

# Configuring Remote Source–Route Bridging

Document ID: 5204

---

## **Introduction**

### **Prerequisites**

- Requirements
- Components Used
- Conventions

### **RSRB Configuration**

- Virtual Ring
- Encapsulation
- Sample Configuration

### **Local Acknowledgment**

### **Passthrough**

### **Prioritizing Across RSRB**

### **RSRB Versions**

### **show Commands**

### **Troubleshooting**

- Dead Peers
- Closed Peers
- Slow Sessions
- CPU Utilization
- Duplicate Rings
- Buffer Tuning

### **Debugging**

### **Related Information**

---

## **Introduction**

Remote Source–Route Bridging (RSRB) enables Cisco routers to expand source–route bridged networks across metropolitan–area networks (MANs) and WANs. Its main function is to transport Logical Link Control (LLC) frames across the network. Consequently, it connects Token Ring networks over non–Token Ring network segments and Ethernet networks over non–Ethernet media. When you have Ethernet segments on one or both ends of the RSRB cloud, source–route translational bridging (SR/TLB) needs to be manually configured in the RSRB router or routers that are connected to the Ethernet network or networks. This document briefly describes configuring RSRB.

## **Prerequisites**

### **Requirements**

Readers of this document should have knowledge of these topics:

- Basics of Source Route Bridging (SRB)
- How to configure Cisco Routers

### **Components Used**

The information in this document is based on Cisco IOS® Software with the IBM feature set.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, make sure that you understand the potential impact of any command.

## Conventions

For more information on document conventions, refer to the Cisco Technical Tips Conventions.

## RSRB Configuration

This section shows how to configure the virtual ring group, encapsulation types, and peer.

### Virtual Ring

The first step in configuring RSRB is to configure the virtual ring of the router. This virtual ring ties in with the source route bridging configuration on the interfaces.

Issue the **source-bridge ring-group** *ring-group* [**virtual-mac-address**] command to define a virtual ring group in the router.

### Encapsulation

Next, you must specify how you intend to transport across the cloud. There are three encapsulation options from which you can choose: Direct, Fast Sequenced Transport (FST), and TCP.

#### Direct Encapsulation

Issue the **source-bridge remote-peer** *ring-group interface interface-name* [**mac-address**] [**If size**] command to use the virtual ring group defined in the previous step and to identify the interface over which to send SRB traffic to another router.

#### FST Encapsulation

To configure FST encapsulation, follow these steps:

1. Issue the **source-bridge fst-peername** *local-interface-address* command to set up an FST local peer name and provide an IP address to be used by this local peer.
2. Issue the **source-bridge remote-peer** *ring-group fst ip-address* [**If size**] command to identify your remote peers and specify the IP address of it.

#### TCP Encapsulation

The most common of all transports for RSRB is TCP encapsulation. It is the slowest of all transports and has the most overhead, but it is also the most reliable for transporting Systems Network Architecture (SNA) across multiprotocol backbones or packet-dropping networks such as Frame Relay.

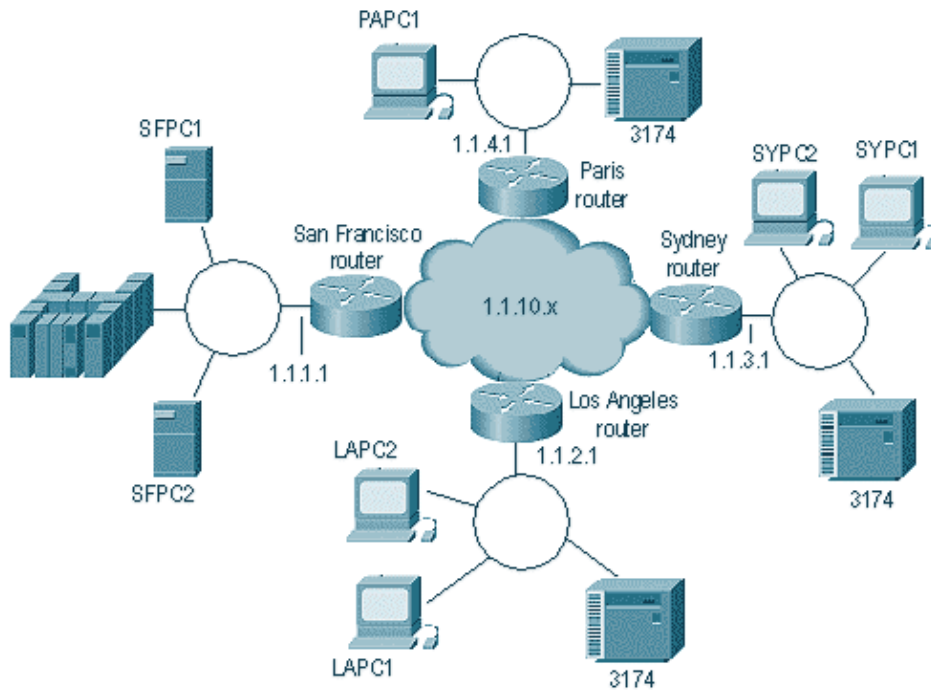
Issue the **source-bridge remote-peer** *ring-group tcp ip-address* [**If size**] [**tcp-receive-window wsize**] [**local-ack**] [**priority**] command to configure the local peer and the IP address to be used.

You must always configure a source-bridge remote-peer local to the system when doing TCP encapsulation. This should be an IP address in the system. If it is a new network and you can spare IP network addresses, create a loopback interface on the router with an IP address, and use this as the local peer IP address. (Refer to the loopback definition for more information on setting up a loopback interface.)

Finally, issue the **source-bridge remote-peer ring-group tcp ip-address [if size] [tcp-receive-window wsize] [local-ack] [priority]** command for the remote router that you are trying to reach, where ip-address is the IP address from that remote peer router.

## Sample Configuration

This section shows a sample configuration for using TCP encapsulation with this network setup:



### San Francisco

```
source-bridge ring-group 100
source-bridge remote-peer 100 tcp 1.1.4.1
source-bridge remote-peer 100 tcp 1.1.2.1
source-bridge remote-peer 100 tcp 1.1.3.1
source-bridge remote-peer 100 tcp 1.1.1.1
!
interface tokenring 0
ip address 1.1.1.1 255.255.255.0
source-bridge 1 1 100
source-bridge spanning
```

### Los Angeles

```
source-bridge ring-group 100
source-bridge remote-peer 100 tcp 1.1.1.1
source-bridge remote-peer 100 tcp 1.1.2.1
!
interface tokenring 0
ip address 1.1.2.1 255.255.255.0
source-bridge 2 1 100
source-bridge spanning
```

### Sydney

```
source-bridge ring-group 100
source-bridge remote-peer 100 tcp 1.1.3.1
source-bridge remote-peer 100 tcp 1.1.1.1
```

```
!  
interface tokenring 0  
ip address 1.1.3.1 255.255.255.0  
source-bridge 3 1 100  
source-bridge spanning
```

#### Paris

```
source-bridge ring-group 100  
source-bridge remote-peer 100 tcp 1.1.4.1  
source-bridge remote-peer 100 tcp 1.1.1.1  
!  
interface tokenring 0  
ip address 1.1.4.1 255.255.255.0  
source-bridge 4 1 100  
source-bridge spanning
```

This example configures a partial mesh RSRB setup. This means that all sites can reach San Francisco, but they can not talk amongst themselves. In order to have any-to-any connectivity, you need to issue the **source-bridge remote-peer** command for each router, pointing to the other. This type of connectivity is usually employed in environments with NetBIOS, but it tends to quickly overload the network unless the WAN setup is very solid.

Use the exact same configuration with FST peers:

#### San Francisco

```
source-bridge ring-group 100  
source-bridge fst-peername 1.1.1.1  
source-bridge remote-peer 100 fst 1.1.4.1  
source-bridge remote-peer 100 fst 1.1.2.1  
source-bridge remote-peer 100 fst 1.1.3.1  
!  
interface tokenring 0  
ip address 1.1.1.1 255.255.255.0  
source-bridge 1 1 100  
source-bridge spanning
```

#### Los Angeles

```
source-bridge ring-group 100  
source-bridge fst-peername 1.1.2.1  
source-bridge remote-peer 100 fst 1.1.1.1  
!  
interface tokenring 0  
ip address 1.1.2.1 255.255.255.0  
source-bridge 2 1 100  
source-bridge spanning
```

#### Sydney

```
source-bridge ring-group 100  
source-bridge fst-peername 1.1.3.1  
source-bridge remote-peer 100 fst 1.1.1.1  
!  
interface tokenring 0  
ip address 1.1.3.1 255.255.255.0  
source-bridge 3 1 100  
source-bridge spanning
```

#### Paris

```

source-bridge ring-group 100
source-bridge fst-peername 1.1.4.1
source-bridge remote-peer 100 fst 1.1.1.1
!
interface tokenring 0
ip address 1.1.4.1 255.255.255.0
source-bridge 4 1 100
source-bridge spanning

```

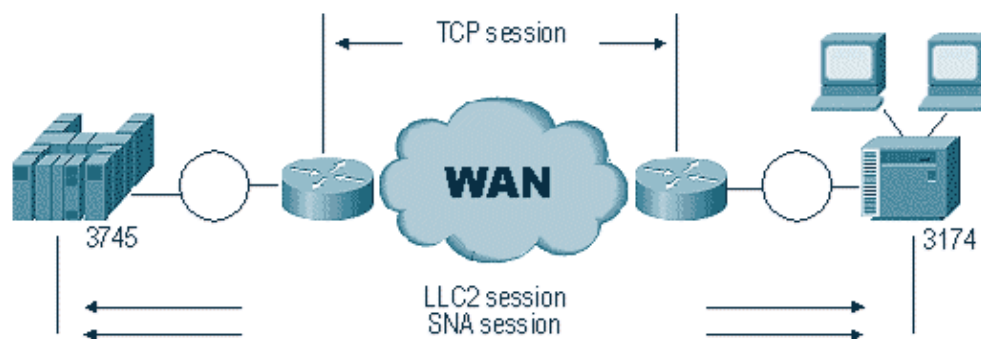
**Note:** When using FST, it is important to remember that the router can not fragment the LLC frame into pieces for transport. So, when the router is transporting 4 KB LLC frames across any media, make sure that the outbound interface can hold at least this frame size. If not, add the **source-bridge largest-frame** parameter to the configuration of the router. It tells the end stations that the largest frame across this path is the value that you define. Refer to Understanding and Troubleshooting Local Source-Route Bridging for more information.

## Local Acknowledgment

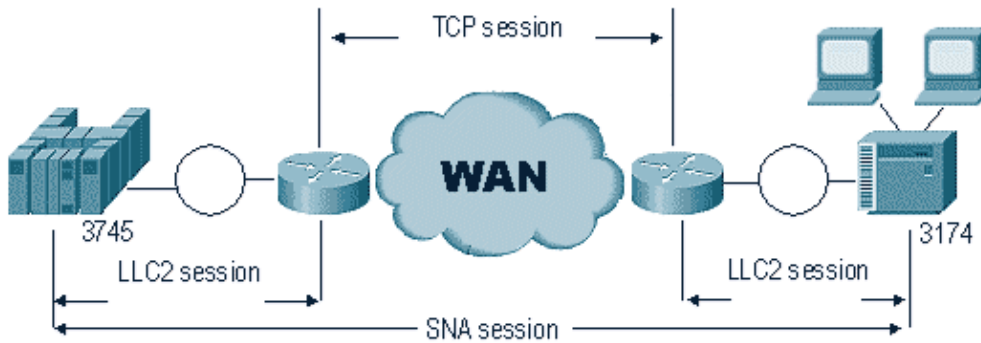
RSRB with local acknowledgment requires the router to maintain a Logical Link Control, type 2, (LLC2) session at each side of the cloud. This method is recommended when the bandwidth of the WAN between the RSRB routers is low. If the two RSRB routers are connected by a high speed WAN or LAN like FDDI, using local acknowledgment is not recommended: the media between the two routers is perfectly capable of handling the traffic, and adding local acknowledgment would just cause an unnecessary overhead in the routers maintaining the local LLC connections.

For slow WANs, RSRB with local acknowledgment is indispensable because session timeouts can occur. In countries that commonly have slow lines (such as 19.2 Kbps or 9.6 Kbps), the sessions can not maintain themselves without local acknowledgement.

In this diagram, all the LLC2 and SNA frames are going across the cloud, which is called passthrough mode :



In this scenario, if the stations are timing out, you can change the LLC T1 timers on the stations so that they wait longer for responses from the host. In LLC2, an acknowledgment is expected in return for every packet that is sent. Each side keeps a record of who sent what. Usually, in LLC2, each station that sends packets transmits a series of packets and expects a response from the receiving station indicating that it received the packets. This response is expected within a certain amount of time, which is the T1 timer. This parameter can be changed in the stations and the hosts, but in large networks, it is easier to add local acknowledgement to the routers than to change the T1 timer.



The routers locally acknowledge that the packet was received on behalf of the station on the other side of the WAN. This makes the host think that the end station received the information and can send more information. These acknowledgment frames do not cross the WAN, which also saves bandwidth. In summary, all frames are locally acknowledged. Data frames are, in turn, sent to the remote peer; but supervisory frames like receiver ready (RR), receiver not ready (RNR), and reject (REJ) never cross the WAN.

**Note:** A drawback of local acknowledgement is that each side needs to maintain fine states. This can cause problems because, if one side of the cloud is in one state and the other in another state, then the session dies.

The router can determine if it is becoming saturated with frames and tell both sides to stop sending data when this happens. Consider this **show source-bridge** command output:

```
This TCP peer: 8.8.2.1
Maximum output TCP queue length, per peer: 100
Peers:
TCP 8.8.2.2      state   bg lv  pkts_rx  pkts_tx  expl_gn  drops  TCP
TCP 8.8.2.1      -      3   0        0        0        0      0  0
```

The TCP column lists the TCP queues. If these queues reach 95 percent capacity, the routers begin to send RNR on both ends to stop the stations from transmitting data. You can change the size of the TCP queues with the **source-bridge tcp-queue-max number** command. This command limits the number of the backup queue for RSRB to control the number of packets that can wait for transmission to a remote ring before they start being thrown away.

**Note:** Be very careful with this command. If the queues are set too large, TCP takes a long time to service all packets across the WAN, making the remote station wait too long for the next information frame. This causes a timeout. It is recommended to not exceed 200 in the TCP queues on the router. Without **local-acknowledgement**, the TCP queues fill until the router begins dropping packets because it can not tell the end stations to stop transmitting. Also, make sure that the TCP queues are not always at the maximum or above it, which may indicate wedged buffers (a problem with RSRB).

The **show local-ack** and **show rif** commands provide valuable information when diagnosing local acknowledgement problems.

## Passthrough

The **source-bridge passthrough** command is used to tell RSRB which remote ring for which the router is not locally acknowledging frames. The reason for this is that only one peer for a specific router is specified, but this router could contain more than one interface. If you want to set up a remote router with local acknowledgement to only one ring of a remote router that has for example four rings, then use the **source-bridge passthrough ring-group** command. This command configures the router so that it does not locally acknowledge frames destined for a specific ring, as illustrated in this diagram:



In this example, the goal is to enable local acknowledgement for remote ring 2, but SR/TLB is configured for the Ethernet. Because you can not use local acknowledgement for SR/TLB, you would have to turn off local acknowledgment for the entire system. By using the **source-bridge passthrough** command, you can avoid turning off local acknowledgement for the entire system. You need to specify the pseudo-ring defined for the Ethernet in the **source-bridge passthrough** command, so that the frames destined to this ring are not locally acknowledged.

**Note:** Make sure that you specify **source-bridge passthrough pseudo-ring-number** when configuring SR/TLB.

## Prioritizing Across RSRB

When setting up TCP peers, you can prioritize traffic according to various criteria. The most simple way is to prioritize RSRB traffic over the other routed protocols in the network. To set up prioritization to differentiate among RSRB traffic itself, specify the **priority** keyword at the end of the RSRB peer. Doing so creates four different peers to the remote router. Each of these four can then be prioritized independently. You can even prioritize based on the LU address.

## RSRB Versions

This table describes RSRB version changes for each Cisco IOS Software Release:

| Cisco IOS Software Release | RSRB Version                                    |
|----------------------------|---|
| 8.2(n) – 8.3(2)            | RSRB version 1 only                             |
| 8.3(3) – 9.1(7.x)          | RSRB version 1 or RSRB version 2 (2 is default) |
| 9.1(8) – current           | RSRB version 2 or RSRB version 3 (3 is default) |



**Caution:** It is risky to mix RSRB versions, but Cisco routers can negotiate the correct version when dealing with older software. For example, if you have an RSRB router with Cisco IOS Software Release 10.3 running RSRB version 3, that router runs RSRB version 2 with a router running code prior to Cisco IOS Software Release 9.1(8). The **show source-bridge** command displays the version of the peers. If you have trouble determining if the peers are negotiating correctly, try issuing the **debug source event** command.

## show Commands

The **show source-bridge** command is important to troubleshoot RSRB. This is sample output from that command:

```
G71# show source-bridge
```

```
Local Interfaces:
      srn bn  trn r p s n  max hops  receive      transmit
      cnt:bytes  cnt:bytes  drops
To0/0 500  1 1000 * * * *  7 7 7  19309:1383649  246:37257  597

Global RSRB Parameters:
  TCP Queue Length maximum: 100

Ring Group 1000:
  This TCP peer: 150.150.5.2
  Maximum output TCP queue length, per peer: 100
Peers:
  state      lv  pkts_rx  pkts_tx  expl_gn  drops  TCPq
TCP 150.150.4.1  closed *3      0        0      8970   4790  0
TCP 150.150.5.1  open  *3     313     20395   8970    1  0
TCP 150.150.5.2  -     3      0        0      22     1  0
Rings:
bn: 1  rn: 500 local ma: 4000.3070.a209 TokenRing2/0      fwd: 313
bn: 1  rn: 100 remote ma: 4000.3040.02e1 TCP 150.150.5.1  fwd: 5100
bn: 1  rn: 1   remote ma: 4000.3040.0211 TCP 150.150.5.1  fwd: 0
Explorers: ----- input -----
      spanning  all-rings  total  spanning  all-rings  total
To0/0  4192639    5726    4198365  0         0         0
Ch1/2  0           0         0        4192639   5726     4198365

Local: process switched 4198365  flushed 0          max Bps 256000
rings      inputs          bursts          throttles      output drops
To0/0      4198514              0              0              0
Ch1/2      0                  0              0              0
```

With RSRB, there is an additional section in the middle of the output that provides this information about the remote peers:

- The TCP peer IP address, including the TCP queue length per peer.
- `state` There should always be a peer in the `-` state (the local peer for the router).
- `l` Contains an asterisk (\*) if local acknowledgment is configured for this peer.
- `v` Contains the RSRB version of the peer.
- `pkts_rx` The number of packets that the router received from this peer.
- `pkts_tx` The number of packets that the router transmitted to this peer.
- `expl_gn` The number of explorers that the router has copied to this peer.
- `drops` The number of packets dropped for this peer.
- `Rings` The remote rings that the remote peers have announced as being locally attached to them. RSRB uses this value to know where to send a packet that already contains a Routing Information Field (RIF).

**Note:** There should *never* be duplicate rings.

When using local acknowledgement, the `show llc2` command is useful for learning LLC2 information about two stations. It can also tell you if the session is up or not. This is sample output from that command:

```
ibu-7206# show llc2

LLC2 Connections: total of 1 connections
TokenRing3/0 DTE: 4001.68ff.0000 4000.0000.0001 04 04 state NORMAL
  V(S)=5, V(R)=5, Last N(R)=5, Local window=8, Remote Window=127
  akmax=3, n2=8, Next timer in 8076
  xid-retry timer      0/60000  ack timer      0/1000
  p timer              0/1000   idle timer     8076/10000
  rej timer            0/3200   busy timer     0/9600
  akdelay timer        0/100    txQ count      0/200
```

That output shows this information:

- `state` Can indicate these possible conditions:
  - ◆ `ADM A` connection is not established.
  - ◆ `SETUP A` request to begin the session has been sent to the remote router, and this station is waiting for the response.
  - ◆ `RESET A` problem has occurred between this station and the station local to it. This station is waiting for a response to the reset.
  - ◆ `D_CONN` The station local to this station has issued a normal disconnect and is waiting for a response.
  - ◆ `ERROR` The station local to this station has detected an error in communication and has sent a packet to the other station.
  - ◆ `NORMAL` Everything is operational between the two stations.
  - ◆ `BUSY` Currently, the station local to this station can not accept any more I-frames because it is congested.
  - ◆ `REJECT` This frame is out of sequence. The other station must retransmit.
  - ◆ `AWAIT` The station local to this station popped a timer and is trying to recover from this failure.
- `V(S)` The sequence number for the next I-frame that the station local to this station will send.
- `V(R)` The sequence number for the next I-frame that should be received from the remote station.
- `Last N(R)` The last sequence number of the station local to this router that was transmitted and acknowledged by the remote router.
- `Local window` The number of frames that the station local to this station can send before an acknowledgment is needed from the remote peer.
- `Remote Window` The number of frames that the station local to this station can accept from the remote.
- `akmax` The number of frames that this station can receive from the station local to it before sending an acknowledgment for them. This parameter can be changed in the **llc2 ack-max** configuration option.
- `n2` The number of times that the router resends a frame to which the station has not replied back. This parameter can be changed with the **llc2 n2** configuration option.
- `Next timer` The time, in milliseconds, before any of the timers pop up.

For each of these `timer` parameters, the first value is the next time that the timer will pop (the amount of time that will pass) and the second value (after the slash [/]) is when the timer will pop itself:

- `xid-retry timer` The amount of time that this station waits for a reply to an exchange identification (XID) before dropping the session. This parameter can be changed with the **llc2 xid-retry-time** configuration option.
- `p timer` The amount of time to wait for a final response to a poll frame before resending the poll frame. This parameter can be changed with the **llc2 tpf-time** configuration option.
- `rej timer` The amount of time that this station waits for another station to retransmit a frame that this station has requested. This parameter can be changed with the **llc2 trej-timer** configuration option.
- `akdelay timer` After receiving a number of frames and not reaching the window size, the station waits a specified period of time, then acknowledges the frames received even though it did not reach the `akmax` value. This parameter can be changed with the **llc2 ack-delay-time** configuration option.
- `ack timer` The T1 timer itself, which indicates the amount of time that the router waits before sending unacknowledged information frames. This parameter can be changed with the **llc2 t1-time** configuration option.
- `idle timer` When there is no activity between two stations, they exchange RRs. The **idle-timer** parameter controls the intervals between these RR polls. This parameter can be changed with the **llc2 idle-time** configuration option.

- `busy timer` Amount of time the router waits while the other LLC2 station is in a busy state before attempting to poll the remote station. This parameter can be changed with the `llc2 tbusy-time` configuration option.
- `txQ count` Queue for holding LLC2 information frames. This parameter can be changed with the `llc2 txq-max` configuration option.

## Troubleshooting

Troubleshooting RSRB focuses on two major components: RSRB itself and the transport (which, in most cases, is IP itself or the WAN protocol itself). This section covers only the basics of these components.

In RSRB, the peers talk between themselves regardless of what transport you use. This brings up the issue of peer states, which is important because both sides must be in the same peering state in order to avoid problems. RSRB has a predefined number of possible states, with these code functions from debugging (in parentheses) and meanings:

- `dead` (RSRB\_PS\_DEAD) Temporarily shut down.
- `closed` (RSRB\_PS\_CLOSED) No activity.
- `opening` (RSRB\_PS\_OPENING) Waiting for connector.
- `openwait` (RSRB\_PS\_OPENWAIT) Waiting for the transport to open.
- `remwait` (RSRB\_PS\_REMWAIT) Transport open, waiting for response.
- `remgo` (RSRB\_PS\_REMGO) Remote peer has agreed to continue to open this peer connection.
- `remopen` (RSRB\_PS\_REMOPENED) Remote opened before this station did.
- `draining` (RSRB\_PS\_DRAINING) Sending start queued packs.
- `open` (RSRB\_PS\_CONNECTED) Fully open.

This is what occurs when the peers open correctly:

```
closed    -> opening
opening   -> openwait
openwait  -> remwait
remwait   -> remgo
remgo     -> draining
draining  -> open
```

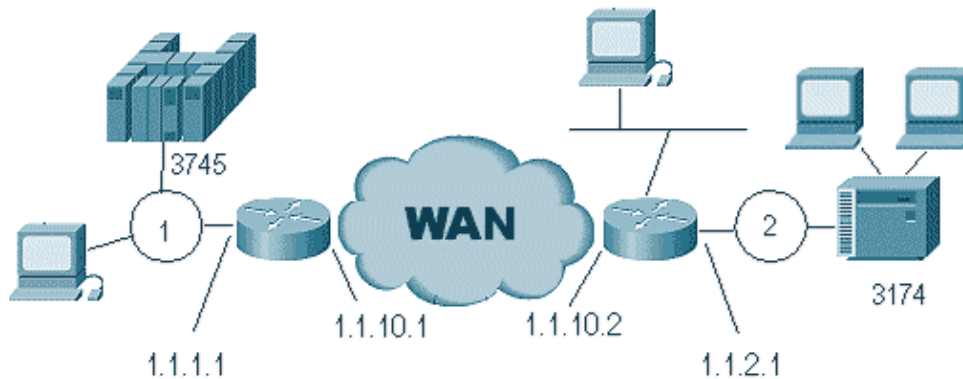
The peer negotiates with its adjacency until it opens up the peer. If you issue a `debug source event` command while the peer is becoming operational, it shows output similar to this:

```
RSRB: peer 100/135.53.1.1 closed [last state 8]
RSRB: peer 100/135.53.1.1/1996 open -> closed (1A0E4A, 14818C)
RSRB: peer 100/135.53.1.1/1996 closed -> opening (19E94E, 1A28D6)
RSRB: peer 100/135.53.1.1/1996 opening -> openwait (19B0E4, 19B396)
RSRB: CONN: opening peer 100/135.53.1.1, 2
RSRB: peer 100/135.53.1.1/1996 openwait -> remwait (19EED2, 19B100)
RSRB: AHDR New version. l=2, r=3, Version Rep from 100/135.53.1.1/1996
RSRB: hdr 3 7 0 64 0 0 87350101
RSRB: peer 100/135.53.1.1/1996 remwait -> remgo (19FFE4, 14825E)
RSRB: peer 100/135.53.1.1/1996 remgo -> draining (19B206, 19B396)
RSRB: peer 100/135.53.1.1/1996 draining -> open (19B28A, 19B396)
RSRB: CONN: connector terminating
RSRB: HDR New version. l=2, r=3, Version Req from 100/135.53.1.1/1996
RSRB: hdr 3 6 0 64 0 0 100008C0
```

That output indicates that there are two different routers: one running RSRB version 3, and one running RSRB version 2. That output shows us that the negotiation continues until the peer is open.

## Dead Peers

When the peer is dead, it usually means that it has lost its connection with its peer.



That diagram shows a normal RSRB and IP setup. Suppose that the two routers are configured for RSRB TCP encapsulation, but you can not get from ring 1 to ring 2. Use these steps to troubleshoot:

1. Check all interfaces in the router to confirm that they are operational and in an up/up state.
2. Check the **show source-bridge** command output to verify that the peers are either open or closed and that the peers are in the same state at each side of the cloud.
3. If the peers are dead, verify that the IP connectivity is operational, using these steps:
  - a. Issue the **show ip route** command for the network of the remote peer.
  - b. Look at the configuration of the router and determine the peer IP addresses.
  - c. If the router shows the correct information for IP routing to reach that location, issue an extended **ping** command from peer to peer.

Ping from your peer IP address to your destination peer IP address. This ensures that the routers can communicate.
  - d. By this time, if you can ping and see IP correctly, but the peer is dead on both sides, issue the **debug source event** command to determine why exactly the peers are failing.
  - e. If pings are failing, determine why this is occurring (with IP).
4. If the peers are open, test the transport across the WAN.

Issue an extended **ping** command, with multiple payloads of 0x0000 0xA0A0 0x0101, to test the clocking of the cloud. Also, issue **ping** commands with extended sizes up to the maximum transmission unit (MTU) of the interface.

5. If you are still experiencing the problem, determine what is occurring in the routers.

Issue the **debug list 1100** command for Token Ring packets to see if the packets are hitting the router. Refer to Understanding and Troubleshooting Local Source-Route Bridging for more information.

6. If no packets show up, you need a Sniffer on the LAN.

## Closed Peers

Peers that are closed usually indicate that no traffic is going across the cloud. Issue a **show source-bridge** command to verify the incoming and outgoing traffic on the interfaces of the router. It is best to issue the **clear source-bridge** command first, then issue the **show source-bridge** command to get the latest statistics. At the end of the **show source-bridge** command output, check how many explorers are getting into the router. Any explorer should be open to the peers because the router should transport the explorer to all peers. To confirm, look at the end of the **show source-bridge** command output for this section:

```

Explorers: ----- input -----
              spanning all-rings      total      spanning all-rings      total
To0/0          99      5726      5825          76      134      210
To0/1          50       92      142          125     5912     6037

Local: fastswitched 6589      flushed 0      max Bps 256000
rings      inputs      bursts      throttles      output drops
To0/0      5825          0          0          0          0
To0/1      142          0          0          0          0

```

That output shows a properly configured router that has a number of explorers and open peers. If those input and output values are all zeros (0), then the router is not receiving any explorers or is not receiving any SRB traffic for that specific peer.

The **show source-bridge** output also contains this important RSRB information:

```

Ring Group 1000:
This TCP peer: 150.150.5.2
Maximum output TCP queue length, per peer: 100
Peers:
TCP 150.150.4.1      state      lv  pkts_rx  pkts_tx  expl_gn  drops TCPq
TCP 150.150.5.1      open      *3   0      8970     8970     0      0
TCP 150.150.5.1      open      *3  15000   20395    8970     1      0
TCP 150.150.5.2      -         3    0       0        22      1      0

```

In that output, when the router transmits an explorer, the `expl_gn` counter increments for each explorer. All peers usually share the same number of explorers, but this fact might change when issuing the **source-bridge proxy-explorers** command. More importantly, you can see that `pkts_rx` has remained 0: the router has sent 8970 explorers to this peer, but has never received a response.

In this case, you should probably check the remote router for correctly configured RSRB and SRB. Another issue could be that no station is capable of responding to the explorers because none of them are configured for SRB themselves.

## Slow Sessions

Slow response across the RSRB cloud could indicate retransmissions across the WAN. Because TCP ensures that the packet is reaching its destination, the router retransmits the packet using TCP to verify that it gets there.

First, determine the number of times that TCP retransmits. The **show tcp** command shows the number of retransmissions that are occurring across the cloud for the specific peer. If you see retransmissions, issue extended **ping** commands from peer to peer to determine exactly why. Use different payloads and sizes, and verify the time it takes to get across the cloud. This is sample output from a **show tcp** command:

```

Stand-alone TCP connection from host 10.17.5.1
Connection state is ESTAB, I/O status: 1, unread input bytes: 0
Local host: 10.17.5.2, Local port: 1994
Foreign host: 10.17.5.1, Foreign port: 11035

```

```

Enqueued packets for retransmit: 0, input: 0, saved: 0

```

```

Event Timers (current time is 0x1B2E50):
Timer      Starts      Wakeups      Next
Retrans      229         0            0x0
TimeWait      0           0            0x0
AckHold      229         0            0x0
SendWnd      0           0            0x0
KeepAlive    0           0            0x0
GiveUp       0           0            0x0
PmtuAger     0           0            0x0

```

```
iss: 2847665974  snduna: 2847667954  sndnxt: 2847667954      sndwnd: 9728
irs: 3999497423  rcvnxt: 3999499452  rcvwnd: 9672  delrcvwnd: 568
```

```
SRTT: 300 ms, RTTO: 607 ms, RTV: 3 ms, KRTT: 0 ms
minRTT: 0 ms, maxRTT: 300 ms, ACK hold: 300 ms
Flags: passive open, higher precedence
```

```
Datagrams (max data segment is 1460 bytes):
Rcvd: 459 (out of order: 0), with data: 229, total data bytes: 2028
Sent: 457 (retransmit: 0), with data: 228, total data bytes: 1979
```

If everything seems normal, it could be that you are sending too small an amount of traffic per packet. For a host-to-physical-unit (host-to-PU) connection, try setting the **maxdata** parameter to 4000. For TCP encapsulation, the router has to break the LLC frame into pieces smaller than 1500 bytes because TCP is coded to a maximum of 1500-byte frames. Changing this value should improve session performance, but introduces more overhead for the router.

Transporting too much traffic or unnecessary traffic across the WAN with RSRB is another possible cause of slow sessions. Putting a SNA filter on the peer to cause the router to return to normal operation often indicates that the router is bridging unnecessary traffic. Doing so requires that the person in charge of the network has a thorough knowledge of what traffic should be going through the routers.

## CPU Utilization

RSRB is a process-level operation, not interrupt-level. This means that RSRB can consume a large portion of the processor depending on the number of peers or the amount of information hitting the system. First, you must determine what is utilizing the CPU in the router. For example, if you are running custom queueing on the serial interfaces of the router or have enabled **route-cache** for any given routed protocol, then the CPU utilization increases for the specific protocol. In this case, issue the **show processes cpu** command to view statistics. If the CPU is at a very high utilization (more than 75 percent) then you may experience slow response time or even session drops.

Each router can handle a limited number of peers before the CPU is overwhelmed, depending on the device and the amount of traffic being transported to each peer. For example, the Cisco 2500 series can not support a large number of peers. In contrast, a Cisco 4700 can support a huge number of peers because of its powerful reduced instruction set computing (RISC) processor. More importantly, determining how many peers can run in a system must be measured on a network basis. A Cisco 4700 can run at 90 percent CPU utilization with just one peer, or at 12 percent CPU utilization with 50 peers. It all depends on the amount and type of traffic in the network.

The **psage** tool is helpful when determining CPU utilization. It is a UNIX tool that reports the most consuming processes in the router.

If you determine that RSRB is the cause of the high CPU utilization, verify why the router is sending so much information across the cloud. There is no **show** command that tells you exactly what type of packet is causing the problem.

## Duplicate Rings

In a router running RSRB, duplicate rings can cause major problems because RSRB learns its partners locally attached rings. When the RSRB router receives an information frame, it checks for the virtual ring in the RIF to determine the next hop. The router then looks in its tables and determines the peer to which this packet is headed, then transmits the packet. If there are duplicate rings, however, then the router sends the packet to the first peer in its table and the other peer does not receive it; if the packet was truly destined to the

other peer, it does not reach it. To determine if you have duplicate rings, look at the **show source-bridge** command output, which appears similar to this:

```
Rings:
bn: 1  rn: 500  local  ma: 4000.3070.a209  TokenRing2/0          fwd: 313
bn: 1  rn: 100  remote ma: 4000.3040.02e1  TCP 150.150.5.1        fwd: 5100
bn: 1  rn: 300  remote ma: 4000.3040.3456  TCP 150.150.6.1        fwd: 7856
bn: 1  rn: 100  remote ma: 4000.3040.1524  TCP 150.150.7.1        fwd: 0
bn: 1  rn: 1    remote ma: 4000.3040.fe43   TCP 150.150.3.1        fwd: 89045
```

You can see ring 100 in two different remote peers. Change the remote ring number at one of those routers to a different value.

## Buffer Tuning

Because RSRB is process-level, it uses system buffers. If you are experiencing too many drops in the router, set up a good buffer scheme.

Refer to Understanding Buffer Misses and Failures for more information.

## Debugging

When using RSRB with local acknowledgment, debugging can provide you with information about network activity. Debugging RSRB without local acknowledgement provides information only about the state of the peers and the frames being passed. Because local acknowledgment spoofs the LLC session between the routers, you need to know what packets are passing by to find out the state of the connection. For example, when the router receives a disconnect from the station, it is time to tear down the session.

**Note:** Before issuing **debug** commands, refer to Important Information on Debug Commands.

This is the output from **debug source event** when an RSRB local-peer router (with IP address 10.48.64.30) is opening the peer connection to a remote RSRB peer (with IP address 10.48.64.14):

```
logos# show debugging
```

```
General bridging:
  Source Bridge Event debugging is on
```

```
logos#
```

```
Mar 21 10:57:56.892: RSRB: remote explorer to 250/10.48.64.14/1996 srn 20 [8830.0141.0FA0.
Mar 21 10:57:56.892: RSRB: peer 250/10.48.64.14/1996 closed -> opening (6101B48C)
Mar 21 10:57:56.892: RSRB: remote explorer to 250/10.48.64.14/1996 srn 20 [8830.0141.0FA0.
Mar 21 10:57:56.892: RSRB: TCPD: wrong state to keep, 250/10.48.64.14/1996 opening EXPLORE
Mar 21 10:57:56.896: RSRB: peer 250/10.48.64.14/1996 opening -> openwait (610173FC)
Mar 21 10:57:56.896: RSRB: CONN: opening peer 250/10.48.64.14, 2
Mar 21 10:57:58.900: RSRB: peer 250/10.48.64.14/1996 openwait -> remwait (6101BBAC)
Mar 21 10:57:58.900: RSRB: VERSION/RING_XREQ sent -> 250/10.48.64.14/1996
Mar 21 10:57:58.904: RSRB: AHDR: connecting, VERSION_REP from 250/10.48.64.14/1996 (ver 3)
Mar 21 10:57:58.904: RSRB: AHDR: CAPS_REQ sent -> 250/10.48.64.14/1996
Mar 21 10:57:58.904: RSRB: peer 250/10.48.64.14/1996 remwait -> remgo (6101CECC)
Mar 21 10:57:58.904: RSRB: DATA: 250/10.48.64.14/1996 RING_XCHG_REP, trn 0, vrn 250, off 0
Mar 21 10:57:58.908: RSRB: peer 250/10.48.64.14/1996 remgo -> draining (61017548)
Mar 21 10:57:58.908: RSRB: added bridge 1, ring 10 for 250/10.48.64.14/1996
Mar 21 10:57:58.908: RSRB: peer 250/10.48.64.14/1996 draining -> open (610175EC)
Mar 21 10:57:58.908: RSRB: CONN: connector terminating
Mar 21 10:57:58.920: RSRB: DATA: 250/10.48.64.14/1996 FORWARD, trn 20, vrn 250, off 25175,
logos#
```

Watch every change in the way that the peer is brought up. For example, if the state went from `openwait` to `remwait` and then did not get any response from the remote router, then the state would then become `dead`.

The `show local-ack` command is another helpful command to use when local acknowledgment is enabled. Its output appears similar to this:

```
logos# show local-ack

local addr      remote addr    lrg  rrg  lsap  dsap  state
4000.4000.4000 0000.302c.9e96 014  00A  04   04   CONNECT
logos#
```

That output shows us that the session is in the `CONNECT` state, which means that there is a valid LLC2 session across RSRB, as far as the router is concerned. Try restarting the session from scratch and issue a `debug local-ack state` command. This is output from the `debug local-ack state` command on the local peer when the LLC2 connection is establishing:

```
logos# debug local-ack state

Local Acknowledgement states debugging is on

logos#
Mar 21 11:02:37.957: LOCACK: (remote) created entity for 4000.4000.4000 0000.302c.9e96 04 0
Mar 21 11:02:37.957: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: CR
Mar 21 11:02:37.957: LOCACK: l2_action 0 0 1
Mar 21 11:02:37.961: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: DLC_IND_OPEN
Mar 21 11:02:37.961: LOCACK: l2_action 21 2 0
Mar 21 11:02:37.965: LOCACK: CONN_POLL_REQ to peer 250/10.48.64.14/1996/4000.4000.4000 000
Mar 21 11:02:37.969: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: CPAR
Mar 21 11:02:37.969: LOCACK: l2_action 22 1 4
Mar 21 11:02:37.969: LOCACK: CONN_POLL_ACK to peer 250/10.48.64.14/1996/4000.4000.4000 000
logos#
```

This is output from the `debug local-ack state` command on the remote peer when the LLC2 connection is establishing:

```
dymock# debug local-ack state

Local Acknowledgement states debugging is on

dymock#
Mar 21 11:02:37.954: LOCACK: (local) created entity for 0000.302c.9e96 4000.4000.4000 04 0
Mar 21 11:02:37.954: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: DLC_IND_EXT CONN
Mar 21 11:02:37.954: LOCACK: l2_action 20 2 0
Mar 21 11:02:37.954: LOCACK: CONN_REQ to peer 250/10.48.64.30/1996/0000.302c.9e96 4000.400
Mar 21 11:02:37.966: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: CPR
Mar 21 11:02:37.966: LOCACK: l2_action 23 2 0
Mar 21 11:02:37.966: LOCACK: CONN_POLL_ACK_REQ to peer 250/10.48.64.30/1996/0000.302c.9e96
Mar 21 11:02:37.970: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: CPA
Mar 21 11:02:37.970: LOCACK: l2_action 0 1 4
dymock#
```

This is the display of the LLC2 connection in the local peer:

```
logos# show local-ack

local addr      remote addr    lrg  rrg  lsap  dsap  state
4000.4000.4000 0000.302c.9e96 014  00A  04   04   CONNECT
logos#
```

This is the display of the LLC2 connection in the remote peer:

```
dymock# show local-ack
```

```
local addr      remote addr      lrg  rrg  lsap  dsap  state
0000.302c.9e96  4000.4000.4000  00A  014  04    04    CONNECT
dymock#
```

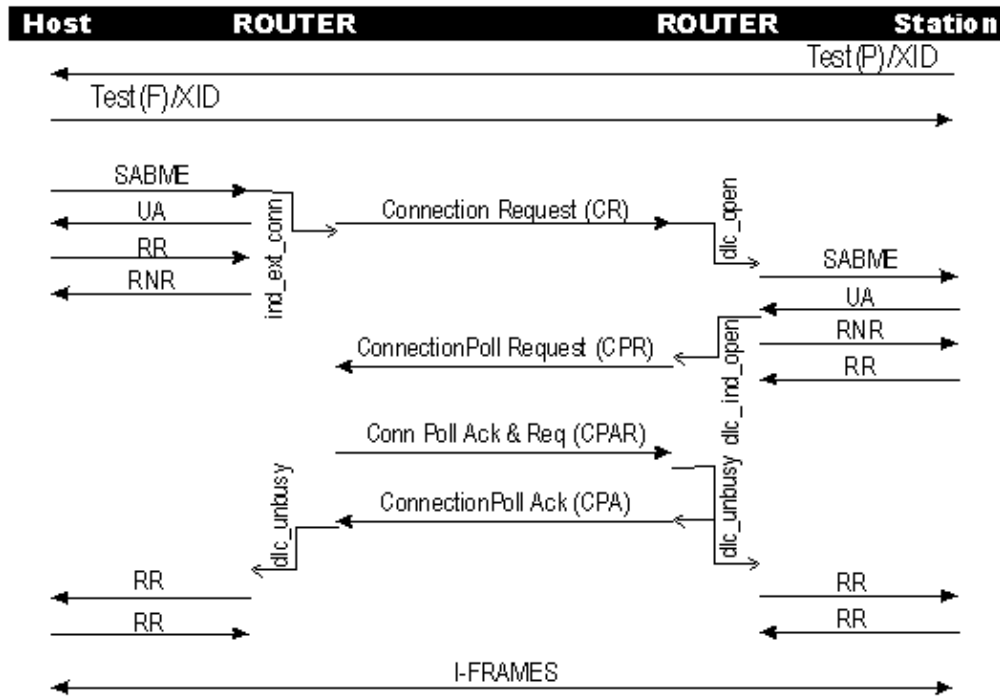
This is output from the **debug local-ack state** command on the local peer when the LLC2 connection is disconnecting:

```
logos#
Mar 21 11:03:16.979: LOCACK: llc cleanup for 4000.4000.4000 0000.302c.9e96 04 04, CONNECT
Mar 21 11:03:16.979: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: DLC_IND_FAIL
Mar 21 11:03:16.979: LOCACK: l2_action 25 2 0
Mar 21 11:03:16.979: LOCACK: DISC_REQ to peer 250/10.48.64.14/1996/4000.4000.4000 0000.302c.9e96 04 04
Mar 21 11:03:16.983: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: DA
Mar 21 11:03:16.983: LOCACK: l2_action 0 2 2
Mar 21 11:03:16.983: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: DLC_IND_FAIL
Mar 21 11:03:16.983: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 addr no longer in use, remote peer disconnected
logos#
```

This is output from the **debug local-ack state** command on the remote peer when the LLC2 connection is disconnecting:

```
dymock#
Mar 21 11:03:16.980: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: DR
Mar 21 11:03:16.980: LOCACK: l2_action 26 2 2
Mar 21 11:03:16.980: LOCACK: DISC_ACK to peer 250/10.48.64.30/1996/0000.302c.9e96 4000.4000.4000 04 04
Mar 21 11:03:16.984: LOCACK: llc cleanup for 0000.302c.9e96 4000.4000.4000 04 04, LOCAL_DISCONNECT
Mar 21 11:03:16.984: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: DLC_IND_CLOSED
Mar 21 11:03:16.988: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 addr no longer in use, remote peer disconnected
dymock#
```

This is the flow and change of states of local acknowledgment:



1. The router creates the finite state machine (FSM) entity for this session.
2. It receives a connection request (CR) from the remote router.
3. The router sends the Set Asynchronous Balanced Mode Extended (SABME) and waits for the unnumbered acknowledgement (UA) from the station.
4. It receives the UA and moves to `dlc_ind_open`.
5. The router sends a connection poll request (CPR) to the remote peer.
6. The router receives a connection poll acknowledgement and request (CPAR).
7. Finally, the router sends a connection poll acknowledgement (CPA) to the peer.

The session disconnection follows these similar steps:

1. The router receives a disconnect request (DR) from the remote peer.
2. The router responds with a disconnect acknowledgement (DA).
3. Finally, the router clean ups the session locally.

For more information, issue the **debug local-ack packets** command. It provides more details about activity in the router.

This is the output from the local RSRB peer:

```
logos# debug local-ack packets
```

```
Local Acknowledgement packets debugging is on
```

```
logos#
```

```
Mar 21 11:33:51.503: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: CR
```

```
Mar 21 11:33:51.507: LOCACK: recv UA P/F, 4000.4000.4000 0000.302c.9e96 04 04, state LOCAL
```

```
Mar 21 11:33:51.507: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: DLC_IND_OPEN
```

```
Mar 21 11:33:51.507: LOCACK: CONN_POLL_REQ to peer 250/10.48.64.14/1996/4000.4000.4000 000
```

```

Mar 21 11:33:51.507: LOCACK: recv RR, 4000.4000.4000 0000.302c.9e96 04 04, state CONN_PEND
Mar 21 11:33:51.515: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: CPAR
Mar 21 11:33:51.515: LOCACK: CONN_POLL_ACK to peer 250/10.48.64.14/1996/4000.4000.4000 000
Mar 21 11:33:51.523: LOCACK: recv LLC2DATA, 4000.4000.4000 0000.302c.9e96 04 04
Mar 21 11:33:51.523: LOCACK: recv IFRAME, 4000.4000.4000 0000.302c.9e96 04 04, state CONN
Mar 21 11:33:51.527: LOCACK: IFRAME frame sent, 4000.4000.4000 0000.302c.9e96 04 04 CONN
Mar 21 11:34:01.522: LOCACK: recv RR, 4000.4000.4000 0000.302c.9e96 04 04, state CONNECT
Mar 21 11:34:11.518: LOCACK: recv RR, 4000.4000.4000 0000.302c.9e96 04 04, state CONNECT
logos#

```

This is the output from the remote RSRB peer:

```

dymock# debug local-ack packets

Local Acknowledgement packets debugging is on

dymock#
Mar 21 11:33:51.500: LOCACK: recv SABME P/F, bogus, state NO_ONES_HOME
Mar 21 11:33:51.500: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: DLC_IND_EXT CONN
Mar 21 11:33:51.500: LOCACK: CONN_REQ to peer 250/10.48.64.30/1996/0000.302c.9e96 4000.400
Mar 21 11:33:51.504: LOCACK: recv RNR, 0000.302c.9e96 4000.4000.4000 04 04, state REM_WAIT
Mar 21 11:33:51.508: LOCACK: recv RR, 0000.302c.9e96 4000.4000.4000 04 04, state REM_WAIT
Mar 21 11:33:51.512: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: CPR
Mar 21 11:33:51.512: LOCACK: CONN_POLL_ACK_REQ to peer 250/10.48.64.30/1996/0000.302c.9e96
Mar 21 11:33:51.516: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: CPA
Mar 21 11:33:51.520: LOCACK: recv IFRAME, 0000.302c.9e96 4000.4000.4000 04 04, state CONN
Mar 21 11:33:51.520: LOCACK: IFRAME frame sent, 0000.302c.9e96 4000.4000.4000 04 04 CONN
Mar 21 11:33:51.528: LOCACK: recv LLC2DATA, 0000.302c.9e96 4000.4000.4000 04 04
Mar 21 11:33:51.712: LOCACK: recv RR, 0000.302c.9e96 4000.4000.4000 04 04, state CONNECT
Mar 21 11:34:01.611: LOCACK: recv RR, 0000.302c.9e96 4000.4000.4000 04 04, state CONNECT
Mar 21 11:34:11.607: LOCACK: recv RR, 0000.302c.9e96 4000.4000.4000 04 04, state CONNECT
dymock#

```

This is the output from the local RSRB peer when the LLC2 connection is disconnecting:

```

logos# debug local-ack packets

Local Acknowledgement packets debugging is on

logos#
Mar 21 11:34:30.517: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: DLC_IND_FAIL
Mar 21 11:34:30.517: LOCACK: DISC_REQ to peer 250/10.48.64.14/1996/4000.4000.4000 0000.302
Mar 21 11:34:30.521: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: DA
Mar 21 11:34:30.521: LOCACK: 4000.4000.4000 0000.302c.9e96 04 04 event: DLC_IND_FAIL
logos#

```

This is the output from the remote RSRB peer when the LLC2 connection is disconnecting:

```

dymock# debug local-ack packets

Local Acknowledgement packets debugging is on

dymock#
Mar 21 11:34:30.518: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: DR
Mar 21 11:34:30.518: LOCACK: DISC_ACK to peer 250/10.48.64.30/1996/0000.302c.9e96 4000.400
Mar 21 11:34:30.522: LOCACK: recv UA P/F, 0000.302c.9e96 4000.4000.4000 04 04, state LOCAL
Mar 21 11:34:30.522: LOCACK: 0000.302c.9e96 4000.4000.4000 04 04 event: DLC_IND_CLOSED
dymock#

```

You can also look at output from the router from the **debug llc2 state** command. This example shows how LLC is moving from a connection request to normal operation.

This is output from the **debug llc2 state** command on the local peer:

```
logos# debug llc2 state
```

```
LLC2 State debugging is on
```

```
logos#
```

```
Mar 21 11:56:21.529: LLC (stsw): 4000.4000.4000 0000.302c.9e96 04 04, ADM -> SETUP (CONN_R
```

```
Mar 21 11:56:21.533: LLC: Connection established: 4000.4000.4000 0000.302c.9e96 04 04, suc
```

```
Mar 21 11:56:21.533: LLC (stsw): 4000.4000.4000 0000.302c.9e96 04 04, SETUP -> NORMAL (REC
```

```
logos#
```

This is output from the **debug llc2 state** command on the remote peer:

```
dymock# debug llc2 state
```

```
LLC2 State debugging is on
```

```
dymock#
```

```
Mar 21 11:56:21.524: LLC (stsw): 0000.302c.9e96 4000.4000.4000 04 04, ADM -> NORMAL (REC_S
```

```
Mar 21 11:56:21.524: LLC: normalstate: set_local_busy 0000.302c.9e96 4000.4000.4000 04 04
```

```
Mar 21 11:56:21.524: LLC (stsw): 0000.302c.9e96 4000.4000.4000 04 04, NORMAL -> BUSY (SET_
```

```
Mar 21 11:56:21.540: LLC: busystate: 0000.302c.9e96 4000.4000.4000 04 04 local busy cleared
```

```
Mar 21 11:56:21.540: LLC (stsw): 0000.302c.9e96 4000.4000.4000 04 04, BUSY -> NORMAL (CLEA
```

```
dymock#
```

The **show llc2** command provides information regarding the LLC2 information of a session between two routers. In this sample output, you can see that the session is identified by MAC addresses; in this instance, the session is in a NORMAL state, which means that it is up and running.

This is the output from the local peer:

```
logos# show llc2
```

```
LLC2 Connections: total of 1 connections
```

```
TokenRing0/0 DTE: 4000.4000.4000 0000.302c.9e96 04 04 state NORMAL
```

```
V(S)=1, V(R)=1, Last N(R)=1, Local window=7, Remote Window=127
```

```
akmax=3, n2=8,
```

```
xid-retry timer      0/60000    ack timer          0/1000
```

```
p timer              0/1000    idle timer        9950/10000
```

```
rej timer            0/3200    busy timer         0/9600
```

```
adm timer            0/60000    llc1 timer         0/0
```

```
akdelay timer        0/100     txQ count          0/200
```

```
logos#
```

This is the output from the remote peer:

```
dymock# show llc2
```

```
LLC2 Connections: total of 1 connections
```

```
TokenRing0/0 DTE: 0000.302c.9e96 4000.4000.4000 04 04 state NORMAL
```

```
V(S)=1, V(R)=1, Last N(R)=1, Local window=7, Remote Window=127
```

```
akmax=3, n2=8,
```

```
xid-retry timer      0/60000    ack timer          0/1000
```

```
p timer              0/1000    idle timer        3880/10000
```

```
rej timer            0/3200    busy timer         0/9600
```

```
adm timer            0/60000    llc1 timer         0/0
```

```
akdelay timer        0/100     txQ count          0/200
```

```
RIF: 0830.0141.0FA1.00A0
```

```
dymock#
```

Issue the **debug llc2 packet** command to see the actual packets passing through the router.

This is the output from the local peer:

logos# debug llc2 packets

LLC2 Packets debugging is on

logos#

```
Mar 21 12:09:19.583: LLC: 4000.4000.4000 0000.302c.9e96 04 04 ADM CONN_REQ (21)
Mar 21 12:09:19.583: LLC: 4000.4000.4000 0000.302c.9e96 04 04 txmt SABME_IND
Mar 21 12:09:19.587: LLC: llc_sendframe
05BB6DD0: 00 .
05BB6DE0: 40400040 00400080 00302C9E 9608B001 @.@.@...0,...0.
05BB6DF0: 410FA100 A004047F A.! . . .
Mar 21 12:09:19.587: LLC: llc2_input
05A00970: 80 .
05A00980: 40000030 2C9E96C0 00400040 00083001 @..0,...@.@..0.
05A00990: 410FA100 A0040573 A.! . . s
Mar 21 12:09:19.587: LLC: i REC_UA_RSP
Mar 21 12:09:19.587: LLC: 4000.4000.4000 0000.302c.9e96 04 04 SETUP REC_UA_RSP (11)
Mar 21 12:09:19.591: LLC: 4000.4000.4000 0000.302c.9e96 04 04 txmt RR_CMD_P1 N(R)=0 p/f=1
Mar 21 12:09:19.591: LLC: llc_sendframe
05BB7690: 0040 .@
05BB76A0: 40004000 40008000 302C9E96 08B00141 @.@.@...0,...0.A
05BB76B0: 0FA100A0 04040101 .!. . . .
Mar 21 12:09:19.591: LLC: llc2_input
05BB7410: 8040 .@
05BB7420: 0000302C 9E96C000 40004000 08300141 ..0,...@.@..0.A
05BB7430: 0FA100A0 04050101 .!. . . .
Mar 21 12:09:19.591: LLC: i REC_RR_RSP N(R)=0 p/f=1
Mar 21 12:09:19.591: LLC: 4000.4000.4000 0000.302c.9e96 04 04 NORMAL REC_RR_RSP (4)
Mar 21 12:09:19.599: LLC: 4000.4000.4000 0000.302c.9e96 04 04 NORMAL CLEAR_LOCAL_BUSY (34)
Mar 21 12:09:19.607: LLC: 4000.4000.4000 0000.302c.9e96 04 04 txmt I N(S)=0 N(R)=0 p/f=0 s
Mar 21 12:09:19.607: LLC: llc_sendframe
05A014D0: 00404000 40004000 .@.@.@.
05A014E0: 8000302C 9E9608B0 01410FA1 00A00404 ..0,...0.A.! . .
05A014F0: 00002D00 0000001E 6B800011 02010500 ..-.....k.....
05A01500: 00000000 ....
Mar 21 12:09:19.611: LLC: llc2_input
05BB6A10: 8040 .@
05BB6A20: 0000302C 9E96C000 40004000 08300141 ..0,...@.@..0.A
05BB6A30: 0FA100A0 04040002 2D000000 001EEB80 .!. . . .-.....k.
05BB6A40: 001111E3 D6D24040 40404000 00070100 ...cVR@@@@@.....
05BB6A50: 00000000 00 . . . .
Mar 21 12:09:19.611: LLC: i REC_I_CMD N(R)=1 N(S)=0 V(R)=0 p/f=0
Mar 21 12:09:19.611: LLC: 4000.4000.4000 0000.302c.9e96 04 04 NORMAL REC_I_CMD (1)
Mar 21 12:09:19.615: LLC (rs): 4000.4000.4000 0000.302c.9e96 04 04 REC_I_CMD N(S)=0 V(R)=0
Mar 21 12:09:19.615: LLC (rs): 4000.4000.4000 0000.302c.9e96 04 04 REC_I_CMD N(R)=2
Mar 21 12:09:19.615: LLC: INFO: 4000.4000.4000 0000.302c.9e96 04 04 v(r) 0
Mar 21 12:09:19.711: LLC: 4000.4000.4000 0000.302c.9e96 04 04 txmt RR_CMD_P0 N(R)=1 p/f=0
Mar 21 12:09:19.711: LLC: llc_sendframe
05A00470: 0040 .@
05A00480: 40004000 40008000 302C9E96 08B00141 @.@.@...0,...0.A
05A00490: 0FA100A0 04040102 .!. . . .
Mar 21 12:09:29.610: LLC: 4000.4000.4000 0000.302c.9e96 04 04 NORMAL INIT_PF_CYCLE (35)
Mar 21 12:09:29.610: LLC: 4000.4000.4000 0000.302c.9e96 04 04 txmt RR_CMD_P1 N(R)=1 p/f=1
Mar 21 12:09:29.610: LLC: llc_sendframe
05A01EB0: 0040 .@
05A01EC0: 40004000 40008000 302C9E96 08B00141 @.@.@...0,...0.A
05A01ED0: 0FA100A0 04040103 .!. . . .
Mar 21 12:09:29.610: LLC: llc2_input
05A01370: 8040 .@
05A01380: 0000302C 9E96C000 40004000 08300141 ..0,...@.@..0.A
05A01390: 0FA100A0 04050103 .!. . . .
Mar 21 12:09:29.610: LLC: i REC_RR_RSP N(R)=1 p/f=1
Mar 21 12:09:29.614: LLC: 4000.4000.4000 0000.302c.9e96 04 04 NORMAL REC_RR_RSP (4)
logos#
```

This is the output from the remote peer:



```

07A00D30:                                8040                                .@
07A00D40: 40004000 40008000 302C9E96 08B00141  @.@.@...0,...0.A
07A00D50: 0FA100A0 04040103                                .!. ....
Mar 21 12:09:29.688: LLC: i REC_RR_CMD N(R)=1 p/f=1
Mar 21 12:09:29.688: LLC: 0000.302c.9e96 4000.4000.4000 04 04 NORMAL REC_RR_CMD (3)
Mar 21 12:09:29.688: LLC (rs): 0000.302c.9e96 4000.4000.4000 04 04 REC_RR_CMD N(R)=2
Mar 21 12:09:29.688: LLC: 0000.302c.9e96 4000.4000.4000 04 04 txmt RR_RSP N(R)=1 p/f=1
Mar 21 12:09:29.688: LLC: llc_sendframe
07A01230:                                0040                                .@
07A01240: 0000302C 9E96C000 40004000 08300141  ..0,..@.@.@...0.A
07A01250: 0FA100A0 04050103                                .!. ....
Mar 21 12:09:39.692: LLC: 0000.302c.9e96 4000.4000.4000 04 04 NORMAL INIT_PF_CYCLE (35)
Mar 21 12:09:39.692: LLC: 0000.302c.9e96 4000.4000.4000 04 04 txmt RR_CMD_P1 N(R)=1 p/f=1
Mar 21 12:09:39.692: LLC: llc_sendframe
07A00830:                                0040                                .@
07A00840: 0000302C 9E96C000 40004000 08300141  ..0,..@.@.@...0.A
07A00850: 0FA100A0 04040103                                .!. ....
Mar 21 12:09:39.696: LLC: llc2_input
07A01870:                                8040                                .@
07A01880: 40004000 40008000 302C9E96 08B00141  @.@.@...0,...0.A
07A01890: 0FA100A0 04050103                                .!. ....
Mar 21 12:09:39.696: LLC: i REC_RR_RSP N(R)=1 p/f=1
Mar 21 12:09:39.696: LLC: 0000.302c.9e96 4000.4000.4000 04 04 NORMAL REC_RR_RSP (4)
dymock#

```

The **debug llc2 packet** command provides information regarding the frames themselves. Consider this portion of that output:

```

Mar 21 12:09:19.583: LLC: 4000.4000.4000 0000.302c.9e96 04 04 txmt SABME_IND
Mar 21 12:09:19.587: LLC: llc_sendframe
05BB6DD0:                                00                                .
05BB6DE0: 40400040 00400080 00302C9E 9608B001  @.@.@...0,...0.
05BB6DF0: 410FA100 A004047F                                A.!. ...

```

You can see the router sending a frame, even though you see this in the text before you can surmise this from the frame itself. By looking at the frame itself, you see the hexadecimal value 7F, which is a SABME. Now, consider this portion of that output:

```

Mar 21 12:09:19.587: LLC: llc2_input
05A00970:                                80                                .
05A00980: 40000030 2C9E96C0 00400040 00083001  @..0,..@.@.@...0.
05A00990: 410FA100 A0040573                                A.!. ...s
Mar 21 12:09:19.587: LLC: i REC_UA_RSP

```

That output shows that the (hexadecimal value 73) UA from the station hit the router. The **debug token ring** command would also show you this information, but it does not tell you the frame type. So, to use the **debug token ring** command, you need to know the different control fields.



**Caution:** The **debug token ring** command is very processor-intensive and should be used carefully.

Refer to Using the Cisco 1100 Access List Debug Filter Utility on an SRB Interface to learn how to limit that command output.

These are the values that equate to each sent command:

| Value              | Command              |
|--------------------|----------------------|
| XX XX              | Unnumbered format    |
| 0F <sup>TM</sup> F | Disconnect Mode (DM) |

|       |  |
|-------|--|
| 43 3  | Disconnect (DISC)                              |
| 63 3  | Unnumbered Acknowledgement (UA)                |
| 6 F F | Set Asynchronous Balance Mode Extended (SABME) |
| 87 97 | Frame Reject (FMR)                             |
| AF BF | Exchange Identification (XID)                  |
| E3 F3 | Test   |

As mentioned earlier, the **debug llc2 packet** command shows the information frames that are flowing through the system. This is an example of several frames in the local peer:

```

Mar 21 12:09:19.591: LLC: llc2_input
05BB7410:                               8040                .@
05BB7420: 0000302C 9E96C000 40004000 08300141 ..0,..@.@@..0.A
05BB7430: 0FA100A0 04050101                .!. ....
Mar 21 12:09:19.591: LLC: i REC_RR_RSP N(R)=0 p/f=1
Mar 21 12:09:19.591: LLC: 4000.4000.4000 0000.302c.9e96 04 04 NORMAL REC_RR_RSP (4)

```

By looking at the LLC2 frame itself, you can tell that that frame is going from the station to the host because the destination is first and the source is second. Remember that, when using SRB, the source MAC address should have the high order bit set. Doing so changes the value of the first byte of the source MAC address.

**Note:** These debugs were issued in a controlled lab environment where there was only one PU and a host, which created very little traffic. In a live network, use the **access-list 1100** command to filter debugs so that you can efficiently debug the network. This process is described in detail in Understanding and Troubleshooting Local Source–Route Bridging.

---

## Related Information

- [Understanding and Troubleshooting Local Source–Route Bridging](#)
- [Technology Support](#)
- [Product Support](#)
- [Technical Support – Cisco Systems](#)

---

[Contacts & Feedback](#) | [Help](#) | [Site Map](#)

© 2008 – 2009 Cisco Systems, Inc. All rights reserved. [Terms & Conditions](#) | [Privacy Statement](#) | [Cookie Policy](#) | [Trademarks of Cisco Systems, Inc.](#)

---

Updated: Sep 09, 2005

Document ID: 5204

---