configure these route circuit parameters in the PVC Section of `grfr.conf`:

- Logical Interface (`lif`) - *required*

  Circuits are grouped onto logical interfaces. You can have all circuits on a given link on the same logical interface, or each circuit on its own logical interface, or any grouping in between.
- Endpoint - *required*

  Specify the DLCI of this circuit, which ends here at the router.
- Peer IP Address - *required*

  Specify the IP address of the host or router at the other end of this circuit. If this parameter is set to 0.0.0.0, Inverse ARP is used to determine the IP address.
- Name - *optional*

  Each route circuit can be named for convenience.
- Enabled Y|N - *optional*

  Enable circuit, default is Y.
- IS-IS Y|N - *optional*

  Enable IS-IS, default is N.
- Traffic shaping parameters - *optional*

  - CIR - Committed Information Rate

  Specify the bits per second that the network should be able to deliver on this circuit without dropping packets.

  The sum of the CIR values on all circuits on a link should not exceed the bandwidth of the link. Default is 55Mb (55000000).

  - Bc - Committed Burst Size

  Specify the amount of data (in bits) that the network should be able to deliver on this circuit without dropping packets during a fixed period of time: Tc. Tc = Bc/CIR.

Typically, `Bc` is set to be the same as `CIR`, for a `Tc` of 1 second. Default is 55Mb (55000000).

– Be - Excess Burst Size

Specify the amount of data (in bits) above `Bc` that the network will attempt to deliver on this circuit during the time period `Tc`.

This data is eligible for discard if the network becomes congested. Default is 0.

Example

In this example, two route circuits need to be configured. One from GRF A to GRF B, and another from GRF D to GRF C:

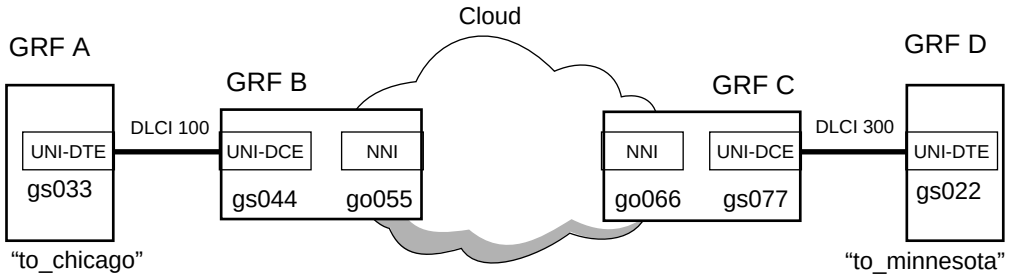


Figure 6-6. Route circuit configuration example

The GRF A and GRF D examples below show the Link section entries in `/etc/grfr.conf` and the interface/logical interface 0 requirements in `/etc/grifconfig.conf` files.

GRF A

- `/etc/grfr.conf` Link entries:

```
#lif DLCI Peer IP Address Optional Parameters
#=== ==== =============== ==================
pvcr gs033 100  192.168.8.8 Name="to_chicago" isis=Y
```

- `/etc/grifconfig.conf` Interface entries:

```
# name address  netmask broad_dest argument
gs030 - - - up
gs0380 - - - up
gs033 192.168.2.2 255.255.255.0
```

GRF D

- `/etc/grfr.conf` Link entries:

```
#lif DLCI Peer IP Address Optional Parameters
#=== ==== =============== ==================
pvcr gs022 300  192.168.2.2 Name="to_minnesota" isis=Y
```

- `/etc/grifconfig.conf` Interface entries:

```
# name address  netmask broad_dest argument
gs020 - - - up
gs0280 - - - up
gs022 192.168.8.8 255.255.255.0
```

Switch circuits - PVCS section

Configure these switch circuit parameters in the PVCS Section of `grfr.conf`:

- Segments (endpoints) - *required*

  Specify the chassis slot, card port, and DLCI of each of the two segments of the circuit that meet here at the switch.

- Name - *optional*

  Each switch circuit can be named for convenience.

- Enabled Y|N - *optional*

  Enable circuit, default is Y.

- Traffic shaping parameters - *optional*

  Same as for route circuit (PVC) above.

Example

In this example, two switch circuits need to be configured. A switch circuit is composed of two segments, each on a different link.

One circuit is the segment pair, GRF B–GRF A and GRF B–GRF C.
The other is the segment pair GRF C–GRF B and GRF C–GRF D:

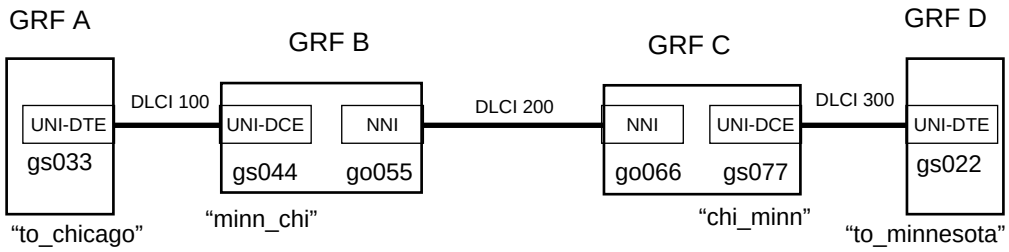


Figure 6-7. Switch circuit configuration example

Here are the PVCS section entries in `/etc/grfr.conf` for each GRF:

GRF B

```
#     EPA      EPB      Optional Parameters
#     ===      ====     ===================
pvcs  4:0:100  5:0:200  Name="minn_chi" CIR=1500000 Bc=1500000 Be=75000
```

GRF C

```
#     EPA      EPB      Optional Parameters
#     ===      ====     ===================
pvcs  6:0:200  7:0:300  Name="chi_minn" CIR=1500000 Bc=1500000 Be=75000
```

One-way multicast - PVCM1 section

In one-way multicast, the root node can send to all leaf nodes. A leaf node can respond to only the root, and only on its unicast circuit.

Configure these one-way multicast parameters in the PVCM1 Section of `grfr.conf`:

- First entry (EPR) must be root circuit endpoint - *required*
  Specify the chassis slot, card port, and DLCI of root endpoint.
- Next *n* entries are the endpoints of each member of the group - *required*
  Specify the chassis slot, card port, and DLCI of *n* leaf endpoints.
- Name - *optional*
  Each multicast group can be named for convenience.
- Enabled Y|N - *optional*
  Enable multicast group, default is Y.
- Traffic shaping parameters - *optional*
  Same as for route circuit (PVC) above.

Example

In this example, a node is the root station for a one-way multicast group consisting of the two leaf nodes. GRF 2 is a Frame Relay switch, and forms a one-node network.

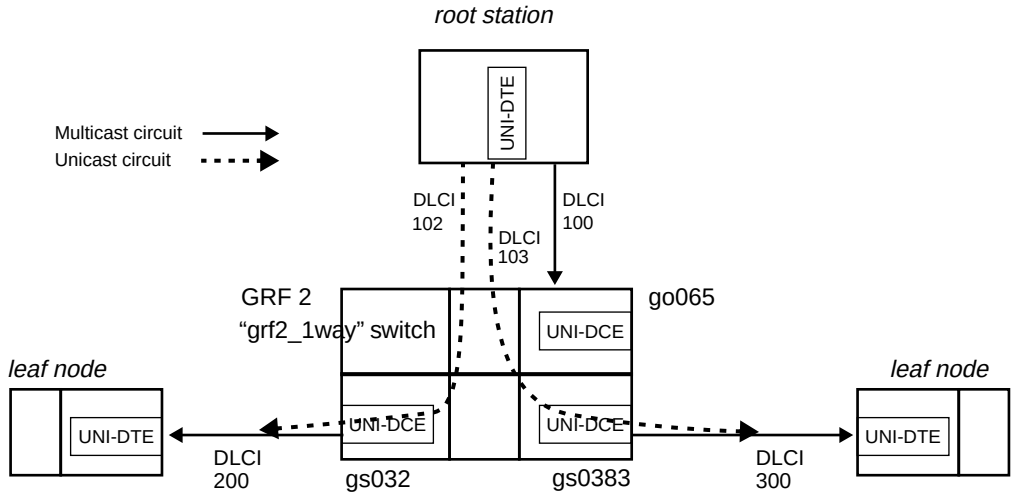


Figure 6-8. One-way multicast example

There are three configuration steps:

1 Configure the unicast circuits in the PVCS section of `/etc/grfr.conf`.

```
      EPA     EPB      Optional Parameters
      ====    ===      ===================
pvcs 6:0:102 3:0:200
pvcs 6:0:103 3:1:300
```

2 Configure the PVCM1 entry in `/etc/grfr.conf` to create the multicast group.

```
      EPR      EP1     EP2       Optional Parameters
      ===      ====    ===       ===================
pvcm1 6:0:100  3:0:200  3:1:300  Name="grf2_1way" CIR=64000 Bc=64000 Be=0
```

3 Configure the interface and logical interface 0 entries in `/etc/grifconfig.conf`.

```
# name address  netmask broad_dest argument
go060 - - - up
gs030 - - - up
gs0380 - - - up
```

Two-way multicast - PVCM2 section

Configure these two-way multicast parameters in the PVCM2 Section of `grfr.conf`:

- First entry (EPR) must be root circuit endpoint - *required*
  Specify the chassis slot, card port, and DLCI of root endpoint.
- Next *n* entries are the endpoints of each member of the group - *required*
  Specify the chassis slot, card port, and DLCI of *n* leaf endpoints.

- Name - *optional*
  Each multicast group can be named for convenience.
- Enabled Y|N - *optional*
  Enable multicast group, default is Y.
- Traffic shaping parameters - *optional*
  Same as for route circuit (PVC) above.

Example

In this example, a node is the root station for a two-way multicast group consisting of two leaf nodes. GRF 2 is a Frame Relay switch, and functions as a one-node network.

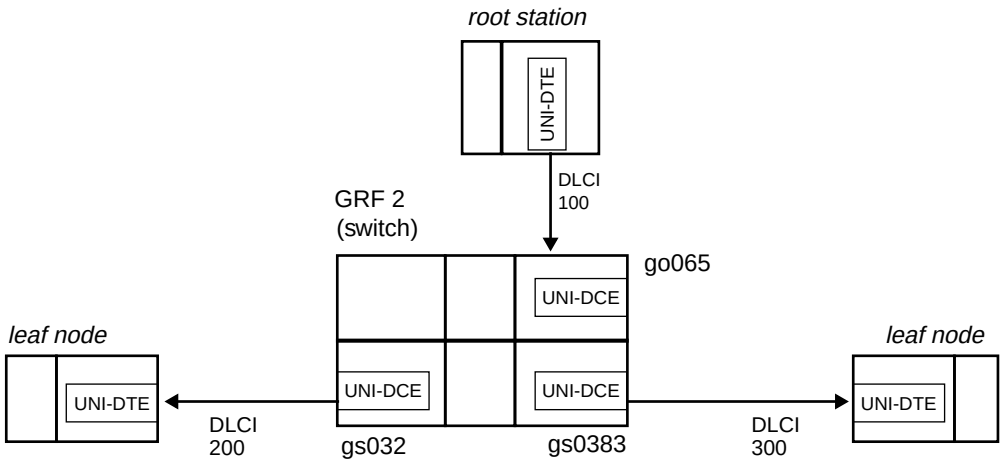


Figure 6-9. Two-way multicast example

1 Configure the PVCM2 entry in `/etc/grfr.conf` to create the group:

```
      EPR      EP1      EP2     Optional Parameters
      ===      ====     ===     ===================
pvcm2 6:0:100  3:0:200  3:1:300  Name="grf2_1way" CIR=64000 Bc=64000 Be=0
```

2 Configure the logical interface 0 entries in `/etc/grifconfig.conf`:

```
# name address  netmask broad_dest argument
go060 - - - up
go0680 - - - up
gs030 - - - up
gs0380 - - - up
```

N-way multicast - PVCMN section

In N-way multicast thare are no leaves, no root, all are equivalent multicast nodes.

Configure these N-way multicast parameters in the PVCMN Section of `grfr.conf`:

- Enter the endpoints of *n* members of the group - *required*
  Specify the chassis slot, card port, and DLCI of *n* member endpoints.

- Name - *optional*
  Each multicast group can be named for convenience.
- Enabled Y|N - *optional*
  Enable multicast group, default is Y.
- Traffic shaping parameters - *optional*
  Same as for route circuit (PVC) above.

Example

In this example, three nodes are in an N-way multicast group. GRF 2 is a Frame Relay switch.

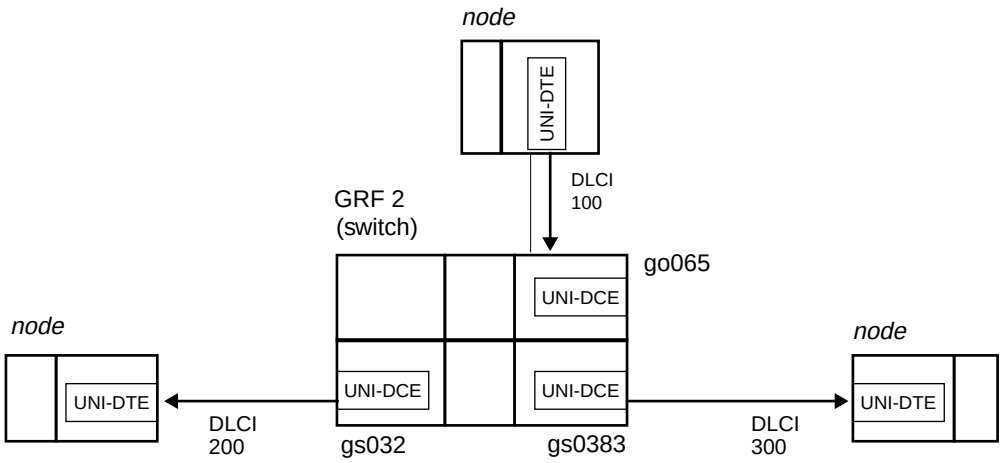


Figure 6-10. N-way multicast example

1 Configure the PVCMN entry in `/etc/grfr.conf` to create the group:

```
      EPR       EP1     EP2     Optional Parameters
      ===       ====    ===     ===================
pvcmn 6:0:100  3:0:200  3:1:300  Name="grf2_nway" CIR=64000 Bc=64000 Be=0
```

2 Configure the logical interface 0 entries in `/etc/grifconfig.conf`:

```
# name address  netmask broad_dest argument
go060 - - - up
go0680 - - - up
gs030 - - - up
gs0380 - - - up
```

Asymmetrical traffic shapes

Configure different traffic shaping parameters for each individual endpoint on a link (an asymmetric circuit) in the PVCEP Section of `grfr.conf`:

- Define target endpoint - *required*
  Specify the chassis slot, card port, and DLCI of endpoint.
- Traffic shaping parameters - *required*
  Same as for route circuit (PVC) above.

Example

This example gives the circuit going to GRF 4 about 2Mb of bandwidth, the traffic coming back to GRF 3 gets 64 Kb.

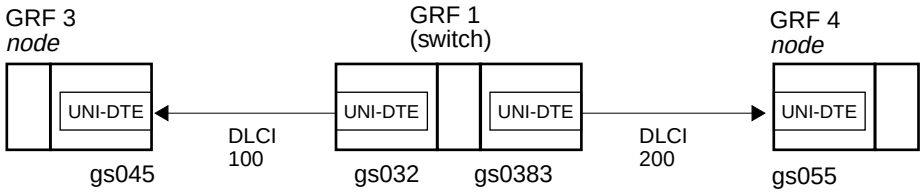


Figure 6-11. Asymmetrical traffic shape example

1 Configure the switch circuit that connects GRF 3 and GRF 4 in the PVCS section of `/etc/grifconfig.conf`:

```
#     EPA      EPB      Optional Parameters
#     ===     ====     ===================
pvcs 4:0:100  5:0:200  Name="grf3-grf4" CIR=64000 Bc=64000 Be=2400
```

2 Configure the endpoint circuit that sends to GRF 3 in the PVCEP section of `/etc/grifconfig.conf`:

```
#     EP       CIR         Bc         Be
#     ===      ====        ===        ===
pvcep 4:0:100  CIR=2000000 Bc=2000000 Be=9600
```

Configure a link on-the-fly

You can add or delete Frame Relay links, or modify a link parameter, without resetting the media card. This example adds a UNI-DTE link on interface `gs073`, DLCI 122.

Here are the steps:

1 In the appropriate Card profile, specify framing protocol, CRC, and clock as required.

2 Configure the logical interface in `/etc/grifconfig.conf`.
Remember that you must also create an entry for logical interface 0 in this file.
Start the UNIX shell and edit `/etc/grifconfig.conf`:

```
super> sh
# vi /etc/grifconfig.conf
# name address  netmask broad_dest argument
#
gs070 - - - up
gs073 192.168.3.3 0.0.0.0
```

Save the file and exit **vi**.

3 Edit the `/etc/grfr.conf` file to add the link in the Link section.

```
# vi /etc/grfr.conf

#Slot Port Optional Parameters
#==== ==== ==================
link 7 1 name="new_link" LMIType=AnnexD
```

Add the logical interface 0 as an active PVC:

```
#   lif    DLCI Peer IP Address Optional Parameters
#   ===    ==== =============== ===================
pvc gs073  122  0.0.0.0  Name="new_link"
```

Save the file and exit **vi**.

4 Use the **grfr -c ccl** command to enable link on slot 7, port 1.

```
# grfr -c ccl -s 7 -l 1
```

Use another **grfr** command to check the status of the new link:

```
# grfr -c dlc

C O N F I G U R E D   L I N K S :
================================

Name:    S/P:  LMI:     Link:    Autogrif: N391: N392: N393: T391: T392: S:
----     ---   ---      ----     --------  ----  ----  ----  ----  ----  --

new_lnk  7/1  ANNEX-D  UNI-DTE  None       6     3     4     10     15   Ae

Total: 1 links configured
```

Configure a PVC on-the-fly

You can add or delete PVCs without resetting the media card by editing the `/etc/grfr.conf` file and then using **grfr -c ccp** to add a PVC or **grfr -c crp** to disable a PVC.

To add a PVC to the HSSI card in slot 13, start the UNIX shell and edit `/etc/grfr.conf`.

To add any additional interfaces, you must edit `/etc/grifconfig.conf`.

```
super> sh
# vi /etc/grfr.conf
```

Then make the PVC entry as usual:

```
#   lif    DLCI Peer IP Address Optional Parameters
#   ===    ==== =============== ===================
pvc gs0d0  606  0.0.0.0   Name="test606"
```

Save the file and exit **vi**.

Use the **grfr -c ccp** command to add a PVC. The configuration file and the PVC DLCI, slot, and link must be specified:

```
# grfr -c ccp -f /etc/grfr.conf -i 606 -s 13 -l 0
```

Here is the response:

```
grfr Adding type 1, lif=gs0d0, dlci=606, peer_ip=0.0.0.0
        Slot =13, link =0, name=test606
PVC slot 13, link 0, dlci 606 defined
```

To delete (disable) a PVC, you do not need to edit the `/etc/grfr.conf` file, the **grfr -c crp** command is sufficient. Specify the target DLCI, slot, and port number to be disabled:

```
# grfr -c crp -i 600 -s 13 -l 0
```

Here is the response:

```
PVC slot 13, link 0, dlci 600 deleted
```

Assigning multiple route PVCs to an interface

DLCIs map point-to-point. One DLCI maps a unique circuit between two endpoints, and so only one destination can be assigned on a given DLCI.

The 0.0.0.0 notation is treated specially in that it says instead of hard-coding the ARP entry for the other end of the circuit, obtain it by sending an inverse ARP to the other end and see what comes back.

If the peer IP addresses are in the same subnet, you can assign multiple DLCIs to the interface:

```
#    lif    DLCI  Peer IP address
PVC  gs047e  405  222.222.10.5
PVC  gs047e  406  222.222.10.6
PVC  gs047e  407  222.222.10.7
PVC  gs047e  408  222.222.10.8
```

If the peer IP addresses are in different subnets, you need multiple interfaces:

```
#    lif   DLCI  Peer IP address
PVC  gs040  405  222.222.10.5
PVC  gs041  406  222.222.11.5
PVC  gs042  407  222.222.12.5
PVC  gs043  408  222.222.13.5
```

Verifying a configuration

The **grfr** display commands return system and interface levels of information that can help you review a Frame Relay configuration.

The **grfr -c dsc** command displays the system-wide Frame Relay configuration.

```
S Y S T E M   P A R A M E T E R S :
==================================
Name:........................ X
Start Time ................... Wed Jan 28 11:13:52 CST 1998
Up-time ...................... 4 days, 20 hours, 10 mins, 57 secs
Configuration File ........... /etc/grfr.conf
grif Configuration File ...... /etc/grifconfig.conf
Debug Level................... 1
Statistics Interval........... 10
Portcard Heartbeat Interval.... 10
Media Types Supported......... HSSI, SONET-OC3
Boards configured ............ 2
Links  configured ............ 4
PVCs  configured ............. 9
   Routed PVCs  configured .... 5
   Switched PVCs  configured .. 4
   Mcasted  PVCs  configured .. 0
   ATMP PVCs configured ....... 0
Active Links ................. XX
Active PVCs .................. XX
```

Active link and PVC data are not available using this command option.

The **grfr -c -dpc** command displays a list of configured PVCs and their configuration parameters.

```
#grfr -c dpc

C O N F I G U R E D   P V C s   :
================================

(A* = Autoadded, D* = Deleted)

Name        Slot  Port  DLCI   Type    CIR   Bc    Be    State     EPs/ISIS
----        ---   ----  ----   ----    ----  ---   ---   -----     --------

13:0:0      13    0     0      Switch  56K   56K   56K   Active    13:0:0
M1-Group    13    0     311    Mcast-R 56K   56K   56K   Active    13:0:312
                                                                   13:0:313
                                                                   13:0:314
M1-Group    13    0     312    Mcast-L 56K   56K   56K   Active    13:0:311
M1-Group    13    0     313    Mcast-L 56K   56K   56K   Active    13:0:311
```

```
M1-Group    13    0     314    Mcast-L 56K   56K   56K   Active    13:0:311
Circ-1      13    0     600    Route   56K   56K   56K   Active    ISIS
Circ-2      13    0     601    Route   56K   56K   56K   Active    ISIS
Circ-3      13    0     602    Route   56K   56K   56K   Active    NO-ISIS
Circ-4      13    0     603    Route   56K   56K   56K   Active    NO-ISIS
Circ-5      13    0     604    Route   56K   56K   56K   Inact     NO-ISIS
atmp-1      13    0     609    ATMP    56K   56K   56K   Active

Total  10 PVCs configured
       5 Routed PVCs
       1 Switched PVCs
       4 Multicast PVCs
       1 ATMP PVCs
```

The **grfr -c dps** command displays PVC statistics.

```
C O N F I G U R E D   P V C s   S T A T s:
=========================================
(S=Slot, P=Port, R=receive, T=Transmit)
(TP=Transmitted Packets, TO=Transmitted Octets)

Name       S/P/DLCI   Type   R-Packets  R-Octets  T-Packets  T-Octets  TPed
----       --------   ----   ---------  --------  ---------  --------  ---
0:0:0      00:0:0     Switch  0          0         0          0         0
0:1:0      00:1:0     Switch  0          0         0          0         0
1:0:0      01:0:0     Switch  63925      1001710   63926      948134    0
south-0    01:0:100   Route   44         3872      41         3608      0
lunar      01:0:101   Route   0          0         0          0         0
1:1:0      01:1:0     Switch  0          0         0          0         0
south-1    01:1:16    Route   0          0         0          0         0
regulu     01:1:101   Route   0          0         0          0         0
south-2    01:1:102   Route   0          0         0          0         0
```

The **grfr -c dlc** command displays the link configuration.

```
C O N F I G U R E D   L I N K S :
================================

Name:   S/P:  LMI:   Link:   Autogrif: N391: N392: N393: T391: T392: S:
----     ---   ---    ----   --------  ----  ----  ----  ----  ----  --

jan0     1 /0  ANNEX-D  NNI      None    6     3     4     10    15  Ae
acme     1 /1  ANNEX-D  UNI-DTE  None    6     3     4     10    15  Ae
Jan      2 /0  ANNEX-D  UNI-DCE  None    6     3     4     10    15  Ie
mike     2 /1  ANNEX-A  UNI-DCE  None    6     3     4     10    15  Ie

Total: 4 links configured
```

grfr command set

Display commands

The **grfr** command has display options that return useful information about Frame Relay links. Display commands are prefaced with the **-c** flag and begin with the letter **d**:

-c dsc, display system configuration and status

-c dlc, display link configuration and status

-c dpc, display PVC configuration and status

-c dic, display interface configuration and status

-c dss, display system status

-c dls, display link status

-c dps, display PVC statistics

-c dbs, display board status

Configuration and debug commands

Configuration commands are prefaced with the **-c** flag and begin with the letter **c**:

-c cel, enable link.
Example: enable a link on slot 3, port 1 `# grfr -c cel -s 3 -l 1`

-c cdl, disable link.
Example: disable link on slot 3, port 0 `# grfr -c cdl -s 3 -l 0`

-c cep, enable PVC.
Example: enable PVC slot port DLCI `# grfr -c cep -i 888 -s 3 -l 0`

-c cdp, disable PVC.
Example: disable PVC slot port DLCI `# grfr -c cdp -i 888 -s 3 -l 0`

-c ccp, configure PVC configuration file slot port DLCI:
Example: config PVC `# grfr -c ccp -f /etc/grfr.conf -i dlci`

-c crp, remove PVC, requires the PVC to be specified.
Example: remove PVC `# grfr -c crp -i dlci`

-c ddl, display debug level. Example: `# grfr -c ddl`

-c csd, set debug level, requires **-d** option to specify level 0–4.
Example: `# grfr -c csd -d 3`

Refer to the *GRF Reference Guide* for more information about the **grfr** command.

States of configured PVCs

Some **grfr** commands return state information, these are the current state options:

Active

An active PVC is correctly configured on both endpoints and the circuit is up.

Inactive

An inactive PVC is correctly configured on both endpoints, but the circuit is not up. If all the PVCs on a port show inactive, the cable could be the problem. If only one is reported inactive, it is likely that the endpoint PVC is down.

Deleted

This state is assigned if the configuration exists on the GRF endpoint but is not configured from the remote endpoint.

Disabled

This is a user-initiated state (via **grfr**) that keeps the configuration information in place but does not let the circuit activate. May also be used when doing an on-the-fly configuration via **grfr**.

Enabled

This is a user-initiated state (via **grfr**) that activates a pre-configured circuit. May also be used when doing an on-the-fly configuration via **grfr**.

Transparent Bridging

This section describes the 1.4 GRF bridging implementation and provides configuration information.

These topics are covered:

GRF bridging implementation

The GRF implements IEEE 802.1D transparent bridging on GRF Ethernet and FDDI interfaces, and on ATM OC-3c interfaces using RFC 1483 encapsulated bridging over PVCs.

Transparent bridging provides a mechanism for interconnecting stations attached to physically separate Local Area Networks (LANs) as if they are attached to a single LAN. This interconnection happens at the 802 MAC layer, and is transparent to protocols operating above this boundary in the Logical Link Control (LLC) or Network layers. Participating stations are unable to identify that peers are on anything other than the directly-attached physical media.

The GRF implementation consists of the transparent bridging function described in 802.1D, and does not include any capability for Source Route or Source Route Transparent (SRT) bridge operation.

Feature summary:

- bridging on FDDI, Ethernet, and ATM OC-3c per the 802.1D standard
- participation in 802.1D spanning tree protocol
- layer-2 transparent bridging of MAC frames through the GRF from one interface to another.
- conversion of frames between Ethernet and FDDI formats as necessary
- fragmentation of IPv4 frames if necessary
- simultaneous bridging and routing over the same interface
  (a GRF interface participating in a bridge group can still route normally)
- routing IP to or from a bridge group from any GRF media
- RFC 1483 encapsulated bridging over ATM OC-3c PVCs with either VC-based multiplexing or LLC encapsulation
- multiple independent bridge groups per GRF
- up to 255 GRF interfaces per bridge group

Specifications

The GRF bridging implementation reflects the following documents:

- International Standard ISO/IEC 10038: 1993;
  ANSI/IEEE Standard 802.1D, 1993 edition
- International Standard ISO 8802-2;
  ANSI/IEEE Standard 802-2, 1989 edition
- RFC 1483, J. Heinanen,
  Multiprotocol Encapsulation over ATM Adaptation Layer 5, 07/20/1993.
  Available via ftp at: `ftp://nic.ddn.mil/rfc/rfc1483.txt`

Simultaneous routing and bridging

Ascend's transparent bridging does not preclude the use of IP packet routing on the same physical interface.

Bridging as well as IP version 4 (IPv4) routing can both be enabled on the same physical interface. In this circumstance, the GRF exchanges traffic between bridging domains and routing domains that exist on the same physical media.

A GRF interface may simultaneously bridge layer-2 frames and route layer-3 packets--that is, forward frames destined to a system attached to another LAN at the MAC layer, but still receive IP packets destined for a remote system attached to a non-broadcast GRF interface and route those packets at the IP layer.

This unique capability eliminates the need for separate pieces of routing equipment to transport packets inter-domain.

To perform the simultaneous functions, the GRF bridging interface examines the destination MAC address of each arriving frame. If the address is *other than* a GRF MAC address for any interface participating in the assigned bridge group, the packet is submitted to the bridging engine for forwarding. When the MAC address is a GRF MAC address, the packet is forwarded to the GRF protocol forwarding engine for routing at the protocol layer. Multicast and broadcast frames are submitted to both engines.

Configuration options

The GRF supports the configuration items specified in 802.1D. A GRF functioning as a bridge will interoperate with other bridges, including equipment of vendors in conformance with the IEEE 802.1D standard, to allow forwarding of frames across multiple LAN hops.

Additionally, the GRF supports up to 64 independent 802.1D bridge groups, and separates traffic between groups. For example, on a GRF with six attached FDDI rings, rings A, B, and C could form one bridge group, rings D and E could form a second bridge group, and ring F could stand alone, using only IP routing for its packets.

A GRF functioning as a bridge also will interoperate with other bridges to forward frames from one bridge to the other over ATM. This will allow two independent bridged LANs at remote locations to function as one logical network transparently connected by ATM. This encapsulated bridging follows the Internet standard specification in RFC 1483.

Interoperability

FDDI - Frame forwarding is compatible with any station sending and receiving FDDI LLC frames.

Ethernet - Frame forwarding is compatible with any station using either DIX Ethernet or IEEE 802.3 frames.

ATM OC-3c - Frame forwarding is compatible with any remote bridge using RFC 1483 bridging encapsulation.

Spanning tree - GRF transparent bridging will interoperate with any other bridge (including other GRFs) compliant with the IEEE 802.1D spanning tree protocols.

Spanning tree

The GRF implementation supports the full Spanning Tree Algorithm specified in the IEEE 802.1D standard.

Using the Spanning Tree, network topologies can contain cycles that can be used as redundant or back-up links. The Spanning Tree controls the bridge's flow of traffic over all potential links to prevent packet storms (bridges repeating a packet or packets to each other, without end).

Consistent with basic GRF architecture, the Spanning Tree Algorithm and all controlling configuration and bridging information is maintained on the control board. A copy of the bridging filtering table is maintained on each media card.

Bridge filtering table

Media card bridge ports forward new MAC source addresses to the operating system for insertion in the global bridge filtering table that is maintained on the control board. Each bridging media card type (FDDI, Ethernet, and ATM OC-3c) also has copy of this table. Batches of table updates are sent out to all bridging media cards in the same way IP route table updates are dispersed to the media cards.

Bridge ports also “age” entries according to the 802.1D protocol. When no activity is associated with a MAC address for the specified time-out interval, the interface sends the operating software a delete request and the address is removed first from the global bridge filtering table and then, via the update packets, from media cards’ tables.

Fragmentation

IPv4 frames are fragmented as necessary, as when bridging an FDDI frame of more than 1500 bytes to an Ethernet interface.

A frame may be too large for the maximum transmission unit of the sending GRF interface. One example is when forwarding a 4500-byte frame from FDDI to an Ethernet interface with an MTU of 1500 bytes. The GRF bridge will attempt to break such a frame into fragments that will fit the sending interface. This is possible if the frame contains an IP datagram; then the GRF may use the fragmentation rules of IP to split the frame. Otherwise, the GRF must drop the frame.

Spamming

Spamming is when a bridging interface forwards a frame to all active interfaces in the bridge group. On the GRF, spamming is done when a broadcast or multicast address is received, or when a frame arrives whose destination address is not in the bridge filtering table.

GateD

GateD treats a bridge group interface as a single interface. Individual member interfaces are not considered in GateD operation.

Bridging components

Bridging daemon – bridged

The bridging daemon, **bridged**, configures and manipulates bridging interfaces on the GRF. It operates the spanning tree algorithm specified in IEEE 802.1D and ensures interoperability with other 802.1D bridges.

bridged reads the `/etc/bridged.conf` configuration file to build an initial bridging topology. The `bridged.conf` file is read whenever **bridged** is restarted. Refer to the **bridged** man page for more information.

bridged is started by the system script `/etc/grstart`. This script monitors the **bridged** daemon and restarts it if **bridged** stops. **bridged** is run from its installed location `/usr/sbin/bridged`.

Configuration file – bridged.conf

The bridging configuration file is `/etc/bridged.conf`. A utility, **bredit**, is used to access the file and create bridge groups and bridging settings.

Parameters in `bridged.conf` can be set to:

- name bridge groups
- assign interfaces (bridge ports) to a group
- assign priority, root path cost, and forwarding addresses to individual interfaces
- assign hello time and forwarding delay values, priority, maximum age, and discard addresses to individual groups

A copy of the `/etc/bridged.conf` file is in the *GRF Reference Guide*.

Editing utility – bredit

The **bredit** utility is used to access and edit the `bridged.conf` configuration file.

bredit opens the configuration file in the **vi** editor. After you make changes, you exit the file with the **vi** exit file **:q** command.

At this point **bredit** asks if you want to make the changes permanent. You also have the option of signaling **bridged** to re-read the updated file immediately. When this option is taken, **bridged** restarts as if it was stopped and restarted for the first time. If you change the file in **vi** but do not choose either of the options, **bredit** tells you that your changes were not committed.

Management tools

A set of tools are provided to manage bridging, primarily through **bridged**. Brief descriptions are provided here, more detail is given in the *Examining and debugging bridge configurations* section near the end of this chapter.

These tools include:

- **brstat**, displays relevant **bridged** status and bridging information
- **brinfo**, displays relevant kernel-based bridging information

brstat

The **brstat** command provides a snapshot of state information directly from **bridged**. A short lag occurs between the time a request is made and when an active **bridged** returns the information.

```
super> brstat
```

brinfo

The **brinfo** command is used to retrieve bridging interface information for administrative debugging and other situations where a simple checking of bridge group or bridge port information is needed.

```
super> brinfo bridge_group | all
```

or

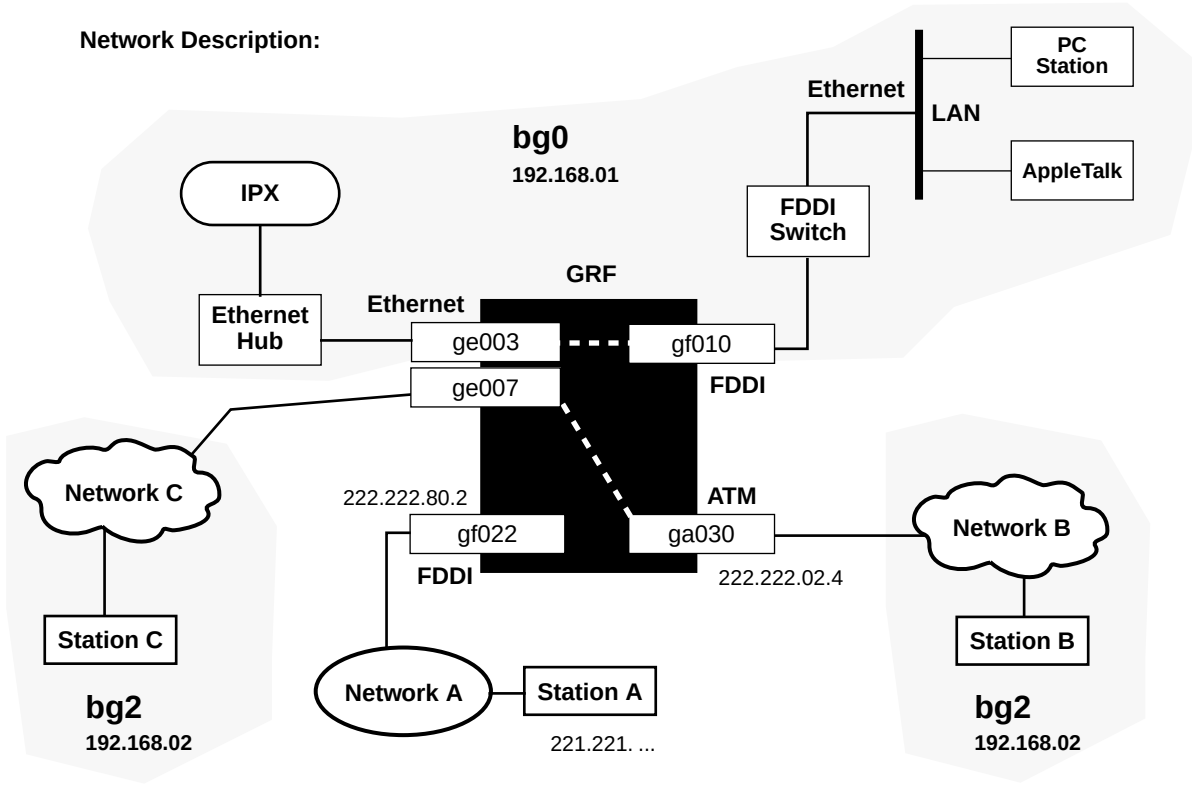
```
super> brinfo bridge_port | all
```

If a bridge group is specified, **brinfo** prints information about the group and the bridge ports (underlying interfaces) that are members of the specified group. If a bridge port (interface name) is specified, **brinfo** displays the specified interface. If no parameters are specified, all groups are reported on by default.

brinfo gets its information directly from the BSD kernel whereas **brstat** gets its information from **bridged**.

Bridging example

In this example, bridge group `bg0` is one shaded area. Two GRF interfaces, one Ethernet and one FDDI (`ge003` and `gf010`), form the bridge between the IPX services and the Ethernet LAN. Bridge group `bg2` has two LANs, each with a GRF interface, one Ethernet (`ge007`) and one ATM (`ga030`). Interface `gf022` is IP routing only. Station A can route to any station in either bridge group.



Configuration tasks:

1. Enter IP addresses in `grifconfig.conf`:

```
bg0     192.168.01
bg2     192.168.02
gf022   221.221. ...
```

2. Define bridge groups in `bridged.conf`:

```
bridge_group bg0      bridge_group bg2
port ge003, gf010     port ge007, ga030
```

3. Set Network A for normal routed network.
4. Configure a PVC on `ga030` in `gratm.conf`.

Figure 2. Bridging example diagram

The GRF currently supports up to 64 bridge groups with as many as 255 logical GRF interfaces assigned to each group. A logical interface can be a member of only one bridge group.

From a GRF perspective, a bridge group equals a virtual LAN.

Configuration file and profile overview

When a new GRF system is installed or a site upgrades to a bridging software release, the bridging daemon, **bridged**, is automatically started.

These are the steps to configure bridging interfaces and parameters:

1. Create bridge groups in bridged.conf

Run **bredit** to access and edit the `/etc/bridged.conf` configuration file. Create and name the bridge groups, and assign bridge ports and parameters to each.

2. Assign an IP address to each bridge group

Step 2 is necessary only if you want to do simultaneous bridging and routing.

Edit `/etc/grifconfig.conf` to identify each bridge group by assigning:

- an IP address
- the GRF interface name
- a netmask, required
- a broadcast address, as required

Note: Members of bridge groups are not assigned IP addresses in `/etc/grifconfig.conf`. In the example from the preceding page, only the FDDI interface, `gf022`, and the bridge groups, `bg0` and `bg2`, are assigned IP addresses.

3. Create ATM OC-3c PVCs for encapsulated bridges

To configure an encapsulated bridge on an ATM circuit, edit the `/etc/gratm.conf` file to create a PVC on the ATM OC-3c logical interface.

1. Create bridge groups in bridged.conf

The **bredit** utility is used to access and edit the `bridged.conf` configuration file.

bredit opens the configuration file in the **vi** editor. After you make changes, you exit the file with the **vi** exit file **:q** command.

Here are the syntax conventions in the `/etc/bridged.conf` file:

{ - the brace after the group name indicates the start of the group's parameters

; - a semi-colon indicates the end of the arguments for a statement

}; - a closing brace and semi-colon indicate the end of the block

The format of a group name is:

```
bridge_group bgA { arguments } ;
```

where `A` is a decimal number from 0 through 63

The only required parameter is the list of GRF interfaces you are assigning to the group.

The format of the group list is:

```
bridge_group bgA {
port interface_name;
};
```

where *`interface_name`* is in the standard GRF interface name format `gx0yz` that uniquely describes a logical FDDI, Ethernet, or ATM OC-3c interface.

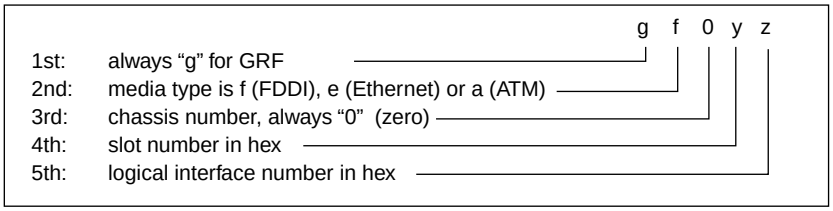


Figure 3. Interface name for FDDI, Ethernet, and ATM OC-3cinterfaces

List one port on a line, or list them all on one line as shown in these examples:

```
port ge003;
port gf010;
```

or

```
port ge003 gf010;
```

A simple bridge group entry is:

```
bridge_group bg0 {
port ge003;
port gf010;
};
```

An empty bridge group is defined in this way:

```
bridge_group bg5 {
          ;
};
```

2. Assign IP addresses to bridge groups

To do simultaneous bridging and routing, assign an IP address to each bridge group in the `/etc/grifconfig.conf` file.

These are the entries in `grifconfig.conf` for bridge group `bg0` and `bg2` and the non-bridging interfaces shown in the example in Figure 2:

```
# name    address          netmask           broad_dest        argument
#
bg0       192.168.01.1   255.255.255.0
bg2       192.168.02.1   255.255.255.0

gf022     222.222.80.2   255.255.255.0
ga030     222.222.02.4   255.255.255.0
```

A netmask entry is required for each bridge group.

3. Create an ATM PVC for an encapsulated bridge

Bridging over ATM can be configured in two ways:

- LLC Encapsulation (RFC 1483, section 4)
- VC Based Multiplexing (RFC 1483, section 5)

When LLC Encapsulation is used, a single PVC is configured to carry all traffic.

When VC Based Multiplexing is used, multiple PVCs are defined for the logical interface. Each PVC carries a specific type of traffic. For example, one PVC carries Ethernet PDUs while another carries FDDI.

Configuration in gratm.conf

Configuration over ATM also requires that new entries be made to three sections of the regular ATM configuration file, `/etc/gratm.conf`.

The next three steps describe ATM bridging configuration requirements and options. Examples of configured PVCs follow.

1 In the Traffic Shaping section of the `/etc/gratm.conf` file, set traffic shaping name and quality of service parameters, use any string. Set a name for each type of service that will be assigned.

The `/etc/gratm.conf` file itself describes how to specify a range of traffic shaping parameters.

```
# Traffic shaping parameters
# Lines beginning with the keyword "Traffic_Shape" define
# traffic shapes which may be used to configure the performance
# characteristics of ATM Virtual Circuits.
#
Traffic_Shape name=high_speed_high_quality \
       peak=155000 sustain=155000 burst=2048 qos=high
```

2 To configure a logical interface for bridging, you create an Interface entry in the Interface section of the `/etc/gratm.conf` file. This entry must include the intended bridging method, specify this with the `bridge_method=` keyword.

Here is a sample Interface entry:

```
Interface ga030 traffic_shape=high_speed_high_quality \
bridge_method=vc_multiplexed,broute_to_ether
```

There are two types of bridging methods that can be specified:

- VC Based Multiplexing, `bridge_method=vc_multiplexed`

  The configuration must include one or more PVCs for this interface specified in the PVC section and defined (as described below) with `proto=vcmux_bridge`.

- LLC Encapsulation, `bridge_method=llc_encapsulated`

  The configuration must include one PVC for this interface specified in the PVC section and defined with `proto=llc,bridging`.

Restrictions

Media and transmission restrictions can be specified using these alternate `bridge_method,xxxx` keywords:

- `bridge_method=xxxx,broute_to_ether`

  IP and ISO datagrams are transmitted as Ethernet frames.

- `bridge_method=xxxx,ether_only`

  All frames except BPDUs (routed datagrams and all bridged LAN frame types) are transmitted as Ethernet frames.

- `bridge_method=xxxx,broute_to_fddi`

  IP and ISO datagrams are transmitted as FDDI frames.

- `bridge_method=xxxx,fddi_only`

  All frames except BPDUs (routed datagrams and all bridged LAN frame types) are transmitted as FDDI frames.

If an interface cannot be used to transmit a particular frame type directly, the GRF attempts to translate the frame to a permitted type. For example, if an interface is defined to send Ethernet frames only and the GRF has an FDDI frame to transmit, the GRF translates the frame to an Ethernet frame first. Similarly, if the GRF has a routed IP datagram to transmit, the GRF adds an Ethernet header and transmits the datagram as an Ethernet frame.

3 One or more Permanent Virtual Circuits (PVCs) must be defined in the PVCs section for each logical interface specified for bridging in the Interfaces section.

A bridging PVC is assigned a protocol value. This value must be consistent with the bridging method defined for the logical interface. Bridging PVCs are assigned either one of these protocol values:

- `proto=llc,bridging`
- `proto=vcmux_bridge,yyyy`

proto=llc,bridging

This type of PVC is used for logical interfaces defined with `bridge_method=llc_encapsulated`. The PVC uses LLC encapsulation for each PDU.

For example, this PVC entry enables bridging on an LLC PVC:

```
PVC ga030 0/32 proto=llc,bridging traffic_shape=high_speed_high_quality
```

proto=vcmux_bridge,yyyy

This type of PVC is used only for logical interfaces defined with `bridge_method=vc_multiplexed`. The PVC carries bridged traffic of a single type.

The ,yyyy represents a second protocol qualifier required for the `proto=` parameter. This qualifier defines the type of bridged traffic the PVC can carry. Traffic types include:

- `proto=vcmux_bridge,ether_fcs`

  Specifies that each PDU is an Ethernet frame, including a Frame Check Sequence.

- `proto=vcmux_bridge,ether_nofcs`

  Specifies that each PDU is an Ethernet frame, without a Frame Check Sequence.
- `proto=vcmux_bridge,fddi_fcs`

  Specifies that each PDU is an FDDI frame, including a Frame Check Sequence.
- `proto=vcmux_bridge,fddi_nofcs`

  Specifies that each PDU is an FDDI frame, without a Frame Check Sequence.
- `proto=vcmux_bridge,bpdu`

  Specifies that each PDU is an 802.1D Bridge Protocol Data Unit.

PVC configuration examples

LLC encapsulated, restricted to Ethernet

Here is a sample LLC Encapsulated configuration, restricted to Ethernet. Note that any IP or ISO routed traffic transmitted on the PVC will be encapsulated as an Ethernet frame.

```
# Traffic shape
Traffic_Shape name=high_speed_high_quality peak=155000 sustain=155000
burst=2048 qos=high
# Logical interface
Interface ga030 traffic_shape=high_speed_high_quality
bridge_method=llc_multiplexed,broute_to_ether
# PVC
PVC ga030 0/32 proto=llc,bridging
```

VC-based multiplexing options

Here is a sample VC Based Multiplexing configuration. Note that routed IP or ISO datagrams are encapsulated as Ethernet frames.

```
# Traffic shape
Traffic_Shape name=high_speed_high_quality peak=155000 sustain=155000
burst=2048 qos=high
# Logical interface
Interface ga030 traffic_shape=high_speed_high_quality
bridge_type=vc_multiplexed,broute_to_ether
# PVCs for bridging
PVC ga030 0/32 proto=vcmux_bridge,ether
PVC ga030 0/33 proto=vcmux_bridge,ether_fcs
PVC ga030 0/34 proto=vcmux_bridge,bpdu
```

Installing configuration changes

When you enter configuration information or make changes, you must do a **grwrite** command to save the `/etc` directory to permanent storage. In the CLI, or from the CLI UNIX shell, enter:

```
# grwrite -v
```

That command saves the `/etc` directory. You can find out at any time if there are unsaved files in that directory, use this version of **grwrite** to get a list of unsaved files:

```
# grwrite -vn
```

You must also reset the media card for the changes to take place. Enter:

```
# grreset <slot_number>
```

Sources of bridging data

Bridging trace log

The **-d level** option for the **bridged** command controls the type of messages collected in the `/var/tmp/bridged.trace` log.

The output shown here reflects level 5, the default, and adequate for most debugging. Enter:

```
# cd /var/tmp
# cat bridged.trace
```

```
1997.11.25.11:39:46 NOTICE main.c:188 main() started
1997.11.25.11:39:46 NOTICE br_init.c:421 add_modify() adding group 'bg0' 1997.11.25.11:39:46 NOTICE
                br_init.c:460 add_modify() adding 'ga030' to 'bg0'
1997.11.25.11:39:46 NOTICE br_init.c:460 add_modify() adding 'ge020' to 'bg0'
1997.11.25.11:39:46 NOTICE br_init.c:460 add_modify() adding 'ge021' to 'bg0'
1997.11.25.11:39:46 NOTICE br_init.c:460 add_modify() adding 'gf000' to 'bg0'
1997.11.25.11:39:46 NOTICE br_init.c:460 add_modify() adding 'gf002' to 'bg0'
1997.11.25.11:39:46 NOTICE standard.c:1103 std_initialisation() bg0 root [me]
1997.11.25.11:39:46 NOTICE standard.c:658 std_become_designated_port() ga030 desig bridge [me] port 1/1
1997.11.25.11:39:46 NOTICE standard.c:658 std_become_designated_port() ge020 desig bridge [me] port 128/2
1997.11.25.11:39:46 NOTICE standard.c:658 std_become_designated_port() ge021 desig bridge [me] port 128/3
1997.11.25.11:39:46 NOTICE standard.c:658 std_become_designated_port() gf000 desig bridge [me] port 2/4
1997.11.25.11:39:46 NOTICE standard.c:658 std_become_designated_port() gf002 desig bridge [me] port 128/5
1997.11.25.11:39:46 NOTICE standard.c:741 std_make_forwarding() bg0.ga030 Listening(1)
1997.11.25.11:39:46 NOTICE standard.c:741 std_make_forwarding() bg0.ge020 Listening(1)
1997.11.25.11:39:46 NOTICE standard.c:741 std_make_forwarding() bg0.ge021 Listening(1)
1997.11.25.11:39:46 NOTICE standard.c:741 std_make_forwarding() bg0.gf000 Listening(1)
1997.11.25.11:39:46 NOTICE standard.c:741 std_make_forwarding() bg0.gf002 Listening(1)
1997.11.25.11:40:01 NOTICE standard.c:997 std_forward_delay_timer_expiry() bg0.ga030 Learning(2)
1997.11.25.11:40:01 NOTICE standard.c:997 std_forward_delay_timer_expiry() bg0.ge020 Learning(2)
1997.11.25.11:40:01 NOTICE standard.c:997 std_forward_delay_timer_expiry() bg0.ge021 Learning(2)
1997.11.25.11:40:01 NOTICE standard.c:997 std_forward_delay_timer_expiry() bg0.gf000 Learning(2)
1997.11.25.11:40:01 NOTICE standard.c:997 std_forward_delay_timer_expiry() bg0.gf002 Learning(2)
1997.11.25.11:40:16 NOTICE standard.c:1000 std_forward_delay_timer_expiry() bg0.ga030 Forwarding(3)
1997.11.25.11:40:16 NOTICE standard.c:799 std_topology_change_detection() bg0 top_change ON
1997.11.25.11:40:16 NOTICE standard.c:1000 std_forward_delay_timer_expiry() bg0.ge020 Forwarding(3)
1997.11.25.11:40:16 NOTICE standard.c:1000 std_forward_delay_timer_expiry() bg0.ge021 Forwarding(3)
1997.11.25.11:40:16 NOTICE standard.c:1000 std_forward_delay_timer_expiry() bg0.gf000 Forwarding(3)
1997.11.25.11:40:16 NOTICE standard.c:1000 std_forward_delay_timer_expiry() bg0.gf002 Forwarding(3)
1997.11.25.11:40:51 NOTICE standard.c:1062 std_topology_change_timer_expiry() bg0 top_change OFF
```

Figure 4. Output from bridging trace file

Bridge group information

brinfo returns configuration information about a bridge group and each of its member ports. The number of ports in a group is stated in the `Ports:` line. Enter:

```
# brinfo bridge_group
```

```
# brinfo bg0
Bridge Daemon: Running
Bridge_group: bg0
     Flags: (0x43) up broadcast running
     Ports:  2
     port ge003
          State: (0xf):Forwarding
          Flags: 0x9143 up broadcast running promisc link0 multicast
          Bridging media: ethernet bpdu
          MAC Address:  0:c0:80:00:55:d1

     port gf010
          State: (0xf):Forwarding
          Flags: 0x9143 up broadcast running promisc link0 multicast
          Bridging media: fddi bpdu
          MAC Address:  0:c0:80:00:55:d2
```

Low-level state information

brstat obtains low-level state information from **bridged**. Enter:

```
# brstat
```

```
Bridged Information:
   Debug Level: 5, Trace Mask: 0xffffffff, Spanning Tree: Enabled
   Log File: "/var/tmp/bridged.trace", Config File: "/etc/bridged.conf"
   bridged started at: Fri Jan 9 14:39:58 1998

Bridge Group bg12
   Spanning Tree:   Enabled
   Designated Root: 32768 00:c0:80:0c:65:53
   Bridge ID:       32768 00:c0:80:83:43:f9

   Root Port: ge066, Root Path Cost: 10
   Topology Change Detected:  No
   Root   Max Age: 20, Hello Time:  2, Forward Delay: 15
   Bridge Max Age: 20, Hello Time:  2, Forward Delay: 15,  Hold Time: 1

                                    Path  Desig Desig                    Desig
   Interface Port ID Con State      Cost  Cost  Bridge                    Port
   --------- ------- --- ---------- ----- ----- ----------------------  -------
   gf080    128 1    No  Disabled   10
   gf081    128 2    No  Disabled   10
   gf082    128 3    No  Disabled   10
   gf082    128 4    No  Disabled   10
   ge065    128 5    Yes Blocking   10    0     32768 00:c0:80:0c:65:53  128 5
  *ge066    128 6    Yes Listening  10    0     32768 00:c0:80:0c:65:53  128 6

Dump snapshot finished at Fri Jan 9 14:40:01 1998
```

Route trees and filtering table

The **netstat -rn** command returns the bridging filtering table of MAC addresses and other related information.

Since the filtering table itself tends to be lengthy, pipe the **netstat** command with **more** to view it easily. Enter:

```
#  netstat -rn

Routing tables

Internet:
Destination      Gateway           Flags     Refs     Use  Interface
198.174.11       198.174.11.33     U          22    21575  ef0
198.174.11.33    127.0.0.1         UGH         2      765  ef0
204.221.156      204.221.156.33    U           1        5  gh010

Bridging:
Destination      Gateway            Flags     Refs     Use  Interface
00:c0:f2:00:1e:a0   00:c0:80:00:55:d2  UHD      0     827173499   gf081
          •                  •          •      •          •       •
          •                  •          •      •          •       •
          •                  •          •      •          •       •
          •                  •          •      •          •       •
          •                  •          •      •          •       •
          •                  •          •      •          •       •
Source MAC address      Port MAC address                    Port interface name

Bridging ARP:
Destination      Gateway            Flags    Refs    Use  Interface
198.174.59.101 00:03:01:80:62:82   UHD     0      0           bg0
#
```

Bridging sockets

The command **netstat -f bridge** displays all active bridging sockets. Sockets are used by **bridged** to transmit and receive Bridge Protocol Data Units:

```
$ netstat -f bridge
Active bridging sockets
Proto Recv-Q Send-Q  Group               Port                (flags)
bridg      0      0  bg0                 *                   3
```

Kernel bridging statistics

The command **netstat -s -f bridge** displays kernel statistics for bridging:

```
$ netstat -s -f bridge
bridging:
      37 packets received
      0 packets received before bridging configured
      0 packets received for unknown interface(s)
      0 packets dropped for pullup failures
```

```
0 packets dropped for socket full
0 packets dropped for no endpoint
37 packets delivered
0 packets dropped for no memory
147052 packets sent
169701 output packets dropped for interface down
0 output packets dropped for link down
44119 output packets unicast
102933 output packets multicast
0 output packets dropped for no memory
9536 output packets copied
93397 output packet copy avoided
1080 copied output packets dropped for no memory
0 output packets with too many copies
```

Examining and debugging bridge configurations

Introduction

There are several places to start debugging bridging problems. To begin, it is probably most useful to try to determine which piece of the system seems to have the problem.

Three pieces of software need to work together for bridging to work correctly:

- **bridged** software (user space)
- GRF kernel software
- media card software

This section describes how tools such as **brinfo** and **brstat** can be helpful in debugging **bridged** and the GRF kernel software. If a problem cannot be isolated using these tools, or if the tools indicate the problem to be elsewhere, then debugging on the media card side needs to be pursued.

This section also describes how to gather traces from **bridged** to help diagnose possible problems. In specific bridging configurations, traces can help to understand **bridged** behavior based on the IEEE 802.1D standard.

Before attempting to debug bridging software, it is helpful to have read the IEEE 802.1D standard, especially those sections describing the behavior of the spanning tree. This bridging implementation uses the spanning tree functionality described in that standard.

Information needed by Ascend support

When you need to send Ascend support information about bridging problems, please include the following:

- a complete description of the problem
- **brinfo** output
- **brstat** output
- a description of the network (text or picture)
- **bridged** trace file(s)
- contents of `/etc/bridged.conf`
- contents of `/etc/gratm.conf`
- contents of `/etc/grifconfig.conf`

Enabling traces via bridged command

bridged writes traces to the `/var/tmp/bridged.trace` file.

This file is periodically archived and saved off in a compressed form to files in the form: `/var/tmp/bridged.trace.x.gz` in which `x` is 0–5.

By default, **bridged** runs with minimal tracing enabled. This saves the system overhead of writing every trace entry and the disk space used by the log file.

Sometimes it is necessary to gather additional **bridged** trace information for a given problem. When this needs to be done, edit `/etc/bridged.conf` using **bredit**, and change the **debug_level** line to read:

```
debug_level 6 ;
```

When this change is committed and **bridged** is reconfigured, additional traces are written to the **bridged** trace file. Once the error condition has been recreated, save the traces and then change the traces level back to 5.

The debug levels correspond to the following list:

- Level 0 (LOG_EMERG): Unusable
- Level 1 (LOG_ALERT): Action must be taken immediately
- Level 2 (LOG_CRIT): Critical conditions
- Level 3 (LOG_ERR): Error conditions
- Level 4 (LOG_WARNING): Warning conditions
- Level 5 (LOG_NOTICE): Normal but significant condition
- Level 6 (LOG_INFO): Informational (basic internal logic)
- Level 7 (LOG_DEBUG): Debugging (low-level internal logic)

Debug level 5, the default, provides more than enough information to resolve most **bridged** issues. Levels 6 and 7 are rarely used.

Note: Error conditions (fatal and non fatal) are always traced.

Displaying useful information

Two commands, **brinfo** and **brstat**, display a majority of the available bridging information.

brinfo queries the kernel to determine state and topology information about the current bridge groups and their operating environment.

brstat queries **bridged** for a superset of this and other information.

Using brinfo

In the example, two bridge groups are configured in the kernel: "`bg0`" and "`bg1`".
While `bg1` has no interfaces defined, `bg0` has two interfaces defined, `gf080` and `gf081`.

Here is the **brinfo** output for the two groups:

```
Bridge group name: bg1
Flags:(0x43) up broadcast running
Ports : 0

Bridge group name:  bg0
Flags:(0x43) up broadcast running
Ports : 2

    Port gf080 : State (0) Blocking
    Flags : (0x9343) : up broadcast running promisc link0 multicast
    Bridging media: fddi bpdu
    MAC address: 0:c0:80:0:55:d1

    Port gf081 : State (0X) Forwarding
    Flags : (0x9343) : up broadcast running promisc link0 multicast
    Bridging media: fddi bpdu
    MAC address: 0:c0:80:0:55:d3
```

Each interface is in one of the following states:

- **disabled**, usually by configuration, or if there is no connection on this port
- **blocking**, by spanning tree logic
- **listening**, spanning tree intermediate state
- **learning**, spanning tree intermediate state
- **forwarding**, spanning tree stable state

The flags correspond to the flags seen when the **ifconfig *interface*** command is used. Flags tell us about the state of the interface from the kernel's perspective. From a bridging perspective, the flags shown in the example are the flags that should be set for normal bridge operations.

All flags should get set automatically when **bridged** starts and interfaces are configured in the **bridged** configuration file.

If the `link0` flag is not set, as in:

```
Flags : (0x9343) : up broadcast running promisc multicast
```

then there is no connection at this interface. Either no wire is connected to the interface, or no host is on the other end of the wire.

State information - brstat

The **brstat** command signals **bridged** to dump out its internal state into the file `/var/tmp/bridged.dump`. This file is massaged by **brstat** to display information of interest. See the **bridged** man page for details about the debug level, log file, and configuration files.

Here is an example of **brstat** output:

```
# brstat

Bridged Information:
   Debug Level: 5, Trace Mask: 0xffffffff, Spanning Tree: Enabled
   Log File: "/var/tmp/bridged.trace", Config File: "/etc/bridged.conf"
   bridged started at: Thu Apr 27 18:43:12 1997

Bridge Group bg0
   Spanning Tree:   Enabled
   Designated Root: 7 08:00:2b:b6:38:80
   Bridge ID:       27 00:c0:80:00:55:d1

   Root Port: gf081, Root Path Cost: 10
   Topology Change Detected: No
   Root   Max Age: 20, Hello Time:  2, Forward Delay: 15
   Bridge Max Age: 20, Hello Time:  2, Forward Delay: 15, Hold Time: 1

                                    Path  Desig Desig                   Desig
   Interface Port ID Con State      Cost  Cost  Bridge                  Port
   --------- ------- --- ---------- ----- ----- ----------------------- -------
   gf080     128 1   Yes Disabled   10    0     32768 08:00:2b:b6:38:80 128 4
  *gf081     138 2   Yes Forwarding 10    0     32768 08:00:2b:b6:38:80 128 6

Dump snapshot finished at Fri Apr 28 15:20:00 1997
```

The configuration information starts at the Bridge Group section. The Designated Root line shows the MAC address of the root bridge.

In the example above, the root bridge is transmitting BPDUs with a priority of 7. The GRF (`bridge ID 00:c0:80:00:55:d1`) is transmitting BPDUs at a priority of 27. If the priorities were equal, the MAC address would be used to determine the root bridge.

The MAC address of the GRF bridge is selected as the numerically lowest MAC address of all the ports in the bridge group, or the MAC address of the maintenance Ethernet port.

The Root Path Cost and other values displayed in the second set of descriptors are spanning tree values that describe spanning tree configuration variables. See the IEEE 802.1D standard for more information about these variables.

The example also shows two configured interfaces, `gf080` and `gf081`. Each interface displays a priority, a unique id, a state, a status, and spanning tree variables associated with the 802.1D standard. Port priority is used to set up redundant bridge connections to the same LAN. An asterisk (*) indicates the root port.

MAC addresses and bridge IDs via netstat -ni

Bridge IDs are listed as **<Bridge>** in the network column.

Use the **netstat -ni** command to check which interfaces have bridge IDs. An excerpt from **netstat** output is shown below:

```
# netstat -ni
Name    Mtu   Network     Address              Ipkts  Ierrs   Opkts Oerrs  Coll
de0     1500  <link1>     00:c0:80:0c:65:53    58217      0   231408 0   287
de0     1500  206.146.164 206.146.164.9        58217      0   231408 0   287
rmb0    616   <link2>     00:00:00:00:00:00  3177659 15624  3278448 0     0
rmb0    616   <GRIT>      0:0x40:0           3177659 15624  3278448 0     0
lo0     1536  <link3>                          61134      0    61134 0     0
lo0     1536  127         127.0.0.1            61134      0    61134 0     0
lo0     1536  <GRIT>      0:0x48:0             61134      0    61134 0     0
atmp0*  1536  <link4>                              0      0        0 0     0
bg0     1500  <link147>                        22410      0        0 0     0
bg0     1500  <Bridge>    00:c0:80:0c:65:53    22410      0        0 0     0
bg0     1500  222.222.169 222.222.169.9        22410      0        0 0     0
bg0     1500  47.0000.8000.0900.0900.0900      22410      0        0 0     0
gl000*  1496  <link5>                              0      0        0 0     0
gf000*  4352  <link142>   00:c0:80:89:24:0e        0      0        0 0     0
gf001*  4352  <link137>   00:c0:80:89:24:0f        0      0        0 0     0
gf002*  4352  <link136>   00:c0:80:89:24:10        5      0      116 0     0
gf002*  4352  <Bridge>    00:c0:80:89:24:10        5      0      116 0     0
gf003*  4352  <link140>   00:c0:80:89:24:11        0      0        0 0     0
ge020   1500  <link141>   00:c0:80:89:09:9d        0      0    35831 0     0
ge020   1500  <Bridge>    00:c0:80:89:09:9d        0      0    35831 0     0
ge021   1500  <link130>   00:c0:80:89:09:9e     1029      0    35763 0     0
ge021   1500  <Bridge>    00:c0:80:89:09:9e     1029      0    35763 0     0
```

The Address field provides the MAC address. Note that the bridge port and the associated link for that interface have the same address.

Restarting bridged during debug

The **brsig** command provides a way to signal **bridged**. **brsig** takes the following parameters:

0 (zero)
USR1
USR2
HUP

This command is not needed for normal operations. It is a low-level debug tool, and should be used carefully. When a signal is received by **bridged**, you see a message similar to this:

```
# bridged signalled successfully.
```

Use **brsig 0** to verify that **bridged** is running.

Use **brsig USR1** to cause **bridged** to write a dump file, `/var/tmp/bridged.dump`. This dump contains detailed information about the state of internal timers and bridging configuration. **bridged** rewrites the dump file each time it receives **USR1**.

Use **brsig USR2** to cause **bridged** to check the state of a bridging interface from the kernel's perspective.

Use **brsig HUP** to cause **bridged** to reread the `bridged.conf` configuration file or an alternate file specified in the **bridged** command line (usually reserved for debugging purposes). The **bredit** command sends a **HUP** to **bridged**.