Configuring Serial Tunnel and Block Serial Tunnel

First Published: Month Day, Year
Last Updated: March 25, 2011

This chapter describes how to configure serial tunnel (STUN) and block serial tunnel (BSTUN).

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest feature information and caveats, see the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the "Feature Information for Configuring Serial Tunnel and Block Serial Tunnel" section on page 51.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to http://www.cisco.com/go/cfn. An account on Cisco.com is not required.

Contents

Information About Configuring Serial Tunnel and Block Serial Tunnel

To configure STUN and BSTUN, you should understand the following concepts:

- Serial Tunnel Overview, page 2
- Block Serial Tunneling (BSTUN) Overview, page 2
- Asynchronous Network Overview, page 5
- Virtual Multidrop Support for Multipoint Security Network Configurations, page 6
- Frame Sequencing, page 7

Serial Tunnel Overview

Cisco's STUN implementation allows Synchronous Data Link Control (SDLC) protocol devices and High-Level Data Link Control (HDLC) devices to connect to one another through a multiprotocol internetwork rather than through a direct serial link. STUN encapsulates SDLC frames in either the Transmission Control Protocol/Internet Protocol (TCP/IP) or the HDLC protocol. STUN provides a straight passthrough of all SDLC traffic (including control frames, such as Receiver Ready) end-to-end between Systems Network Architecture (SNA) devices.

Cisco's SDLC local acknowledgment provides local termination of the SDLC session so that control frames no longer travel the WAN backbone networks. This means end nodes do not time out, and a loss of sessions does not occur. You can configure your network with STUN, or with STUN and SDLC local acknowledgment. To enable SDLC local acknowledgment, the Cisco IOS software must first be enabled for STUN and routers must be configured to appear on the network as primary or secondary SDLC nodes. TCP/IP encapsulation must be enabled. Cisco's SDLC Transport feature also provides priority queueing for TCP encapsulated frames.

Cisco's BSTUN implementation enables Cisco series 2500, 4000, 4500, 4700, and 7200 series routers to support devices that use the Binary Synchronous Communications (Bisync) data-link protocol and asynchronous security protocols that include Adplex, ADT Security Systems, Inc., Diebold, and asynchronous generic traffic. BSTUN implementation is also supported on the 4T network interface module (NIM) on the Cisco router 4000 and 4500 series. Our support of the Bisync protocol enables enterprises to transport Bisync traffic and SNA multiprotocol traffic over the same network.

Block Serial Tunneling (BSTUN) Overview

Cisco's implementation of BSTUN provides the following features:

- Encapsulates Bisync, Adplex, ADT Security Systems, Inc., Diebold, and asynchronous generic traffic for transfer over router links. The tunneling of asynchronous security protocols (ASP) feature enables your Cisco 2500, 3600, 4000, 4500, or 7200 series router to support devices that use the following asynchronous security protocols:
  - adplex
  - adt-poll-select
  - adt-vari-poll
  - diebold
  - async-generic

  - mdi
- Provides a tunnel mechanism for BSTUN over Frame Relay, without using TCP/IP encapsulation.
- Supports Bisync devices and host applications without modification.
- Uses standard synchronous serial interfaces on Cisco 2500 series and the 4T network interface module (NIM) on the Cisco 4000 series and Cisco 4500 series.
- Supports point-to-point, multidrop, and virtual multidrop configuration.

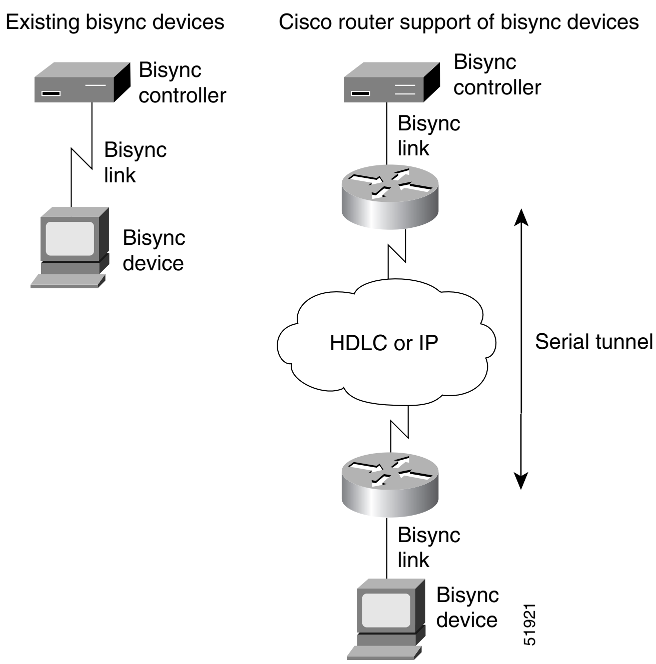
Note The async-generic item is not a protocol name. It is a command keyword used to indicate generic support of other asynchronous security protocols that are not explicitly supported.

Bisync Network Overview

The Bisync feature enables Cisco 2500, 3600, 4000, 4500, 4700, and 7200 series routers to support devices that use the Bisync data-link protocol. This protocol enables enterprises to transport Bisync traffic over the same network that supports their SNA and multiprotocol traffic, eliminating the need for separate Bisync facilities.

At the access router, traffic from the attached Bisync device is encapsulated in IP. The Bisync traffic can then be routed across arbitrary media to the host site where another router supporting Bisync will remove the IP encapsulation headers and present the Bisync traffic to the Bisync host or controller over a serial connection. HDLC can be used as an alternative encapsulation method for point-to-point links. Figure 1 shows how you can reconfigure an existing Bisync link between two devices and provide the same logical link without any changes to the existing Bisync devices.

Figure 1 Routers Consolidating Bisync Traffic by Encapsulation in IP or HDLC



The routers transport all Bisync blocks between the two devices in pass-through mode using BSTUN as encapsulation. BSTUN uses the same encapsulation architecture as STUN, but is implemented on an independent tunnel.

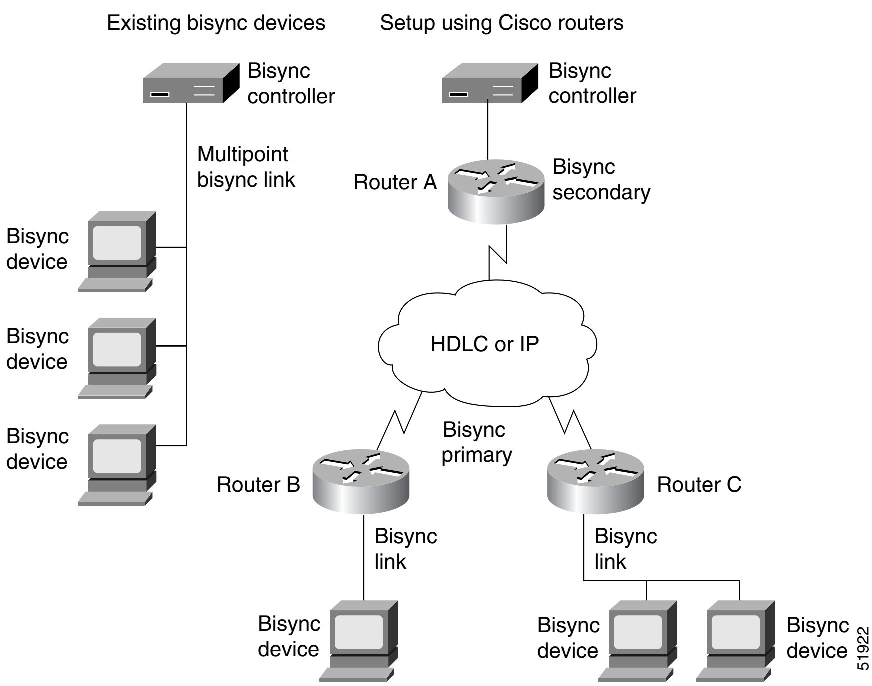
Point-to-Point and Multidrop Support

The Bisync feature supports point-to-point, multidrop, and virtual multidrop Bisync configurations.

In point-to-point operation, the Bisync blocks between the two point-to-point devices are received and forwarded transparently by the Cisco IOS software. The contention to acquire the line is handled by the devices themselves.

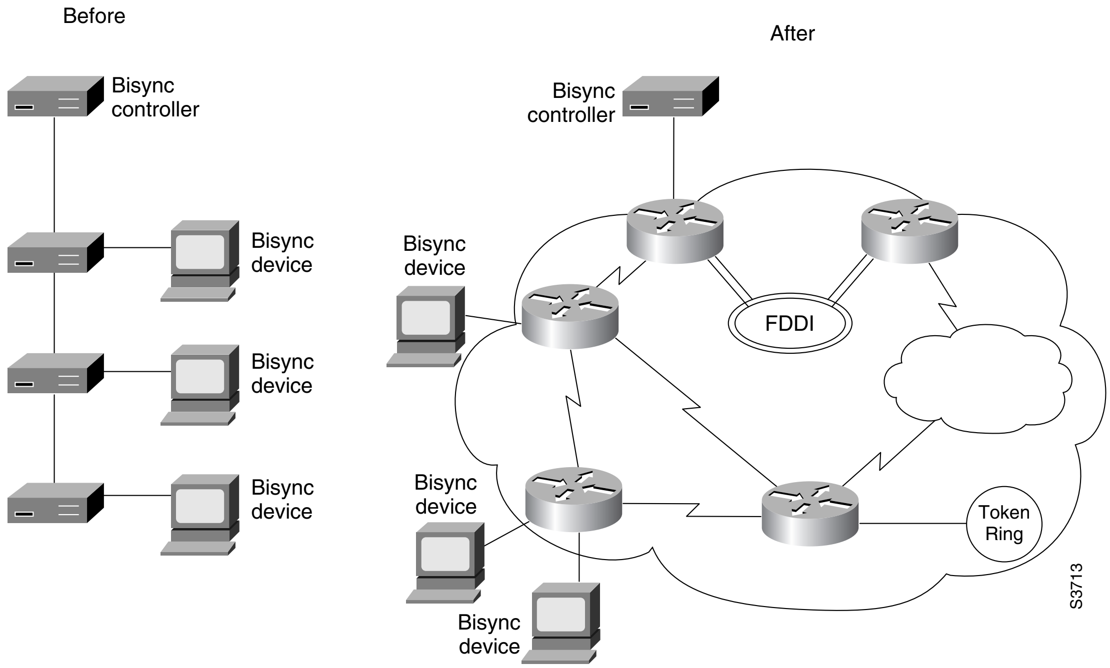
Cisco's Bisync multipoint operation is provided as a logical multipoint configuration. Figure 2 shows how a multipoint Bisync link is reconfigured using Cisco routers. Router A is configured as Bisync secondary. It monitors the address field of the polling or selection block and uses this address information to put into the BSTUN frame for BSTUN to deliver to the correct destination router. To simulate the Bisync multidrop, an EOT block is sent by the Bisync primary router before a poll or selection block. This ensures that Bisync tributary stations are in control mode before being polled or selected.

Figure 2 Multipoint Bisync Link Reconfigured Using Routers



Multidrop configurations are common in Bisync networks where up to 8 or 10 Bisync devices are frequently connected to a Bisync controller port over a single low-speed link. Bisync devices from different physical locations in the network appear as a single multidrop line to the Bisync host or controller. Figure 3 illustrates a multidrop Bisync configuration before and after implementing routers.

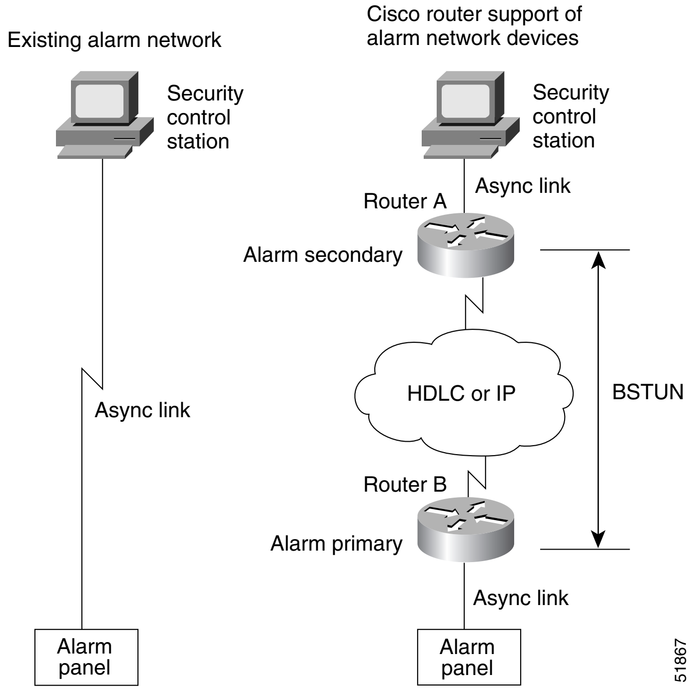
Figure 3 Integrating Bisync Devices over a Multiprotocol Network



Asynchronous Network Overview

These protocols enable enterprises to transport polled asynchronous traffic over the same network that supports their SNA and multiprotocol traffic, eliminating the need for separate facilities. Figure 4 shows how you can reconfigure an existing asynchronous link between two security devices and provide the same logical link without any changes to the existing devices.

Figure 4 Routers Consolidate Polled Asynchronous Traffic Using Encapsulation in IP or HDLC



Router A is configured as the secondary end of the BSTUN asynchronous link and is attached to the security control station; Router B is configured as the primary end of the BSTUN asynchronous link and has one or more alarm panels attached to it.

At the downstream router, traffic from the attached alarm panels is encapsulated in IP. The asynchronous (alarm) traffic can be routed across arbitrary media to the host site where the upstream router supporting these protocols removes the IP encapsulation headers and presents the original traffic to the security control station over a serial connection. High-Level Data Link Control (HDLC) can be used as an alternative encapsulation method for point-to-point links.

The routers transport all asynchronous (alarm) blocks between the two devices in passthrough mode using BSTUN for encapsulation. BSTUN uses the same encapsulation architecture as STUN, but is implemented on an independent tunnel. As each asynchronous frame is received from the line, a BSTUN header is added to create a BSTUN frame, and then BSTUN is used to deliver the frame to the correct destination router.

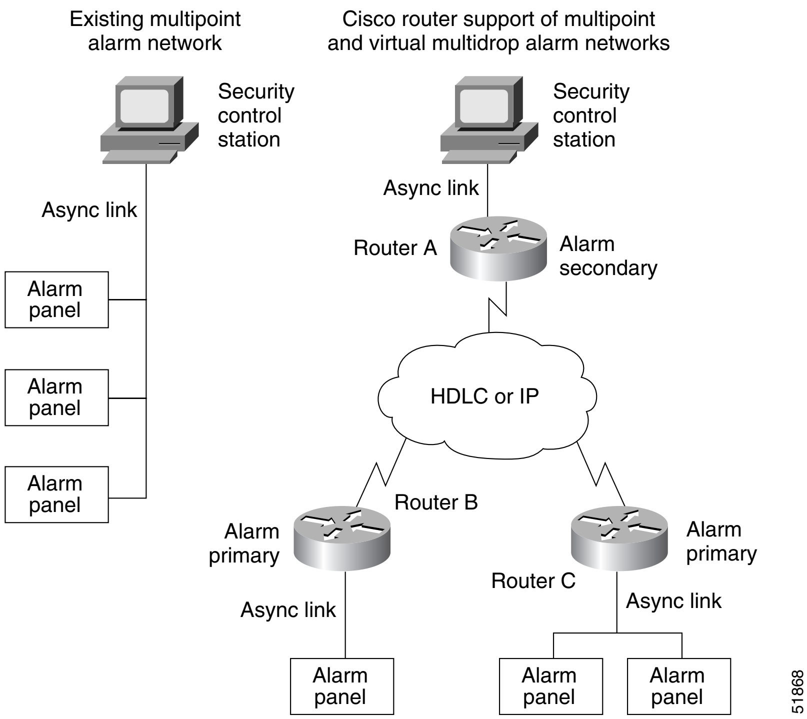
The Cisco routers do not perform any local acknowledgment or cyclic redundancy check (CRC) calculations on the asynchronous alarm blocks. The two end devices are responsible for error recovery in the asynchronous alarm protocol.

Virtual Multidrop Support for Multipoint Security Network Configurations

Multipoint configurations are common in security networks, where a number of alarm panels are frequently connected to a security control station over a single low-speed link. Our virtual multidrop support allows alarm panels from different physical locations in the network to appear as a single multidrop line to the security control station. Both Adplex and ADT are virtual multidropped protocols.

Multidrop operation is provided as a logical multipoint configuration. Figure 5 shows how a multipoint security network is reconfigured using Cisco routers. Router A is configured as an alarm secondary node, routers B and C are configured as alarm primary nodes. Router A monitors the address field of the polling or selection block and puts this address information in the BSTUN frame so BSTUN can deliver the frame to the correct downstream node.

Figure 5 Multipoint Asynchronous Security Protocol Link Reconfigured Using Routers



Frame Sequencing

Both Bisync and asynchronous alarm protocols are half-duplex protocols; data can be sent in either direction, but only in one direction at a time. Each block sent is acknowledged explicitly by the remote end. To avoid the problem associated with simultaneous sending of data, there is an implicit role of primary and secondary station.

Frame Sequencing in Bisync Networks

In a multidrop setup in Bisync networks, the Bisync control station is primary and the tributary stations are secondary. In a point-to-point configuration, the primary role is assumed by the Bisync device that has successfully acquired the line for sending data through the ENQ bidding sequence. The primary role stays with this station until it sends EOT.

To protect against occasional network latency, which causes the primary station to time out and resend the block before the Bisync block sent by the secondary is received, the control byte of the encapsulating frame is used as a sequence number. This sequence number is controlled and monitored by the primary Bisync router. This allows the primary Bisync router to detect and discard “late” Bisync blocks sent by the secondary router and ensure integrity of the Bisync link.

Note Frame sequencing is implemented in passthrough mode only.

Frame Sequencing in Asynchronous Networks

Network delays in asynchronous networks make it possible for a frame to arrive "late," meaning that the poll-cycling mechanism at the security control station has already moved on to poll the next alarm panel in sequence when it receives the poll response from the previous alarm panel.

To protect against this situation, routers configured for adplex or for adt-poll-select protocols use a sequence number built into the encapsulating frame to detect and discard late frames. The "upstream" router (connected to the security control station) inserts a frame sequence number into the protocol header, which is shipped through the BSTUN tunnel and bounced back by the "downstream" router (connected to the alarm panel). The upstream router maintains a frame-sequence count for the line, and checks the incoming frame-sequence number from the downstream router. If the two frame-sequence numbers do not agree, the frame is considered late (out of sequence) and is discarded.

Because the adt-vari-poll option allows the sending of unsolicited messages from the alarm panel, frame sequencing is not supported for this protocol.

Note Polled asynchronous (alarm) protocols are implemented only in passthrough mode. There is no support for local acknowledgment.

How to Configure Serial Tunnel and Block Serial Tunnel

To configure and monitor STUN or STUN local acknowledgment, perform the tasks in the following sections:

- Enabling STUN, page 9
- Specifying STUN Protocol Group, page 9
- Enabling STUN Keepalive, page 11
- Enabling STUN Remote Keepalive, page 11
- Enabling STUN Quick-Response, page 11
- Enabling STUN Interfaces, page 12
- Configuring SDLC Broadcast, page 12
- Establishing the Frame Encapsulation Method, page 13
- Configuring STUN with Multilink Transmission Groups, page 16
- Setting Up STUN Traffic Priorities, page 18

The Bisync feature is configured similar to SDLC STUN, but is configured as a protocol within a BSTUN feature. To configure and monitor Bisync with BSTUN, perform the tasks in the following sections:

- Enabling BSTUN, page 20
- Defining the Protocol Group, page 20
- Enabling BSTUN Keepalive, page 21
- Enabling BSTUN Remote Keepalive, page 21

Enabling STUN

To enable STUN, use the following command in global configuration mode:

Command	Purpose
Router(config)# **stun peer-name** *ip-address*	Enables STUN for a particular IP address.

When configuring redundant links, ensure that the STUN peer names you choose on each router are the IP addresses of the most stable interfaces on each device, such as a loopback or Ethernet interface. See the "Configuration Examples for Serial Tunnel and Block Serial Tunnel" section on page 27.

You must also configure SDLC address FF on Router A for each of the STUN peers. To do so, use the following command in interface configuration mode:

Command	Purpose
Router(config-if)# **stun route address** *address-number* **tcp** *ip-address* [**local-ack**] [**priority**] [**tcp-queue-max**] [**passive**]	Configures SDLC address FF on Router A for each STUN peer.

Specifying STUN Protocol Group

Place each STUN interface in a group that defines the ISO 3309-compliant framed protocol running on that link. Packets will only travel between STUN interfaces that are in the same protocol group.

There are three predefined STUN protocols:

- Basic
- SDLC
- SDLC transmission group (TG)

You can also specify a custom STUN protocol.

To specify STUN protocols, you must perform the tasks in the following sections:

- Creating and Specifying a Custom STUN Protocol, page 11

If you want to use the STUN Local Acknowledgment feature, you must specify either the SDLC protocol or the SDLC TG protocol.

Note Before you can specify a custom protocol, you must first define the protocol; see the "Creating and Specifying a Custom STUN Protocol" section on page 11 for the procedure.

Specifying a Basic STUN Group

The basic STUN protocol does not depend on the details of serial protocol addressing and is used when addressing is not important. Use this when your goal is to replace one or more sets of point-to-point (not multidrop) serial links by using a protocol other than SDLC. Use the following command in global configuration mode:

Command	Purpose
`Router(config)#` **`stun protocol-group`** *`group-number`* **`basic`**	Specifies a basic protocol group and assigns a group number.

Specifying an SDLC Group

You can specify SDLC protocol groups to associate interfaces with the SDLC protocol. Use the SDLC STUN protocol to place the routers in the midst of either point-to-point or multipoint (multidrop) SDLC links. To define an SDLC protocol group, enter the following command in global configuration mode:

Command	Purpose
`Router(config)#` **`stun protocol-group`** *`group-number`* **`sdlc`**	Specifies an SDLC protocol group and assigns a group number.

If you specify an SDLC protocol group, you cannot specify the **stun route all** command on any interface of that group.

For an example of how to configure an SDLC protocol group, see the "Serial Link Address Prioritization Using STUN TCP/IP Encapsulation Example" section on page 17.

Specifying an SDLC Transmission Group

An SNA TG is a set of lines providing parallel links to the same pair of SNA front-end-processor (FEP) devices. This provides redundancy of paths for fault tolerance and load sharing. To define an SDLC TG, use the following command in global configuration mode:

Command	Purpose
`Router(config)#` **`stun protocol-group`** *`group-number`* **`sdlc sdlc-tg`**	Specifies an SDLC protocol group, assigns a group number, and creates an SNA transmission group.

All STUN connections in a TG must connect to the same IP address and use the SDLC local acknowledgment feature.

Creating and Specifying a Custom STUN Protocol

To define a custom protocol and tie STUN groups to the new protocol, use the following commands in global configuration mode:

	Command	Purpose
Step 1	`Router(config)#` **`stun schema`** *`name`* **`offset`** *`constant-offset`* **`length`** *`address-length`* **`format`** `format-keyword`	Creates a custom protocol.
Step 2	`Router(config)#` **`stun protocol-group`** *`group-number`* **`schema`**	Specifies the custom protocol group and assigns a group number.

Enabling STUN Keepalive

To define the number of times to attempt a peer connection before declaring the peer connection to be down, use the following command in global configuration mode:

Command	Purpose
`Router(config)#` **`stun keepalive-count`**	Specifies the number of times to attempt a peer connection.

Enabling STUN Remote Keepalive

To enable detection of the loss of a peer, use the following command in global configuration mode:

Command	Purpose
`Router(config)#` **`stun remote-peer-keepalive`** *`seconds`*	Enables detection of the loss of a peer.

Enabling STUN Quick-Response

You can enable STUN quick-response, which improves network performance when used with local acknowledgment. When STUN quick-response is used with local acknowledgment, the router responds to an exchange identification (XID) or a Set Normal Response Mode (SNRM) request with a Disconnect Mode (DM) response when the device is not in the CONNECT state. The request is then passed to the remote router and, if the device responds, the reply is cached. The next time the device is sent an XID or SNRM, the router replies with the cached DM response.

Note Using STUN quick-response avoids an AS/400 line reset problem by eliminating the Non-Productive Receive Timer (NPR) expiration in the AS/400. With STUN quick-response enabled, the AS/400 receives a response from the polled device, even when the device is down. If the device does not respond to the forwarded request, the router continues to respond with the cached DM response.

To enable STUN quick-response, use the following command in global configuration mode:

Command	Purpose
`Router(config)#` **`stun quick-response`**	Enables STUN quick-response.

Enabling STUN Interfaces

Caution When STUN encapsulation is enabled or disabled on an RSP platform, the memory reallocates memory pools (recarve) and the interface shuts down and restarts. The recarve is caused by the change from STUN to another protocol, which results in a change in the MTU size. No user configuration is required.

You must enable STUN on serial interfaces and place these interfaces in the protocol groups you have defined. To enable STUN on an interface and to place the interface in a STUN group, use the following commands in interface configuration mode:

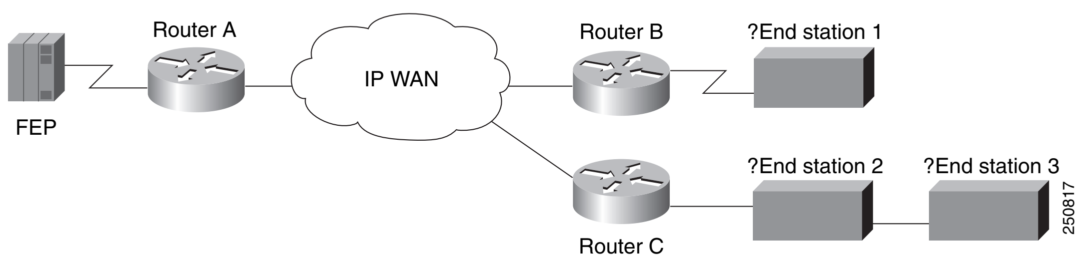
	Command	Purpose
Step 1	`Router(config-if)#` **`encapsulation stun`**	Enables STUN function on a serial interface.
Step 2	`Router(config-if)#` **`stun group`** *`group-number`*	Places the interface in a previously defined STUN group.

When a given serial link is configured for the STUN function, it is no longer a shared multiprotocol link. All traffic that arrives on the link will be transported to the corresponding peer as determined by the current STUN configuration.

Configuring SDLC Broadcast

The SDLC broadcast feature allows SDLC broadcast address FF to be replicated for each of the STUN peers, so that each of the end stations receives the broadcast frame. For example, in Figure 6, the FEP views the end stations 1, 2, and 3 as if they are on an SDLC multidrop link. Any broadcast frame sent from the FEP to Router A is duplicated and sent to each of the downstream routers (B and C).

Figure 6 SDLC Broadcast across Virtual Multidrop Lines



To enable SDLC broadcast, use the following command in interface configuration mode:

Command	Purpose
`Router(config-if)#` **`sdlc virtual-multidrop`**	Enables SDLC broadcast.

Only enable SDLC broadcast on the device that is configured to be the secondary station on the SDLC link (Router A in Figure 6).

Establishing the Frame Encapsulation Method

To allow SDLC frames to travel across a multimedia, multiprotocol network, you must encapsulate them using one of the methods in the following sections:

- Configuring HDLC Encapsulation without Local Acknowledgment, page 13
- Configuring TCP Encapsulation without Local Acknowledgment, page 13
- Configuring TCP Encapsulation with SDLC Local Acknowledgment and Priority Queueing, page 14
- Configuring Local Acknowledgment for Direct Frame Relay Connectivity, page 16

Configuring HDLC Encapsulation without Local Acknowledgment

You can encapsulate SDLC or HDLC frames using the HDLC protocol. The outgoing serial link can still be used for other kinds of traffic. The frame is not TCP encapsulated. To configure HDLC encapsulation, use one of the following commands in interface configuration mode, as needed:

Command	Purpose
`Router(config-if)#` **stun route all interface serial** *number*	Forwards all HDLC or SDLC traffic of the identified interface number.
or	or
`Router(config-if)#` **stun route all interface serial** *number* **direct**	Forwards all HDLC or SDLC traffic on a direct STUN link.
or	or
`Router(config-if)#` **stun route address** *address-number* **interface serial** *number*	Forwards HDLC or SDLC traffic of the identified address.
or	or
`Router(config-if)#` **stun route address** *address-number* **interface serial** *number* **direct**	Forwards HDLC or SDLC traffic of the identified address across a direct STUN link.

Use the **no** forms of these commands to disable HDLC encapsulation.

Note You can forward all traffic only when you are using basic STUN protocol groups.

Configuring TCP Encapsulation without Local Acknowledgment

If you do not want to use SDLC local acknowledgment and only need to forward all SDLC frames encapsulated in TCP, use the following commands in interface configuration mode:

	Command	Purpose
Step 1	`Router(config-if)#` **stun route all tcp** *ip-address* [**passive**]	Forwards all TCP traffic for this IP address.
Step 2	`Router(config-if)#` **stun route address** *address-number* **tcp** *ip-address* [**local-ack**] [**priority**] [**tcp-queue-max**] [**passive**]	Specifies TCP encapsulation.

Use the **no** form of these commands to disable forwarding of all TCP traffic.

This configuration is typically used when two routers can be connected via an IP network as opposed to a point-to-point link.

Configuring TCP Encapsulation with SDLC Local Acknowledgment and Priority Queueing

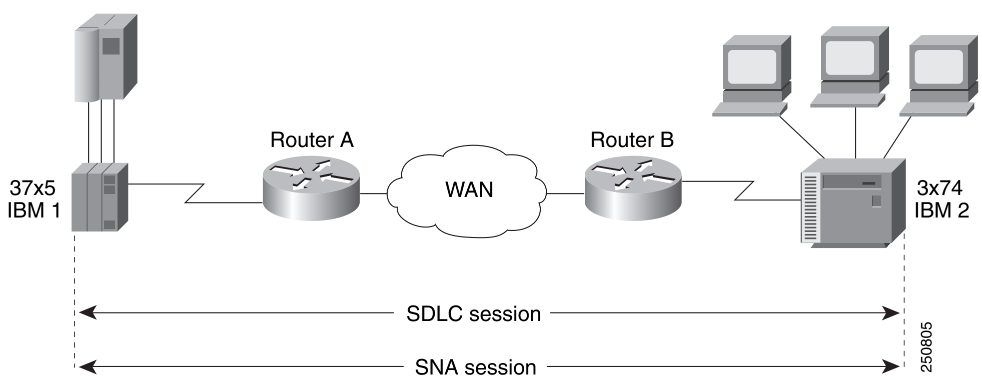
You configure SDLC local acknowledgment using TCP encapsulation. When you configure SDLC local acknowledgment, you also have the option to enable support for priority queueing.

Note To enable SDLC local acknowledgment, you must specify an SDLC or SDLC TG.

SDLC local acknowledgment provides local termination of the SDLC session so that control frames no longer travel the WAN backbone networks. This means that time-outs are less likely to occur.

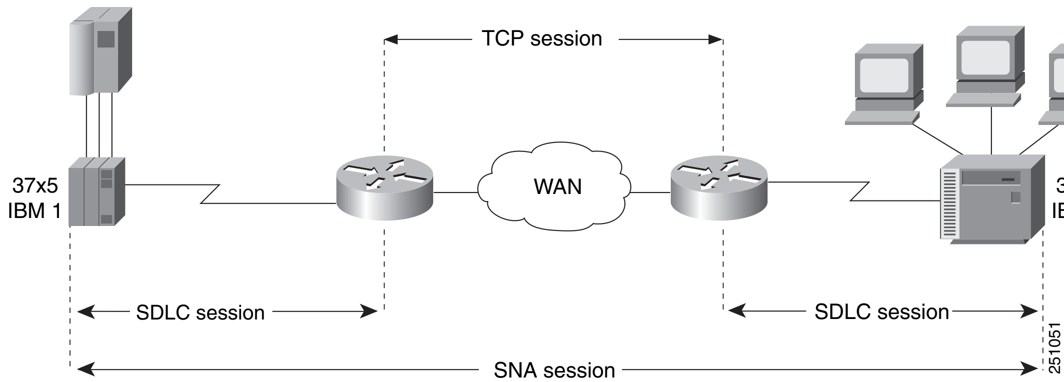
Figure 7 illustrates an SDLC session. IBM 1, using a serial link, can communicate with IBM 2 on a different serial link separated by a wide-area backbone network. Frames are transported between Router A and Router B using STUN, but the SDLC session between IBM 1 and IBM 2 is still end-to-end. Every frame generated by IBM 1 traverses the backbone network to IBM 2, which, upon receipt of the frame, acknowledges it.

Figure 7 SDLC Session Without Local Acknowledgment



With SDLC local acknowledgment, the SDLC session between the two end nodes is not end-to-end, but instead terminates at the two local routers, as shown in Figure 8. The SDLC session with IBM 1 ends at Router A, and the SDLC session with IBM 2 ends at Router B. Both Router A and Router B execute the full SDLC protocol as part of SDLC Local Acknowledgment. Router A acknowledges frames received from IBM 1. The node IBM 1 treats the acknowledgments it receives as if they are from IBM 2. Similarly, Router B acknowledges frames received from IBM 2. The node IBM 2 treats the acknowledgments it receives as if they are from IBM 1.

Figure 8 SDLC Session with Local Acknowledgment



To configure TCP encapsulation with SDLC local acknowledgment and priority queueing, perform the tasks in the following sections:

- Assigning the Router an SDLC Primary or Secondary Role, page 15
- Enabling the SDLC Local Acknowledgment Feature, page 15
- Establishing Priority Queueing Levels, page 16

Assigning the Router an SDLC Primary or Secondary Role

To establish local acknowledgment, the router must play the role of an SDLC primary or secondary node. Primary nodes poll secondary nodes in a predetermined order. Secondaries then send outgoing data, if they have any outgoing data.

For example, in an IBM environment, an FEP is the primary station and cluster controllers are secondary stations. If the router is connected to an FEP, the router should appear as a cluster controller and must be assigned the role of a secondary SDLC node. If the router is connected to a cluster controller, the router should appear as an FEP and must be assigned the role of a primary SDLC node. Devices connected to SDLC primary end-stations must play the role of an SDLC secondary and routers attached to SDLC secondary end stations must play the role of an SDLC primary station.

To assign the router a primary or secondary role, use one of the following commands in interface configuration mode, as needed:

Command	Purpose
`Router(config-if)#` **`stun sdlc-role primary`** or `Router(config-if)#` **`stun sdlc-role secondary`**	Assigns the STUN-enabled router an SDLC primary role. or Assigns the STUN-enabled router an SDLC secondary role.

Enabling the SDLC Local Acknowledgment Feature

To enable SDLC local acknowledgment, use the following command in interface configuration mode:

Command	Purpose
`Router(config-if)#` **`stun route address`** *`address-number`* **`tcp`** *`ip-address`* [**`local-ack`**] [**`priority`**] [**`tcp-queue-max`**] [**`passive`**]	Establishes SDLC local acknowledgment using TCP encapsulation.

The **stun route address 1 tcp local-ack priority tcp-queue-max** interface configuration command enables local acknowledgment and TCP encapsulation. Both options are required to use TGs. You should specify the SDLC address with the echo bit turned off for TG interfaces. The SDLC broadcast address 0xFF is routed automatically for TG interfaces. The **priority** keyword creates multiple TCP sessions for this route. The **tcp-queue-max** keyword sets the maximum size of the outbound TCP queue for the SDLC. The default TCP queue size is 100. The value for **hold-queue in** should be greater than the value for **tcp-queue-max**.

You can use the **priority** keyword (to set up the four levels of priorities to be used for TCP encapsulated frames) at the same time you enable local acknowledgment. The **priority** keyword is described in the following section. Use the **no** form of this command to disable SDLC Local Acknowledgment. For an example of how to enable local acknowledgment, see the "Example: Serial Link Address Prioritization Using STUN TCP/IP Encapsulation" section on page 29.

Establishing Priority Queueing Levels

With SDLC local acknowledgment enabled, you can establish priority levels used in priority queueing for serial interfaces. The priority levels are as follows:

- Low
- Medium
- Normal
- High

To set the priority queueing level, use the following command in interface configuration mode:

Command	Purpose
`Router(config-if)# ` **`stun route address`** *`address-number`* **`tcp`** *`ip-address`* `[`**`local-ack`**`]` **`priority`** `[`**`tcp-queue-max`**`] [`**`passive`**`]`	Establishes the four levels of priorities to be used in priority queueing.

Use the **no** form of this command to disable priority settings. For an example of how to establish priority queueing levels, see the "Example: Serial Link Address Prioritization Using STUN TCP/IP Encapsulation" section on page 29.

Configuring Local Acknowledgment for Direct Frame Relay Connectivity

To implement STUN with local acknowledgment using direct Frame Relay encapsulation, use the following command in interface configuration mode:

Command	Purpose
`Router(config-if)# ` **`stun route address`** *`sdlc-addr`* **`interface`** *`frame-relay-port`* **`dlci`** `number` *`localsap`* **`local-ack cls`**	Configures Frame Relay encapsulation between STUN peers with local acknowledgment.

Configuring STUN with Multilink Transmission Groups

You can configure multilink SDLC TGs across STUN connections between IBM communications controllers such as IBM 37x5s. Multilink TGs allow you to collapse multiple WAN leased lines into one leased line.

SDLC multilink TGs provide the following features:

- Network Control Program (NCP) SDLC address allowances, including echo and broadcast addressing.
- Remote NCP load sequence. After a SIM/RIM exchange but before a SNRM/UA exchange, NCPs send numbered I-frames. During this period, I-frames are not locally acknowledged, but instead are passed through. After the SNRM/UA exchange, local acknowledgment occurs.
- Rerouting of I-frames sent by the Cisco IOS software to the NCP if a link is lost in a multilink TG.
- Flow control rate tuning causes a sending NCP to “feel” WAN congestion and hold frames that would otherwise be held by the Cisco IOS software waiting to be sent on the WAN. This allows the NCP to perform its class-of-service algorithm more efficiently based on a greater knowledge of network congestion.

STUN connections that are part of a TG must have local acknowledgment enabled. Local acknowledgment keeps SDLC poll traffic off the WAN and reduces store-and-forward delays through the router. It also might minimize the number of NCP timers that expire due to network delay. Also, these STUN connections must go to the same IP address. This is because SNA TGs are parallel links between the same pair of IBM communications controllers.

Design Recommendations

This section provides some recommendations that are useful in configuring SDLC multilink TGs.

The bandwidth of the WAN should be larger than or equal to the aggregate bandwidth of all serial lines to avoid excessive flow control and to ensure response timed does not degrade. If other protocols are also using the WAN, ensure that the WAN bandwidth is significantly greater than the aggregate SNA serial line bandwidth to ensure that the SNA traffic does not monopolize the WAN.

When you use a combination of routed TGs and directly connected NCP TGs, you need to plan the configuration carefully to ensure that SNA sessions do not stop unexpectedly. Assuming that hardware reliability is not an issue, single-link routed TGs are as reliable as direct NCP-to-NCP single-link TGs. This is true because neither the NCP nor the Cisco IOS software can reroute I-frames when a TG has only one link. Additionally, a multilink TG directed between NCPs and a multilink TG through a router are equally reliable. Both can perform rerouting.

However, you might run into problems if you have a configuration in which two NCPs are directly connected (via one or more TG links) and one link in the TG is routed. The NCPs treat this as a multilink TG. However, the Cisco IOS software views the TG as a single-link TG.

A problem can arise in the following situation: Assume that an I-frame is being sent from NCP A (connected to router A) to NCP B (connected to router B) and that all SDLC links are currently active. Router A acknowledges the I-frame sent from NCP A and sends it over the WAN. If, before the I-frame reaches Router B, the SDLC link between router B and NCP B goes down, Router B attempts to reroute the I-frame on another link in the TG when it receives the I-frame. However, because this is a single-link TG, there are no other routes, and Router B drops the I-frame. NCP B never receives this I-frame because Router A acknowledges its receipt, and NCP A marks it as sent and deletes it. NCP B detects a gap in the TG sequence numbers and waits to receive the missing I-frame. NCP B waits forever for this I-frame, and does not send or receive any other frames. NCP B is technically not operational and all SNA sessions through NCP B are lost.

Finally, consider a configuration in which one or more lines of an NCP TG are connected to a router and one or more lines are directly connected between NCPs. If the network delay associated with one line of an NCP TG is different from the delay of another line in the same NCP TG, the receiving NCP spends additional time resequencing PIUs.

Setting Up STUN Traffic Priorities

To determine the order in which traffic should be handled on the network, use the methods described in the following sections:

- Assigning Queueing Priorities, page 18
- Prioritizing STUN Traffic over All Other Traffic, page 19

Assigning Queueing Priorities

To assign queueing priorities, perform the tasks in one of the following sections:

- Prioritizing by Serial Interface Address or TCP Port, page 18
- Prioritizing by Logical Unit Address, page 19

Prioritizing by Serial Interface Address or TCP Port

You can prioritize traffic on a per-serial-interface address or TCP port basis. You might want to do this so that traffic between one source-destination pair is always sent before traffic between another source-destination pair.

Note You must first enable local acknowledgment and priority levels as described earlier in this chapter.

To prioritize traffic, use one of the following commands in global configuration mode, as needed:

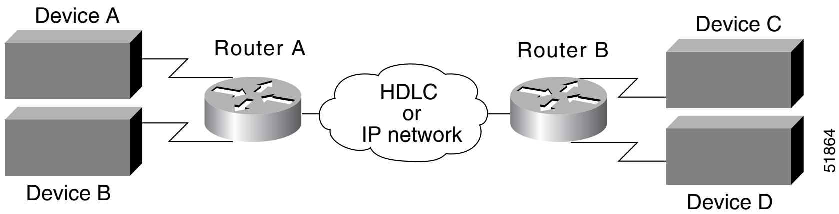
Command	Purpose
`Router(config)#` **`priority-list`** *`list-number`* **`protocol stun`** *`queue`* **`address`** *`group-number address-number`*	Assigns a queueing priority to the address of the STUN serial interface.
or	or
`Router(config)#` **`priority-list`** *`list-number`* **`protocol`** *`ip queue`* **`tcp`** *`tcp-port-number`*	Assigns a queueing priority to a TCP port.

You must also use the following command in interface configuration mode:

Command	Purpose
`Router(config-if)#` **`priority-group`** *`list-number`*	Assigns a priority list to a priority group.

Figure 9 illustrates serial link address prioritization. Device A communicates with Device C, and Device B communicates with Device D. With the serial link address prioritization, you can choose to give A-C a higher priority over B-D across the serial tunnel.

Figure 9 Serial Link Address Prioritization



To disable priorities, use the **no** forms of these commands.

For an example of how to prioritize traffic according to serial link address, see the "Example: Serial Link Address Prioritization Using STUN TCP/IP Encapsulation" section on page 29.

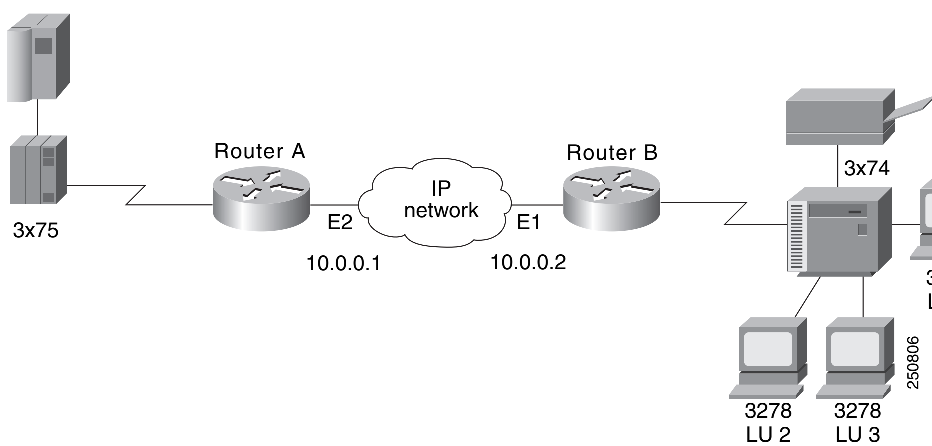
Prioritizing by Logical Unit Address

SNA local logical unit (LU) address prioritization is specific to IBM SNA connectivity and is used to prioritize SNA traffic on either STUN or remote source-route bridging (RSRB). To set the queueing priority by LU address, use the following command in global configuration mode:

Command	Purpose
`Router(config)#` **`locaddr-priority-list`** *`list-number address-number queue-keyword`*	Assigns a queueing priority based on the LU address.

In Figure 10, LU address prioritization can be set so that particular LUs receive data in preference to others or so that LUs have priority over the printer, for example.

Figure 10 SNA LU Address Prioritization



To disable this priority, use the **no** form of this command.

For an example of how to prioritize traffic according to logical unit address, see the "Example: LOCADDR Priority Groups for STUN" section on page 34.

Prioritizing STUN Traffic over All Other Traffic

You can prioritize STUN traffic to be routed first before all other traffic on the network. To give STUN traffic this priority, use the following command in global configuration mode:

Command	Purpose
Router(config)# **priority-list** *list-number* **protocol stun** *queue* **address** *group-number address-number*	Prioritizes STUN traffic in your network over that of other protocols.

To disable this priority, use the **no** form of this command.

For an example of how to prioritize STUN traffic over all other traffic, see the "Example: Serial Link Address Prioritization Using STUN TCP/IP Encapsulation" section on page 29.

Monitoring and Maintaining STUN Network Activity

You can list statistics regarding STUN interfaces, protocol groups, number of packets sent and received, local acknowledgment states, and more. To get activity information, use the following command in privileged EXEC mode:

Command	Purpose
Router# **show stun**	Lists the status display fields for STUN interfaces.

Enabling BSTUN

To enable BSTUN in IP networks, use the following commands in global configuration mode:

	Command	Purpose
Step 1	Router(config)# **bstun peer-name** *ip-address*	Enables BSTUN.
Step 2	Router(config)# **bstun lisnsap** *sap-value*	Configures a SAP on which to listen for incoming calls.

The IP address in the **bstun peer-name** command defines the address by which this BSTUN peer is known to other BSTUN peers that are using the TCP transport. If this command is unconfigured or the **no** form of this command is specified, all BSTUN routing commands with IP addresses are deleted. BSTUN routing commands without IP addresses are not affected by this command.

The **bstun lisnsap** command specifies a SAP on which to detect incoming calls.

Defining the Protocol Group

Define a BSTUN group and specify the protocol it uses. To define the protocol group, use the following command in global configuration mode:

Command	Purpose
Router(config)# **bstun protocol-group** *group-number* {**bsc** | **bsc-local-ack** | **adplex** | **adt-poll** | **adt-poll-select** | **adt-vari-poll** | **diebold** | **async-generic** | **mdi**}	Defines the protocol group.

The **bsc-local-ack** protocol option only works for 3270 Bisync uses.

The block serial protocols include bsc, bsc-local-ack, adplex, adt-poll-select, adt-vari-poll, diebold, async-generic, and mdi.

Traditionally, the adt-poll-select protocol is used over land-based links, while the adt-vari-poll protocol is used over satellite (VSAT) links. The adt-vari-poll protocol typically uses a much slower polling rate when alarm consoles poll alarm panels because adt-vari-poll allows alarm panels to send unsolicited messages to the alarm console. In an adt-vari-poll configuration, alarm panels do not have wait for the console to poll them before responding with an alarm, they automatically send the alarm.

Interfaces configured to run the adplex protocol have their baud rate set to 4800 bps, use even parity, 8 data bits, 1 start bit, and 1 stop bit.

Interfaces configured to run the adt-poll-select and adt-vari-poll protocols have their baud rate set to 600 bps, use even parity, 8 data bits, 1 start bit, and 1.5 stop bits. If different line configurations are required, use the **rxspeed**, **txspeed**, **databits**, **stopbits**, and **parity** line configuration commands to change the line attributes.

Interfaces configured to run the diebold protocol have their baud rate set to 300 bps, use even parity, 8 data bits, 1 start bit, and 2 stop bits. If different line configurations are required, use the **rxspeed**, **txspeed**, **databits**, and **parity** line configuration commands to change the line attributes.

Interfaces configured to run the async-generic protocol have their baud rate set to 9600 bps, use no parity, 8 data bits, 1 start bit, and 1 stop bit. If different line configurations are required, use the **rxspeed**, **txspeed**, **databits**, **stopbits**, and **parity** line configuration commands to change the line attributes.

Interfaces configured to run the mdi protocol have their baud rate set to 600 bps, use even parity, 8 data bits, 1 start bit, and 1.5 stop bits. If different line configurations are required, use the **rxspeed**, **txspeed**, **databits**, **stopbits**, and **parity** line configuration commands to change the line attributes. The mdi protocol allows alarm panels to be sent to the MDI alarm console.

Enabling BSTUN Keepalive

To define the number of times to attempt a peer connection before declaring the peer connection be down, use the following command in global configuration mode:

Command	Purpose
`Router(config)#` **`bstun keepalive-count`**	Specifies the number of times to attempt a peer connection.

Enabling BSTUN Remote Keepalive

To enable detection of the loss of a peer, use the following command in global configuration mode:

:

Command	Purpose
`Router(config)#` **`bstun remote-peer-keepalive`** *`seconds`*	Enables detection of the loss of a peer.

Enabling Frame Relay Encapsulation

To enable Frame Relay encapsulation, use the following commands beginning in global configuration mode:

	Command	Purpose
Step 1	`Router(config)#` **`interface serial`** *`number`*	Specifies a serial port.
Step 2	`Router(config)#` **`encapsulation frame-relay`**	Enables Frame Relay encapsulation on the serial port.

Defining Mapping Between BSTUN and DLCI

To configure the mapping between BSTUN and the DLCI, use one of the following commands in interface configuration mode, as needed:

Command	Purpose
`Router(config-if)#` **`frame-relay map bstun`** *`dlci`*	Defines the mapping between BSTUN and the DLCI when using BSC passthrough.
`Router(config-if)#` **`frame-relay map llc2`** *`dlci`*	Defines the mapping between BSTUN and the DLCI when using BSC local acknowledgment.

Note Direct encapsulation over Frame Relay is supported only for an encapsulation type of cisco, configured using the **encapsulation frame-relay** command.

Configuring BSTUN on the Serial Interface

Configure BSTUN on the serial interface before issuing any further BSTUN or protocol configuration commands for the interface. To configure the BSTUN function on a specified interface, use the following commands in interface configuration mode:

	Command	Purpose
Step 1	`Router(config-if)#` **`interface serial`** *`number`*	Specifies a serial port.
Step 2	`Router(config-if)#` **`encapsulation bstun`**	Configures BSTUN on an interface.

Note Configure the encapsulation bstun command on an interface before configuring any other BSTUN commands for the interface.

Placing a Serial Interface in a BSTUN Group

Each BSTUN-enabled interface on a router must be placed in a previously defined BSTUN group. Packets will only travel between BSTUN-enabled interfaces that are in the same group. To assign a serial interface to a BSTUN group, use the following command in interface configuration mode:

Command	Purpose
`Router(config-if)#` **`bstun group`** *`group-number`*	Assigns a serial interface to a BSTUN group.

Specifying How Frames Are Forwarded

To specify how frames are forwarded when received on a BSTUN interface, use one of the following commands in interface configuration mode, as needed:

Command	Purpose
`Router(config-if)#` **`bstun route address`** *`address-number`* **`interface serial`** *`number`*	Propagates the serial frame that contains a specific address. HDLC encapsulation is used to propagate the serial frames.
`Router(config-if)#` **`bstun route all interface serial`** *`number`*	Propagates all BSTUN traffic received on the input interface, regardless of the address contained in the serial frame. HDLC encapsulation is used to propagate the serial frames.
`Router(config-if)#` **`bstun route address`** *`address-number`* **`tcp`** *`ip-address`*	Propagates the serial frame that contains a specific address. TCP encapsulation is used to propagate frames that match the entry.
`Router(config-if)#` **`bstun route all tcp`** *`ip-address`*[1]	Propagates all BSTUN traffic received on the input interface, regardless of the address contained in the serial frame. TCP encapsulation is used to propagate frames that match the entry.
`Router(config-if)#` **`bstun route address`** *`cu-address`* **`interface serial`** *`serial-int`* [**`dlci`** *`dlci`*]	Propagates the serial frame that contains a specific address. Specifies the control unit address for the Bisync end station. Frame Relay encapsulation is used to propagate the serial frames.
`Router(config-if)#` **`bstun route all interface serial`** *`serial-int`* [**`dlci`** *`dlci`*]	Propagates all frames regardless of the control unit address for the Bisync end station. Frame Relay encapsulation is used to propagate the serial frames in bisync passthrough mode.
`Router(config-if)#` **`bstun route address`** *`cu-address`* **`interface serial`** *`serial-int`* [**`dlci`** *`dlci rsap`*] [**`priority`** *`priority`*]	Propagates the serial frame that contains a specific address. Specifies the control unit address for the bisync end station. Frame Relay encapsulation is used to propagate the serial frames for Bisync local acknowledgment mode.
`Router(config-if)#` **`bstun route all interface serial`** *`serial-int`* [**`dlci`** *`dlci rsap`*] [**`priority`** *`priority]`*]	Propagates all BSTUN traffic received on the input interface, regardless of the address contained in the serial frame. Frame Relay encapsulation is used to propagate the serial frames.

1. The **bstun route all tcp** command functions in either passthrough or local acknowledgment mode.

Note Every BSTUN route statement must have a corresponding route statement on the BSTUN peer. For example, a **bstun route address** *address1* **tcp** *peer2ip* statement on PEER1 must have a corresponding **bstun route address** *address1* **tcp** *peer1ip* statement on PEER2. Similarly, a **bstun route address** statement cannot map to a **bstun route all** statement, and vice versa.

For Bisync local acknowledgment, we recommend that you use the **bstun route all tcp** command. This command reduces the amount of duplicate configuration detail that would otherwise be needed to specify devices at each end of the tunnel.

Setting Up BSTUN Traffic Priorities

You can assign BSTUN traffic priorities based on either the BSTUN header or the TCP port. To prioritize traffic, use one of the following commands in global configuration mode, as needed:

Command	Purpose
Router(config)# **priority-list** *list-number* **protocol bstun** *queue* [**gt** *packetsize*] [**lt** *packetsize*] **address** *bstun-group bsc-addr*	Establishes BSTUN queueing priorities based on the BSTUN header.
Router(config)# **priority-list** *list-number* **protocol ip** *queue* **tcp** *tcp-port-number*	Assigns a queueing priority to TCP port.

You can customize BSTUN queueing priorities based on either the BSTUN header or TCP port. To customize priorities, use one of the following commands in global configuration mode, as needed:

Command	Purpose
Router(config)# **queue-list** *list-number* **protocol bstun queue** [**gt** *packetsize*][**lt** packetsize] **address** *bstun-group bsc-addr*	Customizes BSTUN queueing priorities based on the BSTUN header.
Router(config)# **queue-list** *list-number* **protocol ip** *queue* **tcp** *tcp-port-number*	Customizes BSTUN queueing priorities based on the TCP port.

Note Because the asynchronous security protocols share the same tunnels with Bisync when configured on the same routers, any traffic priorities configured for the tunnel apply to both Bisync and the various asynchronous security protocols.

Configuring Protocol Group Options on a Serial Interface

Depending on the selected block serial protocol group, you must configure one or more options for that protocol group. The options for each of these protocol groups are explained in the following sections:

- Configuring Bisync Options on a Serial Interface, page 24
- Configuring Asynchronous Security Protocol Options on a Serial Interface, page 25

Configuring Bisync Options on a Serial Interface

To configure Bisync options on a serial interface, use one of the following commands in interface configuration mode, as needed:

Command	Purpose
Router(config-if)# **bsc char-set** {**ascii** | **ebcdic**}	Specifies the character set used by the Bisync support feature.
Router(config-if)# **bsc contention** *address*	Specifies an address on a contention interface.
Router(config-if)# **bsc dial-contention** *time-out*	Specifies that the router at the central site will behave as a central router with dynamic allocation of serial interfaces. The timeout value is the length of time an interface can be idle before it is returned to the idle interface pool.
Router(config-if)# **bsc extended-address** *poll-address select-address*	Specifies a nonstandard Bisync address.
Router(config-if)# **full-duplex**	Specifies that the interface can run Bisync in full-duplex mode.
Router(config-if)# **bsc pause** *time*	Specifies the amount of time between the start of one polling cycle and the next.
Router(config-if)# **bsc poll-timeout** *time*	Specifies the timeout for a poll or a select sequence.
Router(config-if)# **bsc host-timeout** *time*	Specifies the timeout for a nonreception of poll or a select sequence from the host. If the frame is not received within this time, the remote connection will be deactivated.
Router(config-if)# **bsc primary**	Specifies that the router is acting as the primary end of the Bisync link.
Router(config-if)# **bsc retries** *retry-count*	Specifies the number of retries before a device is considered to have failed.
Router(config-if)# **bsc secondary**	Specifies that the router is acting as the secondary end of the Bisync link.
Router(config-if)# **bsc spec-poll**	Specifies specific polls, rather than general polls, used on the host-to-router connection.
Router(config-if)# **bsc servlim** *servlim-count*	Specifies the number of cycles of the active poll list that are performed between polls to control units in the inactive poll list.

Configuring Asynchronous Security Protocol Options on a Serial Interface

To configure asynchronous security protocol options on a serial interface, use one or more of the following commands in interface configuration mode:

Command	Purpose
Router(config-if)# **asp role primary**	Specifies that the router is acting as the primary end of the polled asynchronous link.
Router(config-if)# **asp role secondary**	Specifies that the router is acting as the secondary end of the polled asynchronous link.
Router(config-if)# **asp addr-offset** *address-offset*	Configures an asynchronous port to send and receive polled asynchronous traffic through a BSTUN tunnel.
Router(config-if)# **asp rx-ift** *interframe-timeout*	For asynchronous-generic configurations, specifies the timeout period between frames to delineate the end of one frame being received from the start of the next frame.

Configuring Direct Serial Encapsulation for Passthrough Peers

To configure direct serial encapsulation for passthrough peers, use the following command in interface configuration mode:

Command	Purpose
`Router(config-if)#` **`frame relay map bstun`**	Configures the Frame Relay interface for passthrough.

Configuring Local Acknowledgment Peers

To configure local acknowledgment peers, use the following command in interface configuration mode:

Command	Purpose
`Router(config-if)#` **`frame-relay map llc2`** *`dlci`*	Configures the Frame Relay interface for local acknowledgment.

Monitoring and Maintaining the Status of BSTUN

To list statistics for BSTUN interfaces, protocol groups, number of packets sent and received, local acknowledgment states, and other activity information, use the following commands in EXEC mode:

Command	Purpose
`Router#` **`show bstun`** [**`group`** *`bstun-group-number`*] [**`address`** *`address-list`*]	Lists the status display fields for BSTUN interfaces.
`Router#` **`show bsc`** [**`group`** *`bstun-group-number`*] [**`address`** *`address-list`*]	Displays status of the interfaces on which Bisync is configured.

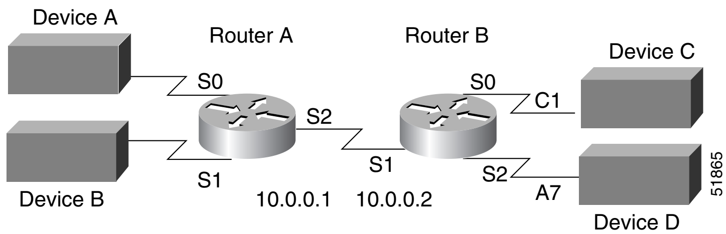
Configuration Examples for Serial Tunnel and Block Serial Tunnel

The following sections provide STUN and BSTUN configuration examples:

Example: STUN Priorities Using HDLC Encapsulation

Assume that the link between Router A and Router B in Figure 11 is a serial tunnel that uses the simple serial transport mechanism. Device A communicates with Device C (SDLC address C1) with a high priority. Device B communicates with Device D (SDLC address A7) with a normal priority.

Figure 11 STUN Simple Serial Transport



The following configurations set the priority of STUN hosts A, B, C, and D.

Router A

```
stun peer-name 10.0.0.1
stun protocol-group 1 sdlc
stun protocol-group 2 sdlc
!
interface serial 0
 no ip address
 encapsulation stun
 stun group 1
 stun route address C1 interface serial 2
!
interface serial 1
 no ip address
 encapsulation stun
 stun group 2
 stun route address A7 interface serial 2
!


interface serial 2
 ip address 10.0.0.1 255.0.0.0
 priority-group 1
!
priority-list 1 protocol stun high address 1 C1
priority-list 1 protocol stun low address 2 A7
```

Router B

```
stun peer-name 10.0.0.2
stun protocol-group 1 sdlc
stun protocol-group 2 sdlc
!
interface serial 0
 no ip address
 encapsulation stun
 stun group 1
 stun route address C1 interface serial 1
!
interface serial 1
 ip address 10.0.0.2 255.0.0.0
 priority-group 1
!
interface serial 2
 no ip address
 encapsulation stun
 stun group 2
 stun route address A7 interface serial 1
!
priority-list 1 protocol stun high address 1 C1
```

```
priority-list 1 protocol stun low address 2 A7
```

Example: SDLC Broadcast

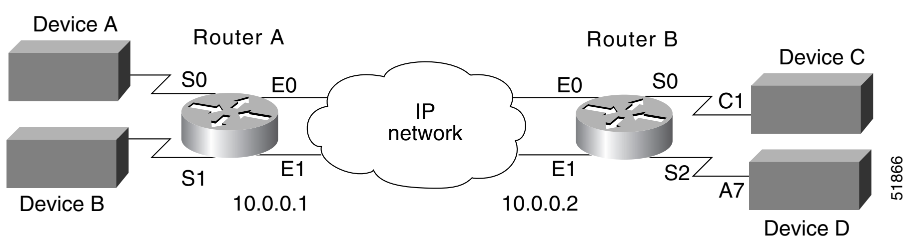
In the following example, an FEP views end stations 1, 2, and 3 as if they were on an SDLC multidrop link. Any broadcast frame sent from the FEP to Router A is duplicated and sent to each of the downstream routers (B and C.)

```
stun peer-name xxx.xxx.xxx.xxx
stun protocol-group 1 sdlc
interface serial 1
encapsulation stun
 stun group 1
 stun sdlc-role secondary
 sdlc virtual-multidrop
 sdlc address 1
 sdlc address 2
 sdlc address 3
 stun route address 1 tcp yyy.yyy.yyy.yyy local-ack
 stun route address 2 tcp zzz.zzz.zzz.zzz local-ack
 stun route address 3 tcp zzz.zzz.zzz.zzz local-ack
 stun route address FF tcp yyy.yyy.yyy.yyy
 stun route address FF tcp zzz.zzz.zzz.zzz
```

Example: Serial Link Address Prioritization Using STUN TCP/IP Encapsulation

Assume that the link between Router A and Router B is a serial tunnel that uses the TCP/IP encapsulation as shown in Figure 12. Device A communicates with Device C (SDLC address C1) with a high priority. Device B communicates with Device D (SDLC address A7) with a normal priority. The configuration file for each router follows the figure.

Figure 12 STUN TCP/IP Encapsulation



Router A

```
stun peer-name 10.0.0.1
stun protocol-group 1 sdlc
stun protocol-group 2 sdlc
!
interface serial 0
 no ip address
 encapsulation stun
 stun group 1
 stun route address C1 tcp 10.0.0.2 local-ack priority
 priority-group 1
!
interface serial 1
 no ip address
```

```
 encapsulation stun
 stun group 2
 stun route address A7 tcp 10.0.0.2 local-ack priority
 priority-group 2
!
interface ethernet 0
 ip address 10.0.0.1 255.0.0.0
 priority-group 3
!
interface ethernet 1
 ip address 10.0.0.3 255.0.0.0
priority-group 3
! This list tells interface Serial 0 which tcp port numbers on the WAN interface
! correspond to the high, medium, normal and low priority queues.
priority-list 1 protocol ip high tcp 1994
priority-list 1 protocol ip medium tcp 1990
priority-list 1 protocol ip normal tcp 1991
priority-list 1 protocol ip low tcp 1992
priority-list 1 protocol stun high address 1 C1


! This list tells interface Serial 1 which tcp port numbers
! on the WAN interface correspond to the high, medium, normal
! and low priority queues.
priority-list 2 protocol ip high tcp 1994
priority-list 2 protocol ip medium tcp 1990
priority-list 2 protocol ip normal tcp 1991
priority-list 2 protocol ip low tcp 1992
priority-list 2 protocol stun normal address 2 A7
! This list establishes the high, medium, normal, and low
! priority queues on the WAN interfaces.
priority-list 3 protocol ip high tcp 1994
priority-list 3 protocol ip medium tcp 1990
priority-list 3 protocol ip normal tcp 1991
priority-list 3 protocol ip low tcp 1992
!
hostname routerA
router igrp
network 1.0.0.0
```

Router B

```
stun peer-name 10.0.0.2
stun protocol-group 1 sdlc
stun protocol-group 2 sdlc
!
interface serial 0
 no ip address
 encapsulation stun
 stun group 1
 stun route address C1 tcp 10.0.0.1 local-ack priority
 priority-group 1
!
interface serial 2
 no ip address
 encapsulation stun
 stun group 2
 stun route address A7 tcp 10.0.0.1 local-ack priority
 priority-group 2
!
interface ethernet 0
 ip address 10.0.0.2 255.0.0.0
priority-group 3
!
```

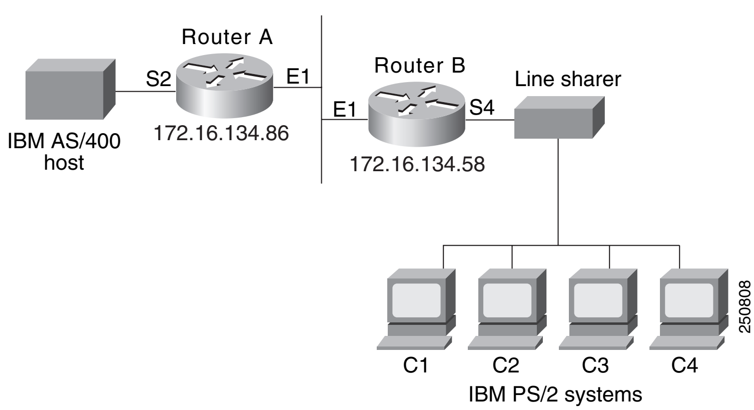
```
interface ethernet 1
 ip address 10.0.0.4 255.0.0.0
priority-group 3
! This list tells interface Serial 0 which tcp port numbers
! on the WAN interface correspond to the high, medium, normal
! and low priority queues.
priority-list 1 protocol ip high tcp 1994
priority-list 1 protocol ip medium tcp 1990
priority-list 1 protocol ip normal tcp 1991
priority-list 1 protocol ip low tcp 1992
priority-list 1 protocol stun high address 1 C1


! This list tells interface Serial 2 which tcp port numbers
! on the WAN interface correspond to the high, medium, normal
! and low priority queues.
priority-list 2 protocol ip high tcp 1994
priority-list 2 protocol ip medium tcp 1990
priority-list 2 protocol ip normal tcp 1991
priority-list 2 protocol ip low tcp 1992
priority-list 2 protocol stun normal address 2 A7
! This list establishes the high, medium, normal, and low
! priority queues on the WAN interface(s).
priority-list 3 protocol ip high tcp 1994
priority-list 3 protocol ip medium tcp 1990
priority-list 3 protocol ip normal tcp 1991
priority-list 3 protocol ip low tcp 1992
!
hostname routerB
router igrp 109
network 1.0.0.0
```

Example: STUN Multipoint Implementation Using a Line-Sharing Device

In Figure 13, four separate PS/2 computers are connected to a line-sharing device off of Router B. Each PS/2 computer has four sessions open on an AS/400 device attached to Router A. Router B functions as the primary station, while Router A functions as the secondary station. Both routers locally acknowledge packets from the IBM PS/2 systems.

Figure 13 STUN Communication Involving a Line-Sharing Device



The configuration file for the routers shown in Figure 13 follows.

Router A

```
! enter the address of the stun peer
stun peer-name 172.16.134.86
! specify that group 4 uses the SDLC protocol
stun protocol-group 4 sdlc
stun remote-peer-keepalive
!
interface ethernet 1
! enter the IP address for the Ethernet interface
 ip address 172.16.134.86 255.255.255.0
! description of IBM AS/400 link
interface serial 2
! description of IBM AS/400 link; disable the IP address on a serial interface
 no ip address
! enable STUN encapsulation on this interface
 encapsulation stun
! apply previously defined stun group 4 to serial interface 2
 stun group 4
! establish this router as a secondary station
stun sdlc-role secondary
! wait up to 63000 msec for a poll from the primary before timing out
 sdlc poll-wait-timeout 63000
! list addresses of secondary stations (PS/2 systems) attached to link
sdlc address C1
sdlc address C2
sdlc address C3
sdlc address C4
! use tcp encapsulation to send frames to SDLC stations C1, C2, C3, or
! C4 and locally terminate sessions with these stations
stun route address C1 tcp 172.16.134.58 local-ack
stun route address C2 tcp 172.16.134.58 local-ack
stun route address C3 tcp 172.16.134.58 local-ack
stun route address C4 tcp 172.16.134.58 local-ack
```

Router B

```
! enter the address of the stun peer
stun peer-name 172.16.134.58
! this router is part of SDLC group 4
stun protocol-group 4 sdlc
stun remote-peer-keepalive
!
interface ethernet 1
! enter the IP address for the Ethernet interface
ip address 172.16.134.58 255.255.255.0
!
! description of PS/2 link
interface serial 4
! disable the IP address on a serial interface
 no ip address
! enable STUN encapsulation on this interface
 encapsulation stun
! apply previously defined stun group 4 to serial interface 2
stun group 4
! establish this router as a primary station
stun sdlc-role primary
sdlc line-speed 9600
! wait 2000 milliseconds for a reply to a frame before resending it
sdlc t1 2000
! resend a frame up to four times if not acknowledged
sdlc n2 4


! list addresses of secondary stations (PS/2 systems) attached to link
```

```
sdlc address C1
sdlc address C2
sdlc address C3
sdlc address C4
! use tcp encapsulation to send frames to SDLC stations C1, C2, C3, or
! C4 and locally terminate sessions with these stations
stun route address C3 tcp 172.16.134.86 local-ack
stun route address C1 tcp 172.16.134.86 local-ack
stun route address C4 tcp 172.16.134.86 local-ack
stun route address C2 tcp 172.16.134.86 local-ack
! set the clock rate on this interface to 9600 bits per second
clock rate 9600
```

Example: STUN Local Acknowledgment for SDLC

The following example shows a sample configuration for a pair of routers performing SDLC local acknowledgment.

Router A

```
stun peer-name 172.16.64.92
stun protocol-group 1 sdlc
stun remote-peer-keepalive
!
interface Serial 0
 no ip address
 encapsulation stun
 stun group 1
 stun sdlc-role secondary
 sdlc address C1
 stun route address C1 tcp 172.16.64.93 local-ack
 clock rate 19200
```

Router B

```
stun peer-name 172.16.64.93
stun protocol-group 1 sdlc
stun remote-peer-keepalive
!
interface Serial 0
 no ip address
 encapsulation stun
 stun group 1
 stun sdlc-role primary
 sdlc line-speed 19200
 sdlc address C1
 stun route address C1 tcp 172.16.64.92 local-ack
 clock rate 19200
```

Example: STUN Local Acknowledgment for Frame Relay

The following example describes an interface configuration for Frame Relay STUN with local acknowledgment:

```
stun peer-name 10.1.21.1 cls 4
stun protocol-group 120 sdlc
!
interface Serial1
 no ip address
 encapsulation frame-relay
```

```
 frame-relay lmi-type ansi
 frame-relay map llc2 22
!
interface Serial4
 no ip address
 encapsulation stun
 clock rate 9600
 stun group 120
 stun sdlc-role secondary
 sdlc address C1
 sdlc address C2
 stun route address C1 interface Serial1 dlci 22 04 local-ack
 stun route address C2 interface Serial1 dlci 22 08 local-ack
```

Example: LOCADDR Priority Groups

The following example shows how to establish queueing priorities on a STUN interface based on an LU address:

```
stun peer-name 131.108.254.6
stun protocol-group 1 sdlc
! give locaddr-priority-list 1 a high priority for LU 02
locaddr-priority-list 1 02 high
! give locaddr-priority-list 1 a low priority for LU 05
locaddr-priority-list 1 05 low
!
interface serial 0
! disable the ip address for interface serial 0
 no ip address
! enable the interface for STUN
 encapsulation stun
 stun group 2
 stun route address 10 tcp 131.108.254.8 local-ack priority
! assign priority group 1 to the input side of interface serial 0
locaddr-priority 1
priority-group 1
```

Example: LOCADDR Priority Groups for STUN

The following configuration example shows how to assign a priority group to an input interface:

Router A

```
stun peer-name 10.0.0.1
stun protocol-group 1 sdlc
locaddr-priority-list 1 02 high
locaddr-priority-list 1 03 high
locaddr-priority-list 1 04 medium
locaddr-priority-list 1 05 low
!
interface serial 0
 no ip address
 encapsulation stun
 stun group 1
 stun route address C1 tcp 10.0.0.2 local-ack priority
 clock rate 19200
 locaddr-priority 1
 priority-group 1
!
interface Ethernet 0
```

```
 ip address 10.0.0.1 255.255.255.0
!
 priority-list 1 protocol ip high tcp 1994
 priority-list 1 protocol ip medium tcp 1990
 priority-list 1 protocol ip normal tcp 1991
 priority-list 1 protocol ip low tcp 1992
```

Router B

```
stun peer-name 10.0.0.2
stun protocol-group 1 sdlc
locaddr-priority-list 1 02 high
locaddr-priority-list 1 03 high
locaddr-priority-list 1 04 medium
locaddr-priority-list 1 05 low
!
interface serial 0
 no ip address
 encapsulation stun
 stun group 1
 stun route address C1 tcp 10.0.0.1 local-ack priority
 clock rate 19200
 locaddr-priority 1
 priority-group 1
!
interface Ethernet 0
 ip address 10.0.0.2 255.255.255.0
!
 priority-list 1 protocol ip high tcp 1994
 priority-list 1 protocol ip medium tcp 1990
 priority-list 1 protocol ip normal tcp 1991
 priority-list 1 protocol ip low tcp 1992
```

BSTUN Configuration Examples

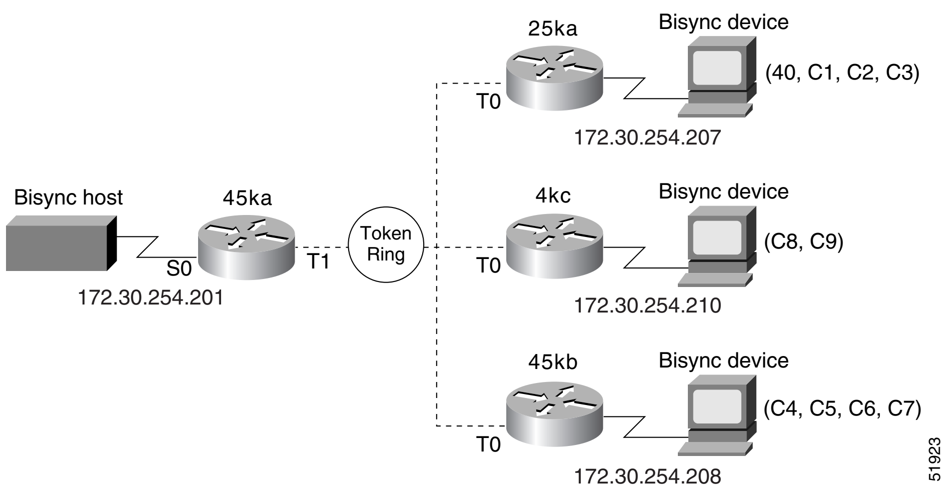
The following sections provide BSTUN configuration examples:

- Example: Simple Bisync Configuration, page 36
- Example: Bisync Addressing on Contention Interfaces, page 40
- Example: Nonstandard Bisync Addressing, page 40
- Example: Priority Queueing: With Priority Based on BSTUN Header, page 40
- Example: Priority Queueing: With Priority Based on BSTUN Header and Packet Sizes, page 41
- Example: Priority Queueing: With Priority Based on BSTUN Header and Bisync Address, page 41
- Priority Queueing: With Priority Based on BSTUN TCP Ports Example, page 41
- Example: Priority Queueing: With Priority Based on BSTUN TCP Ports and Bisync Address, page 42
- Example: Custom Queueing: With Priority Based on BSTUN Header, page 42
- Example: Custom Queueing: With Priority Based on BSTUN Header and Packet Size, page 43
- Example: Custom Queueing: With Priority Based on BSTUN Header and Bisync Address, page 43
- Example: Custom Queueing: With Priority Based on BSTUN TCP Ports, page 43

Example: Simple Bisync Configuration

Figure 14 shows a simple Bisync configuration example.

Figure 14 Simple Bisync Configuration



The configuration files for the routers shown in Figure 14 follows.

Router 45ka

```
version 10.2
!
hostname 45ka
!
no ip domain-lookup
!
bstun peer-name 172.30.254.201
bstun protocol-group 1 bsc
!
interface ethernet 0
 ip address 198.92.0.201 255.255.255.0
 media-type 10BaseT
!
interface ethernet 1
 no ip address
 shutdown
 media-type 10BaseT
!


interface serial 0
 no ip address
 encapsulation bstun
```

```
 clock rate 19200
 bstun group 1
 bsc char-set ebcdic
 bsc secondary
 bstun route address C9 tcp 172.30.254.210
bstun route address C8 tcp 172.30.254.210
bstun route address C7 tcp 172.30.254.208
bstun route address C6 tcp 172.30.254.208
bstun route address C5 tcp 172.30.254.208
bstun route address C4 tcp 172.30.254.208
bstun route address C3 tcp 172.30.254.207
bstun route address C2 tcp 172.30.254.207
bstun route address C1 tcp 172.30.254.207
bstun route address 40 tcp 172.30.254.207
!
interface serial 1
no ip address
shutdown
!
interface serial 2
 no ip address
 shutdown
!
interface serial 3
 no ip address
 shutdown
!
interface tokenring 0
 no ip address
 shutdown
!
interface tokenring 1
 ip address 172.30.254.201 255.255.255.0
 ring-speed 16
!
line con 0
line aux 0
line vty 0 4
login
!
end
```

Router 25ka

```
version 10.2
!
hostname 25ka
!
no ip domain-lookup
!
bstun peer-name 172.30.254.207
bstun protocol-group 1 bsc
!
interface serial 0
 no ip address
 shutdown
!


interface serial 1
 no ip address
 encapsulation bstun
 clock rate 19200
 bstun group 1
```

```
 bsc char-set ebcdic
 bsc primary
 bstun route address C3 tcp 172.30.254.201
 bstun route address C2 tcp 172.30.254.201
 bstun route address C1 tcp 172.30.254.201
 bstun route address 40 tcp 172.30.254.201
!
interface tokenring 0
 ip address 172.30.254.207 255.255.255.0
 ring-speed 16
!
interface bri 0
 no ip address
 shutdown
!
line con 0
line aux 0
line vty 0 4
login
!
end
```

Configuration for Router 4kc

```
version 10.2
!
hostname 4kc
!
no ip domain-lookup
!
bstun peer-name 172.30.254.210
bstun protocol-group 1 bsc
!
interface ethernet 0
 ip address 198.92.0.210 255.255.255.0
 media-type 10BaseT
!
interface serial 0
 no ip address
 encapsulation bstun
 clock rate 19200
 bstun group 1
 bsc char-set ebcdic
 bsc primary
 bstun route address C9 tcp 172.30.254.201
 bstun route address C8 tcp 172.30.254.201
!
interface serial 1
 no ip address
 shutdown
!
interface serial 2
 no ip address
 shutdown
!


interface serial 3
 no ip address
 shutdown
!
interface tokenring 0
 ip address 172.30.254.210 255.255.255.0
 ring-speed 16
```

```
!
interface tokenring 1
 no ip address
 shutdown!
line con 0
line aux 0
line vty 0 4
login
!
end
```

Router 25kb

```
version 10.2
!
hostname 25kb
!
no ip domain-lookup
!
bstun peer-name 172.30.254.208
bstun protocol-group 1 bsc
!
interface serial 0
 no ip address
 encapsulation bstun
 no keepalive
 clock rate 19200
 bstun group 1
 bsc char-set ebcdic
 bsc primary
 bstun route address C7 tcp 172.30.254.201
 bstun route address C6 tcp 172.30.254.201
 bstun route address C5 tcp 172.30.254.201
 bstun route address C4 tcp 172.30.254.201
!
interface serial 1
 no ip address
 shutdown
!
interface tokenring 0
 ip address 172.30.254.208 255.255.255.0
 ring-speed 16
!
!
line con 0
line aux 0
line vty 0 4
login
!
end
```

Example: Bisync Addressing on Contention Interfaces

The following examples show user-configurable addressing on contention interfaces:

Remote Devices

```
bstun peer-name 1.1.1.20
bstun protocol-group 1 bsc
interface serial 0
 bstun group 1
 bsc contention 20
 bstun route address 20 tcp 1.1.1.1
```

Host Device

```
bstun peer-name 1.1.1.1
bstun protocol-group 1 bsc
interface serial 0
 bstun group 1
 bsc dial-contention 100
 bstun route address 20 tcp 1.1.1.20
 bstun route address 21 tcp 1.1.1.21
```

Example: Nonstandard Bisync Addressing

This example specifies an extended address on serial interface 0:

```
bstun peer-name 1.1.1.1
bstun protocol-group 1 bsc
!
interface serial 0
 bstun group 1
 bsc extended-address 23 83
 bsc extended-address 87 42
 bsc primary
 bstun route address 23 tcp 1.1.1.20
```

Example: Priority Queueing: With Priority Based on BSTUN Header

In the following example, the output interface examines header info and places packets with the BSTUN header on specified output queue:

```
priority-list 1 protocol bstun normal
interface serial 0
 priority-group 1
interface serial 1
 encapsulation bstun
 bstun group 1
 bsc char-set ebcdic
 bstun route all interface serial 0
                ...or...
bstun route address C1 interface serial 0
```

Example: Priority Queueing: With Priority Based on BSTUN Header and Packet Sizes

In the following example, the output interface examines header information and packet size and places packets with the BSTUN header that match criteria (gt or lt specified packet size) on specified output queue:

```
priority-list 1 protocol bstun low gt 1500
priority-list 1 protocol bstun hi lt 500
interface serial 0
 priority-group 1
interface serial 1
 encapsulation bstun
 bstun group 1
 bsc char-set ebcdic
 bstun route all interface serial 0
                ...or...
bstun route address C1 interface serial 0
```

Example: Priority Queueing: With Priority Based on BSTUN Header and Bisync Address

In the following example, the output interface examines header information and Bisync address and places packets with the BSTUN header that match Bisync address on specified output queue:

```
priority-list 1 protocol bstun normal address 1 C1
interface serial 0
 priority-group 1
interface serial 1
 encapsulation bstun
 bstun group 1
 bsc char-set ebcdic
 bstun route address C1 interface serial 0
```

Priority Queueing: With Priority Based on BSTUN TCP Ports Example

In the following example, the output interface examines TCP port number and places packets with the BSTUN port number (1976) on specified output queue:

```
priority-list 1 protocol ip high tcp 1976
interface serial 0
 priority-group 1
interface serial 1
 encapsulation bstun
 bstun group 1
 bstun route all tcp 200.190.30.1
```

Example: Priority Queueing: With Priority Based on BSTUN TCP Ports and Bisync Address

In the following example, four TCP/IP sessions (high, medium, normal, and low) are established with BSTUN peers using BSTUN port numbers. The input interface examines the Bisync address and uses the specified output queue definition to determine which BSTUN TCP session to use for sending the packet to the BSTUN peer.

The output interface examines the TCP port number and places packets with the BSTUN port numbers on the specified output queue.

```
priority-list 1 protocol ip high   tcp 1976
priority-list 1 protocol ip medium tcp 1977
priority-list 1 protocol ip normal tcp 1978
priority-list 1 protocol ip low    tcp 1979
!
priority-list 1 protocol bstun normal address 1 C1
!
interface serial 0
 priority-group 1
!
interface serial 1
 encapsulation bstun
 bstun group 1
 bsc char-set ebcdic
 bstun route address C1 tcp 200.190.30.1 priority
 priority-group 1
```

Example: Custom Queueing: With Priority Based on BSTUN Header

In the following example, the output interface examines header info and places packets with the BSTUN header on specified output queue.

```
queue-list 1 protocol bstun normal
!
interface serial 0
 custom-queue-list 1
!
interface serial 1
 encapsulation bstun
 bstun group 1
 bstun route all interface serial 0
```

Example: Custom Queueing: With Priority Based on BSTUN Header and Packet Size

In the following example, the output interface examines header information and packet size and places packets with the BSTUN header that match criteria (gt or lt specified packet size) on specified output queue.

```
queue-list 1 protocol bstun low gt 1500
queue-list 1 protocol bstun high lt 500
!
interface serial 0
 custom-queue-list 1
!
interface serial 1
 encapsulation bstun
 bstun group 1
 bstun route all interface serial 0
```

Example: Custom Queueing: With Priority Based on BSTUN Header and Bisync Address

In the following example, the output interface examines header info and Bisync address and places packets with the BSTUN header that match Bisync address on specified output queue.

```
queue-list 1 protocol bstun normal address 1 C1
!
interface serial 0
 custom-queue-list 1
!
interface serial 1
 encapsulation bstun
 bstun group 1
 bsc char-set ebcdic
 bstun route address C1 interface serial 0
```

Example: Custom Queueing: With Priority Based on BSTUN TCP Ports

In the following example, the output interface examines the TCP port number and places packets with the BSTUN port number (1976) on specified output queue:

```
queue-list 1 protocol ip high tcp 1976
!
interface serial 0
 custom-queue-list 1
!
interface serial 1
 encapsulation bstun
 bstun group 1
 bstun route all tcp 200.190.30.1
```

Example: Custom Queueing: With Priority Based on BSTUN TCP Ports and Bisync Address

In the following example, four TCP/IP sessions (high, medium, normal, and low) are established with BSTUN peers using BSTUN port numbers. The input interface examines the Bisync address and uses the specified output queue definition to determine which BSTUN TCP session to use.

The output interface examines the TCP port number and places packets with the BSTUN port numbers on the specified output queue.

For Bisync addressing, output queues map as shown in Table 1:

Table 1 Bisync Addressing Output Queues

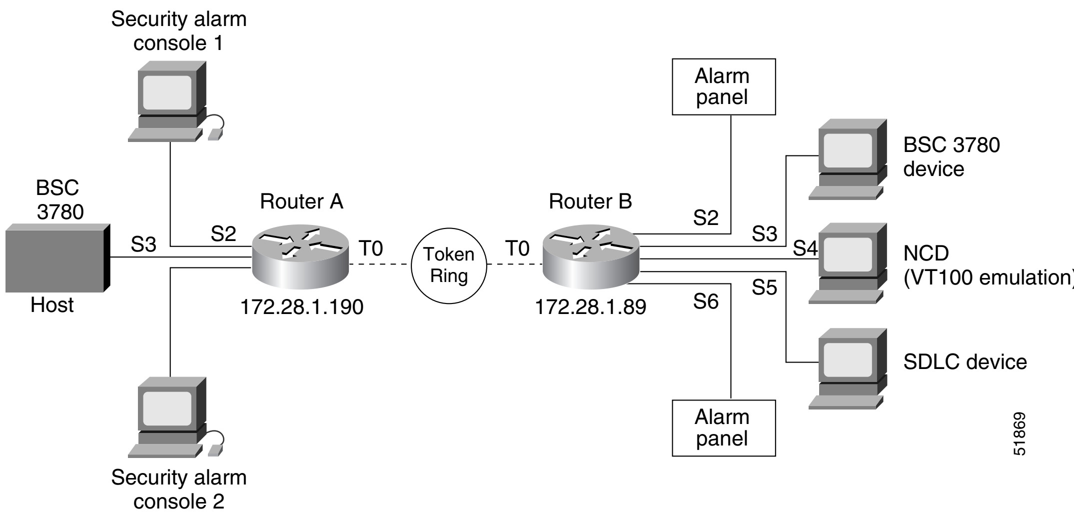
Output Queue	Session Mapped	BSTUN Port
1	Medium	1977
2	Normal	1978
3	Low	1979
4–10	High	1976

```
queue-list 1 protocol ip high tcp 1976
queue-list 1 protocol ip medium tcp 1977
queue-list 1 protocol ip normal tcp 1978
queue-list 1 protocol ip low tcp 1979
!
priority-list 1 protocol bstun normal address 1 C1
!
interface serial 0
 custom-queue-list 1
!
interface serial 1
 encapsulation bstun
 bstun group 1
 bsc char-set ebcdic
 bstun route address C1 tcp 200.190.30.1 priority
 custom-queue-list 1
```

Example: Asynchronous Configuration

In the following example, Router A and Router B are configured for both Adplex and Bisync across the same BSTUN as shown in Figure 15.

Figure 15 Combined Adplex and Bisync Configuration Example



Router A

```
version 11.0
!
hostname router-a
!
bstun peer-name 172.28.1.190
bstun protocol-group 1 bsc
bstun protocol-group 2 adplex
bstun protocol-group 3 adplex
!
interface serial 0
 no ip address
!
interface serial 1
 no ip address
!
interface serial 2
 physical-layer async
 description Connection to 1st Security Alarm Console.
 no ip address
 encapsulation bstun
 no keepalive
 bstun group 2
 bstun route address 2 tcp 172.28.1.189
 bstun route address 3 tcp 172.28.1.189
 adplex secondary
!


interface serial 3
 description Connection to BSC 3780 host.
 no ip address
 encapsulation bstun
```

```
 no keepalive
 clock rate 9600
 bstun group 1
 bstun route all tcp 172.28.1.189
 bsc char-set ebcdic
 bsc contention
!
interface serial 4
 physical-layer async
 description Connection to 2nd Security Alarm Console.
 no ip address
 encapsulation bstun
 no keepalive
 bstun group 3
 bstun route address 2 tcp 172.28.1.189
 bstun route address 3 tcp 172.28.1.189
 adplex secondary
!
interface serial 5
 no ip address
!
interface serial 6
 no ip address
!
interface serial 7
 no ip address
!
interface serial 8
 no ip address
!
interface serial 9
 no ip address
!
interface tokenring 0
 ip address 172.28.1.190 255.255.255.192
 ring-speed 16
!
interface BRI0
 ip address
 shutdown
!
ip host ss10 172.28.0.40
ip host s2000 172.31.0.2
ip route 0.0.0.0 0.0.0.0 172.28.1.129
!
snmp-server community public RO
!
line con 0
 exec-timeout 0 0
line 2
 no activation-character
 transport input-all
 parity even
 stopbits 1
 rxspeed 4800
 txspeed 4800
line 4
 transport input all
 parity even
 stopbits 1
 rxspeed 4800
 txspeed 4800
line aux 0
 transport input all
```

```
line vty 0 4
 password mango
 login
!
end
```

Router B

```
version 11.0
!
hostname router-b
!
bstun peer-name 172.28.1.189
bstun protocol-group 1 bsc
bstun protocol-group 2 adplex
bstun protocol-group 3 adplex
source-bridge ring-group 100
!
interface serial 0
 no ip address
!
interface serial 1
 no ip address
!
interface serial 2
 physical-layer async
 description Connection to Security Alarm Panel.
 no ip address
 encapsulation bstun
 no keepalive
 bstun group 2
 bstun route all tcp 172.28.1.190
 adplex primary
!
interface serial 3
 description Connection to BSC 3780 device.
 no ip address
 encapsulation bstun
 no keepalive
 clock rate 9600
 bstun group 1
 bstun route all tcp 172.28.1.190
 bsc char-set ebcdic
 bsc contention
!
interface serial 4
 physical-layer async
 description Connection to async port on NCD (VT100 terminal emulation).
 no ip address
!


interface serial 5
 no ip address
 encapsulation sdlc-primary
 no keepalive
 nrzi-encoding
 clock rate 9600
 sdllc traddr 4000.0000.4100 222 2 100
 sdlc address C1
 sdllc xid C1 05D40003
 sdllc partner 4000.0000.0307 C1
!
interface serial 6
```

```
 description Connection to alarm panel.
 physical-layer async
 no ip address
 encapsulation bstun
 no keepalive
 bstun group 3
 bstun route all tcp 172.28.1.190
 adplex primary
!interface serial 7
 no ip address
!
interface serial 8
 no ip address
!
interface serial 9
 no ip address
!
interface tokenring 0
 ip address 172.28.1.189 255.255.255.192
 ring-speed 16
 source-bridge 4 1 100
!
interface BRI0
 ip address
 shutdown
!
ip host ss10 172.28.0.40
ip host s2000 172.31.0.2
ip route 0.0.0.0 0.0.0.0 172.28.1.129
!
snmp-server community public RO
!
line con 0
 exec-timeout 0 0
line 2
 no activation-character
 transport input-all
 parity even
 stopbits 1
 rxspeed 4800
 txspeed 4800
line 4
 transport input all
 stopbits 1
line 6
 transport input all
 parity even
 stopbits 1
 rxspeed 4800
 txspeed 4800
line 7
 transport input all
line aux 0
 transport input all
line vty 0 4
 password mango
 login
!
end
```

Example: BSTUN-over-Frame Relay Configuration with Local Acknowledgment

The following example configures BSTUN over Frame Relay with local acknowledgment configured:

```
bstun protocol-group 1 bsc-local-ack

interface Serial1
 encapsulation frame-relay ietf
 clock rate 125000
 frame-relay map llc2 16

interface Serial4
 no ip address
encapsulation bstun
bstun group 1
bsc secondary
bstun route address C3 interface Serial1 dlci 16 C
bstun route address C2 interface Serial1 dlci 16 8
bstun route address C1 interface Serial1 dlci 16 4
```

Example: BSTUN-over-Frame Relay Configuration with Passthrough

The following example configures BSTUN over Frame Relay with Passthrough configured:

```
bstun protocol-group 1 bsc

interface Serial1
 encapsulation frame-relay
 clock rate 125000
 frame-relay map bstun 16
 frame-relay map llc 16

interface Serial4
 no ip address
encapsulation bstun
bstun group 1
bsc secondary
bstun route address C3 interface Serial1 dlci 16
bstun route address C2 interface Serial1 dlci 16
bstun route address C1 interface Serial1 dlci 16
```

Additional References

Related Documents

Related Topic	Document Title
Cisco IOS commands	*Cisco IOS Master Commands List, All Releases*
Bridging and IBM Networking commands	*Cisco IOS Bridging and IBM Networking Command Reference*
Routers that support BSTUN	*Cisco BSTUN/BSC/Bisync Hardware Support Levels*

Standards

Standard	Title
None	—

MIBs

MIB	MIBs Link
None	To locate and download MIBs for selected platforms, Cisco software releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

RFCs

RFC	Title
None	—

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

Feature Information for Configuring Serial Tunnel and Block Serial Tunnel

Table 2 lists the features in this module and provides links to specific configuration information.

Use Cisco Feature Navigator to find information about platform support and software image support. Cisco Feature Navigator enables you to determine which software images support a specific software release, feature set, or platform. To access Cisco Feature Navigator, go to http://www.cisco.com/go/cfn. An account on Cisco.com is not required.

Note Table 2 lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Table 2 Feature Information for Configuring Serial Tunnel and Block Serial Tunnel

Feature Name	Releases	Feature Information
BSTUN (Block Serial Tunneling)	11.2(1) 12.2SR 12.4T	The Bisync (BSC) feature encapsulates Bisync, Adplex, ADT Security Systems, Inc., Diebold, and asynchronous generic traffic for transfer over router links. The following sections provide information about this feature: • Block Serial Tunneling (BSTUN) Overview, page 2 • Enabling BSTUN, page 20
Bisync Enhancements	11.3(1) 12.2SR 12.4T	The Bisync Enhancements feature provides support for Frame Relay and provides extended support for addressing. In 12.3(11)T, support was added for the Cisco 3780. The following sections provide information about this feature: • Bisync Network Overview, page 3 • Enabling Frame Relay Encapsulation, page 21
STUN (Serial Tunnel)	12.2(8)T 12.2SR 12.4T	The STUN (Serial Tunnel) feature allows Synchronous Data Link Control (SDLC) protocol devices and High-Level Data Link Control (HDLC) devices to connect to one another through a multiprotocol internetwork The following sections provide information about this feature: • Serial Tunnel Overview, page 2 • Enabling STUN, page 9