TIBCO Rendezvous®

Administration

Version 8.8.0 | February 2025

Contents

Preface

TIBCO Rendezvous® is a messaging infrastructure product.

TIBCO is proud to announce the latest release of TIBCO Rendezvous software. This release is the latest in a long history of TIBCO products that leverage the power of the Information Bus® technology to enable truly event-driven IT environments. To find out more about how TIBCO Rendezvous software and other TIBCO products are powered by TIB® technology, please visit us at www.tibco.com.

This manual explains administration of TIBCO Rendezvous software and the distributed systems that use it. It is part of the documentation set for Rendezvous Software Release 8.8.0.

Parts of this book describe the configuration of Rendezvous components using a graphical browser administration interface. The book TIBCO Rendezvous Configuration Tools describes a programmer interface and an XML tool for configuring the same parameters.

Manual Organization

This document begins with important information for system administrators:

- Do This First—Administrator’s Checklist

The following section describes several details upon which programmers and administrators must agree for correct program operation:

- Network Details

The next several sections describe Rendezvous components that run as executable processes. Administrators must ensure correct set-up and operation of these components:

- Rendezvous Daemon (rvd)
- Routing Daemon (rvrd)
- Secure Daemons (rvsd and rvsrd)
- Current Value Cache

These sections describe utilities for measuring overall system capacity and performance, and for diagnosing network problems.

- Performance Assessment (rvperf)
- Latency Assessment (rvlat)
- Measuring Tools for IPM
- Protocol Monitor (rvtrace)
- Prometheus Endpoints

Additional administrative tasks apply when distributed systems use advanced features of Rendezvous software:

- Certified Message Delivery
- Fault Tolerance
- Distributed Queues

Do This First—Administrator’s Checklist

This checklist outlines several important tasks for system administrators.

We recommend that you review this checklist when you install TIBCO Rendezvous® software on any of your networks, when you add new networks, when you add new platforms to your networks, when you reconfigure networks, and whenever users report problems with Rendezvous software.

Install the Rendezvous Software

Install Rendezvous software at your site. The TIBCO Rendezvous Installation guide describes the installation procedure on various hardware and operating system platforms.

Product Activation

You must activate this release as part of an installation or upgrade. The release will not function unless it is activated with a license.

Generate a license to activate this product within the TIBCO Software Download site at http://www.tibco.com/downloads. Only customers with an active entitlement to this release can generate a license.

Activate TIBCO products using TIBCO® Activation Service software, which you can download from the TIBCO Software Download site.

To learn how to configure this product for activation, visit: https://docs.tibco.com/.

Enable Access to Executable Binary Files

Microsoft Windows Platforms

Skip this step—it is done automatically during the installation procedure.

UNIX Platforms

Add the Rendezvous binary directory to the execution path of each programmer and end user of Rendezvous programs. The exact directory name varies depending on where you installed Rendezvous; the installation procedure prints the correct location for your convenience (usually a name constructed like *installation_point*/`tibco/tibrv/bin`).

Add Service Entries

Rendezvous transports use the service `rendezvous` as a default (when programmers do not explicitly specify a service). If `rendezvous` is not defined as a service, the secondary default is UDP port 7500.

We recommend that you define `rendezvous` as a service name in your network database. If port 7500 is already in use on your network, you *must* define `rendezvous` as a service (designating an available port number). The examples below define `rendezvous` as port 7500, but you may use any UDP port number.

Some organizations may want to define additional services to segregate Rendezvous communications. For example, by convention, fault tolerance messages between Rendezvous components use service 7504. You may also define those services at this time. For more information, see Service Selection.

On all platforms, Rendezvous software obtains service names by calling the function `getservbyname()`. Ensure that this function returns the correct port numbers.

UNIX Platforms

Add these definitions to the services database:

```
rendezvous      7500/udp
rendezvous-ft   7504/udp
```

Microsoft Windows Platforms

On all supported Windows platforms, add these definitions to the services database:

```
rendezvous      7500/udp
rendezvous-ft   7504/udp
```

Enable Packet Checksums

Rendezvous software *requires* that the operating system compute packet checksums. You *must* configure the operating system to enable packet checksums on every host computer that runs Rendezvous programs or executable components.

Most operating systems enable packet checksums by default. Nonetheless, we recommend that you ensure that this setting is still in effect.

Arrange Internetwork Communications

This step extends the Rendezvous software from intranetwork message exchange to internetwork message exchange.

- If you plan to run Rendezvous programs on a single network, you may skip this step.
- If you plan to link several networks, read Routing Daemon (rvrd).

Arrange your network appropriately.

File Descriptor Limits

On UNIX, IBM i and z/OS platforms, the operating system can limit the maximum number of file descriptors per process, as well as the total number of file descriptors summed over all processes. Because each connection uses a file descriptor, this limitation in turn limits the capacity of Rendezvous components:

- In `rvd` and its variants, it limits the maximum number of client connections (that is, transports) that a daemon can accept.
- In `rvrd` and its variants, it limits the combined total of neighbor connections and client connections.
- In client programs, it limits the number of transport objects.

Symptoms

When operating system file descriptor limits are set too low, Rendezvous components might report errors indicating that too many files are open, or that file descriptor limits have been exceeded. In many situations, you can eliminate this problem by raising the limit.

UNIX Resource Inheritance

In a UNIX environment, when a Rendezvous client program automatically starts a daemon process, the daemon becomes a child process of the client. As a child, the daemon inherits all the file descriptors and sockets that are open in the client process (for example, the descriptor associated with an open log file). Even after the client process (parent) closes the resource and exits, the resource remains open in the daemon process (child) until the daemon exits. Furthermore, the operating system does not release the space associated with a file that was open in the parent process until the child process exits.

For this reason we do not recommend using the auto-start feature in production environments. Consider the following solutions:

- Start daemons explicitly (instead of relying on the client to automatically start its daemon).

  For a tactic that might be helpful in implementing this solution, see Suppress Daemon Auto-Start on page 86 in TIBCO Rendezvous Concepts.

- Ensure that clients auto-start the daemon before opening any file descriptors or sockets. In this way, the daemon does not inherit these resources (other than `stderr`).

- Create a script that explicitly closes the inherited resources before starting the daemon.

  The auto-start feature calls an executable file named `rvd`. You can substitute your own script named `rvd` that explicitly closes inherited resources.

Product Activation

A Rendezvous daemon must be activated before it will start. Activation is performed by TIBCO Activation Service software which can be obtained from the TIBCO Software download site. Please see the TIBCO License Activation Service documentation for complete details on activating TIBCO products.

A Rendezvous daemon is configured to connect to a TIBCO Activation Service with a URL in the form of `https://url[?fp=activation` service fingerprint]. A fingerprint of the Activation Service can be included in the URL for peer verification. The URL can be supplied to the Rendezvous daemon through the `-license` command-line parameter or the environment variable `TIBRV_LICENSE`. If the URL is not supplied in the command-line or environment, the Rendezvous daemon will search its environment PATH for a license file named `tibrvlic.bin` and, failing that, will attempt to contact an Activation Service at `https://tib-activate:7070`.

> **Note:** Production environments must use a TIBCO Activation Service for TIBCO product activation. Using a license file to activate a Rendezvous daemon is only supported for development environments. The URL format for a license file is `file://pathname`.

License Expiration and Caching

Licenses obtained from the TIBCO Activation Service are cached locally and used by the Rendezvous daemon on subsequent process restarts to avoid unnecessary network traffic and improve start up time. Cached licenses are invalidated if the Rendezvous daemon `-listen` or `-reuse-port` values change and the process is restarted, requiring a new license to be obtained from the TIBCO Activation Service.

Cached licenses may expire sooner than the final expiration date of the license itself. This is done to ensure that the cached license is periodically refreshed and remains valid.

When the Rendezvous daemon cannot reach the TIBCO Activation Service during a license refresh, a warning is logged indicating the potential for service disruption.

If the cached license expires while the TIBCO Activation Service is unreachable, the Rendezvous daemon will continue to operate until the final expiration date of the license or the process is stopped, allowing for continued operation during temporary service outages.

If the cached license has not expired and the Rendezvous daemon is restarted while the Activation Service is unreachable, the effective expiration date is that of the cached license, not the final expiration of the license maintained by the TIBCO Activation Service.

If the cached license expires and the Rendezvous daemon is restarted while the Activation Service is unreachable, the process will be unlicensed and will terminate immediately with an error message, as no valid license can be obtained.

Logging

Message	Frequency	Remarks	Result
Entitlement to TIBCO Rendezvous for <COMPANY_NAME> is confirmed through dd-mmm-yyyy.	Once	At start up and whenever a license is re-acquired.	
Notice: License for TIBCO Rendezvous for <COMPANY_NAME> expires on dd-mmm-yyyy. Request a new license soon.	Daily	<90 days from expiration	
ALERT: License for TIBCO Rendezvous for <COMPANY_NAME> expires on dd-mmm-yyyy. Request a new license immediately to avoid potential disruption.	Hourly	<7 days from expiration	
ALERT: Unable to connect to TIBCO Activation Service instance at URL. Restore connectivity in <DAYS> days to avoid potential disruption.	Once	On failure to connect to the licensing server.	
TERMINAL WARNING: Entitlement to	Once or	License expiration.	Daemon

Message	Frequency	Remarks	Result
TIBCO Rendezvous for <COMPANY_ NAME> ended on dd-mmmyyyy. Renew your entitlement immediately to avoid potential product shutdown.	Hourly		exits
ERROR: Your organization does not have entitlement to TIBCO Rendezvous. Contact TIBCO at https://www.tibco.com/contactus to purchase entitlement to this product.	Once	At startup or if the license is revoked, for example, the license server successfully responds, but does not respond with the requested feature.	Daemon exits

Network Details

Rendezvous software hides most networking details from applications programmers. In some cases (in cooperation with network and system administrators), programmers may supply three optional networking parameters—`network`, `service` and `daemon`—to the transport creation calls. This section describes those parameters for the administrator.

Note

The TIBCO Rendezvous Concepts guide describes these parameters in even greater detail than this section. See also these sections:

- Service Parameter in TIBCO Rendezvous Concepts
- Network Parameter in TIBCO Rendezvous Concepts
- Daemon Parameter in TIBCO Rendezvous Concepts

Transport Parameters

Network transport creation calls accept three parameters that govern the behavior of the Rendezvous daemon: `service`, `network` and `daemon`. In simple networking environments, the default values of these parameters are sufficient (in C, the program can supply `NULL` for all three).

Most programmers will use default values for these parameters unless advised otherwise by their network administrator. To determine whether your environment requires special treatment, consider whether any of these conditions apply:

- Several independent distributed applications run on the same network, and you must isolate them from one another (`service` parameter).
- Programs use the Rendezvous routing daemon, `rvrd`, to cooperate across a WAN with programs that belong to a particular service group, and the local programs must join the same service group (`service` parameter).
- A Rendezvous program runs on a computer with more than one network interface, and you must choose a specific network for Rendezvous communications (`network` parameter).

- Computers on the network use multicast addressing to achieve even higher efficiency, and programs must specify a set of multicast groups to join (`Network` parameter).
- A program runs on one computer, but connects with a Rendezvous daemon process running on a different computer, and you must specify the remote daemon to support network communications (`daemon` parameter).
- Two programs use direct communication. Both programs must enable this feature and specify its service (`service` parameter).

If none of these conditions apply, then programmers can use default values for the transport parameters.

If your network environment requires special treatment for any these parameters, please notify applications programmers developing software for your environment. If your organization runs Rendezvous programs developed by a third party, consult the third-party documentation for information about network and service configuration.

In addition, certain components of Rendezvous software, local programs and third-party programs may also require special configuration:

- The Rendezvous routing daemon, `rvrd`, must specify the service and network for each local network. Exchange this information with the network administrators at each remote site.
- The Rendezvous secure daemon limits clients to communication on a set of authorized network and service pairs.
- The current value cache, `rvcache`, accepts all three transport parameters. When you configure this program, include any special values as needed.
- Many Rendezvous programs accept transport parameters on their command lines. Inform all users of any special values that apply.

Service Selection

Rendezvous daemon (`rvd`) processes communicate using UDP services. The `service` parameter instructs the Rendezvous daemon to use this service whenever it does network communications on behalf of this transport.

As a direct result, services divide the network into logical partitions. Each transport communicates over a single service; a transport can communicate only with other transports on the same service. To communicate with more than one service, a program must create more than one transport.

Interaction between Service and Network Parameters

Within each Rendezvous daemon, all the transports that use a specific service must also use the same network parameter. That is, if the `service` parameters resolve to the same UDP port, then the `network` parameter must also be identical. (This restriction extends also to routing daemons.)

For example, suppose that the program `foo`, on the computer named `orange`, has a transport that communicates on the service `svc1` over the network `lan1`. It is illegal for *any* program to subsequently create a transport to that `rvd` on `orange` to communicate on `svc1` over any other network—such as `lan2`. Once `rvd` binds `svc1` to `lan1`, that service cannot send outbound broadcast messages to any other network. Attempting to illegally rebind a service to a new network fails; the transport creation call produces the status code `TIBRV_INIT_FAILURE`.

To work around this limitation, use a separate service for each network.

The limitation is not as severe as it might seem at first, because it only affects *outbound broadcast* messages.

- *Point-to-point* messages on the transport's service travel on the appropriate network (as determined by the operating system) irrespective of the transport's `network` parameter.
- *Inbound* broadcast messages on the transport's service can arrive from any network, irrespective of the transport's `network` parameter.

Specifying the UDP Service

Programs can specify the service in one of several ways, listed in order of preference in Specify UDP Service.

Specify UDP Service

Port number	When a program specifies a UDP port number, it must be a string representing a decimal integer. For example: `"7890"`
Service name	When a program specifies a service name, the transport creation function searches the network database using `getservbyname()`, which searches a network database such as NIS, DNS or a flat file such as `/etc/services` (in some versions of UNIX).
Default (Non-Secure Daemons)	If a program does not specify a service, or it specifies null, the transport creation function searches for the service name `rendezvous`. If `getservbyname()` does not find `rendezvous`, the Rendezvous daemon instructs the transport creation function to use a hard default of port `7500`. We strongly recommend that administrators define `rendezvous` as a service, especially if port `7500` is already in use. For example, network administrators might add the following service entry to the network database: `rendezvous 7500/udp` Once this entry is in the network database, programmers can conveniently specify `NULL` or the empty string as the `service` argument to create a transport that uses the default Rendezvous service.
Default (Secure Daemons)	Secure daemons use internal defaults, which must be set explicitly by the administrator; see Default Network and Service.
Direct Communication	To enable direct communication, specify two parts separated by a colon: • UDP service for regular communication

	• UDP service for direct communication You may specify both parts either as a service name or a port number. Direct communication is not available when connecting to a remote daemon. For a general overview, see Direct Communication on page 91 in TIBCO Rendezvous Concepts.

The `rvd` daemon interprets this service as a UDP service.

Network Selection

Every network transport object communicates with other transport objects over a network. On computers with only one network interface, the Rendezvous daemon communicates on that network without further instruction from the program.

On computers with more than one network interface, the `network` parameter instructs the Rendezvous daemon to use a particular network for all communications involving this transport. To communicate over more than one network, a program must create a separate transport object for each network. For further details, see Limitation on Computers with Multiple Network Interfaces.

The network parameter also specifies multicast addressing details (for a brief introduction, see Multicast Addressing).

To connect to a remote daemon, the `network` parameter must refer to the network from the perspective of the remote computer that hosts the daemon process.

Constructing the Network Parameter

The network parameter consists of up to three parts, separated by semicolons—network, multicast groups, send address—as in these examples:

`"lan0"`	network only

`"lan0;225.1.1.1"`	one multicast group
`"lan0;225.1.1.1,225.1.1.5;225.1.1.6"`	two multicast groups, send address
`"lan0;;225.1.1.6"`	no multicast group, send address

Part One—Network

Part one identifies the network, which you can specify in several ways:

Specify Network

Host name	When a program specifies a host name, the transport creation call uses an operating system call that searches a network database to obtain the IP address. The maximum length of a host name string is 256 characters.
Host IP address	When a program specifies an IP address, it must be a string representing a multi-part address. For example: `"101.120.115.111"`
Network name (where supported)	When an application specifies a network name, the transport creation function calls `getnetbyname()`, which searches a network database such as Network Information Services (NIS) or a flat file (such as `networks`) in the system directory.
Network IP number	If a program specifies a host IP address or a network IP number it must be in dotted-decimal notation. For example, `101.55.1.10`.
Interface name (where supported)	When an application specifies an interface name, the transport creation function searches the interface table for the specified interface name.

	For example, `lan0`. The interface name must be one that is known to `ifconfig` or `netstat`.
Default (Non-Secure Daemons)	If a program does not specify a network, the transport creation function uses the default network that corresponds to the host name of the system as determined by the C function `gethostname()`.
Default (Secure Daemons)	Secure daemons use internal defaults, which must be set explicitly by the administrator; see Default Network and Service.

The use of the UDP broadcast protocol has generally been superseded by the IP multicast protocol. To use broadcast protocols without multicast addressing, specify only part one of the network parameter, and omit the remaining parts.

Part Two—Multicast Groups

Part two is a list of zero or more multicast groups to join, specified as IP addresses, separated by commas. Each address in part two must denote a valid multicast address. Joining a multicast group enables listeners on the resulting transport to receive data sent to that multicast group.

For a brief introduction to multicasting, see Multicast Addressing.

Part Three—Send Address

Part three is a single send address. When a program sends broadcast data on the resulting transport, it is sent to this address. (Point-to-point data is not affected.) If present, this item must be an IP address—not a host name or network name. The send address *need not* be among the list of multicast groups joined in part two.

If you join one or more multicast groups in part two, but do not specify a send address in part three, the send address defaults to the first multicast group listed in part two.

Multicast Addressing

Multicast addressing is a focused broadcast capability implemented at the operating system level. In the same way that the Rendezvous daemon filters out unwanted messages based on service groups, multicast hardware and operating system features filter out unwanted messages based on multicast addresses.

When no broadcast messages are present on the service, multicast filtering (implemented in network interface hardware) can be more efficient than service group filtering (implemented in software). However, transports that specify multicast addressing still receive broadcast messages, so combining broadcast and multicast traffic on the same service can defeat the efficiency gain of multicast addressing.

Rendezvous software supports multicast addressing only when the operating system supports it. If the operating system does not support it, and you specify a multicast address in the network argument, then transport creation calls produce an error status `TIBRV_NETWORK_NOT_FOUND).(`

Limitation on Computers with Multiple Network Interfaces

On computers with more than one network interface, Rendezvous programs must not attempt to combine communications over different network interfaces using the same UDP service. To understand this limitation, consider these examples of *incorrect* usage in the context of multiple network interfaces.

Erroneous Examples

- A program, `mylistener`, creates a transport using service 7500 and network `lan0`; it listens to broadcast subjects using that transport. Other program processes on both `lan0` and `lan1` send broadcast messages using service 7500.

  As a result, `mylistener` might unexpectedly receive inbound messages from `lan1`.

- An administrator configures the Rendezvous routing daemon on a computer with two network interfaces (`lan0` and `lan1`) using service 7500. Since the administrator does not specify a `-network` parameter, the routing daemon uses the default network interface.

As a result, the routing daemon forwards messages from its neighbor only to the default network interface; however, it might forward messages from both `lan0` and `lan1` to its neighbor.

- A program creates two network transports. Both use service 7500, but one uses network `lan0`, while the other uses network `lan1`.

  As a result, the call to create the second transport produces an error.

- Two programs on the same computer each create a transport. Both use service 7500, but one uses network `lan0`, while the other uses network `lan1`. Even though these transports are in different processes, both transports connect to the same Rendezvous daemon—which is subject to the limitation.

  As a result, the program fails to create the second transport.

We recommend *caution* when you deploy Rendezvous programs or Rendezvous routing daemons on any computer with multiple network interfaces.

Source of the Limitation

The roots of this limitation are in the underlying IP broadcast protocols. Consider this asymmetry:

- When sending an *outbound* broadcast packet, IP software sends the packet on exactly one network.

  Rendezvous programs can specify this network with the transport creation function's `network` parameter.

- In contrast, IP software collects *inbound* broadcast packets from all network interfaces.

  Furthermore, when IP software presents an inbound packet to a client program (such as `rvd`) it does not include any indication of the network on which that packet arrived.

Because of this asymmetry, the actual behavior of IP broadcast protocols can be different than one might expect.

Avoiding the Limitation

You can use two strategies to avoid problems associated with this limitation:

- Use a separate service for Rendezvous messages on each network.
- Use multicast addressing to precisely define a range of reachable transports.

Using a separate service can rectify all four of the erroneous examples. Multicast addressing can rectify the first two erroneous examples, but not the latter two. In all four examples, a single Rendezvous daemon is sufficient.

For example, consider these two approaches to rectifying the first erroneous example:

- Separate Service

  A program, `mylistener`, creates a transport using service 7500 and network `lan0`; it listens to broadcast subjects using that transport. Other processes on `lan0` send messages using service 7500, but all processes on `lan1` send messages using service 7510. Separating by service prevents the transport from receiving interference from `lan1`.

- Multicast Addressing

  A program, `mylistener`, creates a transport using service 7500 and multicast network `lan0;225.1.1.1`. This transport selectively receives only those inbound multicast messages that are sent on network `lan0`, to multicast address `225.1.1.1`, on service 7500. Multicast addressing (where available) filters out messages sent on other networks using any other multicast address.

  However, multicast addressing does not filter out IP *broadcast* messages sent on the same UDP service. Once again, the roots of this limitation are in the underlying IP broadcast protocols.

Daemon Client Socket—Establishing Connections

The Rendezvous daemon (`rvd`) and the Rendezvous routing daemon (`rvrd`) both open a client socket to establish communication with their clients (Rendezvous programs). The `-listen` option to `rvd` and `rvrd` lets you specify where the daemon should listen for new client connections. Conversely, Rendezvous programs request connections with the daemon at that client socket. The `daemon` parameter of the transport creation function determines how and where to find the Rendezvous daemon and establish communication.

Each transport object establishes a communication conduit with a Rendezvous daemon, as described in these steps:

Procedure

1. The daemon process opens a (TCP) *client socket*, and waits for a client to request a connection.

   The -listen option of the Rendezvous daemon specifies where the daemon listens for new client transports.

2. The program calls the transport creation function, which contacts the daemon at the client socket specified in its daemon parameter.

   The daemon parameter of the transport creation function *must* correspond to the -listen option of the daemon process; that is, they must specify the same communication type and socket number.

   If no daemon process is listening on the specified client socket, then the transport creation call automatically starts a new daemon process (which listens on the specified client socket) and then attempts to connect to it.

3. The daemon process opens a conduit for private communication with the new transport. All future communication uses that private conduit.

   The request socket is now free for additional requests from other client transports.

Specifying a Local Daemon

Specify the daemon's client socket as a character string.

For *local* daemons, specify a TCP socket number; for example: "6555"

If you omit the daemon parameter of the transport creation function (in C, supply NULL), it uses 7500 as the default. Similarly, to start a daemon process using the default socket, omit the -listen option to the daemon command line.

In all cases, the communication type and socket number in the daemon parameter of the transport creation call must match those given to rvd through its -listen parameter.

See Also

POSIX Local IPC Sockets

Remote Daemon

In most cases, programs use a local daemon, running on the same host as the program. Certain situations require a *remote* daemon, for example:

- The program runs on a laptop computer that is not directly connected to the network. Instead, the laptop connects to a workstation on the network, and the daemon runs on that workstation.
- The program connects to a network at a remote site.

For *remote* daemons, specify two parts (introducing the remote host name as the first part):

- Remote host name.
- TCP socket number.

For example: `"purple_host:6555"`.

Once again the communication type and socket number in the `daemon` parameter of the transport creation call must match those given to `rvd` through its `-listen` parameter. However, the `-listen` parameter still receives only a two-part argument—without a remote host name.

When a client specifies a remote daemon that is not present, the client does *not* auto-start a daemon in that remote location.

Note For a general overview, see Direct Communication on page 91 in TIBCO Rendezvous Concepts.

Barring Remote Connections

A Rendezvous daemon or routing daemon can prohibit connections from remote programs by specifying `-listen "127.0.0.1"`. The special network address `127.0.0.1` represents the local host, so this parameter specifies that only local programs may connect.

This configuration is especially important when a routing daemon runs on a firewall computer.

POSIX Local IPC Sockets

POSIX local IPC sockets, also known as UNIX domain sockets, are an alternative to the TCP/IP sockets that Rendezvous normally uses for communication between client and daemon. When available, IPC sockets yield faster performance than TCP/IP sockets. All other Rendezvous behavior is transparent to this choice of socket protocol.

Availability

IPC sockets are available only on UNIX platforms that support `AF_UNIX` or `AF_LOCAL` socket types.

Behavior

On UNIX platforms where IPC is available, they are the default behavior of Rendezvous (release 8.3 and later). That is, Rendezvous automatically uses IPC sockets for communication between client and daemon processes on the same host computer. To override (or force) this default behavior, you can explicitly configure either the daemon or the client (see below).

On platforms where IPC is not available, requests to use IPC always fail.

Daemon Configuration

The daemon's `-listen` parameter accepts values that specify the available socket protocols.

Socket Type for Client-Daemon Communication—Daemon

Value	Behavior
`tcp:`*port*	The daemon listens only for TCP connection requests. If the operating system prevents the daemon from listening for TCP connections, the daemon exits immediately.
`ipc:`*port*	The daemon listens for both IPC and TCP connection requests. If the operating system prevents the daemon from listening for either TCP or IPC connections, the daemon exits immediately.

Value	Behavior
port	The daemon listens for both IPC and TCP connection requests. However, if the operating system prevents the daemon from listening for IPC connections, the daemon does not exit; instead, it degrades gracefully, listening only for TCP connection requests.
`ipc:`*ip_addr*`:`*port* *ip_addr*`:`*port*	When an *ip_addr* is present, the daemon listens for both IPC and TCP connection requests, listening *only* on the interface *ip_addr* (on *port*). If *ip_addr* is `127.0.0.1`, then the daemon listens for connection requests only from local clients. If the prefix `ipc:` is present, and the operating system prevents the daemon from listening for either TCP or IPC connections, the daemon exits immediately. If the prefix `ipc:` is absent, and the operating system prevents the daemon from listening for IPC connections, the daemon does not exit; instead, it degrades gracefully, listening only for TCP connection requests.

More than One Daemon

When starting two (or more) daemons on the same host computer, you must specify distinct *port* numbers. If one daemon is already using a *port* number, and another daemon attempts to reuse it, the second daemon exits immediately.

Client Configuration

Client programs can specify the preferred socket protocol for connecting to the daemon; specify the socket preference in the *daemon* parameter of the transport creation call.

Socket Type for Client-Daemon Communication—Client

Value	Behavior
`tcp:`*port*	The client connects to the daemon using a TCP/IP socket.
`ipc:`*port*	The client connects to the daemon using an IPC socket. If the connection

Value	Behavior
	request fails, then the transport creation call fails too.
port	The client first attempts to connect to the daemon using an IPC socket; if that attempt fails, then the client attempts to connect using a TCP/IP socket.

Note IPC sockets are available only if the client and the daemon reside on the same host computer.

Daemon Auto-Start

If no daemon is listening for client connections on *port*, then the transport creation call attempts to start one automatically. The transport creation call replicates its own *daemon* argument as the new daemon's `-listen` argument.

When a client specifies a remote daemon that is not present, the client does *not* auto-start a daemon in that remote location.

Default Port and Service Numbers

For convenient reference, these tables list port and service numbers with special meaning to Rendezvous components.

Browser Administration Interface

Rendezvous components use HTTP ports for browser administration interfaces.

Configurable daemons distributed with Rendezvous also open an ephemeral HTTPS port (to keep the daemon configuration secure against unauthorized modification). To find the actual HTTPS port that the operating system has assigned, check the daemon's start banner or log file. The configurable daemons are `rvrd`, `rvsd`, `rvsrd`, and `rvcache`—but not `rvd`.

Default HTTP Port Numbers

HTTP Port	Rendezvous Component
Daemons Distributed with Rendezvous	
7580	rvd, rvrd, rvsd, rvsrd
7581	rvcache
Daemons Distributed with Related Products	
7590	`rvtxd`

Using the Browser Administration Interface

To use the browser administration interface, point your browser at an address like this template:

```
http://host:port
```

- *host* can be the host name or IP address of the host computer where the daemon is running. In some networks, a fully qualified host name is required (for example, `myhost.tibco.com`).
- *port* is either the default port from *Default HTTP Port Numbers, or the port supplied as the –* `http` *command line argument when starting the daemon; see Command Line Parameters.*

Service Ports

Default UDP Service Numbers

Service	Description
`rendezvous` 7500	Program transport objects use these UDP services as defaults. For more detail, see Specifying the UDP Service.

Service	Description
`rendezvous-ft` 7504	Fault tolerance members use these UDP services as defaults for fault-tolerance protocol communications.

Listen Ports

Default TCP Port Numbers

TCP Port	Rendezvous Component
7500	rvd and rvrd use this TCP port as the default to listen for new connections from program transports. Program transport objects use this TCP port as the default to establish connections to rvd or rvrd.

Reliability and Message Retention Time

The *reliability interval* (or *retention time*) is the time interval during which a sending daemon retains outbound messages. Retaining message data allows the sending daemon to retransmit message packets upon request from another daemon process (which did not correctly receive the data). Conversely, the reliability interval also specifies the time for which a receiving daemon can request retransmission of missing data from a sender.

A related concept, the *reliability window*, refers to the set of outbound message data that a sending daemon has retained (and not yet discarded). Data within the reliability window is available for the sending daemon to retransmit upon request.

You can specify the reliability interval in several ways, depending on the needs and complexity of your enterprise:

- For all services of a daemon, using a factory default (60 seconds)
- For all services of a daemon, using a daemon command-line argument
- For a specific service, using a client API call

See Also

For additional background information, see also Reliable Message Delivery on page 34 in TIBCO Rendezvous Concepts.

Factory Default

The factory default reliability interval is 60 seconds. In environments with low data rates, this default is sufficient for reliability and does not impose high memory requirements on the daemon process. Since it is a hard-coded default, it is simple to use, requiring no further administrative action.

Using a Non-Default Reliability Interval

Decreasing

Some situations require a shorter reliability interval, in order to decrease memory requirements. For example:

- High-Speed Sender

  Consider a program that sends messages at very high data rates. Using the factory default retention time, the daemon must retain 60 seconds worth of data in its process memory; this volume of data might overwhelm the host computer's available memory. To reduce the `rvd` process memory requirement, consider shortening the retention time.

- Time-Sensitive Data

  In some programs, data becomes obsolete before 60 seconds elapse. For example, in real-time multi-player game networks, data might become obsolete in less than one second. Retaining and retransmitting obsolete data burdens the daemons and the network without any benefit. A shorter retention time would be more consistent with application requirements, and can improve overall performance.

  In some situations, it is not unreasonable to reduce retention time to zero.

Consequences

Decreasing retention time decreases reliability, and increases the probability of lost data. `DATALOSS` advisory messages indicate message data lost because the sending daemon no longer retains it.

Zero

Zero is a special value for retention time. When the retention time is zero, the sending daemon does not store message data for retransmission (nor does it retransmit packets). Conversely, when the retention time is zero, the receiving daemon does not request that sending daemons retransmit data packets.

Zero is a legal value for a daemon’s `-reliability time` `parameter (see below). However, zero is not a legal value when a client program requests reliability for a specific service (see below).`

Increasing

We strongly discourage *increasing* the retention time beyond 60 seconds. Memory requirements increase in direct proportion to the retention time. If greater reliability is required, consider using certified delivery features instead (see Certified Message Delivery on page 115 in TIBCO Rendezvous Concepts).

Lower Value Rule

It is *not necessary* that all daemons in a network specify the same reliability time. If a sending daemon and a receiving daemon have different reliability intervals, the lower value governs retransmission interactions between the two.

- The sender’s actual retention time follows the sender’s reliability interval.
- The receiver requests retransmissions only until the lower value of the reliability interval (either the receiver’s or the sender’s) has elapsed.

Contrast this rule with the Service Reliability Rule that applies among transports that connect to the *same* daemon; see below.

Routing Daemons are Exempt

The reliability interval affects only the messages within a local network. It does not affect the operation of `rvrd` as it transfers messages across network boundaries.

Changing the Reliability Interval at a Daemon

You can override the factory default reliability interval for a specific daemon by supplying the `-reliability time` parameter on the command line that starts the daemon.

The value (measured in seconds) that you supply replaces the 60-second factory default, becoming the new default reliability interval for the daemon. It applies to all services on which the daemon communicates—*except* when a lower value in turn overrides it for a specific service or multicast group (see below).

Changing the Reliability Interval within an Application Program

Application programs can override both the factory default and the daemon's `-reliability` parameter. An API call can *request reliability* on a specific service. Any transport can request a reliability interval. The daemon uses the requested value as one of several inputs when it computes the effective reliability interval (see below).

Each request pertains to an individual transport, and is independent of other transports on the service or within an application process. A transport that does not request a specific reliability interval implicitly requests the daemon's governing value.

Programs can request reliability only from daemons of release 8.2 or later.

An application can request a shorter retention time than the value that governs the daemon as a whole (either the factory default or the daemons `-reliability` parameter). The daemon silently overrides calls that request a retention time longer than the daemon's governing value.

Service Reliability Rule

Client transport objects that connect to the same daemon could specify different reliability intervals on the same service—whether by requesting a reliability value, or by using the daemon's governing value. In this situation, the daemon resolves the difference using a method that favors more stringent reliability requirements, yet limits the maximum value to the daemon's governing value.

Procedure

1. The daemon begins by selecting the *largest* potential value from among all the transports on that service.
2. The daemon then compares that maximum value to the daemon's governing value, and uses the smaller of the two as the effective reliability interval for the service (that is, for all the transports on the service). That is, the daemon's governing value limits the maximum requested value.

Contrast this rule with the Lower Value Rule that applies *between* two daemons; see above.

Service Reliability Rule

Consider a situation in which the daemon's command line specifies 40 seconds as the `-reliability time` value. Two client transports on service `7500` request reliability values of 30 seconds and 50 seconds. The daemon selects the largest value, 50, and then limits it to the daemon's governing value of 40 seconds.

Now consider a separate situation, in which the daemon uses the factory default reliability (60 seconds) as its governing value. A client transport requests 75 seconds. The daemon limits that request to 60 seconds.

Recomputing the Reliability

Whenever a transport connects, requests reliability, or disconnects from the daemon, the daemon recalculates the reliability interval for the corresponding service, by selecting the largest value of all transports communicating on that service.

When recomputing the reliability interval would result in a shorter effective retention time, the daemon delays using the new value until after an interval equivalent to the older (longer) retention time. This delay ensures that the daemon retains message data at least as long as the effective reliability interval at the time the message is sent.

Disabling Multicast

When the command line for any daemon includes `-no-multicast`, the daemon disables all UDP traffic to and from the affected daemon (multicast, broadcast and point-to-point). This section describes the behavior of the daemons (`rvd`, `rvrd`, `rvsd`, `rvsrd`) when multicasting is disabled.

API

All changes in behavior occur within the daemon. These behavior changes are transparent to Rendezvous API calls. Client programs can create transports that specify multicast addressing service, send messages to any subjects, and listen to any subjects. No changes to client programs are required.

Daemon Behavior

Disabling multicast communication changes daemon behavior in these ways:

- When a client sends a message to a public subject, the daemon does not multicast it (nor broadcast it) to the network.
- When a *routing* daemon receives multicast or broadcast messages from the network, it does not forward them to other daemons within the local network.

When multicast communication is disabled, daemons continue to operate in these ways:

- All messages (including public subjects) flow among all the clients of the daemon.

  For example, in Disabling Multicast: Public Subjects Still Flow Among Local Clients, even though they connect to a daemon that has disabled multicast, clients A, B and C can still exchange public subjects among themselves. However, they can exchange only point-to-point messages with D, E and F (clients of another daemon).

- All messages (including public subjects) flow in both directions between local clients of a *routing* daemon and the daemon's neighbors.

  For example, in Disabling Multicast: Routing Daemons, J and K are local clients of a routing daemon that has disabled multicast. Nonetheless, they can exchange public subjects with transports on net Z. In contrast, L (a client of another daemon on the same network cannot exchange public subjects with transports on net Z, nor with J and K on net Y. (All clients on nets Y and Z can exchange point-to-point messages.)

  For another example, see rvsrd.

Figure 1: Disabling Multicast: Public Subjects Still Flow Among Local Clients

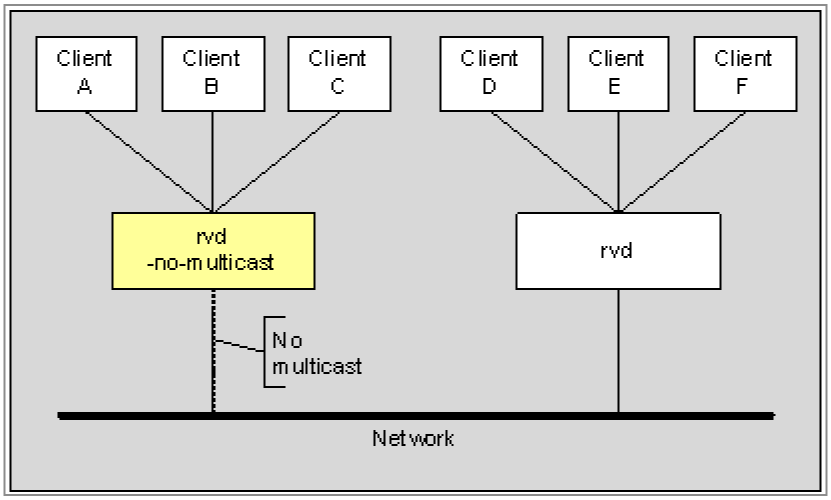
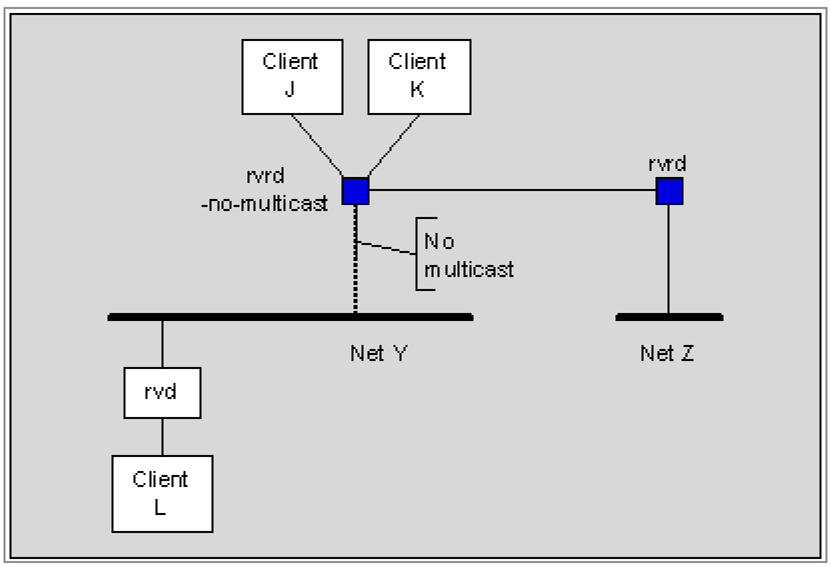


Figure 2: Disabling Multicast: Routing Daemons



Rendezvous Daemon (rvd)

The Rendezvous daemon (`rvd`) is the background process that supports all Rendezvous communications. Distributed processes depend on it for reliable and efficient network communication. All information that travels between and among processes passes through a Rendezvous daemon when it enters or exits a host computer.

The Rendezvous daemon fills these roles:

- Route messages to program processes.
- Deliver messages reliably.
- Filter subject-addressed messages.
- Shield programs from operating system idiosyncrasies, such as low-level sockets and file descriptor limits.

The Rendezvous daemon process, `rvd`, starts automatically when needed, runs continuously and may exit after a period of inactivity.

For further general information about the Rendezvous daemon and reliable broadcast delivery, see Rendezvous Daemon in TIBCO Rendezvous Concepts.

rvd

Command

Syntax

```
rvd     [-http [ip_address:]http_port]
        [-no-http]
        [-license url]
        [-listen [socket_protocol:|ip_address:]tcp_port]
        [-no-permanent]
        [-no-lead-wc | -lead-wc]
        [-no-multicast]
        [-reliability time]
```

```
        [-max-consumer-buffer size]
        [-rxc-max-loss loss]
        [-rxc-recv-threshold bps]
        [-rxc-send-threshold bps]
        [-reuse-port inbox_port]
        [-logfile log_filename]
        [-log-max-size size]
        [-log-max-rotations n]
        [-foreground]
        [-udp-buffer-size size]
        [-udp-ttl hops]
        [-transport-batch-size size]
        [-tls-min-proto-version version]
        [-tls-max-proto-version version]
        [-tls-ciphers string1:string2:stringN]
        [-tls-ciphersuites name1:name2:nameN]
        [-no-wc]
```

Purpose

The command `rvd` starts the Rendezvous daemon process. The Rendezvous daemon is the network I/O handler for all Rendezvous programs on a computer.

Remarks

Usually, the Rendezvous daemon (`rvd`) process starts automatically. When a Rendezvous program creates a transport, Rendezvous software determines whether a daemon is already listening for connections (as specified by the `daemon` parameter). If so, the new transport connects to that daemon. If not, it automatically starts a new daemon and connects to it.

However, when the `daemon` parameter of the transport creation call specifies a remote daemon, the daemon does not start automatically—you must start it manually on the remote computer.

The `rvd` command starts the Rendezvous daemon manually. You might do this to specify optional parameters, or a start a daemon that will accept connections from programs running on remote computers.

When started automatically by a client, `rvd` can also exit automatically. If `rvd` is not connected to any valid client transports for 1 minute, then `rvd` automatically exits. However, when started manually, `rvd` does not exit automatically. To override this behavior, start it manually with the `-no-permanent` option.

The Rendezvous routing daemon (`rvrd`) subsumes the behavior of `rvd`, so it is usually not necessary to run `rvd` on computers that already run `rvrd`.

IPM

TIBCO Rendezvous® Server In-Process Module (IPM) uses many of the same parameters as `rvd`, with parallel behavior. The table of parameters below notes exceptions to this rule.

Command Line Parameters

Parameter	Description
`-http` *ip_address*`:`*http_port* `-http` *http_port*	The browser administration interface accepts connections on this HTTP port. Permit administration access only through the network interface specified by this IP address. To limit access to a browser on the `rvd` host computer, specify `127.0.0.1` (the local host address). When the IP address is absent, the daemon accepts connections through any network interface on the specified HTTP port. If the explicitly specified port is already occupied, the program exits. When this parameter is entirely absent, the default behavior is to accept connections from any computer on HTTP port `7580`. If this default port is unavailable, the operating system assigns an ephemeral port number. In all cases, the program prints the actual HTTP port where it accepts connections. This parameter is not available with IPM.
`-no-http`	Disable *all* HTTP connections, overriding `-http`. This parameter is not available with IPM.

Parameter	Description
`-license` *`url`*	The URL to a TIBCO Activation Service in the form of `https://url[?fp=activation service fingerprint]`. A fingerprint of the Activation Service can be included in the URL for peer verification. The URL can also be supplied using the environment variable `TIBRV_LICENSE`. If the URL is not explicitly configured using `-license` or `TIBRV_LICENSE`, the Rendezvous daemon will search its environment PATH for a license file named `tibrvlic.bin` and, failing that, will attempt to contact an Activation Service at `https://tib-activate:7070`. **Note:** Production environments must use a TIBCO Activation Service for TIBCO product activation. Using a license file to configure the Rendezvous daemon is only supported for development environments. The URL format for a license file is `file://pathname`.
`-listen` *tcp_port* `-listen` *ip_address*`:`*tcp_port* `-listen` *socket_protocol*`:`*tcp_port*	`rvd` (and by extension, `rvrd` operating within the local network) opens a TCP client socket to establish communication between itself and its client programs. The `-listen` parameter specifies the TCP port where the Rendezvous daemon listens for connection requests from client programs. This `-listen` parameter of `rvd` corresponds to the `daemon` parameter of the transport creation call (they must specify the same TCP port number). The IP address specifies the network interface through which this daemon accepts TCP connections. To bar connections from remote programs, specify IP address `127.0.0.1` (the loopback interface). When the IP address is absent, the daemon accepts connections from any computer on the specified TCP port. When this parameter is entirely absent, the default

Parameter	Description
	behavior is to accept connections from any computer on TCP port 7500. For more detail about the choreography that establishes conduits, see Daemon Client Socket—Establishing Connections. **Warning** This parameter does *not* correspond to the `service` parameter of the transport creation call—but rather to the `daemon` parameter. This parameter is not available with IPM.
`-no-permanent`	If present (or when `rvd` starts automatically), `rvd` exits after 1 minute during which no transports are connected to it. If not present, `rvd` runs indefinitely until terminated. This parameter is not available with IPM.
`-permanent`	This flag is deprecated in release 7.0 and later. To preserve backward compatibility with existing scripts, `rvd` ignores this flag, rather than rejecting it. This parameter is not available with IPM.
`-no-lead-wc` | `-lead-wc`	Sending to subjects with lead wildcards (for example, `>` or `*.foo`) can cause unexpected behavior in some applications, and cause network instability in some configurations. This option lets you selectively screen wildcard sending. When `-no-lead-wc` is present, the daemon quietly rejects client requests to send outbound messages to subjects that contain wildcards in the lead element. The daemon does *not* report excluded messages as errors.

Parameter	Description
	When `-lead-wc` is present (or when neither flag is present), the daemon allows sending messages to subjects with lead wildcards. This parameter is not available with IPM.
`-no-multicast`	When present, the daemon disables multicast (and broadcast) communication. For details, see Disabling Multicast. This parameter is not available with IPM.
`-reliability` *time*	Rendezvous daemons compensate for brief network failures by retaining outbound messages, and retransmitting them upon request. This parameter is one of several ways to control the message reliability interval. For a complete discussion the concept of reliability, the various ways to control it, the interaction among those ways, and reasonable values, see Reliability and Message Retention Time. If this parameter is absent, `rvd` uses the factory default (60 seconds). If this parameter is present, `rvd` (and by extension, `rvrd` operating within the local network) retains messages for *time* (in seconds). The value must be a non-negative integer.
`-max-consumer-buffer` *size*	When present, the daemon enforces this upper bound (in bytes) on each consumer buffer (the queue of messages for a client transport). When data arrives faster than the client consumes it, the buffer overflows this size limit, and the daemon discards the oldest messages to make space for new messages. The client transport receives a `CLIENT.SLOWCONSUMER` advisory. When absent or zero, the daemon does not enforce a size limit on the consumer buffer. (However, a 60-

Parameter	Description
	second time limit on messages still limits buffer growth, independently of this parameter.) This parameter is not available with IPM.
`-rxc-max-loss` *loss* `-rxc-recv-threshold` *bps* `-rxc-send-threshold` *bps*	These three parameters configure the retransmission control (RXC) feature, which suppresses retransmission requests from chronically-lossy receivers. If `-rxc-max-loss` is absent or zero, then RXC is disabled. If it is an integer in the range `[1,100]`, it determines the maximum percentage acceptable loss rates above which a receiver is considered chronically-lossy. `-rxc-recv-threshold` configures the threshold receive rate (in bits per second) above which a chronically-lossy receiver censors its own retransmission requests. When absent, the default value is zero (always censor a chronically-lossy receiver). `-rxc-send-threshold` configures the threshold send rate (in bits per second) above which the daemon suppresses (that is, ignores requests from) chronically-lossy receivers. When absent, the default value is zero (always suppress retransmissions to chronically-lossy receivers). For a complete explanation, see Retransmission Control .
`-reuse-port` *inbox_port*	When present, other daemons on the same host computer can reuse service ports. When absent, other daemons cannot reuse a service port that is in use by this daemon. For correct operation, all the daemons that use a common service port on a host computer must specify this option. For background and details, see Reusing Service Ports.

Parameter	Description
	The *inbox_port* argument (required) specifies the UDP port that this daemon uses for point-to-point communications. This value must be unique for each daemon (that reuses service ports) on a common host computer. Furthermore, you must not use the *inbox_port* in any transport specification on the same host computer.
`-logfile` *log_filename*	Send log output to this file. When absent, the default is `stderr`.
`-log-max-size` *size* `-log-max-rotations` *n*	When present, activate the log rotation regimen (see Log Rotation). When you specify these options, you must also specify `-logfile`. *size* is in kilobytes. If *size* is non-zero, it must be in the range [100, 2097152]. Values outside this range are automatically adjusted to the nearest acceptable value. Zero is a special value, which disables log rotation. When `-log-max-size` is zero or absent, a single log file may grow without limit (other than the limit of available storage). *n* indicates the maximum number of files in the rotation. When `-log-max-rotations` is absent, the default value is 10.
`-foreground`	Available only on UNIX platforms. When present, `rvd` runs as a foreground process. When absent, `rvd` runs as a background process. This parameter is not available with IPM.
`-udp-buffer-size` *size*	UDP Buffer Size

Parameter	Description
	When present, the daemon requests buffers of this *size* (in bytes) for inbound and outbound UDP multicast. (Operating system constraints can limit this request.) The value of *size* must be a non-negative integer. When absent or zero, the daemon requests the default buffer size, 16MB (16*1024*1024 bytes). In most situations we recommend the default buffer size. In some situations larger outbound buffers can yield higher throughput, at the cost of longer latency (waiting for the operating system to flush the buffer). You can use rvlat to empirically test the effect on latency. In some situations larger inbound buffers can reduce the probability that the operating system cannot write packets into a full buffer (such events trigger retransmission requests, which increase network bandwidth usage).
`-udp-ttl` *hops*	UDP TTL When present, the daemon sends UDP packets with a TTL value of *hops* (a positive integer, less than or equal to 256). When absent, the default TTL is 16 hops.
`-transport-batch-size` *size*	IPM Transport Batch Size When present, enable outbound batching of data from IPM, and set the batch size (in bytes). When the batch size is greater than zero, IPM transfers data to the network in batches. This option can increase throughput, at the cost of higher latency. When absent, the batch size is zero, and IPM transfers data to the network immediately, for lowest latency.

Parameter	Description
	This parameter is available *only* with IPM.
`-tls-min-proto-version` *version* `-tls-max-proto-version` *version*	Set the minimum or maximum supported protocol versions for the ctx using OpenSSL calls `SSL_CTX_set_min_proto_version` and `SSL_CTX_set_max_proto_version`.
`-tls-ciphers` *string1:string2:stringN*	Set the list of available ciphers (TLSv1.2 and earlier) using OpenSSL call `SSL_CTX_set_cipher_list`.
`-tls-ciphersuites` *name1:name2:nameN*	Configure the available TLSv1.3 ciphersuites using OpenSSL call `SSL_CTX_set_ciphersuites`.
`-no-wc`	Silently drop any messages published by clients that contain any wildcard tokens.

Utility Scripts

You can create utilities to start Rendezvous daemons with specific command line arguments. For models, see the sample `rvd` scripts (or the sample Windows program `rvd.c`) in the Rendezvous subdirectory `src/examples/utilities/`.

You can use such utilities to customize daemon behavior; for example, your utilities can select 64-bit daemons, or specify log file parameters or reliability parameters.

Retransmission Control

Note

The retransmission control feature addresses the issue of excessive retransmissions, which can occur in some deployments. Unless your deployment exhibits this specific behavior, this feature can degrade performance. We discourage use of this feature except in consultation with a TIBCO professional.

Motivation

When a receiving daemon consistently misses many packets, we categorize it as a *chronically-lossy receiver*. This condition could indicate a host computer that is slower than other computers in the network; an overloaded host computer; a hardware problem with the NIC, connectors or cables; mismatched NIC capacity; or a network problem involving routing or switching hardware.

The effects of a chronically-lossy receiver can include wasted network bandwidth, wasted CPU resources, and decreased performance for the entire distributed application system. When a sender retransmits excessively, its data send rate can increase dramatically, which in turn can exhaust network capacity.

Retransmission control (RXC) is a feature of the daemon that can help ameliorate the adverse effects of chronically-lossy receivers, conserve network and CPU resources, and locate problem hosts. When RXC is enabled, it lets sending daemons suppress retransmissions to chronically-lossy receivers, and it lets chronically-lossy receiving daemons censor their own retransmission requests.

Identifying Excessive Loss

When RXC is enabled for a receiving daemon, the daemon tracks inbound packet loss for each service. When RXC is enabled for a sending daemon, the daemon tracks packet loss and retransmission statistics for each receiving daemon and service. To distinguish chronic loss from temporary loss, the daemons compute a set of related measurements that pinpoint receivers for which retransmission is an ineffective solution.

Max Loss

When starting a daemon, an administrator can set the `-rxc-max-loss` command line parameter (a percentage, expressed as an integer between zero and 100, inclusive). If greater than zero, then RXC is enabled; the sending daemon measures loss statistics, and compares them against the configured threshold, and against other thresholds derived from it. Using several metrics lets the daemon distinguish between temporary loss and chronic loss. If measured rates exceed their corresponding threshold values, then a receiver is categorized as chronically-lossy.

Remediation at Sender

When a sender identifies a chronically-lossy receiver, it can suppress retransmission to that receiver, with two main effects:

- The sending daemon ignores retransmission requests from that receiver; that is, the daemon does not retransmit the requested data.
- The sending daemon produces an `INFO` advisory, indicating the chronically-lossy receiver. For details, see `RETRANSMISSION.OUTBOUND.SUPPRESSED` in TIBCO Rendezvous Concepts.

Remediation at Receiver

When a receiver identifies itself as a chronically-lossy receiver, it can censor its own retransmission requests with two main effects:

- The receiving daemon does not send retransmission requests to the network.
- The receiving daemon produces an `INFO` advisory, indicating that it is a chronically-lossy receiver. For details, see `RETRANSMISSION.INBOUND.REQUEST_NOT_SENT` in TIBCO Rendezvous Concepts.

Bandwidth Usage & Thresholds

In some deployments it might be acceptable to retransmit to chronically-lossy receivers while both the receiver's and the sender's bandwidth usage remain low. To configure these thresholds, set the command line parameters `-rxc-recv-threshold` and `-rxc-send-threshold`.

Send

For each service, the receiving daemon measures its inbound bandwidth usage (in bits per second). The daemon does not censor retransmission requests until its inbound bandwidth usage exceeds `-rxc-recv-threshold`.

Receive

For each service, the sending daemon measures its outbound bandwidth usage (in bits per second). The daemon does not suppress retransmissions to chronically-lossy receivers until its outbound bandwidth usage exceeds `-rxc-send-threshold`.

The default value of both threshold parameters is zero, a special value specifying that the daemons *always* suppress retransmissions and requests whenever RXC is enabled and chronically-lossy receivers are identified.

Other Details

Daemons compute all statistics separately for each service (see Service Selection).

Routing daemons include `rvd` functionality; in this capacity, RXC *does* apply to routing daemons. However, when a routing daemon forwards messages to another routing daemon, it uses TCP protocols, and RXC does not apply.

Reusing Service Ports

In Rendezvous release 8.1.x and earlier, a service port was available only to the first daemon (on a particular host computer) that bound it. That is, client transports would fail when requesting that same service from another daemon on the same host computer.

In release 8.2 and later, you can allow daemons to reuse service ports that are already bound by another daemon (or IPM) on the same host computer. The daemon parameter `-reuse-port inbox_port` `enables this feature.`

Note	HP Tru64 UNIX does not support this feature.

Motivation

This section presents two situations in which reusing a service port would be advantageous.

IPM

Application processes that communicate using IPM can use this feature. When several such processes must communicate on the same service, and run on one host computer, then they necessarily reuse that service, because each such process acts as its own daemon. To allow this reuse, each IPM must specify -reuse-port inbox_port.

Enabling Service Reuse

To enable a set of daemons on the same host computer to reuse service ports, you must explicitly enable this feature on all the daemons in the set (otherwise the behavior is undefined).

Inbox Port

While daemons on the same host can reuse service ports for broadcast or multicast messages, they cannot reuse ports for point-to-point (`_INBOX`) messages. For each daemon and IPM on that common host, you must designate a unique UDP port to carry point-to-point communications. (That is, you must not assign the same inbox port number to two daemons on the same host.) Supply that port as the *inbox_port* argument to the -reuse-port inbox_port option.

Migration from Earlier Releases

Warning

Before using this feature, you must first upgrade all the interoperating daemons and IPM libraries to release 8.2 or later.

Furthermore, all application programs that connect to daemons that actually reuse service ports must relink to Rendezvous libraries from release 8.2 or later.

Incorrect migration can result in dataloss.

Reusing Service Ports in Routing Daemons

Release 8.2 updates `rvrd` (and `rvsrd`) with the ability to reuse service ports, and to correctly interoperate with daemons that reuse service ports. However, note the following restriction:

Warning

Two routing daemons on the same host cannot serve the same local network.

If two routing daemons violate this restriction, the second router to start detects the conflict, reports a configuration error, and could exit (depending on other parameters).

In contrast:

- Two routing daemons can serve the same local network, if the routing daemons run on different host computers.
- Two routing daemons can run on the same host computer, if they do not serve any local networks in common.

Log Destination

Each Rendezvous daemon and component process—`rvd`, `rvrd`, `rvsd`, `rvsrd`, `rvtrace`—produces log output. The content of log output varies, but the semantics of command line options that affect logging are identical for all of these components:

- When all of the command line options that affect logging are absent, daemons send log output to `stderr`.
- When `-logfile` *log_filename* is present, daemons send log output to the file you specify, namely *log_filename*.
- When `-log-max-size` *size* is present and non-zero, daemons use a log rotation regimen. For details, see Log Rotation below. The parameter `-log-max-rotations` *n* determines the number files in the rotation.
- When `-log-config` *config_log_filename* is present, daemons log duplicate copies of configuration changes to the file you specify, namely *config_log_filename*. Daemons never rotate nor remove this file, so a permanent record of this important information remains. (This parameter is available *only* for `rvrd`, `rvsd` and `rvsrd`.)

Log Rotation

To enable log rotation, you must specify a non-zero value for `-log-max-size`. (When absent, the default value is zero.)

The command line option `-log-max-size` *size* limits the growth of log files. The *size* parameter specifies the maximum disk space (in kilobytes) that a log file can occupy (approximately) before it is rotated.

The command line option `-log-max-rotations` *n* specifies the number of log files in the rotation sequence. Notice that the storage devoted to log files can grow at most to approximately *size* `* (`*n*`+1)`.

The daemon rotates the log files according to this renaming plan:

- The parameter `-logfile` *log_filename* specifies the name of the *current* log file, which receives log output. For example, for `rvd` one might specify `rvd.log` (without any numerical suffix). This name also becomes the base for a sequence of rotation files.
- When the current log file reaches its maximum size, the daemon rotates log files:
  - — It closes the current log file (in our example, `rvd.log`).
  - — It appends the next available numerical suffix to the base name, and renames the full log file with that name; for example, `rvd.log1`. Suffixes begin with `1`, and continue through *n* before rotating back to 1. This renaming operation overwrites any existing file from a previous rotation.
  - — It opens a new current log file (once again, `rvd.log`).
- When the daemon terminates and restarts, logging resumes by appending to the current log file, but the rotation state is reset (that is, the first rotation overwrites the file with suffix `1`).

  You can determine the most recent file by comparing either packet time stamps within the files, or file modification times.

Log Rotation Backward Compatibility

The command line option `-log-rotate` *total_size* is deprecated in release 7.5, and will become obsolete in a subsequent release. We recommend migrating to the new log rotation parameters at your earliest convenience.

In the meantime, we preserve backward compatibility by converting the value of this deprecated parameter to corresponding values of the new parameters:

- `-log-rotate` *total_size* retains its old meaning—specifying the total size for all log files. The maximum size for each individual file in the rotation is *total_size*`/10`.
- If both old and new parameters are present, the new parameters take precedence (overriding the old parameter).

Current Log Page

The browser interfaces for all daemon components include a **Current Log** page, which displays the most recent 4 kilobytes of log output (for convenience).

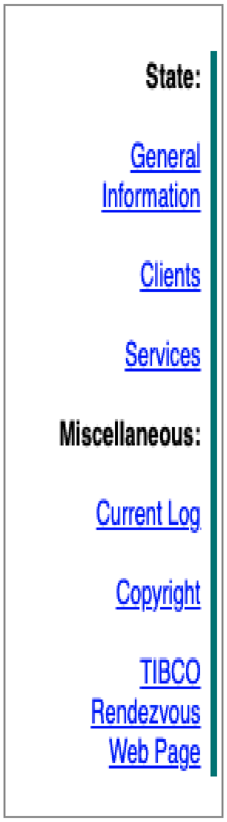
Browser Administration Interface — rvd

The browser administration interface lets you control `rvd` from a web browser. Although `rvd` does not have any configurable operating parameters, you can view internal data structures that reflect network conditions.

Navigation

All browser administration interface pages display a navigation panel at the left side of the page. Use these links to display other pages.

Figure 3: rvd Navigation Panel



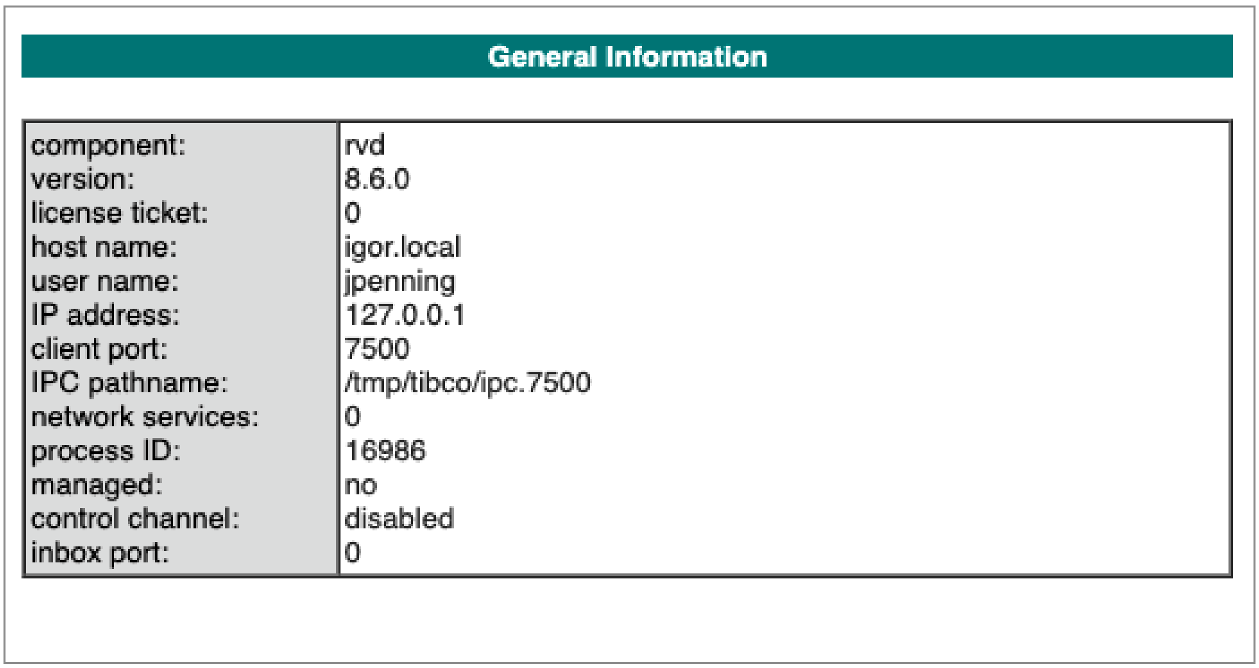
Category	Item	Description
State	General Information	This page displays information about an `rvd` process; see General Information.
	Clients	This page summarizes the client transports; see Clients.
	Services	This page summarizes network services activity; see Services.
Miscellaneous	Current Log	This page displays the most recent 4 kilobytes from the log file.
	TIBCO Rendezvous Web Page	The product page from the TIBCO web site.

Category	Item	Description
	Copyright	The Rendezvous copyright page.

General Information

`rvd` (like all Rendezvous components) displays information about itself on this page.

To display this page, click **General Information** in the left margin of any page of the `rvd` browser administration interface.


General Information

component:	rvd
version:	8.6.0
license ticket:	0
host name:	igor.local
user name:	jpenning
IP address:	127.0.0.1
client port:	7500
IPC pathname:	/tmp/tibco/ipc.7500
network services:	0
process ID:	16986
managed:	no
control channel:	disabled
inbox port:	0

Item	Description
component	The name of the program—`rvd` (or `rvsd`).
version	Version number of the program.
license ticket	Not applicable.
host name	The hostname of the computer where the daemon process runs.

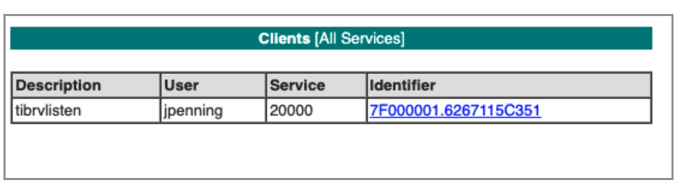
Item	Description
	Notice that the daemon process can run on one computer, while you access its browser interface from another computer.
username	The user who started the daemon process.
IP address	The IP address of the computer where the daemon process runs.
client port	The TCP port where the daemon listens for client connections.
network services	The number of network services on which this daemon's clients communicate.
process ID	The operating system's process ID number for the component.
managed	Not applicable
control channel	Not applicable.
inbox port	When the daemon reuses service ports, this field displays the unique inbox port. When the daemon does not reuse service ports, this field displays zero.

Clients

`rvd` displays information about its clients on this page.

To display this page, click **Clients** in the left margin of any page of the `rvd` browser administration interface.

Figure 4: rvd Clients Page



Item	Description
table rows	Each row of the table describes one client transport.
Description	The description string of the transport object. Client programs set this string using an API call.
User	The username of the user that started the client program process.
Service	The UDP service on which the client transport communicates.
Identifier	A globally unique identifier for the transport object. Click this identifier to view Client Information page.

Client Information

This page displays additional detail about a particular client transport.

To display this page, click any transport identifier in the Clients page.

Figure 5: rvd Client Information Page

Item	Description
Description	The description string of the transport object. Client programs set this string using an API call.
User	The username string of the user that started the client program process.
Service	The UDP service on which the client transport communicates.
Original Service	Not applicable.
Host	IP address of the client's host computer.
Port	TCP port number that the daemon uses to communicate with this client.
Pid	Process ID of the client (on its host computer). (This information requires that both client and daemon be of release 7.5 or later.)
Serial Number	Not applicable
Expiration	Not applicable.
Identifier	A globally unique identifier for the transport object.
Version	Version number of the Rendezvous API library that this client uses.
Subscriptions	Number of subscriptions that this client transport has registered with `rvd`. Click this link to view a list of the subscription subjects; see Subscription List.

Subscription List

This page displays additional detail about the subscriptions of a particular client transport. Each row displays the subject name of one subscription.

To display this page, click **Clients** in the Client Information page.

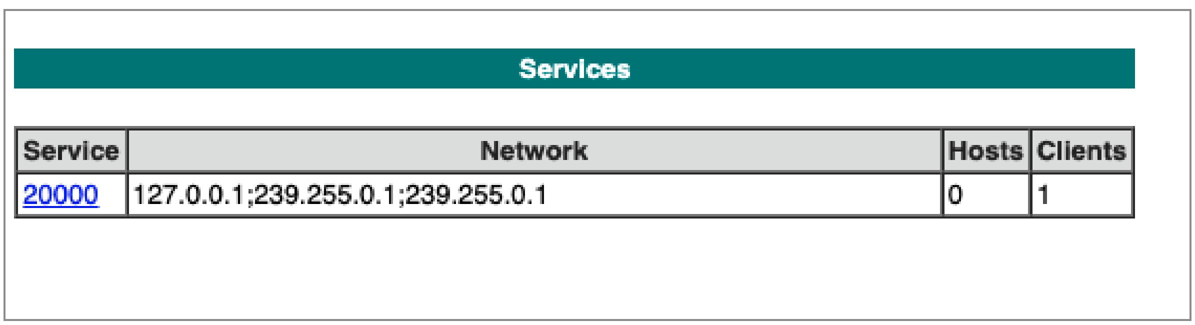
Figure 6: rvd Subscription List Page

Services

On this page `rvd` displays information about the network services it mediates for its clients.

To display this page, click **Services** in the left margin of any page of the `rvd` browser administration interface.

Figure 7: rvd Services Page



Item	Description
table rows	Each row of the table describes one network service—that is, a UDP service on a particular network interface.
Service	The UDP service number. Clicking this number displays the Service Information page.
Network	The network number or multicast specification.

Item	Description
Hosts	The number of *other* host computers with Rendezvous daemons that communicate on this network and service.
Clients	The number of client transports that use this network and service.

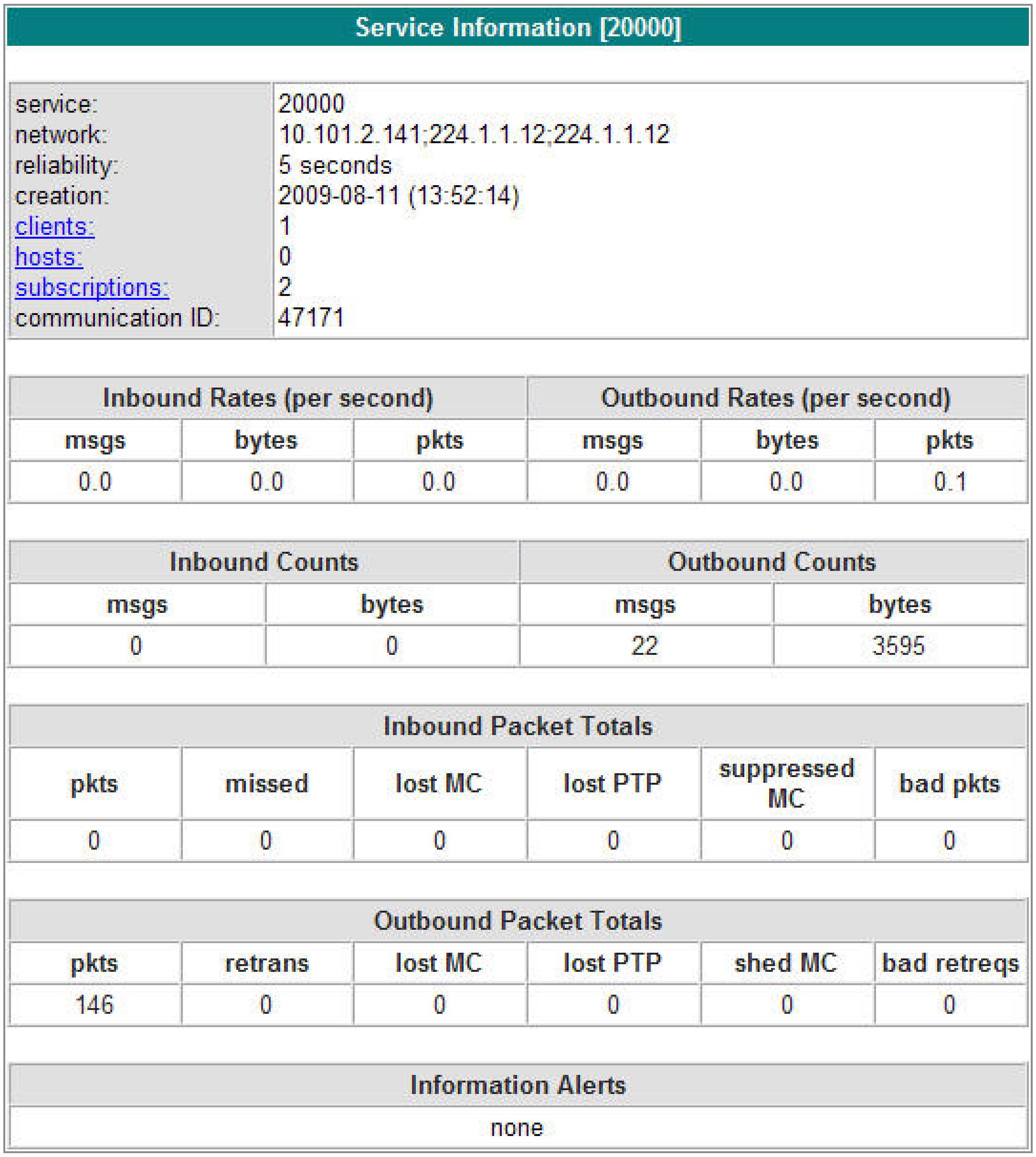
Service Information

Service

This page (see rvd Service Information Page) displays additional detail and operational statistics about a particular network service.

To display this page, click any service number in the Services page.

Figure 8: rvd Service Information Page



Service Information [20000]	
service:	20000
network:	10.101.2.141;224.1.1.12;224.1.1.12
reliability:	5 seconds
creation:	2009-08-11 (13:52:14)
clients:	1
hosts:	0
subscriptions:	2
communication ID:	47171

Inbound Rates (per second)			Outbound Rates (per second)		
msgs	bytes	pkts	msgs	bytes	pkts
0.0	0.0	0.0	0.0	0.0	0.1

Inbound Counts		Outbound Counts	
msgs	bytes	msgs	bytes
0	0	22	3595

Inbound Packet Totals					
pkts	missed	lost MC	lost PTP	suppressed MC	bad pkts
0	0	0	0	0	0

Outbound Packet Totals					
pkts	retrans	lost MC	lost PTP	shed MC	bad retreqs
146	0	0	0	0	0

Information Alerts
none

Item	Description
Service	The UDP service number.
Network	The network number or multicast specification.

Item	Description
Reliability	On this service, `rvd` retains outbound message data for retransmission. After this interval, it discards the data. For complete details, see Reliability and Message Retention Time.
Creation	Date and time that this service became active.
Clients	The number of client transports that use this network service. To view the Clients page, click this item.
Hosts	The number of *other* host computers with Rendezvous daemons that communicate on this network and service. To view the Host List page, click this item.
Subscriptions	The number of subscriptions registered with the daemon on this network service. To view the list of subscriptions, click this item.
Communication ID	Identifies the daemon and service. TIBCO support staff may request this value for diagnostic purposes.
Inbound Rates	The rate (per second) at which inbound messages, bytes and packets arrived on this network service during the most recent sampling period.
Outbound Rates	The rate (per second) at which the daemon sent outbound messages, bytes and packets on this network service during the most recent sampling period.
Inbound Counts	Cumulative statistics about inbound data messages; running totals since the start of the daemon process: • msgs—number of messages • bytes—number of bytes
Outbound Counts	Cumulative statistics about outbound data messages; running totals since the start of the daemon process: • msgs—number of messages • bytes—number of bytes

<table>
<tr><th>Item</th><th>Description</th></tr>
<tr><td>Inbound Packet Totals</td><td>Cumulative statistics about inbound packets; running totals since the start of the daemon process:
pkts—number of data packets
missed—number of missed packets (detected as a packet sequence gap)
lost MC—number of multicast packets lost because the sending daemon could not retransmit them
lost PTP—number of point-to-point packets lost because the sending daemon could not retransmit them
suppressed MC—number of multicast packets for which the daemon suppressed the sending of retransmission requests (see Retransmission Control)
bad pkts—number of unreadable data packets
These bad packets correspond to <code>DATALOSS</code> advisories in which the error description string includes the words <code>multicast destination</code>. For information about non-zero values see .</td></tr>
<tr><td>Outbound Packet Totals</td><td>Cumulative statistics about outbound packets; running totals since the start of the daemon process:
pkts—number of data packets
retrans—number of packets retransmitted (multicast and point-to-point)
lost MC—number of multicast packets the daemon could not retransmit (too old)
lost PTP—number of point-to-point packets the daemon could not retransmit (too old)
shed MC—number of multicast packets for which the daemon ignored retransmission requests from a chronically-lossy receiver, and did not retransmit the data (see Retransmission Control)
bad retreqs—number of unreadable retransmission request packets</td></tr>
</table>

Item	Description
	These bad packets correspond to `DATALOSS` advisories in which the error description string includes the words `multicast destination`. For information about non-zero values see .
Information Alerts	This panel displays the 50 most recent `DATALOSS` advisories.

Host List

This page displays Rendezvous daemon process instances on other host computers that communicate on the same network and service. From this page, you can view the service pages of those other daemons.

To display this page, click the word `hosts` in the Service Information page.

Note This page lists any Rendezvous communications daemon host, whether the process is `rvd`, `rvsd`, `rvrd`, or `rvsrd`.

Figure 9: rvd Host List Page


Host List [service 5239]

Hostname	IP address	Version	Serial	Uptime
bigdog.rv.tibco.com	10.101.2.35	7.0.8	1	03:03:02

Item	Description
host	Each row of this table represents one Rendezvous daemon process and its host computer.
Hostname	The name of the computer where the other daemon is running.
IP Address	The IP address of the computer where the other daemon is running.

Item	Description
Version	The version of the Rendezvous daemon on the other host.
Serial	Not applicable.
Uptime	The elapsed time that the daemon has been using the common UDP service.

Routing Daemon (rvrd)

Warning Proper configuration of `rvrd` is one of the more complex administration tasks, and it is critically important for enterprises that use Rendezvous routing daemons. We recommend that network administrators give special attention to this section.

See Also

Upgrading rvrd to a New Release

Routing Daemon Overview

Rendezvous daemons (`rvd`) deliver messages to programs on computers *within* a single network. Delivering messages *beyond* network boundaries requires an additional software component—Rendezvous routing daemons (`rvrd`).

Routing daemons efficiently connect Rendezvous programs on separate IP networks, so that messages flow between them as if on a single network. Communicating programs remain decoupled from internetwork addresses and other details.

The routing daemons forward Rendezvous messages between networks. When routing daemons are present, Rendezvous programs on one network can listen for subject names and receive messages from other networks *transparently*—neither the sending nor the receiving programs require any code changes. Administrators retain control over the subject names that can flow in or out of each network.

Situations

Use the routing daemon in situations where *one or more* of these conditions apply:

- Participating networks lie in distant geographic areas.

- Participating networks lie in nearby geographic areas, but are not connected by multicast routing hardware.
- Participating networks are separated by a firewall.
- Messages must traverse expensive or slow WAN links.

Subsumes Rendezvous Daemon

In addition to routing Rendezvous messages to and from other networks, a routing daemon process also serves its host computer as a Rendezvous daemon (`rvd`). It is not necessary to run a separate `rvd` on a computer that is already running an `rvrd` process.

Concepts

This section compares routing daemon *software* to a *hardware* router, using an extended analogy to introduce the operational concepts of Rendezvous routing daemons.

Goal

The goal of routing daemon software is to take Rendezvous *messages* from one network, and make them available on other networks. The effect is to connect a set of networks into a larger network.

Compare this goal to the goal of routing hardware—to take *packets* from one network, and make them available on other networks. Once again, the effect is to connect a set of networks into a larger network.

Connections

Routing daemon software uses a *routing table* to define *connections* to local networks, and to other routing daemons.

Compare this tool to a hardware router, which uses a routing table to define the connections between the router and its interfaces.

Each *entry* in the routing table describes one routing daemon and its connections. Although each routing daemon specifies only its own routing table entry, all the routing daemons in a WAN cooperate to share this information, so that every routing daemon builds a copy of the complete *global routing table*.

Local Network

A routing daemon *serves* a set of *local networks* by forwarding messages between those networks and other networks (usually, by way of other routing daemons).

While routing hardware specifies its local networks primarily in terms of network interfaces, routing daemon software specifies each local network as a pair combining network and UDP service. UDP services effectively divide the physical network into separate logical networks—even though they use the same hardware.

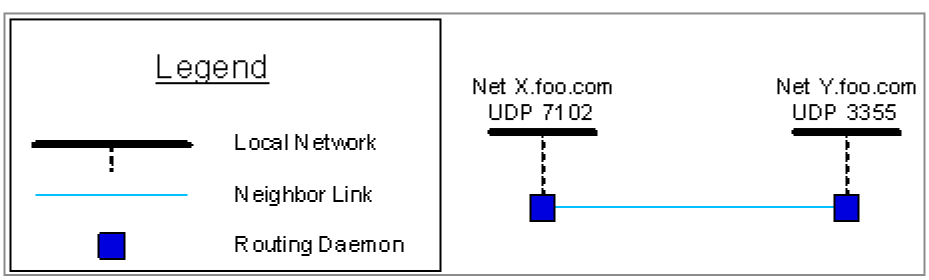
A routing daemon filters messages by subject name, restricting the subjects that its local networks can import and export. Filtering messages by subject in routing daemon software yields a finer granularity of control than filtering packets in a hardware router. Routing daemons control the set of subjects that each network can export to other networks, and import from other networks. For more information, see Subject Gating.

Neighbor

To achieve the goal of forwarding message between networks, routing daemons connect to other routing daemons. A routing daemon declares its potential *neighbors*—the other routing daemons to which it can directly connect.

Two *potential* neighbors become *actual* neighbors when they establish a TCP connection.

Figure 10: Routing Daemons



Route

The set of connections through which a message travels between its originating network and its destination network is called a *route*. Several potential routes can exist between the originating and destination networks; routing daemons select the actual route for each message.

Requirements

These four conditions enable delivery of Rendezvous messages between networks:

Routing Daemons

A routing daemon must exist on at least one computer of each local network that participates by sending or receiving Rendezvous messages.

Neighbor Connections

The network administrator must allow the routing daemons to establish TCP or TLS connections, so the routing daemons can become neighbors.

Subject Gating

Each routing daemon must *export* the relevant subject names from its local network, and *import* the relevant subject names from other networks.

For details, see Subject Gating, and Subject Filtering with Wildcards.

Subject Interest

Import and export gating is not sufficient to start the flow of messages. To receive forwarded messages, programs within the local network must express *interest* in the relevant subject names, by listening for those subjects.

Whenever a routing daemon detects interest in a subject within one of its local networks, it cooperates with other routing daemons to forward that subject to that local network. When programs in the network no longer retain interest in a subject, the routing daemons stop forwarding it.

For more details, see Routing Daemons Filter Interest to Permitted Subjects.

Restricting Message Flow

Routing daemons can be very selective in allowing messages to flow between networks. Network administrators can use this selectivity in several important ways:

- Restrict sensitive information to particular networks.
- Limit the volume of messages between networks.
- Constrain information to flow in only one direction between two networks.

Restricting Messages by Service or Port

For coarse-grained control over information flow, limit communication between networks to particular UDP services.

Recall that Rendezvous programs can segregate messages by specifying the `service` parameter to the transport creation function. The UDP service is part of the definition of a local network; the routing daemon exchanges with its neighbors only information that arrives on the designated service.

For example, if your organization adopts a convention to send sensitive information via particular UDP services, then you can use the routing daemon to regulate (or even completely disable) the import and export of messages sent via those services.

Restricting Messages by Subject Name

For fine-grained control over all the information flowing in or out of your networks, limit communication by subject name.

Subject names specify exactly which messages may enter and leave a local network—the routing daemon blocks all other Rendezvous messages. For details, see Subject Gating.

rvrd Process

Initial State

In its initial state an `rvrd` process operates identically to an `rvd` process; it does not route messages *yet*.

Administrators use the browser administration interface to configure the routing daemon, and to start its operation as a software router.

Administrative Store File

An `rvrd` process can store its routing table entry and parameters in a file. When the process restarts, it can read that file to resume its previous operating configuration.

The format of administrative store files is *not* human-readable. To examine or change the routing table entry or parameters, use the browser administration interface.

Important	Administrative store files require physical security safeguards and operating system protection. Store these files in a location that is accessible only to the system administrators who maintain it.

HTTP Administration Interface

You can configure `rvrd` using the browser administration interface. For more information, see Browser Administration Interface—rvrd, and the command line parameter `-http` for rvrd.

Logging

An `rvrd` process can output a log of its activity. For details, see Routing Daemon Logging.

Routing Table Entry

Each `rvrd` process specifies its routing table entry. For details, see Routing Table Entry.

Routing Table Entry

Routing table entries are the basic building blocks of a Rendezvous routing system. In most situations, each routing daemon process embodies a single routing table entry, which denotes that daemon throughout the WAN, and describes its operation.

In rare situations one routing daemon process can embody several routing table entries. Each entry defines a separate and independent software router, but without the cost associated with process switching. For more information, see Independent Routing Table Entries in One Process.

Combining all the routing table entries of all the routing daemons produces the global routing table. Each routing daemon uses its copy of the global routing table to forward messages efficiently to other routing daemons and their networks.

Router Name

Each routing table entry has a name. Routing daemons use these names to identify one another—so names must be unique throughout the entire WAN.

One convenient way to ensure unique names is to use the fully-qualified DNS names of the `rvrd` host computers; for example, `frobitz.yellowNet.baz.com`. (When one process embodies several routing table entries, you can use a prefix to create unique names; for example, `1.frobitz.yellowNet.baz.com`).

Other naming conventions are acceptable, as long as the names are unique.

The name is a string of alphanumeric, dot, and dash characters. The maximum total length of the string is 64 characters (including the dot separators).

Local Networks

Each routing daemon can serve zero or more local networks. For details, see Local Network.

Notice that a routing daemon need not serve any local networks. In this configuration, it operates as a *way station*, forwarding message traffic between other routing daemons—for example, to cross a firewall. For an illustration of this role, see Security and Firewalls.

Neighbors

Each routing daemon can connect to zero or more neighbors (routing daemons on other networks). For details, see Neighbors.

Local Network

Each routing daemon can serve zero or more local networks.

Network and Service

Two parameters together define the local network:

- UDP Service

  For details, see Specifying the UDP Service.

- Network Specification

  For details, see Constructing the Network Parameter.

Local Network Name

Each local network must have a *globally unique* network name.

One convenient way to generate globally unique network names is to concatenate the UDP service, the network specification, and the DNS domain name. For example, `7500.fooNet.baz.com` could refer to a local network using service 7500; in contrast, the name `7522.fooNet.baz.com` would refer to the local network using service 7522 on the same physical network.

Although that naming scheme is convenient, it can sometimes adversely affect network bandwidth use. Consider using shorter unique names in these situations:

- When WAN bandwidth is severely limited.

- When the average message is very small (smaller than 50 bytes).

Like router names, each local network name is a string of alphanumeric, dot, and dash characters. The maximum total length of the string is 64 characters (including the dot separators).

Warning

When several routing daemons serve one local network, each routing daemon *must* specify the same name for that network.

That is, if two local networks use the same physical network and the same service, then they are really the same local network. It is an error to refer to that local network with two or more different names.

Subject Gating

The router configuration determines the set of public subjects that can *potentially* pass between the routing daemon and the local network:

- *Export* subjects can flow out from the local network to the routing daemon, and from there to other networks.
- *Import* subjects can flow into the local network from the routing daemon.

Gating of System Subjects

As a rule, routing daemons do not forward Rendezvous system subjects (such as `_RV.>` and `_RVRD.>`). If you specify these subjects for export or import, the router ignores them.

However, system subjects required for feature operation (such as `_RVCM.>` and `_RVFT.>`) are the exception to this rule. Routing daemons *do* forward these subjects, and you can specify them for export and import.

Point-to-Point Gating

Routing daemons *automatically* transmit point-to-point messages as appropriate:

- When a routing daemon receives a point-to-point message whose destination is elsewhere in the global routing table, it forwards that message to the routing daemon that serves the destination network.

- When a routing daemon receives a point-to-point message whose destination is in one of its local networks, it forwards that message directly to `rvd` on the destination computer.
- Administrators do not need to explicitly import or export inbox subject names.

Subject Filtering with Wildcards

The wildcard characters, `*` and `>`, have the same semantics in import, export and exchange parameters as they do in listening calls:

- `*` matches any single element.
- `>` in the last (rightmost) position matches one or more trailing elements.

Recall that these rules of import parameter behavior apply also to routing daemons.

Importing Wildcard Subjects

Importing this wildcard name	Matches messages with names like these	But not names like these (reason)
`FOO.*`	`FOO.BAR`	`FOO.BAR.BAZ` (extra element)

`FOO.>`	`FOO.BAR`
	`FOO.BAR.BAZ`
	`FOO.BAR.BAZ.BOX`
		`FOO` (missing element)

`FOO.*.>`	`FOO.BAR.BAZ` `FOO.BAR.BAZ.BOX`	`FOO.BAR` (missing element) `FOO` (missing elements)
`FOO.*.STOP`	`FOO.BAR.STOP` `FOO.FOZ.STOP`	`FOO.STOP` (missing element) `FOO.BAR.BAZ` (unmatched 3rd element)

Routing Daemons Filter Interest to Permitted Subjects

Routing daemons filter local listening interest according to the subjects that the local networks can import and export. The general rule is that routing daemons disregard listening interest that would include subjects in either of these categories:

- Subjects that the listener's local network cannot import.
- Subjects that the sender's local network cannot export.

Note: Customers frequently deploy application programs that listen to wildcard subjects that are *more inclusive* than the wildcard subjects that `rvrd` imports or exports. As a result, the routing daemon filters this application subject interest, and the listeners do not receive any messages.

For example, consider a situation in which the local network imports `FOO.>` (that is, it does not permit any other subjects to enter from the WAN). When a process, `L1`, within the local network listens to the subject `>` (that is, the wildcard that matches any subject), the routing daemon first compares it to the permitted import subjects; since `>` is not a subset of `FOO.>`, the routing daemon does not forward any messages into the local network, so `L1` does not receive any messages.

When a second process, `L2`, in the same local network, listens to the subject `FOO.BAR`, the routing daemon begins importing messages (because the subject matches a subject for which import is permitted); both `L1` and `L2` receive the imported messages.

When a third process, `L3`, listens to the subject `FOO.>`, the routing daemon widens the set of messages it imports; both `L1` and `L3` receive the additional message subjects.

Correctly Importing Wildcard Subjects summarizes this example, and presents further examples.

Correctly Importing Wildcard Subjects

Importing this wildcard name	While listening to this subject	Imports these subjects	Reason
`FOO.>`	`>`	*none*	`>` is more inclusive than `FOO.>`

Importing this wildcard name	While listening to this subject	Imports these subjects	Reason
			`rvrd` filters out `>` because it isn't imported.
`FOO.>`	`FOO.BAR`	`FOO.BAR`	`FOO.BAR` is included within `FOO.>` `rvrd` filters out everything else because (for example, `FOO.BAZ`) no listener is requesting it.
`FOO.>`	`FOO.>`	`FOO.>`	`FOO.>` is identical to `FOO.>`
`A.B.C`	`A.*.C`	*none*	`A.*.C` is more inclusive than `A.B.C` `rvrd` filters out `A.*.C` because it isn't imported.
`A.*.C`	`A.*.C`	`A.*.C`	`A.*.C` is identical to `A.*.C`
`A.B.>`	`A.B.*`	`A.B.*`	`A.B.*` is included within `A.B.>` `rvrd` filters out everything else (for example, `A.B.C.D`) because no listener is requesting it.

See Also

Using Wildcards to Receive Related Subjects in TIBCO Rendezvous Concepts

Fixed Subject Interest

The concept of fixed subject interest is obsolete in release 6 (and later). Instead, subject interest dynamically determines the set of subjects that *actually* flow to and from a network.

Restriction on Local Networks

Two routing daemons on the same host cannot serve the same local network. For further explanation, see Reusing Service Ports in Routing Daemons.

Neighbors

Neighbor links connect routing daemons. A routing daemon declares its *potential* neighbors in its routing table entry. Two routing daemons become *actual* neighbors when they establish a TCP connection.

To declare potential neighbors, see Neighbor Interfaces. To examine actual neighbors, see Connected Neighbors.

Neighbor Pairs

Neighbors operate in pairs—one router at each end of a neighbor connection. Administrators can specify the pairs in four ways; see Adding Neighbor Interfaces.

Local Connection Information

These parameters specify the local end of a neighbor connection.

Local Host

The default value denotes the host computer’s default interface. You may override this default by specifying another network interface on the local host computer—either as a resolvable hostname, or as the IP address of the interface.

Local Connect Port

In each neighbor declaration, a routing daemon designates a TCP port number where the routing daemon accepts connection requests from that neighbor.

When a routing daemon declares several neighbors, it can designate a unique local connect port for each neighbor, or some of its neighbors can share a local connect port.

However, when a routing daemon process operates several routing table entries, the routing entries may *not* share any local connect ports.

Remote Connection Information

These parameters specify the remote end of a neighbor connection.

Remote Router Name

In most situations, a routing daemon identifies a neighbor using its unique router name (see Router Name).

(For a counterexample, see Seek Neighbor with Any Name.)

Remote Host

Specify the location of a neighbor either as a resolvable hostname, or as the IP address of the computer in a remote network where the neighboring daemon is running.

Remote Connect Port

The remote port is the TCP port number where the remote neighbor listens for a connection request from this routing daemon.

This parameter must match the local connect port of a routing table entry within the `rvrd` process on the neighboring host computer.

Remote Public Certificate

When neighbors communicate using TLS, you must enter the public certificate of the authorized neighbor. For background information, see Certificates and Security on page 29 in TIBCO Rendezvous Concepts.

Network Administration

Neighboring routing daemon processes must be able to establish a TCP connection. The network administrator (at each site) must configure the hardware (or software) routers and firewalls to permit this TCP connection between the two routing daemon host computers.

Data Compression

Routing daemons can compress data to reduce the network volume that travels between neighbors.

- Compression is most useful when you pay for WAN transmission by volume.
- Compression reduces volume at the cost of speed. Compression and decompression slows `rvrd` processing at both ends of a neighbor link.

To enable data compression, select an appropriate option on the neighbor interface forms of *both* neighbors; see Add New Neighbor Interface.

Adding Neighbor Interfaces

Routing daemons can declare neighbors in four ways:

- Active Neighbor
- Passive Neighbor
- Accept Any as Neighbor
- Seek Neighbor with Any Name

To specify a potential neighbor connection, see Add New Neighbor Interface.

Active Neighbor

A routing daemon can declare another routing daemon as its neighbor, and actively initiate a connection to it. If the connection is broken, the routing daemon actively attempts to restore it.

Consider an example situation in which a routing daemons link several networks within an enterprise. Each routing daemon within the enterprise declares every other routing daemon as an active neighbor.

To specify an active neighbor, you must supply this information:

- Remote Router Name
- Remote Host
- Remote Connect Port
- Local Connect Port

Passive Neighbor

A routing daemon can declare that it passively accepts connections from its neighbor, but does not actively initiate the connection itself.

Consider these example situations:

- Unidirectional firewall.

  Routing daemon `abc.homeNet.myDom.com` is located behind a firewall that allows connection requests in only one direction—outward. Active connection attempts by its neighbor, `mno.lyonNet.myDom.com`, would invariably fail, marking each attempt as a potential security violation at the firewall. When `mno.lyonNet.myDom.com` declares `abc.homeNet.myDom.com` as a neighbor, it can specify passive connect, reflecting its inability to initiate a connection to `abc.homeNet.myDom.com`. To become *actual* neighbors, `abc.homeNet.myDom.com` must initiate the connection to `mno.lyonNet.myDom.com`.

- Modem restriction.

  Routing daemon `fgh.oshkoshNet.myDom.com` is located on a host that depends on a modem for network access; the modem settings permit `fgh.oshkoshNet.myDom.com` to dial out, but the modem does not accept incoming calls. Active connection attempts by its neighbor, `klm.chicagoNet.myDom.com`, would invariably fail, while wasting resources. When `klm.chicagoNet.myDom.com` declares `fgh.oshkoshNet.myDom.com` as a neighbor, it can specify passive connect, reflecting its inability to initiate a connection to `fgh.oshkoshNet.myDom.com`. To become *actual* neighbors, `fgh.oshkoshNet.myDom.com` must initiate the connection to `klm.chicagoNet.myDom.com`.

Specifying Passive Neighbors

To specify a passive neighbor, you must supply this information:

- Remote Router Name
- Local Connect Port

Accept Any as Neighbor

Instead of declaring a specific set of neighbors, a routing daemon can declare that it accepts neighbor connections from any routing daemon.

This configuration is especially useful for hub topologies, dial-in connections, and any situation in which a routing daemon might operate with a large number of potential neighbors.

Specify the local connect port where this routing daemon accepts TCP connections from any (remote) routing daemon.

A routing daemon can simultaneously specify individual neighbors *and* declare that it accepts any other routing daemons as neighbors.

Seek Neighbor with Any Name

Instead of declaring a neighbor with a specific name, a routing daemon can seek out any available member from a set of routing daemons, without regard to its name.

This configuration is especially useful for load balancing among a set of potential neighbors with identical routes.

Specify the potential neighbors with two pieces of information:

- Remote Host, which must be either a *DNS* hostname that can resolve to more than one IP address, or a *virtual* IP address.

- Remote Connect Port—all potential neighbors must listen for connection requests on this port).

Each potential neighbor must accept connections from the seeking routing daemon, without actively attempting to connect to it. The potential neighbors can specify this in either of two ways:

- Accept connections from any neighbor, including the seeking routing daemon (see Accept Any as Neighbor).
- Passively accept connections specifically from the seeking routing daemon (see Passive Neighbor).

Redundant Routing Daemons for Fault Tolerance

Rendezvous routing daemons can cooperate for fault-tolerant service. Fault tolerance protects routing daemons against hardware failures, process failures and network segmentation.

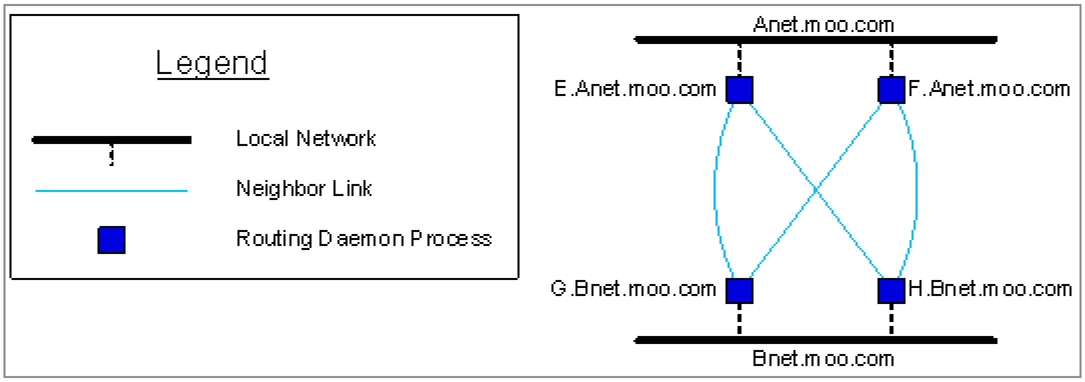
In Fault Tolerance among Routing Daemons, two routing daemon processes, `E.Anet.moo.com` and `F.Anet.moo.com`, run on separate host computers, and serve the local client network `Anet.moo.com`; similarly, routing daemons `G.Bnet.moo.com` and `H.Bnet.moo.com` both serve local client network `Bnet.moo.com`. Neighbor links connect `E` with `G` and `H`, and also `F` with `G` and `H`. Although these neighbor links offer redundant paths from `Anet` to `Bnet`, the routing daemons cooperate to forward each message only once. In failure situations, the routing daemons automatically readjust to continue service smoothly.

The concepts of primary and secondary do not apply to redundant routing daemons. Instead load balancing parameters govern fault-tolerant behavior (see Load Balancing).

In groups of redundant routers (such as `E` and `F`), the router names *must* be distinct, the local network configurations *must* be identical, while the load balancing parameters along neighbor links may differ.

Notice that `E` and `F` are not neighbors, nor are `G` and `H`. It would be an error for neighbors to serve the same local network (see Common Topology Errors).

Figure 11: Fault Tolerance among Routing Daemons



Load Balancing

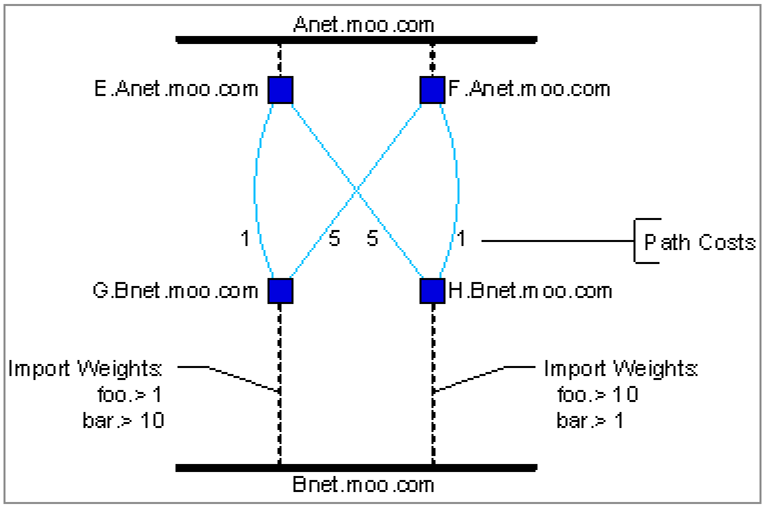
You can balance network load by directing messages along preferred routes. Routing daemons let you specify preferred routes using two cooperating mechanisms:

- **Path costs** determine a preference to route messages along specific neighbor links.
- **Subject import weights** determine a preference to import particular subjects into a network through a specific routing daemon.

Example

Path Cost and Subject Import Weight repeats the fault-tolerant configuration from Fault Tolerance among Routing Daemons—messages generally travel downward from `Anet` to `Bnet`. In this variation, the administrator specifies parameters to balance the load during normal operation—divide the message volume by subject, and direct messages along the outer links in preference to the crossover links. (In failure situations, messages continue to flow along the alternate routes.)

Figure 12: Path Cost and Subject Import Weight



- Path costs direct the message flow through the two outer links.
- Import weights split the traffic by subject:
  - Messages with subjects `bar.>` enter `Bnet` through routing daemon `G`.
  - Messages with subjects `foo.>` enter `Bnet` through routing daemon `H`.

Cooperating Mechanisms

Notice that effective load balancing depends on *both* mechanisms together. With path costs alone, all messages might flow only through `F` and `H`, while `E` and `G` have idle capacity. With subject import weights alone, all messages might flow only through `F`, while `E` has idle capacity. When both mechanisms cooperate, subject import weights divide the message volume between `G` and `H`, and path costs propagate that division back to `E` and `F`.

Path Cost

You can specify the path cost of each neighbor link. Routing protocols seek the route with the lowest cost.

For example, in Path Cost and Subject Import Weight, the outer links—between `E` and `G`, and between `F` and `H`—each specify a cost of 1. In contrast, the inner crossover links—between `F` and `G`, and between `E` and `H`—each specify a cost of 5. When all the components operate normally, messages flow across the lower cost (outer) links. When components fail, messages flow across the lowest cost link that remains operational.

Symmetric Path Costs

You must specify *symmetric* path costs. That is, if you specify a path cost of *n* at `G`'s neighbor link to `E`, the you must also specify the same path cost, *n*, at `E`'s neighbor link to `G`. This rule applies even when you intend that messages flow only in one direction (for example, from top to bottom in Path Cost and Subject Import Weight). Asymmetric path costs can result in unpredictable and inefficient routing behavior.

Backward Compatibility

For routing daemons from release 6, the cost of every path is always 1, and you cannot change this value. You can set a higher value for path costs only when configuring routers from release 7 or later.

See Also

To configure path cost between neighbors, see Neighbor Interfaces.

To configure path cost from a router instance to a local network, see Local Network Interfaces Configuration.

Subject Import Weight

You can specify weight values as annotations on import subject gating. When a message could travel two paths with equal cost, import weights break the tie. Routing protocols seek the path with the greatest weight.

For example, in Path Cost and Subject Import Weight, the administrator has specified that `G` imports `foo.>` with weight 1, and `bar.>` with weight 10. Conversely, `H` imports `foo.>` with weight 10, and `bar.>` with weight 1. When all the components operate properly, messages with subjects `foo.>` travel through `H` (which draws them through `F`), while messages with subjects `bar.>` travel through `G` (which draws them through `E`). If `E` were to fail, all messages would travel through `F` and `H` (because that route has the lowest path cost).

See Also

To configure subject import weight, see Subject Gating.

Border Routing

When `rvrd` is configured as a border router, then path cost and subject import weight affect only first-tier routing decisions—that is, routing within a zone. They do not affect second-tier routing decisions—that is, routing across a border between two zones. For background information about these concepts, see Border Routing.

Independent Routing Table Entries in One Process

In most situations, each routing daemon process embodies a single routing table entry. Nonetheless, in rare situations one routing daemon process can embody several routing table entries. Each entry defines a separate and independent software router, but without the cost associated with process switching.

This section explores two situations in which multiple routing table entries are appropriate:

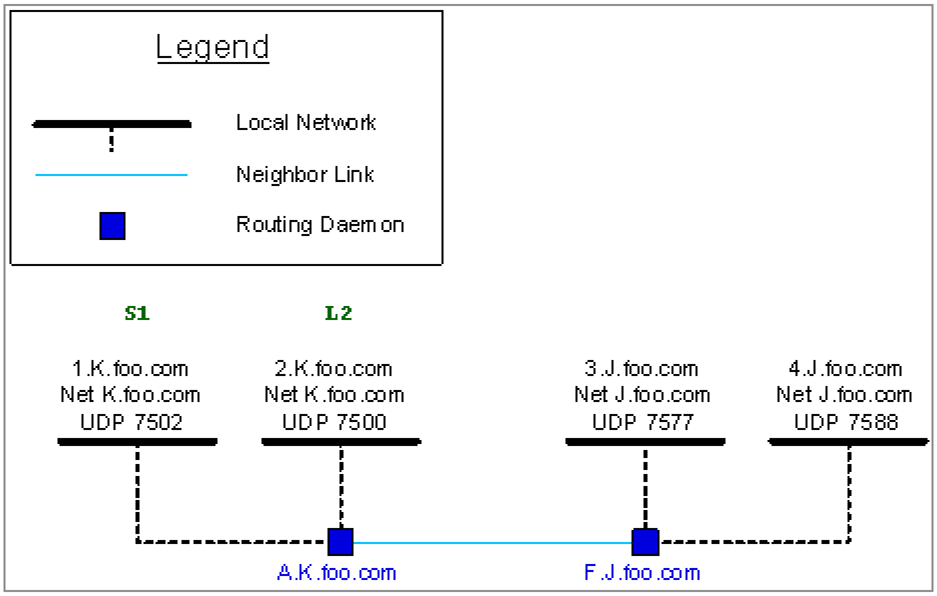
- Overlapping Subject Space
- Bandwidth Contention

Overlapping Subject Space

Consider two distinct distributed programs that use overlapping subject spaces—that is, they use some of the same subjects for their messages. When the two programs are deployed on the same physical network, each one receives messages from the other, which is inappropriate. To eliminate interference within the network, isolate each program to a separate UDP service.

Yet this solution within one network does not ordinarily keep the subject spaces separate when routing daemons connect two or more networks, because the routing daemon merges the subject spaces of its local networks.

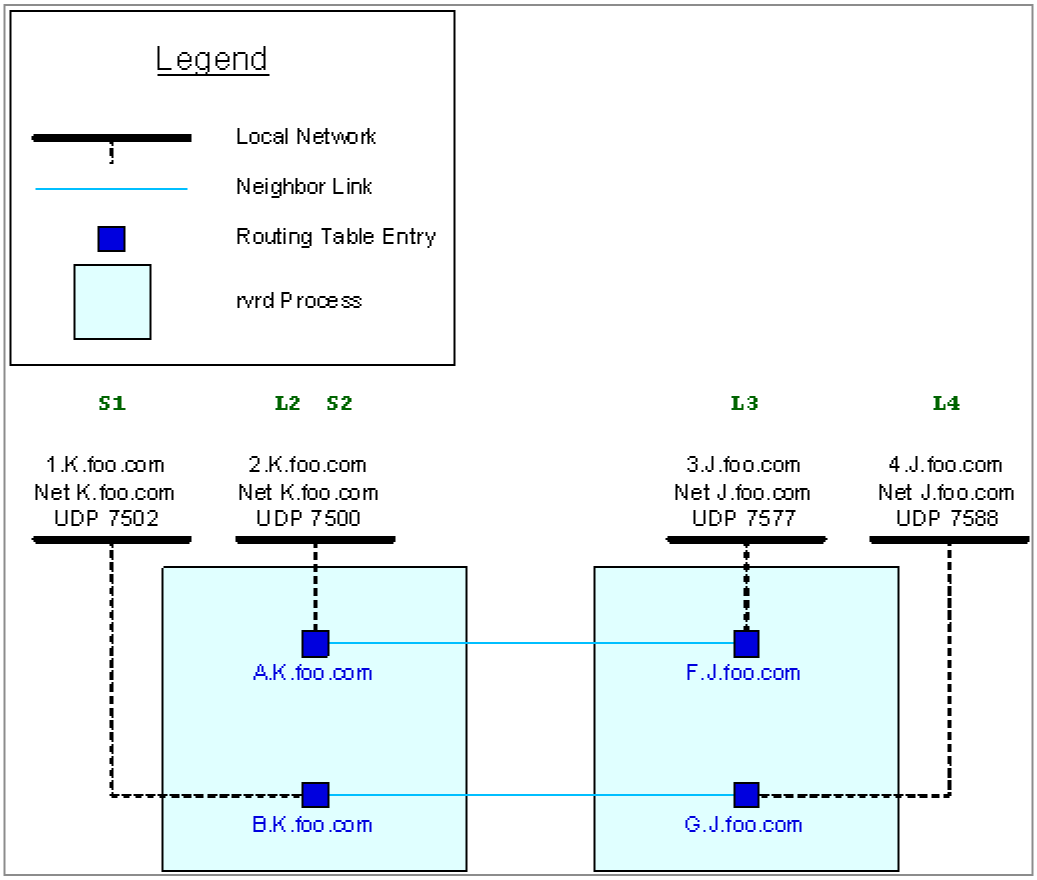
Figure 13: Routing Daemons Merge Subject Spaces



For example, on the left side of Routing Daemons Merge Subject Spaces, the two UDP services `7500` and `7502` effectively separate one physical network (`K.foo.com`) into two disjoint subject spaces; that is, program `L2` *cannot* receive messages from program `S1`. Similarly, on the right side of Routing Daemons Merge Subject Spaces, two UDP services `7577` and `7588` effectively separate one physical network (`J.foo.com`) into two disjoint subject spaces. However, the routing daemons in this configuration merge the subject spaces of their local networks—effectively canceling the separation; that is, program `L2` *does* receive messages from program `S1`.

To restore the separation, configure an independent routing table entry for each local network, as in Independent Routing Table Entries Keep Subject Spaces Separate.

Figure 14: Independent Routing Table Entries Keep Subject Spaces Separate



In Independent Routing Table Entries Keep Subject Spaces Separate, each `rvrd` process contains two independent routers, which act as parts of two disjoint routes—keeping the data and subject spaces separate:

- Routing table entries `A` and `F` form a route connecting network `2` with network `3`.
- Routing table entries `B` and `G` form a route connecting network `1` with network `4`.

Notice that once again, program `L2` *cannot* receive messages from `S1`.

Bandwidth Contention

Bandwidth contention is the second reason to separate programs using disjoint routes.

Consider two programs that are deployed on the same physical network—a program `S2` that sends messages at a moderate data rate, and a program `S1` that sends messages at a relatively high data rate. However, messages from S2 are much more important to the enterprise as a whole than messages from S1.

When forwarding these messages across a WAN, routing daemons would ordinarily send them across the same TCP connection. The many unimportant messages from S1 could delay the more important messages from S2.

To solve this throughput problem, configure an independent route for each set of messages, as in Independent Routing Table Entries Keep Subject Spaces Separate. On the left side of Independent Routing Table Entries Keep Subject Spaces Separate, S1 and S2 use distinct UDP services within the same physical network, effectively separating their messages into two logical network spaces. Disjoint routes carry the two sets of messages:

- Important messages from S2 travel through routing entries A and F.
- Messages from S1 travel through routing entries B and G.

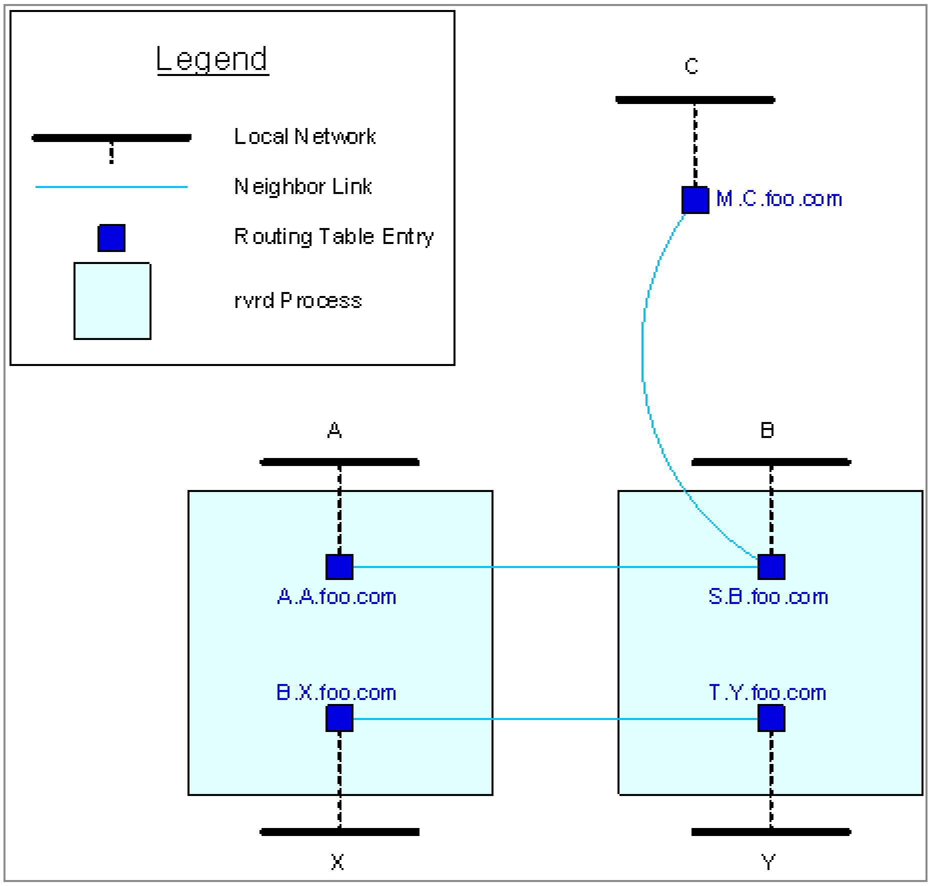
  The heavy volume on this route does not interfere with crucial message throughput on the S2 route, because a separate TCP connection carries each route.

Defeating Independence

The routing table entries within an `rvrd` process operate as independent pathways; that is, data does not flow directly between routing table entries within a routing daemon process instance.

Nonetheless, data can flow indirectly by way of a mutual neighbor. In Mutual Neighbors Merge Routes, notice that adding a neighbor link between M and T would merge the route connecting networks A, B and C, with the otherwise disjoint route connecting X and Y (defeating their independence). Use caution when altering a network of routing daemons.

Figure 15: Mutual Neighbors Merge Routes



Common Topology Errors

This section describes two variants of an erroneous routing configuration.

Neighbors on the Same Network

It is an error to configure two neighbors to serve the same logical local network (network and service).

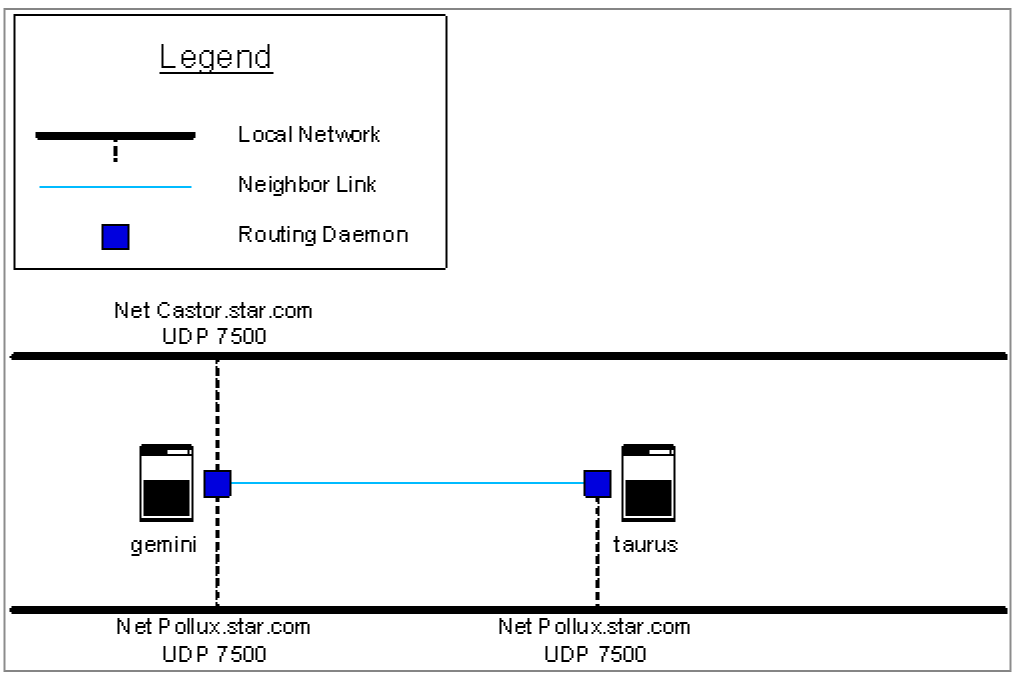
Since no gain could possibly result from forwarding messages from a network to the same network, it might seem that this error is rather rare. Nonetheless, in actual practice this error occurs rather frequently as an oversight.

Consider the situation in Erroneous Neighbors on the Same Network. In the desired outcome, neighbors on computers `gemini` and `taurus` exchange messages on UDP service `7500` between the two networks, `Castor.star.com` and `Pollux.star.com`. Because computer `gemini` has two network interfaces, the administrator attempts to limit `rvrd` operation to only `Castor.star.com`. Nonetheless, the routing daemon on `gemini` still receives messages from `Pollux.star.com` through its other interface (to understand the reason for this behavior, see Limitation on Computers with Multiple Network Interfaces). Because the two neighbors both serve the same network, `Pollux.star.com`, erroneous behavior results.

If `gemini` had only one network interface, `Castor.star.com`, the routing daemons would operate correctly.

When the routing daemon detects a topology error of this kind, it outputs an error message. Administrators must correct this situation immediately.

Figure 16: Erroneous Neighbors on the Same Network



Duplicating Effort

It is an error to use routing daemons to duplicate the effort of another forwarding mechanism (for example, a hardware router, or another pair of routing daemon neighbors. (This error is actually a variation of the error described in Neighbors on the Same Network.)

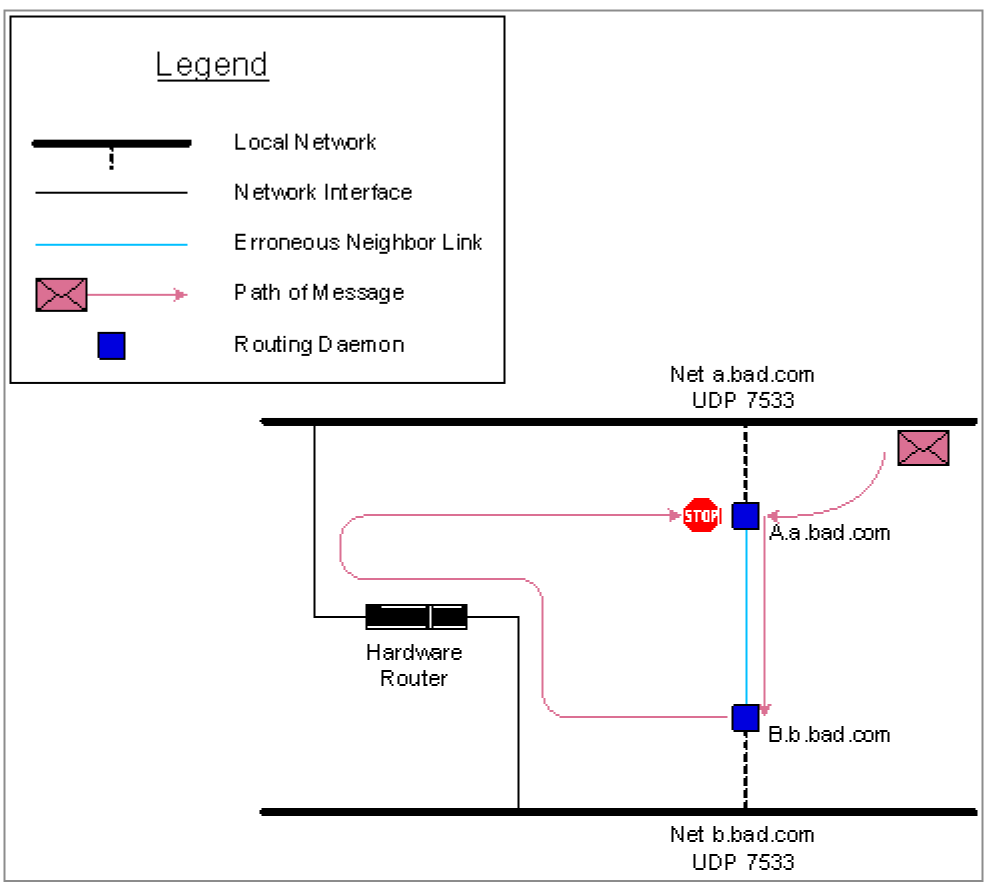
Consider the situation in Routing Daemons and Duplication. Two mechanisms forward messages between the two networks—the hardware router and a pair of routing daemons (`A.a.bad.com` and `B.b.bad.com`). When a program on network `a.bad.com` sends a message, routing daemon `A` forwards it to its neighbor `B`, which redistributes it on network `b.bad.com`. When the hardware router receives the redistributed message, it forwards it back to network `a.bad.com`, where `A` detects the duplication and produces an error message.

This kind of error can occur in either broadcast or multicast situations. However, it is especially common in environments where hardware routers enable multicast routing. Upgrading a hardware router can trigger this error.

Upgrading `rvrd` from release 5 to release 6 (or later) provides another fertile environment for this error. When both routing daemons run concurrently in the same network, be careful to avoid duplicate service.

To repair the situation, remove one of the routing daemons, or disable hardware multicast routing.

Figure 17: Routing Daemons and Duplication

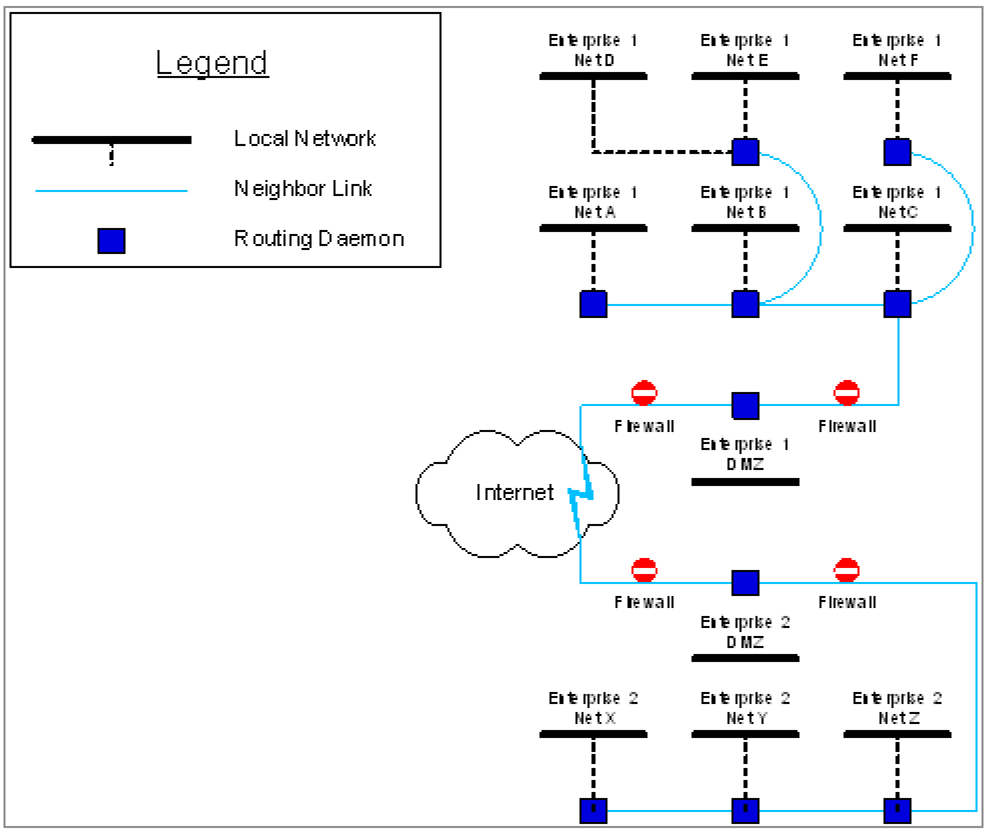


Security and Firewalls

Routing daemons offer security controls based on UDP service groups and subject names (see Restricting Message Flow). In addition, the routing daemon works in concert with firewalls to constrain information flow.

The WAN in Routing Daemon WAN with Firewalls connects two enterprises across the Internet. Each enterprise protects its networks with firewalls. Notice that the routing daemon within the DMZ does not serve any local network; instead that routing daemon operates as a *way station*, forwarding messages across the firewalls on either side of it.

Figure 18: Routing Daemon WAN with Firewalls



Neighbors Across a Firewall

Firewalls restrict the flow of information across organizational boundaries. For Rendezvous messages to flow between routing daemons, the daemons must establish TCP connections between neighbors. Security administrators can permit this connection using any technique they prefer; for example:

- Configure the firewall to permit TLS connections on the routing daemon's local port. Configure the routing daemons to connect with one another using TLS neighbor connections.
- Configure VPN connectivity between neighbor host computers.
- Configure the firewall to permit TCP connections on the routing daemon's local port.
- Configure the neighbors to connect using an SSH tunnel through the firewall.

Retransmission

Within its local networks, a routing daemon is the source (that is, the sending daemon) of all the forwarded messages that it rebroadcasts. That routing daemon handles retransmission requests (for example, if a listening application in the local network misses a packet).

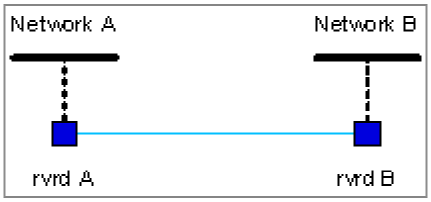
Retransmission and rvrd illustrates this concept:

Procedure

1. An application in network A sends a message.
2. Routing daemon A forwards the message to routing daemon B.
3. Routing daemon B rebroadcasts the message on network B.
4. A receiving application in network B misses a packet, and its `rvd` requests retransmission.
5. If the packet is within the reliability window of routing daemon B, then it retransmits the packet.

   Otherwise, it denies the retransmission request; it does *not* attempt to get a new copy of the message from routing daemon A.

Figure 19: Retransmission and rvrd



Border Routing

Border routing introduces a second tier of organization to `rvrd` routing networks, connecting several networks into a larger grouping while preserving their independence.

To configure border routing, see Border Routing and Border Policy.

Tip	Border routing is an advanced feature. We recommend that you consult with TIBCO before deploying this feature.

Advantages

Border routing can be advantageous in some situations:

- In networks with many routers, border routers can limit the spread of routing-related information, resulting in increased network stability.
- When routing messages between separate enterprises—for example, to a business partner outside your intranet—border routers can isolate intranet topology information.
- Border router policies enforce subject gating restrictions pairwise between routing table entries (that is, neighbors and local networks). You can specify a separate policy for each ordered pair.

Concepts

First-Tier Router

In the context of border routing, the term *first-tier router* denotes the kind of router that is already familiar from the preceding sections of this chapter.

First-tier routers share a global routing table. Every first-tier router has its own copy of the entire routing table, spanning the entire routing network; any change in the routing network propagates to every router. In networks with many routers, the resulting overhead can be noticeable.

Border Router

A *border router* or *second-tier router* is an `rvrd` process that can serve as a border, dividing a routing network into separate zones (see Zone, below).

You can configure a border router with neighbors and local networks (in the same way as you would configure a first-tier router). The border router connects these elements, and forwards messages among them.

Policy

A *policy* defines the set of subjects that a border router forwards from one of its neighbors or local networks (called the *From interface*) to another of its neighbors or local networks (the *To interface*). To configure policy, see Border Policy.

A border router can restrict a subject, forwarding only those messages that have not yet crossed a border; see First Border.

Note

Adding an interface to a border router automatically creates a default policy for all pairings of existing interfaces with the new interface. The default policy allows forwarding of `_INBOX.>` (that is, all point-to-point messages) in both directions (that is, from the new interface to each existing interface, and to the new interface from each existing interface). This default behavior mimics the behavior of first-tier routers (namely, inbox messages flow automatically, without explicit configuration).

This behavior is automatic when you add a border router interface using either the browser administration interface, or these Java configuration

API methods:

- `Router.addActiveInterface()`
- `Router.addLocalNetworkInterface()`
- `Router.addPassiveInterface()`

Nonetheless, you may explicitly remove this subject (`_INBOX.>`) from the border policy to disable forwarding of inbox messages.

In contrast, specifying a new interface by editing an XML configuration overrides this default border policy. An XML document engenders a router configuration that matches the XML specification exactly; an interface will *not* have *any* border policy unless the XML document explicitly specifies one.

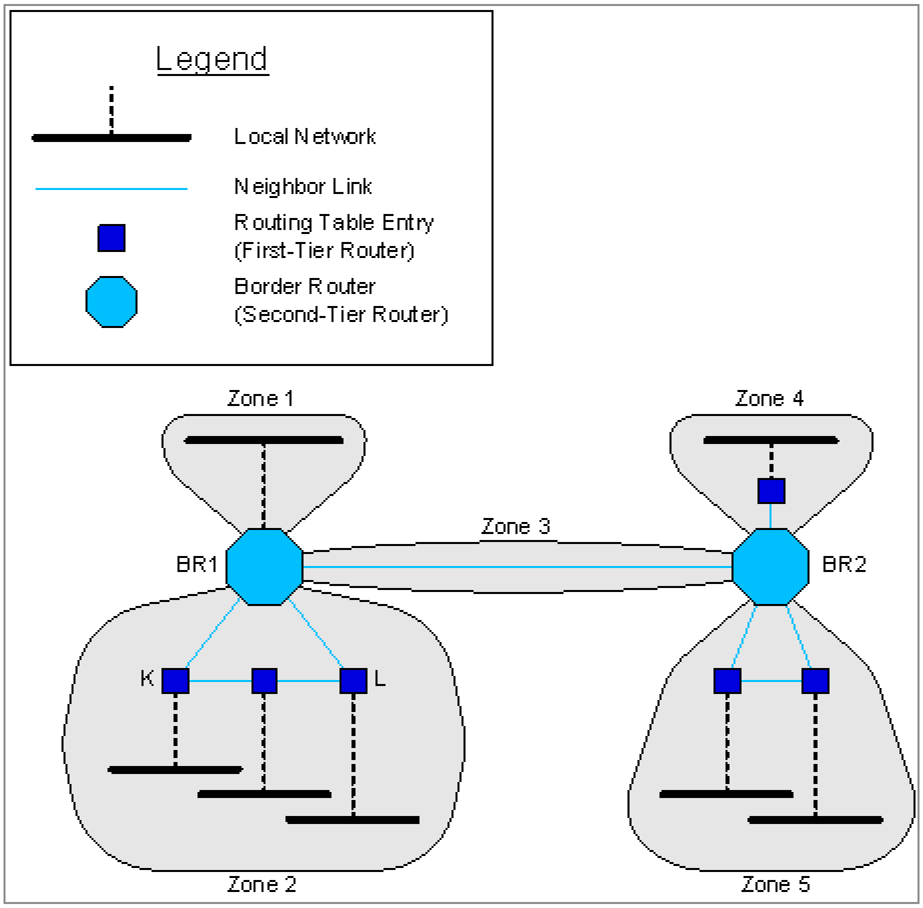
Zone

A *zone* or *first-tier routing network* is a collection of routers and local networks, in which every pair in the collection is connected by a route that does not cross through a border router.

Administrators do not explicitly configure zones. Instead, border routers periodically examine the network, and dynamically partition it into zones based on network connectivity.

In Border Router: Concepts, border router BR1 has two first-tier neighbors (K and L), and a route connects those neighbors without crossing through BR1 nor any other border router; so K and L are in the same zone.

Figure 20: Border Router: Concepts



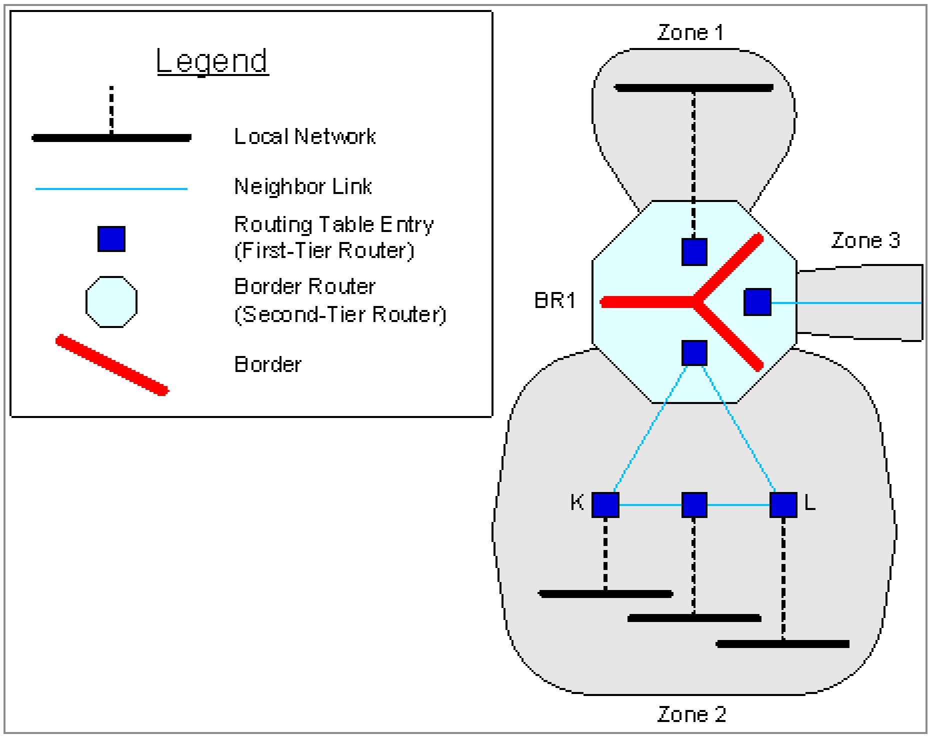
The *effective policy* of a zone is the union of the policies of all its constituents. In other words, a message that can enter or leave through any constituent can enter or leave the zone as a whole. For example, if BR1 allows `foo.*` to cross from K to BR2, then `foo.*` can cross from anywhere in zone 2 to anywhere in zone 3.

Implicit Internal First-Tier Routers

Each border router process embodies several *implicit internal first-tier routers*. When a border router automatically groups its neighbors and local networks into zones, it tacitly instantiates one first-tier router (within itself) for each zone that it serves.

When we say that a border router participates in a zone, we really mean that one of the implicit first-tier routers within the border router participates in that zone. Border Router: Implicit Internal First-Tier Routers expands a portion of Border Router: Concepts; it illustrates that border router BR1 contains three implicit internal first-tier routers, which serve zones 1, 2 and 3. Each one participates in one zone, as a representative of BR1.

Figure 21: Border Router: Implicit Internal First-Tier Routers



Implicit first-tier routers are similar to the embodied routers described in Independent Routing Table Entries in One Process, except that you cannot configure them—the border router creates and configures them automatically. (Border routers can embody only implicit first-tier routers; they cannot configure *explicit* internal first-tier routers.)

Internal representatives of a border router are invisible to the external first-tier interfaces that they serve. All representatives of a border router present the same routing name, which is identical to the name of the border router (in our example, all three are named BR1). As a result, all external interfaces appear to communicate with BR1.

Border

Within a border router, a *border* separates every pair of implicit internal first-tier routers (see Border Router: Implicit Internal First-Tier Routers). Border routers dynamically determine borders, just as they dynamically determine zones.

A message can cross a border when a policy allows the message subject.

First-tier routing table information cannot cross a border.

Borders are not directly accessible to administrators; they remain internal to border routers.

Second-Tier Routing Network

A *second-tier routing network* is a collection of border routers in which every pair in the collection is connected by a route.

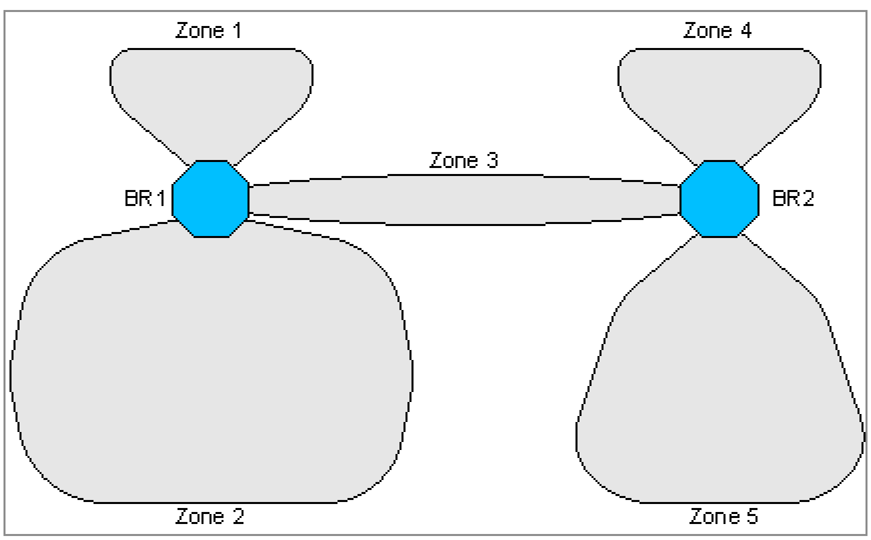
First-tier routing information includes information about first-tier routers, local networks, and all the subjects that can flow among them (within a zone).

Second-tier-routing information includes information about border routers, zones, and all the subjects that can flow among them; it specifically excludes all first-tier routing information.

All border routers in a second-tier network share second-tier routing information, but not first-tier information. Conversely, first-tier constituents of zones cannot access second-tier information—except for information about subjects available through a participating border router.

For example, Border Router: Second-Tier Routing Network illustrates the view of the second-tier network shared by BR1 and BR2 (based on the example of Border Router: Concepts). BR1 and BR2 share second-tier information so that both can create an internal routing table that includes both border routers, all five zones, and all the subjects that can flow among them. However, BR1 cannot access first-tier information about the constituents of zones 4 and 5, and BR2 cannot access first-tier information about the constituents of zones 1 and 2.

Figure 22: Border Router: Second-Tier Routing Network



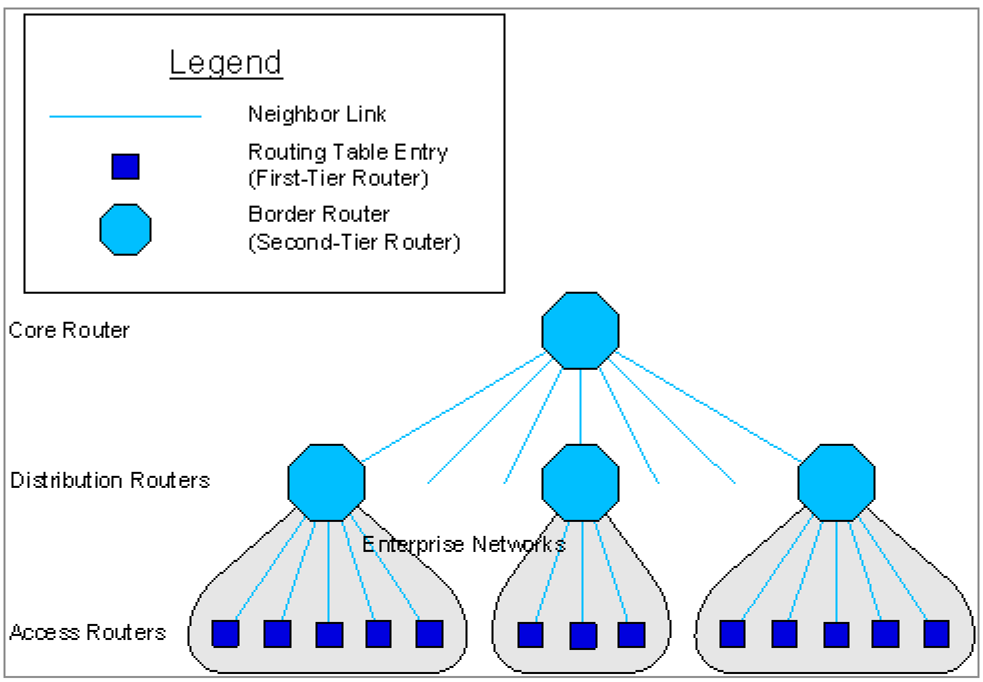
High-Fanout Second-Tier Networks

You can use border routers to construct a high-fanout network, like the standard reference architecture in Border Router: High-Fanout Network.

Within an enterprise, this architecture can promote network stability, use network bandwidth efficiently, and effectively control the flow of data.

When this architecture spans several enterprises, distribution-level border routers can isolate each enterprise network within a separate zone. With appropriate policy configuration, this architecture addresses privacy issues among partners in business-to-business applications.

Figure 23: Border Router: High-Fanout Network

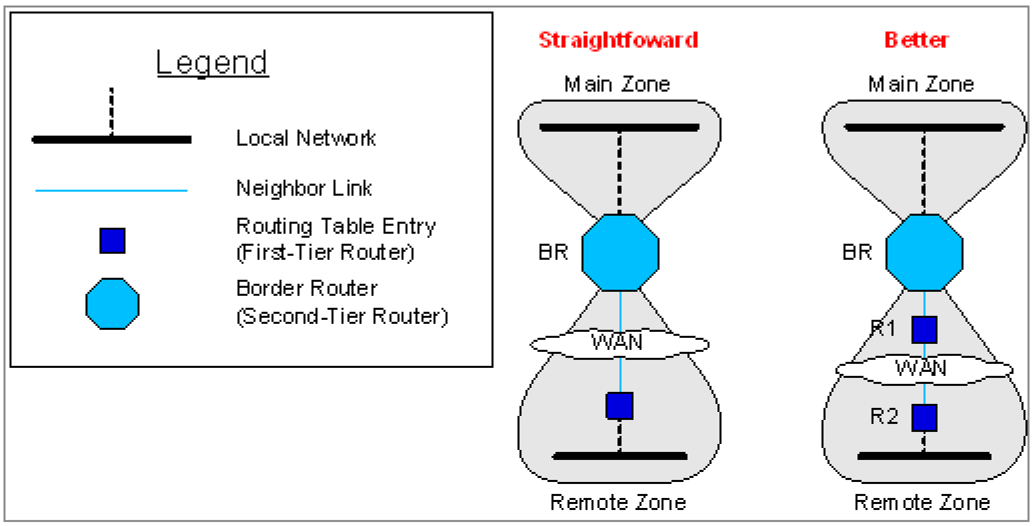


Best Practice: Zone Stability in Second-Tier Networks

In some topologies, a communication link failure can remove an entire zone from a second-tier network. For example, in the straightforward topology (left in Border Router: Zone Stability), a WAN failure disconnects the entire Remote Zone from border router BR. The border router automatically assesses the situation and rebuilds its zone map, however, this process can disrupt message flow for several minutes. A similar disruption can occur when the WAN resumes normal operation.

Contrast the better topology (right in Border Router: Zone Stability). BR and the first-tier router R1 both reside on the same side of the WAN link (possibly even on the same host computer). In this topology, WAN failure does not disconnect the entire Remote Zone, since BR remains in contact with R1. BR need not reconfigure its zone map, so it avoids associated delays.

Figure 24: Border Router: Zone Stability

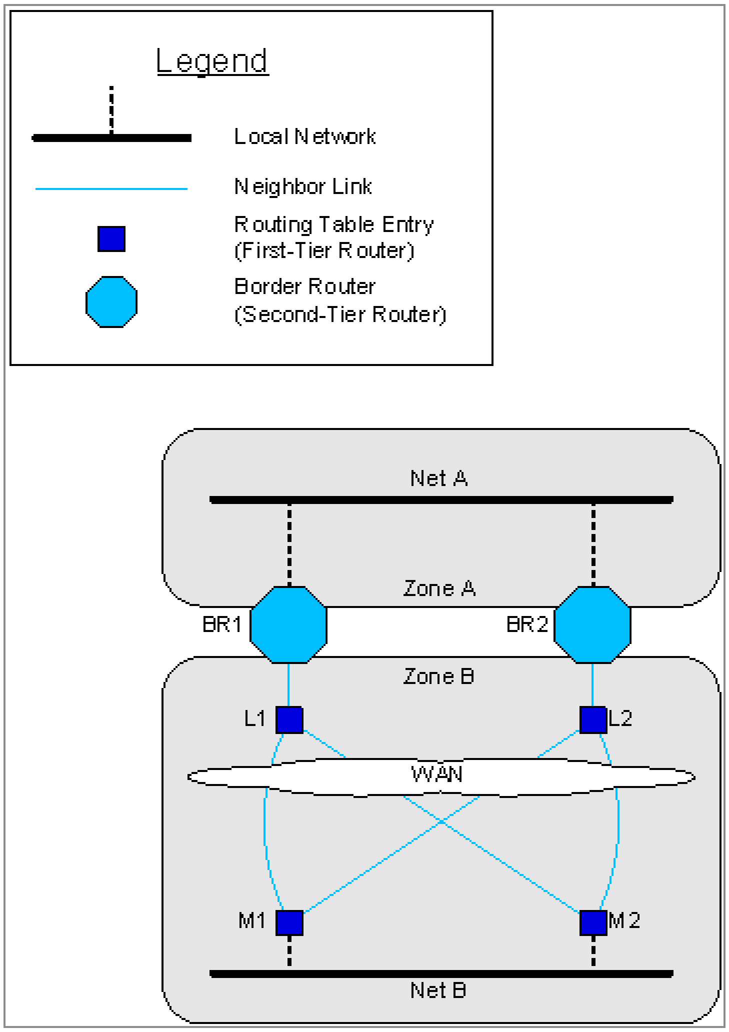


Best Practice: Fault Tolerance in Second-Tier Networks

Border Router: Fault Tolerance illustrates fault tolerance in a second-tier network that crosses a WAN link. Notice that this topology addresses two aspects of fault-tolerance:

- Redundant border routers—BR1 and BR2
- Redundant routing across WAN communications—the X pattern spanning the first-tier routers L1, L2, M1 and M2 within zone B

Figure 25: Border Router: Fault Tolerance

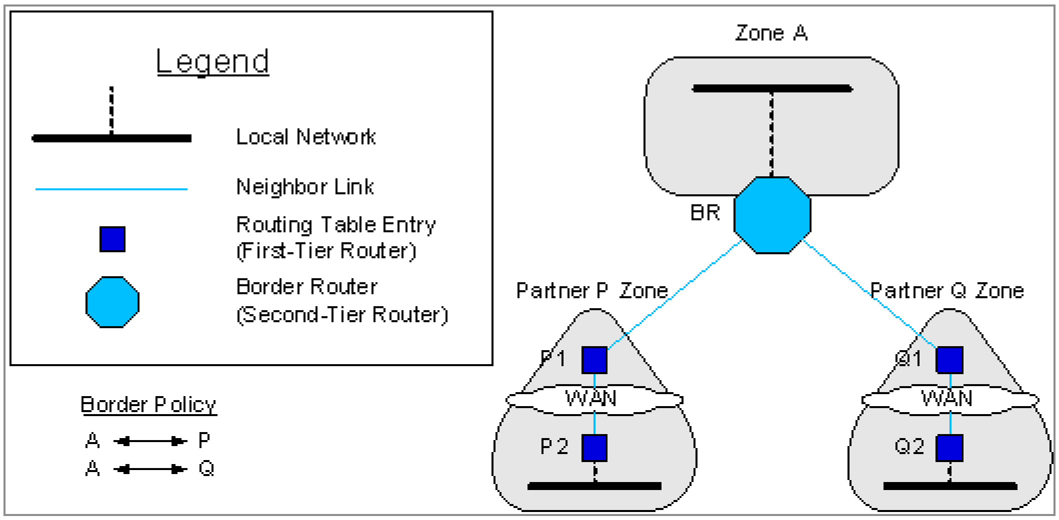


Best Practice: Isolating Enterprise Zones in Second-Tier Networks

Business-to-business (B2B) networks often require strict isolation of each partner's data at the same time as they require data to flow in two directions between the hub and each partner. Border Router: Isolating Enterprise Zones illustrates a situation in which the main network in zone A must exchange data freely with partner P and partner Q—but data from either partner must not flow to the other partner. The border policy configures this separation as required.

(Incidentally, notice that Border Router: Isolating Enterprise Zones features zone stability, co-locating routers P1 and Q1 near border router BR; see Best Practice: Zone Stability in Second-Tier Networks.)

Figure 26: Border Router: Isolating Enterprise Zones



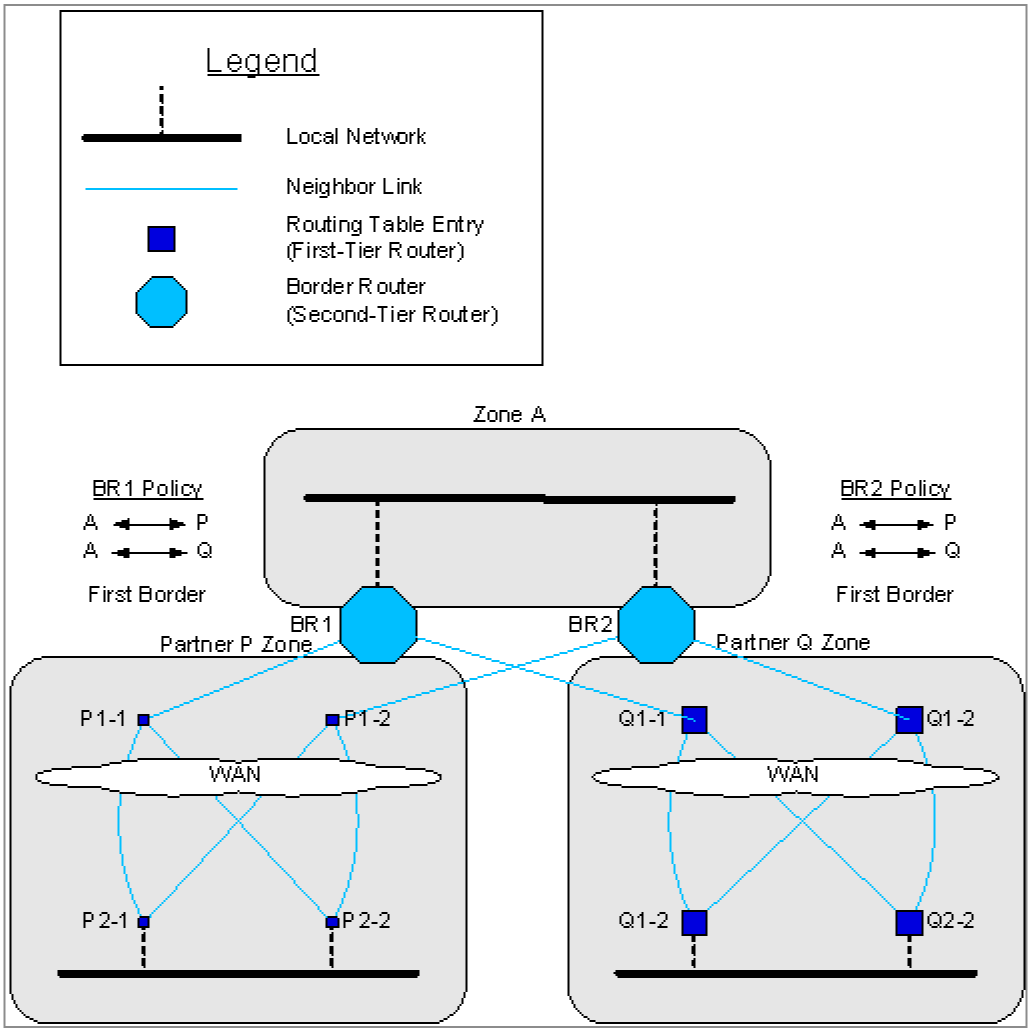
Fault Tolerance

Adding fault tolerance to the topology of Border Router: Isolating Enterprise Zones would seem straightforward, but actually adds an unexpected complication. Border Router: Isolating Enterprise Zones and Fault Tolerance depicts the resulting topology. Notice that we address two aspects of fault tolerance (as we did in Border Router: Fault Tolerance):

- Redundant border routers (BR1 and BR2) guard against failure of either member of this pair. BR1 and BR2 each connect the hub to both partner zones (zone P and zone Q).
- Redundant routing across WAN communications guards against WAN link failure. The familiar X pattern is repeated within each partner zone.

Yet this topology introduces a prohibited behavior—messages can flow between the two partners. BR1 routes messages from partner P to hub A; BR2 forwards the messages from hub A to partner Q. To block this unintended data flow, administrators must properly configure border policy on BR1 and BR2; it is crucial to restrict the flow to those messages which have not yet crossed a border (see First Border). Configure this restriction separately for each subject that these border routers forward.

Figure 27: Border Router: Isolating Enterprise Zones and Fault Tolerance



Backlog Protection

Every WAN connection has a maximum capacity. Routing daemons cannot exceed this physical limitation. When the volume of routed data is greater than WAN capacity, `rvrd` buffers the outbound data.

Data backlog can occur in several scenarios; for example:

- An unexpected burst of data exceeds WAN capacity.
- A temporary problem with the WAN sharply decreases its capacity.
- WAN capacity is insufficient for the required volume of data.

- WAN capacity is generally sufficient, but `rvrd` is misconfigured to route more data than expected. The total data volume exceeds WAN capacity.

The Connected Neighbors page displays the peak backlog for each neighbor; see Connected Neighbors.

Maximum Backlog

An extremely large backlog can cause severe problems for `rvrd` and its host computer. Administrators can configure `rvrd` to protect against this possibility.

When enabling this feature, the administrator specifies the maximum permissible backlog (in kilobytes). When an outbound backlog of this size accumulates for any neighbor connection, `rvrd` automatically disconnects from that neighbor, clears the corresponding outbound data buffer, and attempts to reconnect to the neighbor.

To obtain a reasonable estimate for the threshold value that triggers this action, calculate the process storage available to `rvrd`, divided by the number of neighbor connections it serves.

You can configure this feature separately for each routing table entry. The router applies that maximum to all of its neighbor connections.

To configure this feature, see Routers.

Warning	Notice that enabling this feature represents a deliberate decision to discard data in certain extreme circumstances. When this feature is disabled (the default), the routing daemon does not protect against backlog. The decision to use this feature must be based on the business requirements of the enterprise.

Idle

`rvrd` can run in either of two states—running or idle.

When *running*, `rvrd` establishes neighbor connections and routes messages.

When *idle*, `rvrd` does no routing operations. However, the browser administration interface is available for configuring parameters. The process still behaves as a Rendezvous daemon (`rvd`).

While `rvrd` is in idle state, you can configure the routing table and other parameters without affecting the network in any way, without binding local resources (such as UDP services or TCP ports), and without resolving any names in the routing table. After saving the configuration in a store file (and terminating the `rvrd` process), you can restart `rvrd` using the stored configuration. Alternatively, you can move the store file to another host computer, and start `rvrd` there.

Routing Daemon Logging

A routing daemon process can output a running log of its activity. System administrators can use the resulting log files to monitor neighbor connections, subject interest and message flow.

To configure the kinds of normal activity to log, see Logging.

To configure the destination of log output, see Log Destination.

> The command line parameter `-log` is obsolete in release 6 (and later). Use the browser administration interface to configure `rvrd` logging categories.

Interpreting Log Output

Each line in the log file describes a significant event in the operation of a routing daemon. A time stamp indicates the date and time of the interaction. The remainder of the line is a string describing the event.

The log file begins with events in the routing daemon's start sequence. First, it discovers its hardware and software operating environment:

```
TIB/Rendezvous daemon
Copyright 1994-2004 by TIBCO Software Inc.
All rights reserved.
Version 7.3.0
2004-09-08 14:03:13 rvrd: Hostname: optimist
2004-09-08 14:03:13 rvrd: Hostname IP address: 10.101.2.140
2004-09-08 14:03:13 rvrd: Detected IP interface: 127.0.0.1 (lo)
2004-09-08 14:03:13 rvrd: Detected IP interface: 10.101.2.140 (eth0)
2004-09-08 14:03:13 rvrd: Using ticket file
```

```
/usr/local/tibco/bin/tibrv.tkt
2004-09-08 14:03:13 rvrd: Using store file /tmp/1.admin
2004-09-08 14:03:13 rvrd: Warning: zlib compression not supported in SSL
initialization.
2004-09-08 14:03:13 rvrd: OpenSSL 0.9.7c 30 Sep 2003
2004-09-08 14:03:13 rvrd: Initializing random pool...
2004-09-08 14:03:13 rvrd: Invoking callback for case 2 and certificate
1.
```

Next, the routing daemon reads its configuration from its store file. In this example, it defines a router (routing table entry) named `optimist`. That router has an *accept any neighbor* interface, and a local network interface.

```
2004-09-08 14:03:13 rvrd: [optimist]: Defined.
2004-09-08 14:03:13 rvrd: [optimist]: Any neighbor is allowed to connect
to local port 9666. Link cost: 1.
2004-09-08 14:03:13 rvrd: [optimist]: Local network 7505.RV.TIBCO
defined. Interface: 10.101.2.140. Service UDP port: 7505, Service spec:
7505, Network spec: 10.101.2. Link cost: 1.
```

The routing daemon finishes its start sequence by reporting the URL bindings of its browser administration interfaces.

```
2004-09-08 14:03:13 rvrd: Http interface -
http://optimist.rv.tibco.com:8000/
```

The administrator sets the logging parameters for normal activity.

```
2004-09-08 14:08:35 rvrd: Logging: [Connections - On], [Subject Interest
- On], [Subject Data - On].
```

Now the routing daemon begins normal operations. Log items reflect neighbor connections to other routers (`viggen-r1`), exchange of subscription interest information, and forwarding of message data.

```
2004-09-08 14:13:26 rvrd: [optimist]: Connected to viggen-r1.
2004-09-08 14:15:40 rvrd: [optimist]: Sending subscription for [TEST] to
viggen-r1 for source 0A65023F/hpux11-viggen-n1.
2004-09-08 14:16:30 rvrd: [optimist]: Received data on [TEST] from
neighbor viggen-r1.
2004-09-08 14:18:03 rvrd: [optimist]: Sending cancel for [TEST] to
viggen-r1 for source 0A65023F/hpux11-viggen-n1.
2004-09-08 14:19:31 rvrd: [optimist]: Disconnected from viggen-r1 (4).
```

Then `optimist` discovers another routing daemon (fanox) serving the same local network and service (redundancy for fault tolerance). Then it loses contact with fanox.

```
2004-09-08 14:23:16 rvrd: [optimist]: Found fanox in 7505.RV.TIBCO.
2004-09-08 14:24:24 rvrd: [optimist]: Lost fanox in 7505.RV.TIBCO.
```

rvrd

Command

Syntax

```
rvrd    -store filename
       [-http [ip_address:]http_port]
       [-https [ip_address:]https_port]
       [-http-only]
       [-https-only]
       [-no-http]
       [-license url]
       [-idle]
       [-listen [socket_protocol:|ip_address:]tcp_port]
       [-no-permanent]
       [-no-lead-wc | -lead-wc]
       [-no-multicast]
       [-reliability time]
       [-max-consumer-buffer size]
       [-rxc-max-loss loss]
       [-rxc-recv-threshold bps]
       [-rxc-send-threshold bps]
       [-compress-level level]
       [-reuse-port inbox_port]
       [-logfile log_filename]
       [-log-max-size size]
       [-log-max-rotations n]
       [-log-config config_log_filename]
       [-foreground]
       [-udp-ttl hops]
       [-tls-min-proto-version version]
       [-tls-max-proto-version version]
       [-tls-ciphers string1:string2:stringN]
       [-tls-ciphersuites name1:name2:nameN]
       [-no-wc]
```

Purpose

The routing daemon efficiently connects Rendezvous programs on distant IP networks, so that messages flow between them as if within a single network. Nonetheless, communicating programs remain decoupled from internetwork addresses and other details.

Remarks

The `rvrd` process subsumes the behavior of `rvd`, so it is not necessary to run a separate `rvd` process on computers that run `rvrd`. We recommend *against* running both components on the same computer.

`rvrd` must run on a host computer with a *permanent* IP address. For example, a temporary address assigned by DHCP is invalid.

Command Line Parameters

Parameter	Description
`-store` *filename*	This file contains the routing table entry and parameters that configure `rvrd`. `rvrd` reads this file when the process starts, and writes this file each time you change the configuration using the browser administration interface. **Important** The store file requires physical security safeguards and operating system protection. Keep it in a location that is accessible only to the system administrators who maintain it. See also Store Files.
`-http` *ip_address*`:`*http_port* `-http` *http_port* `-https` *ip_address*`:`*https_port*	The browser administration interface accepts connections on this HTTP or HTTPS port. Permit administration access only through the network interface

Parameter	Description
`-https` *https_port*	specified by this IP address. To limit access to a browser on the `rvrd` host computer, specify `127.0.0.1` (the local host address). When the IP address is absent, the daemon accepts connections through any network interface on the specified HTTP or HTTPS port. If the explicitly specified HTTP port is already occupied, the program exits. If the explicitly specified HTTPS port is already occupied, the program selects an ephemeral port. When the `-http` parameter is entirely absent, the default behavior is to accept connections from any computer on HTTP port `7580`; If this default port is unavailable, the operating system assigns an ephemeral port number. When the `-https` parameter is entirely absent, the default behavior is to accept secure connections from any computer on an ephemeral HTTPS port. In all cases, the program prints (in its start banner and log file) the actual HTTP and HTTPS ports where it accepts browser administration interface connections.
`-http-only`	Disable HTTPS (secure) connections, leaving only an HTTP (non-secure) connection.
`-https-only`	Disables HTTP (non-secure) connections, leaving only an HTTPS (secure) connection.
`-no-http`	Disables *all* HTTP and HTTPS connections, overriding `-http` and `-https`.
`-license` *url*	The URL to a TIBCO Activation Service in the form of `https://url[?fp=activation service fingerprint]`. A fingerprint of the Activation Service can be included in

Parameter	Description
	the URL for peer verification. The URL can also be supplied using the environment variable `TIBRV_LICENSE`. If the URL is not explicitly configured using `-license` or `TIBRV_LICENSE`, the Rendezvous daemon will search its environment PATH for a license file named `tibrvlic.bin` and, failing that, will attempt to contact an Activation Service at `https://tib-activate:7070`. **Note:** Production environments must use a TIBCO Activation Service for TIBCO product activation. Using a license file to configure the Rendezvous daemon is only supported for development environments. The URL format for a license file is `file://pathname`.
`-idle`	Starts `rvrd` in its *idle* state (if present). When absent, start `rvrd` in its *running* state—routing messages.
`-listen` *tcp_port* `-listen` *ip_address*`:`*tcp_port* `-listen` *socket_protocol*`:`*tcp_port*	`rvd` (and by extension, `rvrd` operating within the local network) opens a TCP client socket to establish communication between itself and its client programs. The `-listen` parameter specifies the TCP port where the Rendezvous daemon listens for connection requests from client programs. This `-listen` parameter of `rvd` corresponds to the `daemon` parameter of the transport creation call (they must specify the same TCP port number). The IP address specifies the network interface through which this daemon accepts TCP connections. To bar connections from remote programs, specify IP address `127.0.0.1` (the loopback interface). When the IP address is absent, the daemon accepts connections from any computer on the specified TCP port.

Parameter	Description
	When this parameter is entirely absent, the default behavior is to accept connections from any computer on TCP port 7500. For more detail about the choreography that establishes conduits, see Daemon Client Socket—Establishing Connections. **Warning** This parameter does *not* correspond to the `service` parameter of the transport creation call—but rather to the `daemon` parameter.
`-no-permanent`	If present (or when `rvd` starts automatically), `rvd` exits after 1 minute during which no transports are connected to it. If not present, `rvd` runs indefinitely until terminated. This parameter is not available with IPM.
`-no-lead-wc` | `-lead-wc`	Sending to subjects with lead wildcards (for example, `>` or `*.foo`) can cause unexpected behavior in some applications, and cause network instability in some configurations. This option lets you selectively screen wildcard sending. When `-no-lead-wc` is present, the daemon quietly rejects client requests to send outbound messages to subjects that contain wildcards in the lead element. The daemon does *not* report excluded messages as errors. When `-lead-wc` is present (or when neither flag is present), the daemon allows sending messages to subjects with lead wildcards.
`-no-multicast`	When present, the daemon disables multicast (and broadcast) communication. For details, see Disabling

Parameter	Description
	Multicast.
`-reliability` *time*	Rendezvous daemons compensate for brief network failures by retaining outbound messages, and retransmitting them upon request. This parameter is one of several ways to control the message reliability interval. For a complete discussion the concept of reliability, the various ways to control it, the interaction among those ways, and reasonable values, see Reliability and Message Retention Time. If this parameter is absent, `rvd` uses the factory default (60 seconds). If this parameter is present, `rvd` (and by extension, `rvrd` operating within the local network) retains messages for *time* (in seconds). The value must be a non-negative integer.
`-max-consumer-buffer` *size*	When present, the daemon enforces this upper bound (in bytes) on each consumer buffer (the queue of messages for a client transport). When data arrives faster than the client consumes it, the buffer overflows this size limit, and the daemon discards the oldest messages to make space for new messages. The client transport receives a `CLIENT.SLOWCONSUMER` advisory. When absent or zero, the daemon does not enforce a size limit on the consumer buffer. (However, a 60-second time limit on messages still limits buffer growth, independently of this parameter.)
`-rxc-max-loss` *loss* `-rxc-recv-threshold` *bps* `-rxc-send-threshold` *bps*	These three parameters configure the retransmission control (RXC) feature, which suppresses retransmission requests from chronically-lossy receivers. (This feature applies to the `rvd` behavior within `rvrd`, but not to routing behavior.)

Parameter	Description
	If `-rxc-max-loss` is absent or zero, then RXC is disabled. If it is an integer in the range `[1,100]`, it determines the maximum percentage acceptable loss rates above which a receiver is considered chronically-lossy. `-rxc-recv-threshold` configures the threshold receive rate (in bits per second) above which a chronically-lossy receiver censors its own retransmission requests. When absent, the default value is zero (always censor a chronically-lossy receiver). `-rxc-send-threshold` configures the threshold send rate (in bits per second) above which the daemon suppresses (that is, ignores requests from) chronically-lossy receivers. When absent, the default value is zero (always suppress retransmissions to chronically-lossy receivers). For a complete explanation, see Retransmission Control .
`-compress-level` *level*	When present, this option guides the trade-off between data compression and data latency. Acceptable values are integers in the range `[1, 10]`. • `1` favors minimum latency, sacrificing compression efficiency. • `10` favors maximal compression, accepting the concomitant cost of latency. This option applies across all neighbor interfaces (it is not possible to specify different values for each neighbor). Furthermore, it applies *only* to neighbor interfaces that are configured for data compression *without* TLS. When absent, the default behavior is equivalent to `10`—favoring compression over latency.
`-reuse-port` *inbox_port*	When present, other daemons on the same host computer can reuse service ports.

Parameter	Description
	When absent, other daemons cannot reuse a service port that is in use by this daemon. For correct operation, all the daemons that use a common service port on a host computer must specify this option. For background and details, see Reusing Service Ports. The *inbox_port* argument (required) specifies the UDP port that this daemon uses for point-to-point communications. This value must be unique for each daemon (that reuses service ports) on a common host computer. Furthermore, you must not use the *inbox_port* in any transport specification on the same host computer.
`-logfile` *log_filename*	Sends log output to this file. When absent, the default is `stderr`.
`-log-max-size` *size* `-log-max-rotations` *n*	When present, activates the log rotation regimen (see Log Rotation). When you specify these options, you must also specify `-logfile`. *size* is in kilobytes. If *size* is non-zero, it must be in the range [100, 2097152]. Values outside this range are automatically adjusted to the nearest acceptable value. Zero is a special value, which disables log rotation. When `-log-max-size` is zero or absent, a single log file may grow without limit (other than the limit of available storage). *n* indicates the maximum number of files in the rotation. When `-log-max-rotations` is absent, the default value is 10.
`-log-config` *config_log_filename*	Sends duplicate log output to this file for log items that

Parameter	Description
	record configuration changes. The daemon never rotates nor removes this special log file. Instead, this file remains as a record of all configuration changes. When absent, the default is `stderr`.
`-foreground`	Available only on UNIX platforms. When present, `rvrd` runs as a foreground process. When absent, `rvrd` runs as a background process.
`-udp-ttl` *hops*	UDP TTL (Available only with TRDP daemons.) When present, the daemon sends UDP packets with a TTL value of *hops* (a positive integer, less than or equal to 256). When absent, the default TTL is 16 hops.
`-tls-min-proto-version` *version* `-tls-max-proto-version` *version*	Sets the minimum or maximum supported protocol versions for the ctx using OpenSSL calls `SSL_CTX_set_min_proto_version` and `SSL_CTX_set_max_proto_version`.
`-tls-ciphers` *string1:string2:stringN*	Sets the list of available ciphers (TLSv1.2 and earlier) using OpenSSL call `SSL_CTX_set_cipher_list`.
`-tls-ciphersuites` *name1:name2:nameN*	Configures the available TLSv1.3 ciphersuites using OpenSSL call `SSL_CTX_set_ciphersuites`.
`-no-wc`	Silently drops any messages published by clients that contain any wild card tokens.
`-pwd-hash password-string`	Prints a base64-encoded PBKDF2 hash password string before exiting. Rendezvous daemons do not store passwords, they

Parameter	Description
	convert the plain text passwords into a secure hash first. Provides a secure hash instead of a plain-text Administrator password in the XML/JSON configuration and you can remove Rendezvous daemons from the chain of custody of the password. Secure hashes are one-way, and passwords cannot be recovered from the secure hash. Ensure that you securely store the original password.

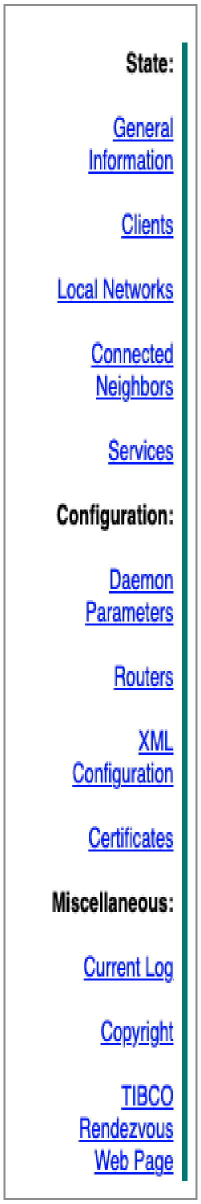
Browser Administration Interface—rvrd

The browser administration interface lets you control `rvrd` from a web browser. You can configure its operating parameters, and view operating statistics.

Navigation

All browser administration interface pages display a navigation panel at the left side of the page. Use these links to display other pages.

Figure 28: rvrd Navigation Panel



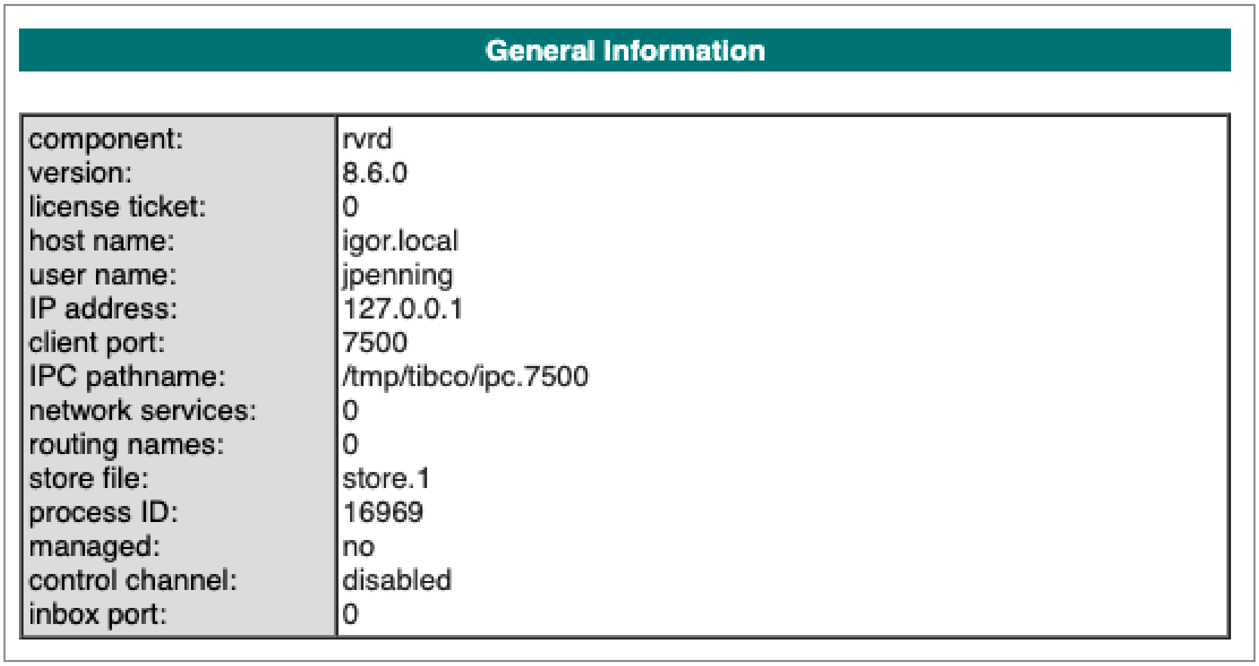
Category	Item	Description
State	General Information	This page displays information about an `rvrd` process; see General Information.
	Clients	This page summarizes the client transports; see Clients.

Category	Item	Description
	Local Networks	This page summarizes the local networks of a router; see Local Networks.
	Connected Neighbors	This page summarizes the actual neighbor connections of a router; see Connected Neighbors.
	Services	This page summarizes network services activity; see Services.
Configuration	Daemon Parameters	This page lets you configure parameters that control configuration access and router logging; see Daemon Parameters.
	Routers	This page lets you configure routers. You can access additional configuration pages through links on this page. See Routers, and the sections that follow it.
	XML Configuration	This page lets you view the current configuration as an XML document, and reconfigure the component by submitting an edited XML document.
	Certificates	This page lets you configure certificates that the daemon uses to identify itself in secure protocols. See Certificates.
	Log Out	This item logs out the current user or Administrator. See Log Out.
Miscellaneous	Current Log	This page displays the most recent 4 kilobytes from the log file.
	Copyright	The Rendezvous copyright page.
	TIBCO Rendezvous Web Page	The product page from the TIBCO web site.

General Information

`rvrd` (like all Rendezvous components) displays information about itself on this page.

To display this page, click **General Information** in the left margin of any page of the `rvrd` browser administration interface.



General Information

component:	rvrd
version:	8.6.0
license ticket:	0
host name:	igor.local
user name:	jpenning
IP address:	127.0.0.1
client port:	7500
IPC pathname:	/tmp/tibco/ipc.7500
network services:	0
routing names:	0
store file:	store.1
process ID:	16969
managed:	no
control channel:	disabled
inbox port:	0

Item	Description
component	The name of the program—`rvrd` (or `rvsrd`).
version	Version number of the program.
license ticket	The license ticket that validates this process.
host name	The hostname of the computer where the daemon process runs. Notice that the daemon process can run on one computer, while you access its browser interface from another computer.
username	The user who started the daemon process.

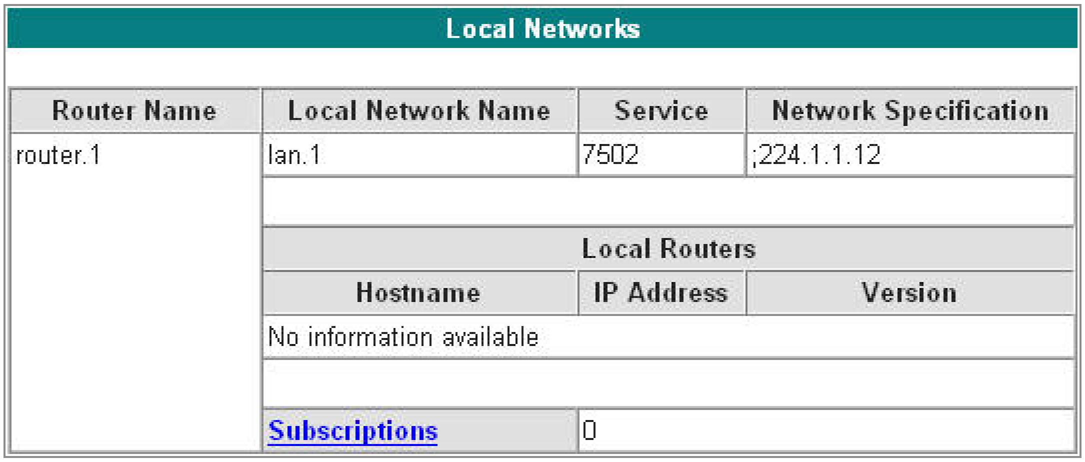
Item	Description
IP address	The IP address of the computer where the daemon process runs.
client port	The TCP port where the daemon listens for client connections.
network services	The number of network services on which this daemon's clients communicate.
routing names	The number of router names that this daemon embodies; see Routing Table Entry.
store file	File name of the daemon's store file; see the command line parameter –`store` for rvsd, and for rvsrd.
process ID	The operating system's process ID number for the component.
managed	Not applicable.
control channel	Not applicable.
inbox port	When the daemon reuses service ports, this field displays the unique inbox port. When the daemon does not reuse service ports, this field displays zero.

Local Networks

`rvrd` displays information about its local networks on this page.

To display this page, click **Local Networks** in the left margin of any page of the `rvrd` browser administration interface.

Figure 29: rvrd Local Networks Page



Item	Description
Router Name	This page groups local networks by router name (routing table entry). A box in this column indicates the name of the routing table entry that serves the local networks shown to its right. See also Routing Table Entry.
Local Network Name	The name of a local network.
Service	The UDP service for communication on the local network.
Network Specification	The network specification (as specified by the routing table entry).
Local Routers	This subtable lists other routers that serve this local network.
Hostname	The name of the host computer where the other routing daemon runs. Click here to view the browser administration interface for the other routing daemon process.
IP Address	The IP address of the host computer where the other routing daemon runs.
Version	The version of the other routing daemon executable.

Item	Description
Subscriptions	Total number of subscriptions over all transports within the local network, for which clients have registered interest. Click this link to view a list of the subscription subjects; see Subscription List.

Connected Neighbors

`rvrd` displays information about its (actual) neighbor connections on this page.

To display this page, click **Connected Neighbors** in the left margin of any page of the `rvrd` browser administration interface.

Note This page is related to—but not the same as—the page described in Neighbor Interfaces.

Figure 30: rvrd Connected Neighbors Page

Connected Neighbors			
Router Name	**Neighbor Name**	**Link Stats**	**Peak Backlog**
shen01	shen03	shen01[2]	1.228 KBytes
	shen02	shen01[3]	106.512 KBytes

Item	Description
table rows	Each row in this table describes one neighbor connection.
Router Name	This page groups neighbors by local router name (routing table entry). A box in this column indicates the name of the local router that connects to the neighbors show to its right. See also Routing Table Entry.
Neighbor	The name of a remote router with which the local router has a neighbor

Item	Description
Name	connection. Click here to view the browser administration interface for the neighbor routing daemon process.
Link Stats	The name of the (local) neighbor interface that specifies this neighbor connection. `rvrd` generated this name automatically when you configured the neighbor interface. Click this name to view the Router Connection Statistics page.
Peak Backlog	Backlog is outbound data awaiting transmission to a neighbor. This column displays the peak backlog for each neighbor. The Reset Statistics button on the Router Connection Statistics page resets this figure to zero. It is also reset to zero when the neighbors become disconnected and subsequently reconnect. See also, Backlog Protection.

Router Connection Statistics

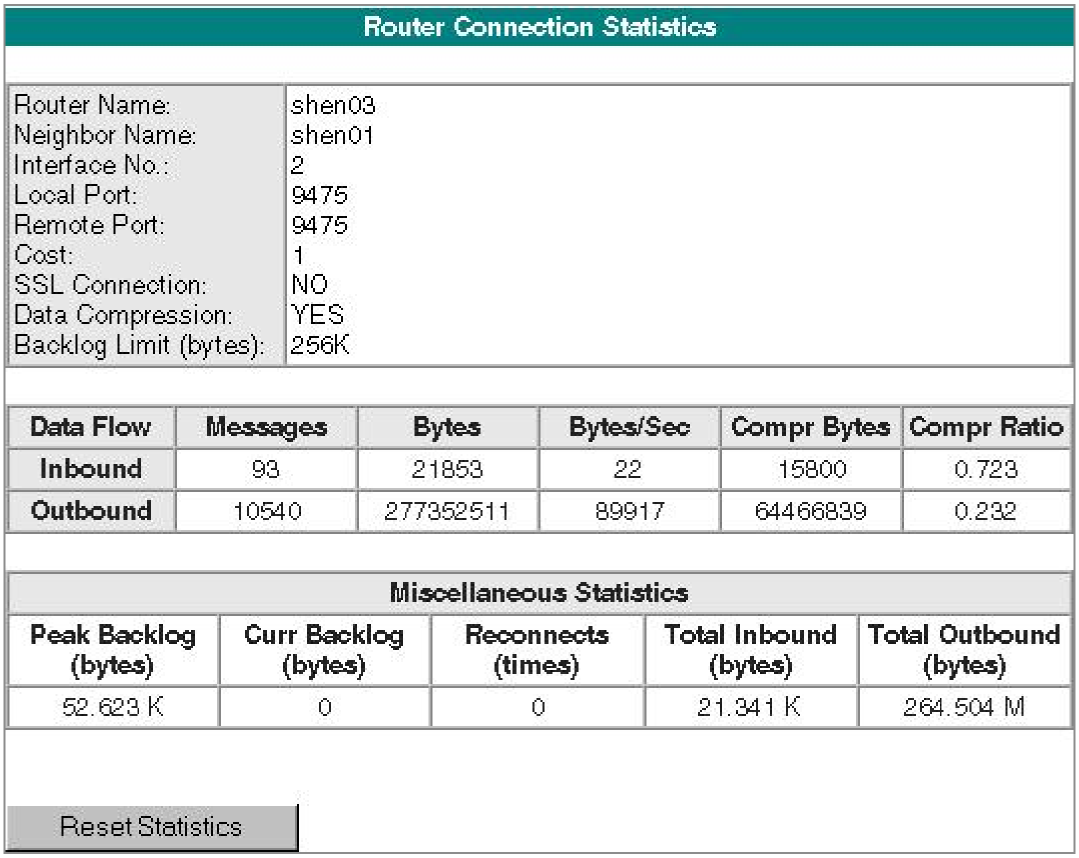
`rvrd` displays statistics about the performance of a neighbor connection on this page.

To display this page, click the **Link Stats** column of the Connected Neighbors page.

Note

Connection statistics are not available when neighbors connect using TLS.
See also SSL Connection with Compression.

Figure 31: rvrd Router Connection Statistics Page



Item	Description
summary	This list presents static information about the neighbor connection.
Router Name	The name of the local router. (See also Routing Table Entry.)
Neighbor Name	The name of the neighbor (remote router).
Interface Number	The name of the (local) neighbor interface that specifies the neighbor connection. `rvrd` generated this number automatically when you configured the neighbor interface, and incorporates it into the neighbor ID.
Local Port	TCP port that this router uses to communicate with the neighbor.

Item	Description
Remote Port	TCP port that the neighbor (remote router) uses to communicate with the local router.
Cost	Path cost of the neighbor link.
SSL Connection	TLS security feature for the neighbor link.
Data Compression	Data compression feature for the neighbor link. When a connection uses TLS with data compression, then *compression* statistics are not available (because TLS does the compression). For background information, see Data Compression, and SSL Connection with Compression.
Backlog Limit	When backlog protection is enabled, this item displays the threshold for disconnect from the neighbor. See Backlog Protection.
Data Flow	This table displays statistics about the volume of data on the neighbor connection. The *Inbound* row displays statistics about inbound data from the remote neighbor to the local router. The *Outbound* row displays statistics about outbound data from the local router to the remote neighbor.
Messages	Cumulative count of messages.
Bytes	Cumulative count of bytes (without compression).
Bytes/Sec	Data transmission rate during the most recent interval.
Compr Bytes	Cumulative count of compressed bytes. This item displays non-zero only when both of the neighbors specify data compression on the Add New Neighbor Interface.
Compr Ratio	Compression ratio.

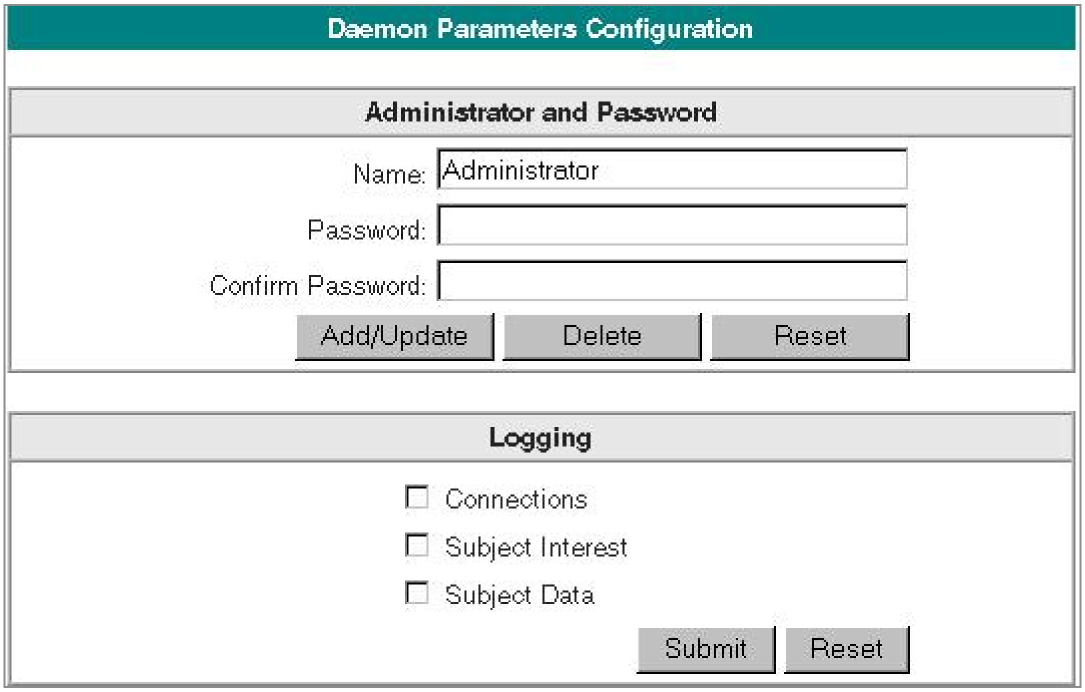
Item	Description
	This item displays non-zero only when both of the neighbors specify data compression on the Add New Neighbor Interface.
Miscellaneous Statistics	This table displays statistics not related to either inbound or outbound data transmission.
Peak Backlog	Peak backlog of outbound data (in bytes) since the last reset of statistics. See also, Backlog Protection.
Curr Backlog	Current backlog of outbound data (in bytes). See also, Backlog Protection.
Reconnects	Cumulative count of times when the neighbor link became disconnected and subsequently reconnected. (For example, network failure or backlog protection could cause a disconnect.)
Total Inbound Total Outbound	Cumulative counts of inbound and outbound bytes (without compression) since the start of the neighbor connection. The Reset Statistics button does not affect these items.
Reset Statistics	Click this button to reset statistical counters to zero.

Daemon Parameters

This page lets you configure parameters that affect overall daemon security.

To display this page, click **Daemon Parameters** in the left margin of any page of the `rvrd` browser administration interface.

Figure 32: rvrd Daemon Parameters Configuration Page



Administrator and Password

Only authorized personnel have access to routing daemons.

When administrator identification information is *not* set, anyone who can connect to the browser administration interface can examine and reconfigure the daemon. This arrangement can be useful during initial configuration and testing phases. However, during regular operation we recommend limiting access.

Once administrator identification information is registered, the browser administration interface is locked against unauthorized access. The daemon prompts administrators to prove identity by typing a name and password. After providing proper identification, an authorized administrator is logged in, and has complete access to configure the daemon. If the administrator does not provide proper identification, the browser displays the General Information page and continues to prompt for a correct name and password.

Warning Browsers remember administrator name and password information for the duration of the browser process. Merely closing the browser *window* does not erase this information. To guard against intruders you must terminate the browser *process* (*all* its windows).

Primary Administrator

The first administrator to register is called the *primary administrator*. In addition to configuring the daemon, the primary administrator can also add, delete and modify identification information pertaining to the other administrators.

Each daemon configuration can store up to 16 additional administrator name and password pairs (after the primary administrator).

One Administrator Session

Each daemon process permits only one administrator session at a time. When one administrator is logged in, other administrators are locked out; this prevents conflicts in which two administrators attempt to modify the configuration at the same time. To terminate a administrator session, see Log Out (below).

Item	Description
Name	Type a name string.
Password	Type a password string.
Confirm Password	Type the password again.
Add/Update	Specify a name and password, then click this button to add a new administrator. The primary administrator can add other administrators and update their passwords. All other administrators can update only their own passwords.
Delete	Click this button to delete administrator identification information. This action is available only to the primary

Item	Description
	Administrator. Deleting the primary administrator also deletes all other administrator.

Log Out

To end an administrative session, click **Log Out** in the left margin of the browser administration interface. This item appears only when you are logged in as an Administrator.

Daemons automatically log out administrator sessions that have been idle for 10 minutes.

Logging

This panel configures the kind of routing activity that the routing daemon routinely outputs to its log file.

Item	Description
Connections	Log connection activity whenever this routing daemon establishes or closes a connection to a neighbor.
Subject Interest	Log all subscription requests (notification of listening) that this routing daemon sends to its neighbors or receives from its neighbors.
Subject Data	Log all messages that this routing daemon forwards to its neighbors or receives from its neighbors.

To configure the destination of log output, see Log Destination.

To interpret the content of log output, see Routing Daemon Logging.

Routers

This page lets you configure routing table entries (router names). For more information, see Routing Table Entry.

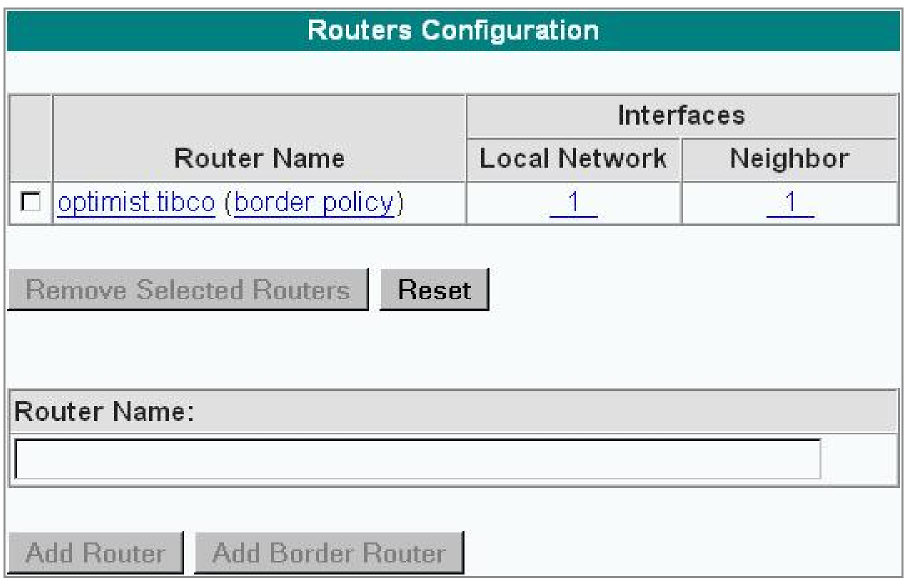
To display this page, click **Routers** in the left margin of any page of the `rvrd` browser administration interface.

Identify each routing table entry by a globally unique name.

You can add a new entry or remove an existing entry at any time. (However, border routing introduces restrictions; see Border Routing.)

For background information, see Routing Table Entry, and Independent Routing Table Entries in One Process.

Figure 33: rvrd Routers Configuration Page



Item	Description
Existing Routers	This panel lists the routing table entries within this routing daemon process. Each row represents one routing table entry.
Router Name	This column displays the router name of a routing table entry. Click here to set the maximum backlog for the routing table entry; see

Item	Description
	Backlog Protection. When configured as a border router, the phrase **(border policy)** appears after the router name. To configure or view the border policy, click this phrase; see Border Policy. (For background information, see Policy.)
Local Network	The number of local networks configured for a routing table entry. Click here to view the page Local Network Interfaces Configuration.
Neighbor	The number of neighbors configured for a routing table entry. Click here to view the page Neighbor Interfaces.
Add Router	To add a first-tier router, type its name, then click this button.
Add Border Router	To add a border router, type its name, then click this button. See also, Border Routing, below.
(border policy)	To view the page Border Policy, click this phrase **(border policy)** following the name of a border router.

Border Routing

For an introduction to the concepts of this feature, see Border Routing.

Tip	Border routing is an advanced feature. We recommend that you consult with TIBCO before deploying this feature.

Border routing restricts permissible configurations. When an `rvrd` process is configured as a border router, that border router must be the only routing table entry for the process.

As a result, you can configure an `rvrd` process either as a collection of one or more first-tier routers, or as exactly one border router. You cannot configure more than one border router in a process, nor mix first-tier and border routers in the same process.

To configure a process as a border router, type the new router name, and click the **Add Border Router** button.

You cannot remove a border router from an `rvrd` process.

Local Network Interfaces Configuration

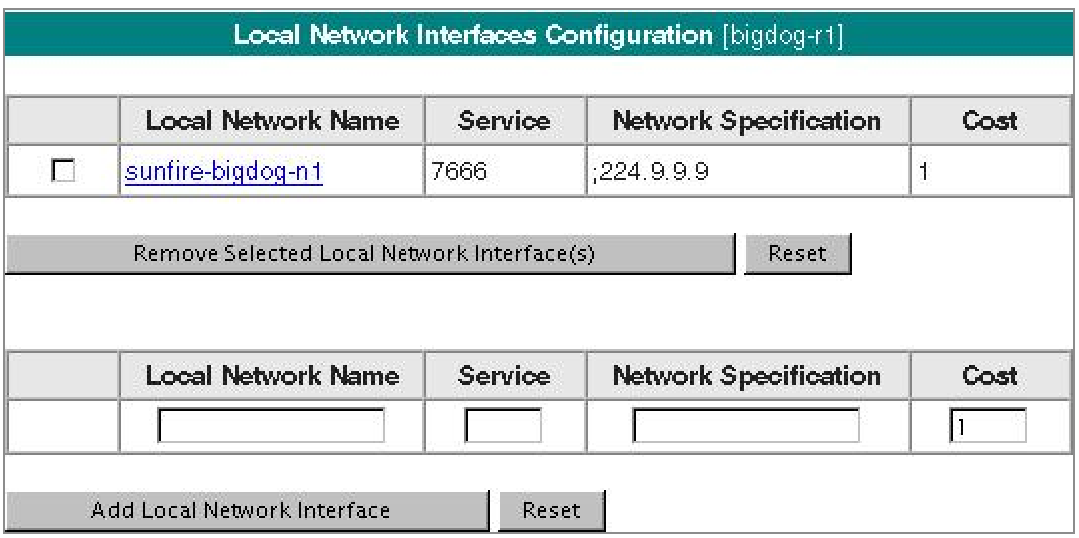
This page lets you configure local networks for a routing table entry.

To display this page, click the number of local networks in a row of the **Routers page.**

For background information, see Local Network.

> **Note**
> This page is not the same as the page described in Local Networks.

Figure 34: rvrd Local Network Interfaces Configuration Page



Item	Description
existing local networks	The upper table lists local networks. Each row represents one local network.
Local Network Name	The name of a local network. Local network names must be globally unique.

Item	Description
	To configure subject gating for a local network, click its name in the table of existing local networks. For more information, see Local Network.
Service	The UDP service for communication on a local network. Programs within the local network communicate using this service. For more information, see Specifying the UDP Service.
Network Specification	The network specification for a local network. For more information, see Constructing the Network Parameter.
Cost	Path cost for routing between a local network and the routing daemon. For more information, see Load Balancing.
Add Local Network Interface	To add a new local network, type the specifications and click this button.

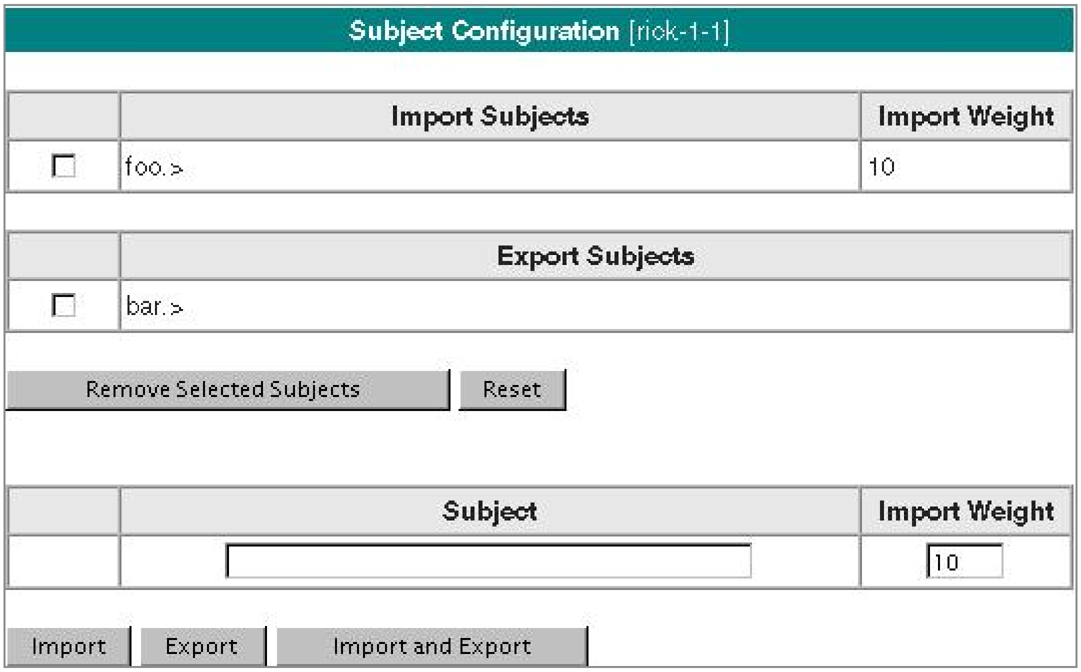
Subject Gating

This page lets you configure subject gating (import and export subjects) for a local network.

To display this page, click the name of a local network in a row of the **Local Network Interfaces Configuration page.**

For background information, see Subject Gating, and Subject Filtering with Wildcards.

Figure 35: rvrd Subject Configuration (Gating) Page



Item	Description
Import Subjects	This table lists import subjects. The local network can import subjects that match these names. You can remove a subject at any time.
Export Subjects	This table lists export subjects. The local network can export subjects that match these names. You can remove a subject at any time.
adding subjects	To add subjects, specify the subject string (which may contain wildcards) here, and click one of three buttons: • Import • Export • Import and Export See also Subject Import Weight.

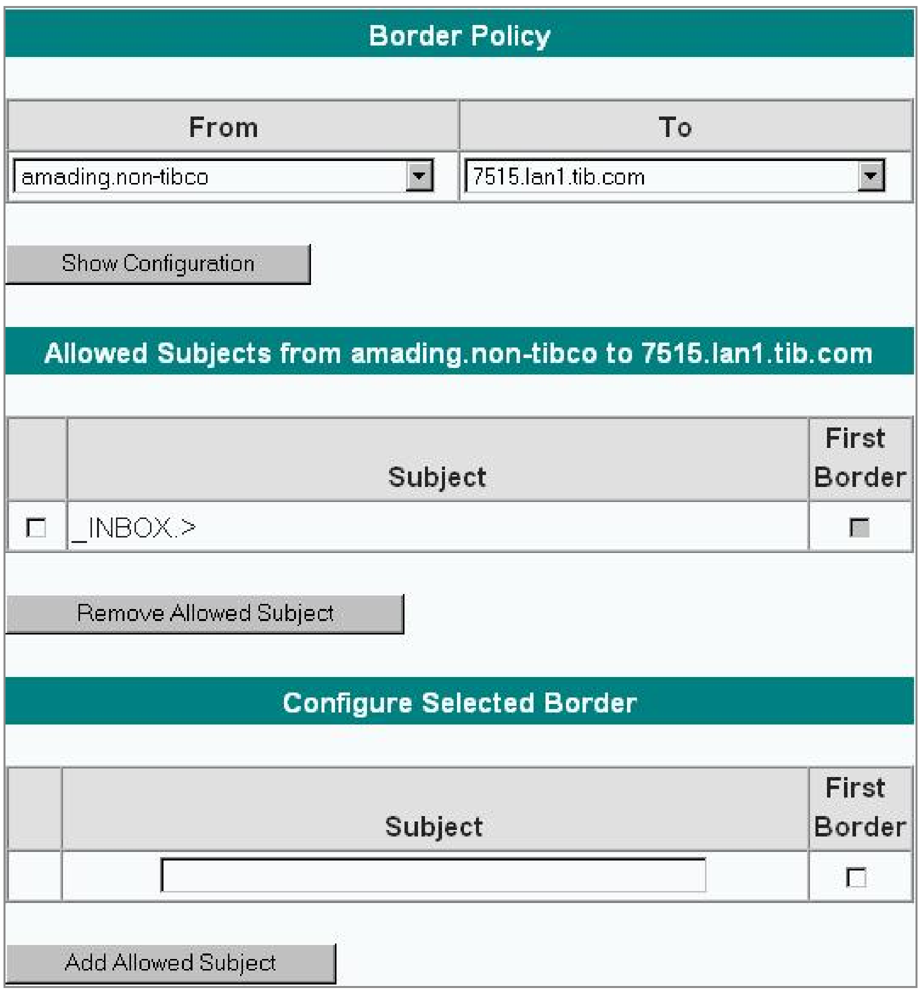
Border Policy

This page lets you configure policy for a border router—that is, the subjects that the router forwards between its interfaces.

To display this page, click the phrase **(border policy)** following the name of a border router in the **Routers page.**

For background information, see Policy (and the items that follow it), and Subject Filtering with Wildcards.

Figure 36: rvrd Border Policy Configuration Page



Item	Description
From	Choose a From interface from this menu.
To	Choose a To interface from this menu.
Show Configuration	To display the current set of subjects that the border router forwards from the From interface to the To interface, click this button. The Allowed Subjects table (below) then displays the current list for that ordered pair.
Allowed Subjects	This table lists subjects that the border router allows for the current pair of From interface and To interface. To update this table, click the Show Configuration button. You can remove a subject at any time.
Remove Allowed Subject	To delete an allowed subject, check its select box, and click this button.
Add Allowed Subject	To add an allowed subject, choose the From interface and To interface, then specify the subject string (which may contain wildcards) and click this button.
First Border	A border router can restrict a subject, forwarding only those messages that have not yet crossed another border. To restrict the new subject in this way, check the First Border box before adding the subject.

Neighbor Interfaces

This page lets you configure the potential neighbor connections of a routing table entry.

To display this page, click the number of neighbors in a row of the **Routers page.**

For background information, see Neighbors.

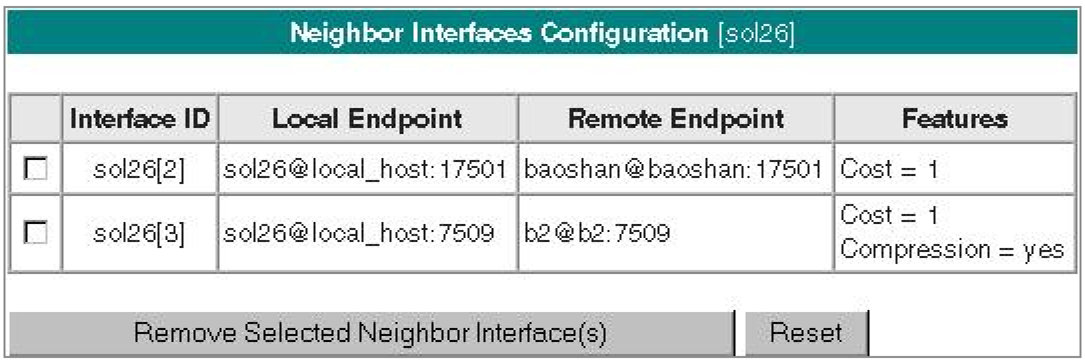
Note

This page is related to—but not the same as—the page described in Connected Neighbors.

Existing Neighbor Interfaces

The first part of this page is a table of existing neighbor interfaces—that is, interface specifications for potential neighbor connections to other routers.

Figure 37: rvrd Neighbor Interfaces Page—Existing



Item	Description
existing neighbor interfaces	The upper table lists configured neighbor interfaces. Each row represents one potential neighbor.
Interface ID	The name of this neighbor interface. `rvrd` generates this name automatically, incorporating the router name.
Local Endpoint	This three-part string denotes the local end of the potential neighbor link. It has the form: *router_name*@*host*:*TCP_connect_port* • *router_name* is the name of the local routing table entry. • *host* is a fixed token, `local_host`, which denotes the local `rvrd` host computer. (Note that this token does *not* denote

Item	Description
	the `LOCALHOST` loopback network address.) • *TCP_connect_port* is the TCP port where the local router accepts neighbor connection requests from remote routers.
Remote Endpoint	This three-part string denotes the remote end of the potential neighbor link. It has the form: *router_name*@*host*:*TCP_connect_port* • *router_name* is the name of the remote routing table entry. • *host* is the hostname or IP address of the remote `rvrd` host computer. • *TCP_connect_port* is the TCP port where the local router attempts to connect to remote routers. The token `Any` can appear in these three parts. For the semantics of this notation see Accept Any as Neighbor, and Seek Neighbor with Any Name. See also, Four Variations of the Form.
Features	This column lists optional features of this neighbor specification: • Cost: the path cost of this neighbor link (see Load Balancing) • Compression: this flag indicates whether this interface specifies data compression (see Data Compression) • SSL: this flag indicates whether this interface requires a TLS connection (see SSL Connection with Compression)

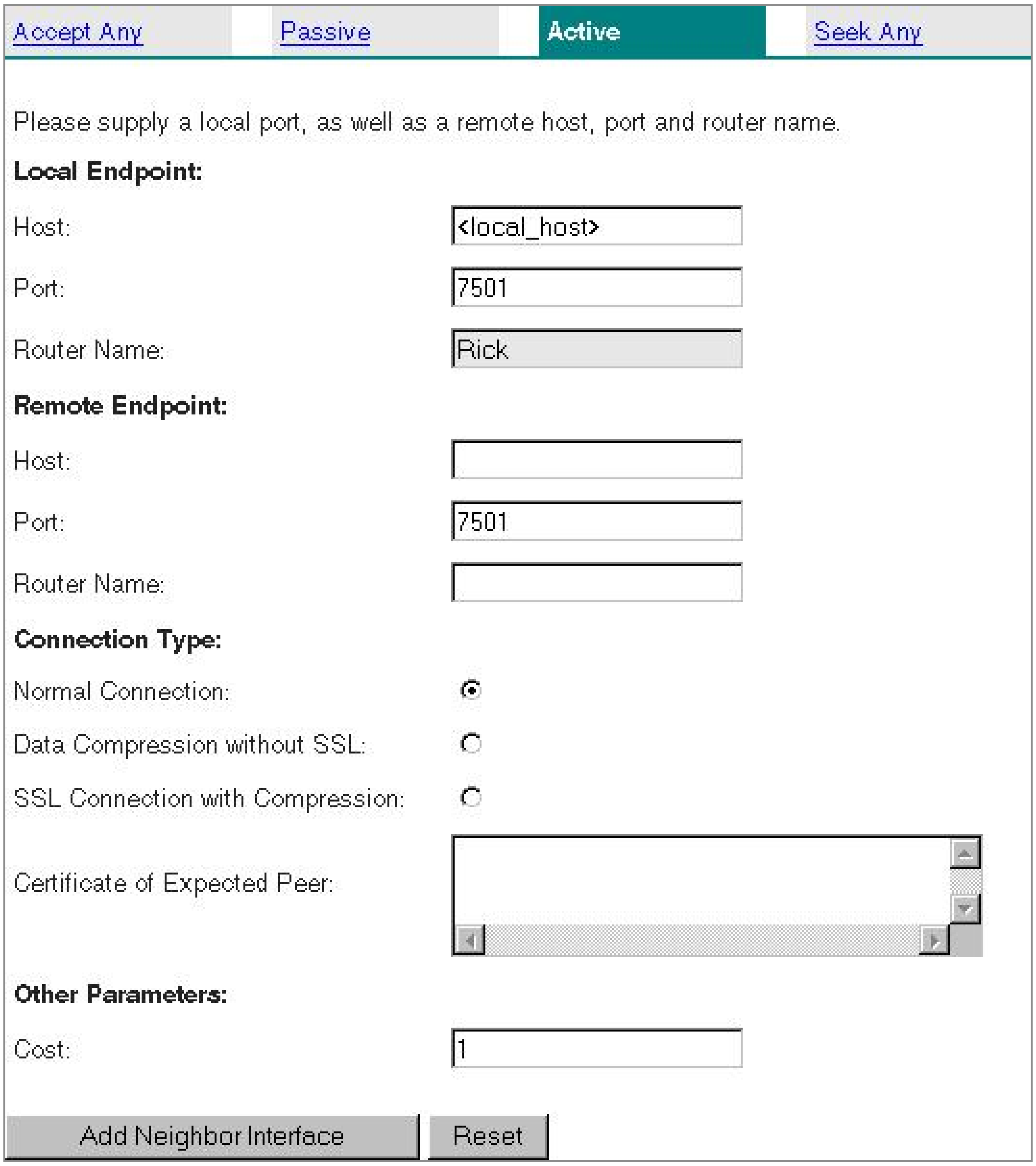
See Also

Router Name

Add New Neighbor Interface

The remainder of this page lets you complete a form to specify a new neighbor interface.

Figure 38: rvrd Neighbor Interface Configuration Form



Four Variations of the Form

Four buttons rearrange the form into four variations, each with a different meaning. In each variation, `rvrd` automatically fills in some fields, and leaves others empty for you to fill.

Four Neighbor Interface Configuration Forms

Item	Description
Accept Any	Use this variation of the form to specify a neighbor interface in which this routing daemon accepts neighbor connections from any other routing daemon. A distinguishing characteristic of *accept any* neighbors is a remote endpoint string in which the router name, the host and the port are all Any. **Restrictions:** • It is not possible to configure more than one *accept any* neighbor interface. • *Accept any* interfaces cannot use TLS neighbor connections. • Border routers cannot configure an *accept any* neighbor interface. For more information, see Accept Any as Neighbor.
Passive	Use this variation of the form to specify a neighbor interface in which the local router does not actively attempt to connect to the remote neighbor. Instead, it passively waits for the remote neighbor to request a connection. A distinguishing characteristic of passive neighbors is a remote endpoint string in which the router name is specified, but the host and port are Any. For more information, see Passive Neighbor.
Active	Use this variation of the form to specify a neighbor interface in which the local router actively attempts to connect to the remote neighbor. A distinguishing characteristic of active neighbors is a remote endpoint string in which the router name, the host and the port are all specified. For an example, see Active Neighbor.
Seek Any	Use this variation of the form to specify a neighbor interface in which this routing daemon attempts to connect to any remote routing daemon that matches the specification. A distinguishing characteristic of *seek any* neighbors is a remote endpoint string in which the router name is Any, but the host and the port are

Item	Description
	specified. In addition, the local endpoint port is Any. **Restrictions:** • It is illegal to configure two or more *seek any* neighbor interfaces with the same host. • *Seek any* interfaces cannot use TLS neighbor connections. • Border routers cannot configure a *seek any* neighbor interface. For more information, see Seek Neighbor with Any Name.

Items in the Neighbor Interface Configuration Form

This table describes the items in rvrd Neighbor Interface Configuration Form.

Item	Description
Local Endpoint	This three-part specification denotes the local end of the potential neighbor link: • *Router Name* is the name of the local routing table entry. `rvrd` always automatically fills in this name. • *Host* is a hostname or IP address corresponding to a network interface in the local `rvrd` host computer. For convenience, `rvrd` automatically fills in this field with the fixed token, `local_host`, which denotes the default network interface of the local `rvrd` host computer. (Note that this token does *not* denote the `LOCALHOST` loopback network address.) You may override this default value by typing an alternate hostname or IP address. • *Port* is the local TCP port where the local router accepts neighbor connection requests from remote routers. For more information, see Local Connect Port.
Remote Endpoint	This three-part specification denotes the remote end of the potential neighbor link:

Item	Description
	• *Router Name* is the name of the remote routing table entry. • *Host* is the hostname or IP address of the remote `rvrd` host computer. • *Port* is the remote TCP port where the local router attempts to connect to remote routers. For more information, see Remote Connection Information.
Normal Connection	With this option, the two neighbors neither compress data nor use TLS protocols for communication on the link between them.
Data Compression without SSL	With this option, the two neighbors compress data on the link between them. To enable compression, you must select this option on both neighbors. For more information, see Data Compression.
SSL Connection with Compression	With this option, the two neighbors communicate using both compression and TLS protocols. To enable TLS, you must select this option on both neighbors—otherwise they cannot establish a connection. This option appears only in the Passive and Active variations of the configuration form. Connection statistics are not available when neighbors connect using TLS. See also Router Connection Statistics. In older releases of the routing daemon, TLS and compression are mutually exclusive features. For backward compatibility with older neighbors, this feature degrades gracefully to TLS without compression.
Certificate of Expected Peer	In TLS protocols, the local router expects the remote router to present this certificate as evidence of its identity. Paste the text of the public certificate (in PEM encoding) in this field. This field appears only in the Passive and Active variations of the configuration form.
Cost	The path cost of this neighbor link (see Load Balancing).

Certificates

This page lets you configure the X.509 certificates that the routing daemon uses to identify itself.

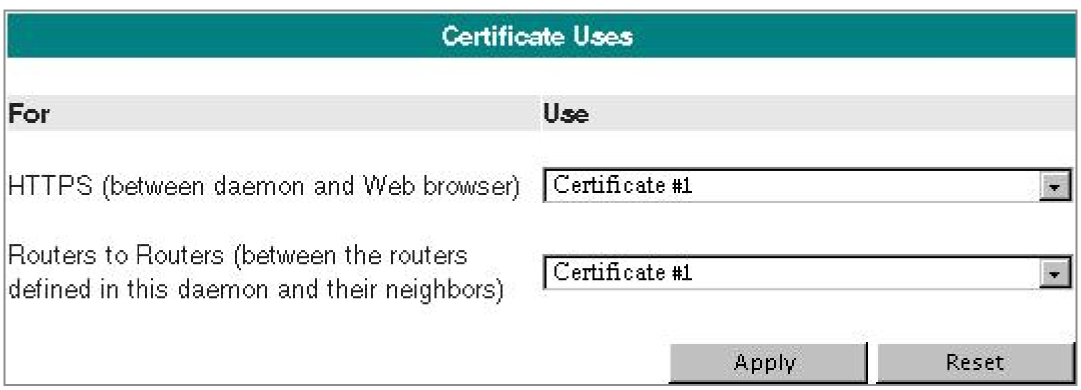
To display this page, click **Certificates** in the left margin of any page of the `rvrd` browser administration interface.

For background information, see Certificates and Security in TIBCO Rendezvous Concepts.

Each daemon process keeps a list of certificates it can use to identify itself. These certificates are numbered for easy reference. The first panel on this page determines which of these certificates the daemon uses for particular tasks. The remainder of the page lets you enter the certificates.

Certificate Uses

Figure 39: rvrd Certificate Uses Form



Item	Description
HTTPS	Set the certificate for the secure browser administration interface. To avoid security warnings from the web browser, distribute this certificate to authorized administrators. For security information, see Level of Trust—CA-Signed versus Self-Signed Certificates.

Item	Description
Routers to Routers	Set the certificate for secure TLS neighbor connections. Distribute this certificate to each applicable neighbor.

Certificate List

Figure 40: rvrd Certificate List

Item	Description
certificate number	Use this number to refer to the certificate in the Certificate Uses panel.
Add from File	Enter a file name and a private key password. When you click Add from File, the daemon reads the certificate with private key from the file. The file may be in either PEM encoding, or PKCS #12 format. See also Security Factors.
Add from Text	Paste the text of a certificate with private key. Enter a private key password. The certificate *must* be in PEM encoding. See also Security Factors.

Self-Signed Certificate

Each daemon process creates a self-signed certificate at start time, and registers it in the list as certificate #1. You may use that certificate as is, add other certificates to the list, or delete it and enter other certificates. For security information, see Level of Trust—CA-Signed versus Self-Signed Certificates.

This self-signed certificate expires one year after creation.

CA-Signed Certificate

You can also supply certificates signed by a certificate authority (CA). To use a CA-signed certificate, you must supply not only the certificate and private key, but also the CA's public certificate (or a chain of such certificates). Concatenate these items in one file or string. For more details, see CA-Signed Certificates.

CA-signed certificates expire at dates recorded within the certificate data.

Secure Daemons (rvsd and rvsrd)

These two daemons use TLS for secure connections to client program transports:

- `rvsd`, the Rendezvous secure communications daemon, corresponds to `rvd`
- `rvsrd`, the Rendezvous secure routing daemon, corresponds to `rvrd`

This section describes the security features of these two daemons, and details the parameters that differentiate them from their non-secure counterparts.

Secure Daemon Overview

This section describes the two daemons that offer secure client connections:

- `rvsd`, the Rendezvous secure communications daemon, corresponds to `rvd`. Rendezvous Daemon (rvd) describes `rvd`, the Rendezvous communications daemon.
- `rvsrd`, the Rendezvous secure routing daemon, corresponds to `rvrd`. Routing Daemon (rvrd) describes `rvrd`, the Rendezvous routing daemon.

Secure Connections

The two ordinary Rendezvous daemons, `rvd` and `rvrd`, communicate with clients over non-secure TCP connections. In contrast, their secure counterparts, `rvsd` and `rvsrd`, communicate with clients over TLS connections, allowing secure client communication over non-secure networks.

Restricting Access

Secure daemons restrict client access in three ways:

- Only authorized clients can connect to a secure daemon.
- Secure daemons restrict the combinations of network and UDP service over which client transports can communicate.
- Secure daemons limit the subject space that its clients can access.

Plaintext Communication

Although they ensure secure client connections, both secure daemons transmit messages as plaintext. That is, when they publish messages from clients to local networks, those messages are not encrypted.

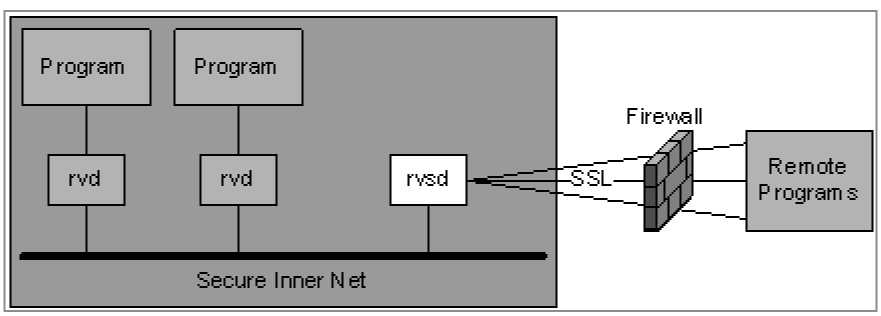
Motivation

Deploy secure daemons when clients must connect securely over a non-secure network. This section illustrates example situations involving remote clients.

rvsd

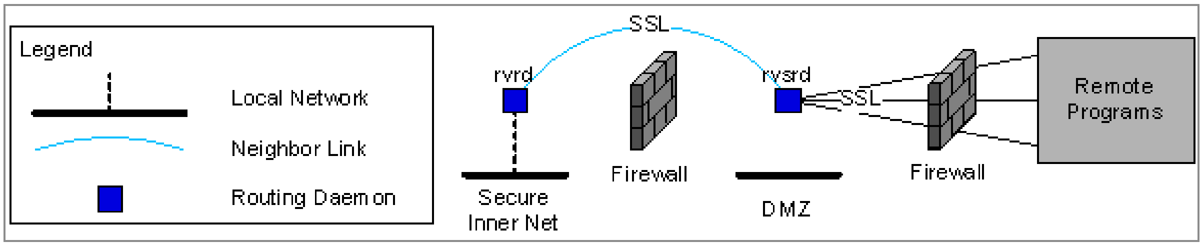
rvsd—Secure Connections across Single Firewall depicts a hub and spoke architecture. An `rvsd` hub runs on a firewall computer, and remote programs access the hub through secure TLS connections. This arrangement lets trusted remote programs communicate with servers and other programs inside the secure inner network. `rvsd` bars untrusted programs from connecting to it.

Figure 41: rvsd—Secure Connections across Single Firewall



rvsrd

Figure 42: rvsrd—Secure Connections across Double Firewall



rvsrd—Secure Connections across Double Firewall depicts a situation with two Rendezvous routing daemons configured to cross a double firewall. Remote programs initiate secure TLS connections to a secure routing daemon hub (`rvsrd`) within the outer firewall (DMZ network). A secure TLS neighbor link connects that secure routing daemon with an ordinary routing daemon (`rvrd`) in the secure inner network.

To configure secure neighbor links, see SSL Connection with Compression.

Preventing Multicast in the DMZ

To prevent `rvsrd` from multicasting client messages within the DMZ network, start `rvsrd` with the `-no-multicast` option. For background information, see Disabling Multicast.

`-no-multicast` is available starting with Rendezvous release 7.2. This feature replaces the following procedure, which was required in earlier releases:

- Configure `rvsrd` so that in *all* of its local networks, the network specification is the loopback address (IP address `127.0.0.1`). To configure, see Local Network Interfaces Configuration.
- Similarly limit the access of client transports to network and service pairs in which the network is the loopback address (IP address `127.0.0.1`). To configure, see Authorize Network and Service Pairs.

Users

Each secure daemon instance authorizes a set of trusted *users*:

- The secure daemon allows a client transport to connect *only* if the client presents valid identification as an authorized user.
- User identification can be either a certificate, or a username and password.

To authorize a user, see Users.

To connect to a secure daemon as a user, see Secure Daemon programming language API.

Certificate Identification

The secure daemon can register zero or more X.509 public key identity certificates per user. The secure daemon limits access to user programs that can sign TLS protocol messages with a corresponding private key.

The secure daemon accepts all certificates in either PEM encoding or PKCS #12 format.

For more details, see CA-Signed Certificates.

Username and Password Identification

The secure daemon registers at most one password per user. The secure daemon limits access to user programs that supply a correct pair of username and password strings.

Important For important information about password security, see Security Factors.

Syntax

Username and password strings must conform to these syntax specifications:

- The username must be less than 128 characters. The combined length of the username and password must be less than 250 characters.
- These strings must consist of printable characters only, from any character set.

  Dot (`.`), star (`*`), and greater-than (`>`) characters are permitted. However, we recommend against using them except in legacy situations (for example, where such names are already in use in another security system).
- These strings cannot contain two adjacent space characters.

- The first and last characters must not be spaces.
- These strings must contain at least one non-space character.
- These strings cannot contain embedded newline characters (`\n`) or null characters.
- The null or empty string is not a legal username nor password.

Limiting Access

A secure daemon controls user access to local communications. Administrators can limit access at two levels of granularity:

- Network and service
- Subject

Network and Service Authorization

Each secure daemon allows its users to communicate over a set of local networks. Two parameters together define a local network:

- Network Specification

  For details, see Constructing the Network Parameter.

- UDP Service

  For details, see Specifying the UDP Service.

You must explicitly authorize each local network by specifying these two parameters. To authorize a local network, see Authorize Network and Service Pairs.

Users can communicate *only* on the local networks that you authorize. A user program cannot create a client transport that specifies an unauthorized local network (the transport create call produces an error status code).

Default Local Network

As an administrator, you can designate a *default* local network. A client transport that does not specify particular network and service parameters automatically communicates over this default local network; see Default Network and Service.

Subject Authorization

Each secure daemon allows its users to communicate using a set of Rendezvous subject names.

- Subjects authorized for *sending* can flow from client transports out to local networks.

  A client transport that sends a message with an unauthorized subject does not receive any error indication; instead, the secure daemon silently discards the message.

- Subjects authorized for *listening* can flow to client transports from local networks.

  A client transport that creates a listener with an unauthorized subject does not receive any error indication—but the resulting listener object never receives any messages.

Subject authorization applies equally to all users and all local networks.

All `_INBOX` subjects are implicitly authorized. It is not necessary to explicitly authorize `_INBOX` subjects.

To authorize secure daemon subjects, see Authorize Subjects.

Note

If clients use fault tolerance, certified message delivery, or distributed queue features, you must authorize the appropriate administrative subjects; see these tables:

- Critical Subjects for Certified Delivery
- Critical Subjects for Fault Tolerance
- Critical Subjects for Distributed Queues

Security Factors

Store Files

Important

The secure daemon store file contains very sensitive information. Store it on the local file system of the secure daemon’s host computer, with tight file access, in a physically secure environment. Ensure timely backup to secure media.

Core-Dump Files

Important

Secure daemon process storage contains sensitive information in unencrypted form. Similarly, user program storage can contain passwords or private key data. It is essential to deny access to these processes and their core image files. We strongly recommend arranging operating system parameters to prevent creation of core files.

To guard against attacks, take these precautions:

- Configure the operating environment to avoid making core dumps.
- Configure the operating environment to prevent access to process memory (if possible).
- Ensure that file system storage is secure.

Daemon Certificates

Important

Administrators must implement a secure mechanism to distribute the secure daemon’s public certificate to users (that is, either programmers of client programs or end users).

Ensure that users verify daemon certificates before using them with client programs. Ensure that users keep daemon certificates in files that are secure from unauthorized modification or tampering. Remember, a false certificate can give a rogue daemon access to user passwords.

CA-Signed Certificates

Rendezvous does *not* support separating a leaf certificate from its signing CA certificate. When arranging certificate data, you may supply either a self-signed certificate, or a complete certificate trust chain, including leaf, intermediate (which are optional), and root certificates—all in one certificate data file. In either case, the entire certificate chain is contained in one package, and Rendezvous components verify trust by comparing the entire package.

To better understand the way in which Rendezvous uses certificates and certificate trust chains, compare it to the familiar model of web browser security.

In the familiar model, web browsers generally store a set of certificates representing trusted certificate authorities (CAs), and use them to deduce the authenticity of many certificates—any certificate signed using one of those trusted CA certificates.

In contrast, to authenticate a user (or another daemon), Rendezvous secure daemons require that a client-supplied certificate must exactly match a trusted certificate previously stored with the daemon. Daemons use certificates to verify digital signatures and message integrity, but they do *not* use CA certificates to authenticate client certificates. Similarly, Rendezvous clients verify certificates from Rendezvous daemons by matching them against trusted certificates previously registered with the client program.

If you require CA-signed certificates, or if your organization already uses CA-signed certificates, you may use them by packaging each one together with its CA root certificate and intermediate certificates—a complete trust chain for each certificate. You can use standard certificate utilities to create certificate files in the appropriate encoding formats.

Level of Trust—CA-Signed versus Self-Signed Certificates

When client connects to a daemon, both forms of certificate (CA-signed and self-signed) represent equivalent levels of trust.

- The daemon accepts the client's certificate only if the daemon is configured to accept that certificate as the identity of a valid user, or as the identity of another trusted daemon.
- The client accepts the daemon's certificate only if the client has previously registered that certificate as the identity of a trusted daemon.

In these situations, self-signed certificates can be more convenient than CA-signed certificates, with no degradation in security.

However, when a browser connects to a daemon's browser administration interface, CA-signed daemon certificate chains do represent a stronger level of trust than self-signed daemon certificates. Furthermore, using CA-signed daemon certificates can help avoid browser security warnings.

Passwords

Important

Private key files use password-encryption for security. Nonetheless, these files are important points of vulnerability.

To guard against attacks, ensure that file system storage is secure, and keep all passwords secure.

- Do not store passwords in non-secure files or on non-secure file systems.
- Control access to sensitive files—even when those files are password-encrypted.
- Never hard-code passwords in application programs, nor accept them as command line parameters.
- Code programs to erase passwords from process storage before exiting.

- Never write passwords in convenient locations.
- Never send passwords in plaintext messages.
- Choose passwords carefully.

Behavioral Differences

Secure daemons exhibit slight differences in behavior from their non-secure counterparts. This section summarizes those differences.

Automatic Start and Stop

`rvd` can start either automatically or by explicit command. In contrast, administrators must start `rvsd` by explicit command.

`rvd` can stop automatically after an interval in which it has no clients (see rvd, and -no-permanent). In contrast, `rvsd` does *not* stop automatically.

Subject Gating

Secure daemons are silent when subject gating parameters preclude send or listen operations:

- Subjects authorized for *sending* can flow from client transports out to local networks.

  A client transport that sends a message with an unauthorized subject does not receive any error indication; instead, the secure daemon silently discards the message.

- Subjects authorized for *listening* can flow to client transports from local networks.

  A client transport that creates a listener with an unauthorized subject does not receive any error indication—but the resulting listener object never receives any messages.

Default Network and Service

Secure daemons and non-secure daemons behave differently when a client transport specifies a default value (that is, null) for its network or service parameter. Non-secure daemons use *external* defaults; see Specifying the UDP Service and Constructing the Network Parameter. In contrast, secure daemons use *internal* defaults—which you can configure using the browser administration interface; see Default Network and Service.

Browser Connections

Secure daemons automatically open both HTTP and HTTPS ports for browser administration interface connections—unless you specify otherwise. When an HTTPS connection is available, the daemon uses it; that is, whenever possible, it transfers non-secure HTTP communication over to its secure HTTPS connection.

You can block the secure HTTPS connection by specifying `-http-only`, which leaves only the non-secure HTTP connection.

You can block all browser administration interface connections by specifying `-no-http`.

See Also

Network and Service Authorization

Default Network and Service

rvsd

Command

Syntax

```
rvsd  -store filename
      [-http [ip_address:]http_port]
      [-https [ip_address:]https_port]
      [-http-only]
      [-https-only]
      [-no-http]
```

```
[-license url]
[-no-permanent]
[-listen [socket_protocol:|ip_address:]tcp_port]
[-no-lead-wc | -lead-wc]
[-no-multicast]
[-reliability time]
[-max-consumer-buffer size]
[-rxc-max-loss loss]
[-rxc-recv-threshold bps]
[-rxc-send-threshold bps]
[-reuse-port inbox_port]
[-logfile log_filename]
[-log-max-size size]
[-log-max-rotations n]
[-log-config config_log_filename]
[-foreground]
[-udp-ttl hops]
[-tls-min-proto-version version]
[-tls-max-proto-version version]
[-tls-ciphers string1:string2:stringN]
[-tls-ciphersuites name1:name2:nameN]
[-no-wc]
```

Purpose

The command `rvsd` starts the Rendezvous secure communications daemon process—the secure counterpart to rvd.

Remarks

This section describes only those aspects where `rvsd` differs from rvd. For details that both daemons share, see rvd.

Although `rvd` usually starts automatically, administrators must start `rvsd` by explicit command.

Command Line Parameters

rvsd

Parameter	Description
`-store` *filename*	This file contains the security parameters that configure `rvsd`. `rvsd` reads this file when the process starts, and writes this file each time you change the configuration using the browser administration interface. The secure daemon store file contains very sensitive information. Store it on the local file system of the secure daemon's host computer, with tight file access, in a physically secure environment. Ensure timely backup to secure media. See also Store Files.
`-http` *ip_address*`:`*http_port* `-http` *http_port* `-https` *ip_address*`:`*https_port* `-https` *https_port*	The browser administration interface accepts connections on this HTTP or HTTPS port. Permit administration access only through the network interface specified by this IP address. To limit access to a browser on the `rvsd` host computer, specify `127.0.0.1` (the local host address). When the IP address is absent, the daemon accepts connections through any network interface on the specified HTTP or HTTPS port. If the explicitly specified HTTP port is already occupied, the program exits. If the explicitly specified HTTPS port is already occupied, the program selects an ephemeral port. When the `-http` parameter is entirely absent, the default behavior is to accept connections from any computer on HTTP port `7580`; If this default port is unavailable, the operating system assigns an ephemeral port number.

Parameter	Description
	When the `-https` parameter is entirely absent, the default behavior is to accept secure connections from any computer on an ephemeral HTTPS port. In all cases, the program prints (in its start banner and log file) the actual HTTP and HTTPS ports where it accepts browser administration interface connections.
`-http-only`	Disable HTTPS (secure) connections, leaving only an HTTP (non-secure) connection.
`-https-only`	Disable HTTP (non-secure) connections, leaving only an HTTPS (secure) connection.
`-no-http`	Disable *all* HTTP and HTTPS connections, overriding `-http` and `-https`.
`-license` *`url`*	The URL to a TIBCO Activation Service in the form of `https://url[?fp=activation service fingerprint]`. A fingerprint of the Activation Service can be included in the URL for peer verification. The URL can also be supplied using the environment variable `TIBRV_LICENSE`. If the URL is not explicitly configured using `-license` or `TIBRV_LICENSE`, the Rendezvous daemon will search its environment PATH for a license file named `tibrvlic.bin` and, failing that, will attempt to contact an Activation Service at `https://tib-activate:7070`. **Note:** Production environments must use a TIBCO Activation Service for TIBCO product activation. Using a license file to configure the Rendezvous daemon is only supported for development environments. The URL format for a license file is `file://pathname`.
`-listen` *tcp_port* `-listen` *ip_address*`:`*tcp_port* `-listen` *socket_protocol*`:`*tcp_port*	`rvsd` (and by extension, `rvsrd` operating within the local network) opens an TLS client socket to establish communication between itself and its client programs.

Parameter	Description
	The `-listen` parameter specifies the TLS port where the Rendezvous daemon listens for connection requests from client programs. This `-listen` parameter of the secure daemon corresponds to the `daemon` parameter of the transport creation call (they must specify the same TLS port number). The IP address specifies the network interface through which this daemon accepts TLS connections. To bar connections from remote programs, specify IP address `127.0.0.1` (the loopback interface). When the IP address is absent, the daemon accepts connections from any computer on the specified TLS port. When this parameter is entirely absent, the default behavior is to accept connections from any computer on TLS port 7500. For more detail about the choreography that establishes conduits, see Daemon Client Socket—Establishing Connections. **Warning** This parameter does *not* correspond to the `service` parameter of the transport creation call—but rather to the `daemon` parameter.
`-no-permanent`	If present (or when `rvd` starts automatically), `rvd` exits after 1 minute during which no transports are connected to it. If not present, `rvd` runs indefinitely until terminated. This parameter is not available with IPM.
`-no-lead-wc`	Sending to subjects with lead wildcards (for example, >

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td><code>-lead-wc</code></td><td>or <code>*.foo</code>) can cause unexpected behavior in some applications, and cause network instability in some configurations. This option lets you selectively screen wildcard sending.

When <code>-no-lead-wc</code> is present, the daemon quietly rejects client requests to send outbound messages to subjects that contain wildcards in the lead element. The daemon does not report excluded messages as errors.

When <code>-lead-wc</code> is present (or when neither flag is present), the daemon allows sending messages to subjects with lead wildcards.

This parameter is not available with IPM.</td></tr>
<tr><td><code>-log-config</code> config_log_filename</td><td>Send duplicate log output to this file for log items that record configuration changes. The daemon never rotates nor removes this special log file. Instead, this file remains as a record of all configuration changes.

When absent, the default is <code>stderr</code>.</td></tr>
<tr><td><code>-reliability</code> time</td><td rowspan="7">These parameters are the same as for rvd.

For details, see Command Line Parameters.</td></tr>
<tr><td><code>-max-consumer-buffer</code> size</td></tr>
<tr><td><code>-rxc-max-loss</code> loss

<code>-rxc-recv-threshold</code> bps

<code>-rxc-send-threshold</code> bps</td></tr>
<tr><td><code>-reuse-port</code> inbox_port</td></tr>
<tr><td><code>-logfile</code> log_filename</td></tr>
<tr><td><code>-log-max-size</code> size</td></tr>
<tr><td><code>-foreground</code></td></tr>
</table>

Parameter	Description
`-udp-ttl` *hops*	
`-tls-min-proto-version` *version* `-tls-max-proto-version` *version*	Set the minimum or maximum supported protocol versions for the ctx using OpenSSL calls `SSL_CTX_set_min_proto_version` and `SSL_CTX_set_max_proto_version`.
`-tls-ciphers` *string1:string2:stringN*	Set the list of available ciphers (TLSv1.2 and earlier) using OpenSSL call `SSL_CTX_set_cipher_list`.
`-tls-ciphersuites` *name1:name2:nameN*	Configure the available TLSv1.3 ciphersuites using OpenSSL call `SSL_CTX_set_ciphersuites`.
`-no-wc`	Silently drop any messages published by clients that contain any wild card tokens.
`-pwd-hash password-string`	Takes the password-string, prints a base64 encoded PBKDF2 hash, and then exits. Rendezvous daemons do not store passwords, they convert the user-supplied plain text passwords into a secure hash first. By supplying a secure hash instead of a plain-text Administrator password in XML/JSON configuration, users remove Rendezvous daemons from the chain of custody of the password. Secure hashes are one-way, and passwords cannot be recovered from the secure hash. You are responsible for securely storing the original password.

rvsrd

Command

Syntax

```
rvsrd  -store filename
       [-http [ip_address:]http_port]
       [-https [ip_address:]https_port]
       [-http-only]
       [-https-only]
       [-no-http]
       [-license url]
       [-idle]
       [-listen [socket_protocol:|ip_address:]tcp_port]
       [-no-permanent]
       [-no-lead-wc | -lead-wc]
       [-no-multicast]
       [-reliability time]
       [-max-consumer-buffer size]
       [-rxc-max-loss loss]
       [-rxc-recv-threshold bps]
       [-rxc-send-threshold bps]
       [-compress-level level]
       [-reuse-port inbox_port]
       [-logfile log_filename]
       [-log-max-size size]
       [-log-max-rotations n]
       [-log-config config_log_filename]
       [-foreground]
       [-udp-ttl hops]
       [-tls-min-proto-version version]
       [-tls-max-proto-version version]
       [-tls-ciphers string1:string2:stringN]
       [-tls-ciphersuites name1:name2:nameN]
       [-no-wc]
```

Purpose

The command `rvsrd` starts the Rendezvous secure routing daemon process—the secure counterpart to rvrd.

Remarks

This section describes only those aspects where `rvsrd` differs from rvrd. For details that both daemons share, see rvrd.

Administrators must start `rvsrd` by explicit command.

Command Line Parameters

rvsrd

Parameter	Description
`-store` *filename*	This file contains the security parameters that configure `rvsrd`, as well as the routing table entry and parameters that configure its routing daemon behavior. `rvsrd` reads this file when the process starts, and writes this file each time you change the configuration using the browser administration interface. The secure daemon store file contains very sensitive information. Store it on the local file system of the secure daemon's host computer, with tight file access, in a physically secure environment. Ensure timely backup to secure media. See also Store Files.
`-http` *ip_address*`:`*http_port* `-http` *http_port* `-https` *ip_address*`:`*https_port* `-https` *https_port*	The browser administration interface accepts connections on this HTTP or HTTPS port. Permit administration access only through the network interface specified by this IP address. To limit access to a browser on the `rvsrd` host computer, specify `127.0.0.1` (the local host address). When the IP address is absent, the daemon accepts connections through any network interface on the specified HTTP or HTTPS port. If the explicitly specified HTTP port is already occupied, the program exits. If the explicitly specified HTTPS port is already occupied, the program selects an ephemeral port. When the `-http` parameter is entirely absent, the default behavior is to accept connections from any computer on HTTP port `7580`; If this default port is unavailable, the operating system assigns an ephemeral port number.

Parameter	Description
	When the `-https` parameter is entirely absent, the default behavior is to accept secure connections from any computer on an ephemeral HTTPS port. In all cases, the program prints (in its start banner and log file) the actual HTTP and HTTPS ports where it accepts browser administration interface connections.
`-http-only`	Disable HTTPS (secure) connections, leaving only an HTTP (non-secure) connection.
`-https-only`	Disable HTTP (non-secure) connections, leaving only an HTTPS (secure) connection.
`-no-http`	Disable *all* HTTP and HTTPS connections, overriding `-http` and `-https`.
`-license` *url*	The URL to a TIBCO Activation Service in the form of `https://url[?fp=activation service fingerprint]`. A fingerprint of the Activation Service can be included in the URL for peer verification. The URL can also be supplied using the environment variable `TIBRV_LICENSE`. If the URL is not explicitly configured using `-license` or `TIBRV_LICENSE`, the Rendezvous daemon will search its environment PATH for a license file named `tibrvlic.bin` and, failing that, will attempt to contact an Activation Service at `https://tib-activate:7070`. **Note:** Production environments must use a TIBCO Activation Service for TIBCO product activation. Using a license file to configure the Rendezvous daemon is only supported for development environments. The URL format for a license file is `file://pathname`.
`-listen` *tcp_port* `-listen` *ip_address*:*tcp_port* `-listen` *socket_protocol*:*tcp_port*	`rvsd` (and by extension, `rvsrd` operating within the local network) opens an TLS client socket to establish communication between itself and its client programs.

Parameter	Description
	The `-listen` parameter specifies the TLS port where the Rendezvous daemon listens for connection requests from client programs. This `-listen` parameter of the secure daemon corresponds to the `daemon` parameter of the transport creation call (they must specify the same TLS port number). The IP address specifies the network interface through which this daemon accepts TLS connections. To bar connections from remote programs, specify IP address `127.0.0.1` (the loopback interface). When the IP address is absent, the daemon accepts connections from any computer on the specified TLS port. When this parameter is entirely absent, the default behavior is to accept connections from any computer on TLS port 7500. For more detail about the choreography that establishes conduits, see Daemon Client Socket—Establishing Connections. **Warning** This parameter does *not* correspond to the `service` parameter of the transport creation call—but rather to the `daemon` parameter.
`-no-permanent`	If present (or when `rvd` starts automatically), `rvd` exits after 1 minute during which no transports are connected to it. If not present, `rvd` runs indefinitely until terminated. This parameter is not available with IPM.
`-no-lead-wc`	Sending to subjects with lead wildcards (for example, >

Parameter	Description
`-lead-wc`	or `*.foo`) can cause unexpected behavior in some applications, and cause network instability in some configurations. This option lets you selectively screen wildcard sending. When `-no-lead-wc` is present, the daemon quietly rejects client requests to send outbound messages to subjects that contain wildcards in the lead element. The daemon does *not* report excluded messages as errors. When `-lead-wc` is present (or when neither flag is present), the daemon allows sending messages to subjects with lead wildcards. This parameter is not available with IPM.

Parameter	Description
`-idle`	These parameters are the same as for rvrd. For details, see Command Line Parameters.
`-reliability` *time*	
`-max-consumer-buffer` *size*	
`-rxc-max-loss` *loss* `-rxc-recv-threshold` *bps* `-rxc-send-threshold` *bps*1	
`-compress-level` *level*	
`-reuse-port` *inbox_port*	
`-logfile` *log_filename*	
`-log-max-size` *size* -log-max-rotations *n*	
`-log-config` *config_log_filename*	
`-foreground`	
`-udp-ttl` *hops*	
`-tls-min-proto-version` *version* `-tls-max-proto-version` *version*	Set the minimum or maximum supported protocol versions for the ctx using OpenSSL calls `SSL_CTX_set_min_proto_version and SSL_CTX_set_max_proto_version`.
`-tls-ciphers` *string1:string2:stringN*	Set the list of available ciphers (TLSv1.2 and earlier) using OpenSSL call `SSL_CTX_set_cipher_list`.
`-tls-ciphersuites` *name1:name2:nameN*	Configure the available TLSv1.3 ciphersuites using OpenSSL call `SSL_CTX_set_ciphersuites`.
`-no-wc`	Silently drop any messages published by clients that

Parameter	Description
	contain any wild card tokens.
`-pwd-hash password-string`	Takes the password-string, prints a base64 encoded PBKDF2 hash, and then exits. Rendezvous daemons do not store passwords, they convert the user-supplied plain text passwords into a secure hash first. By supplying a secure hash instead of a plain-text Administrator password in XML/JSON configuration, users remove Rendezvous daemons from the chain of custody of the password. Secure hashes are one-way, and passwords cannot be recovered from the secure hash. You are responsible for securely storing the original password.

Browser Administration Interface—rvsd and rvsrd

The browser administration interface lets you control `rvsd` and `rvsrd` from a web browser. You can configure their operating parameters and view internal data structures.

This section describes only those pages specific to the secure daemons. For information about pages they share with their non-secure counterparts, see Browser Administration Interface — rvd, and Browser Administration Interface—rvrd.

Navigation

All browser administration interface pages display a navigation panel at the left side of the page. Use these links to display other pages.

Figure 43: rvsd Navigation Panel

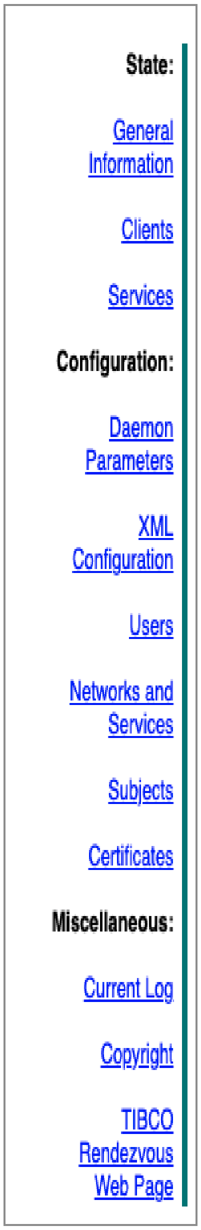
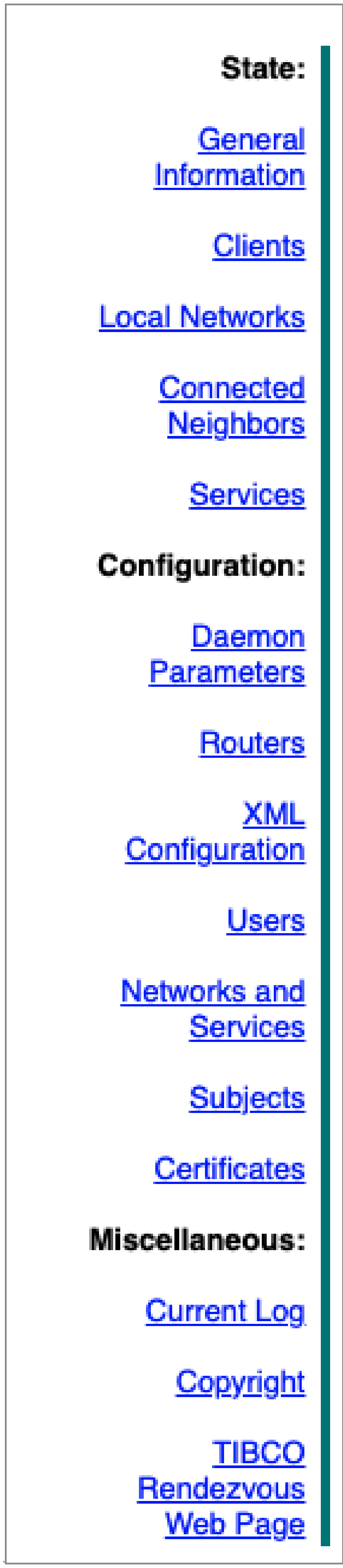


Figure 44: rvsrd Navigation Panel


State:
General Information
Clients
Local Networks
Connected Neighbors
Services
Configuration:
Daemon Parameters
Routers
XML Configuration
Users
Networks and Services
Subjects
Certificates
Miscellaneous:
Current Log
Copyright
TIBCO Rendezvous Web Page

Category	Item	Description
State	General Information	This page displays information about an `rvsd` or `rvsrd` process; see General Information.
	Clients	This page summarizes the client transports; see Clients.
	Local Networks	This page summarizes the local networks of a router; see Local Networks.
	Connected Neighbors	This page summarizes the actual neighbor connections of a router; see Connected Neighbors.
	Services	This page summarizes network services activity; see Services.
Configuration	Daemon Parameters	This page lets you configure parameters that control configuration access and secure default values for service and network parameters; see Daemon Parameters. For `rvsrd`, this page also configures router logging; see Logging.
	Routers	This page lets you configure routers. You can access additional configuration pages through links on this page. See Routers, and the sections that follow it.
	XML Configuration	This page lets you view the current configuration as an XML document, and reconfigure the component by submitting an edited XML document.
	Users	These pages let you register authorized users; see Users.
	Networks and Services	This page lets you configure the network and service pairs that client transports can use for Rendezvous communication; see Authorize Network and Service Pairs.

Category	Item	Description
	Subjects	This page lets you configure the subjects that client transports of a secure daemon can use for sending or listening; see Authorize Subjects.
	Certificates	This page lets you configure certificates that the daemon uses to identify itself in secure protocols. See Certificates.
	Log Out	This item logs out the current user or administrator. See Log Out.
Miscellaneous	Current Log	This page displays the most recent 4 kilobytes from the log file.
	Copyright	The Rendezvous copyright page.
	TIBCO Rendezvous Web Page	The product page from the TIBCO web site.

General Information

`rvsd` and `rvsrd` (like all Rendezvous components) display information about themselves on this page.

To display this page, click **General Information** in the left margin of any page of the secure daemon browser administration interface.

General Information

component:	rvsd
version:	8.6.0
license ticket:	0
host name:	igor.local
user name:	jpenning
IP address:	127.0.0.1
client port:	7500
IPC pathname:	/tmp/tibco/ipc.7500
network services:	0
store file:	store.1
process ID:	16937
managed:	no
control channel:	disabled
inbox port:	0

Item	Description
component	The name of the program—`rvsd` or `rvsrd`.
version	Version number of the program.
license ticket	The license ticket that validates this process.
host name	The hostname of the computer where the daemon process runs. Notice that the daemon process can run on one computer, while you access its browser interface from another computer.
username	The user who started the daemon process.
IP address	The IP address of the computer where the daemon process runs.
client port	The TLS port where the daemon listens for client connections.
network services	The number of network services on which this daemon's clients communicate.
store file	File name of the daemon's store file; see the command line parameter –

Item	Description
	store for rvsd, and for rvsrd.
process ID	The operating system's process ID number for the component.
managed	Not applicable.
control channel	Not applicable.
inbox port	When the daemon reuses service ports, this field displays the unique inbox port. When the daemon does not reuse service ports, this field displays zero.

Daemon Parameters

This page lets you configure parameters that affect overall daemon security.

To display this page, click **Daemon Parameters** in the left margin of any page of the secure daemon browser administration interface.

For rvsd, this page contains two areas—Administrator and Password panel, and a Default Network and Service panel. For rvsrd, this page adds a third panel for logging parameters; see Logging.

Administrator and Password

Figure 45: Secure Daemon Administrator and Password Area

Only authorized personnel have administrative access to secure daemons. (In contrast, to configure client program usernames, see Users.)

When administrator identification information is *not* set, anyone who can connect to the browser administration interface can examine and reconfigure the daemon. This arrangement can be useful during initial configuration and testing phases. However, during regular operation we recommend limiting access.

Once administrator identification information is registered, the browser administration interface is locked against unauthorized access. The daemon prompts administrators to prove identity by typing a name and password. After providing proper identification, an authorized administrator is logged in, and has complete access to configure the daemon. If the administrator does not provide proper identification, the browser displays the General Information page and continues to prompt for a correct name and password.

Warning

Browsers remember administrator name and password information for the duration of the browser process. Merely closing the browser *window* does not erase this information. To guard against intruders you must terminate the browser *process* (*all* its windows).

Primary Administrator

The first administrator to register is called the *primary administrator*. In addition to configuring the daemon, the primary administrator can also add, delete and modify identification information pertaining to the other administrators.

Each daemon configuration can store up to 16 additional administrator name and password pairs (after the primary administrator).

One Administrator Session

Each daemon process permits only one administrator session at a time. When one administrator is logged in, other administrators are locked out; this prevents conflicts in which two administrators attempt to modify the configuration at the same time. To terminate a administrator session, see Log Out (below).

Item	Description
Name	Type a name string.
Password	Type a password string.
Confirm Password	Type the password again.
Add/Update	Specify a name and password, then click this button to add a new user. This action is available only to the primary administrator.
Delete	Click this button to delete administrator identification information. This action is available only to the primary administrator. Deleting the primary administrator also deletes all other administrators.

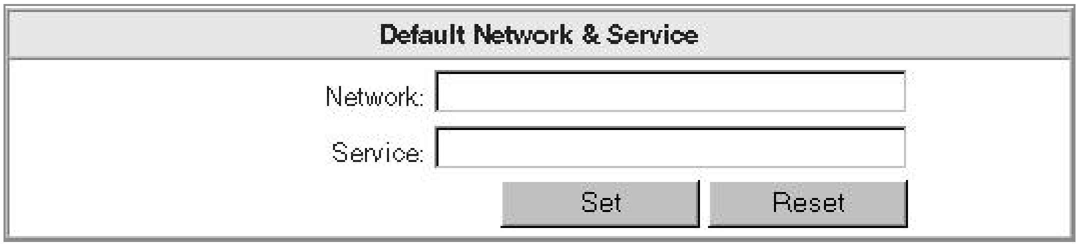
Log Out

To end an administrative session, click **Log Out** in the left margin of the browser administration interface. This item appears only when you are logged in as an Administrator.

Daemons automatically log out administrator sessions that have been idle for 10 minutes.

Default Network and Service

Figure 46: Secure Daemon Default Network and Service



Secure daemons and non-secure daemons behave differently when a client transport specifies a default value for its network or service parameter. Non-secure daemons use external defaults; see Specifying the UDP Service and Constructing the Network Parameter. In contrast, secure daemons use internal defaults—which you can configure using this panel.

Note

Unless you explicitly set values for these default parameters, they remain null—indicating the absence of any default value.

When either default is absent, the secure daemon will refuse connections from programs that rely on the default. As a result the transport creation call in the program fails.

For background information, see Network and Service Authorization.

Item	Description
Network	Type the default network.
Service	Type the default UDP service.

See Also

Default Network and Service

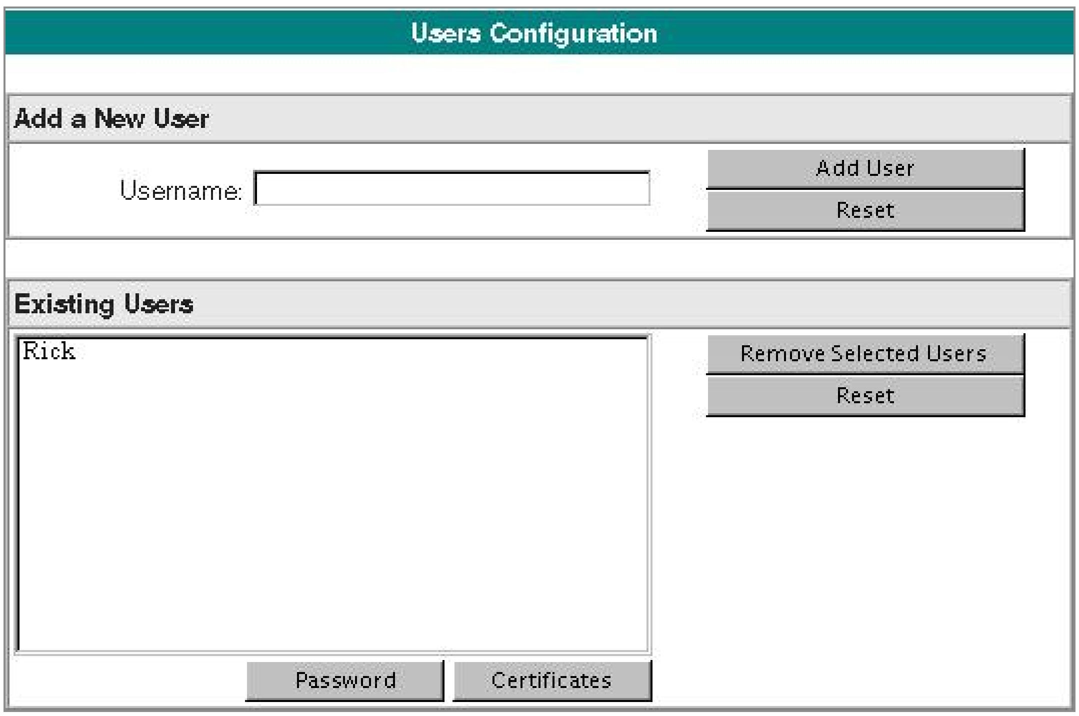
Users

This page lets you configure the set of users that can connect to a secure daemon. (In contrast, to configure administrative users, see Administrator and Password.)

To display this page, click **Users** in the left margin of any page of the secure daemon browser administration interface.

For background information, see Users.

Figure 47: Secure Daemon Users Page



Add a New User

This panel lets you create new users.

Item	Description
User Name	Required. Every user must have a unique name, distinct from every other user that this secure daemon administers.

Item	Description
	For syntax rules governing usernames, see Username and Password Identification. To add a user, type the username, and click the Add User button.

Existing Users

This list displays the users currently authorized to connect to the secure daemon.

Buttons operate on selected users from the list.

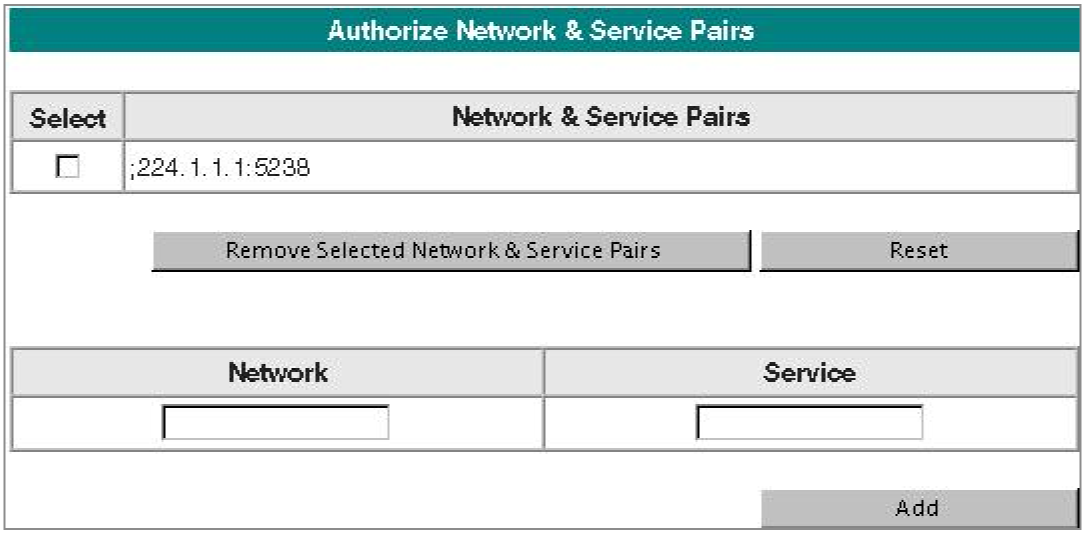
Button	Description
Password	Displays a page that lets you view and set the password for the selected user.
Certificates	Displays a series of pages that let you view and set public certificates for the selected user.
Remove Selected Users	Deletes one or more selected users from the list.

Authorize Network and Service Pairs

This page lets you configure the network and service pairs that users can access through the secure daemon.

To display this page, click **Networks and Services** in the left margin of any page of the secure daemon browser administration interface.

Figure 48: Secure Daemon Authorize Network and Service Pairs Page



Item	Description
Network & Service Pairs	This table lists the pairs of network and service that all authenticated users may access through this secure daemon. To remove a pair from this list, click to check its Select box, then click the **Remove Selected Network & Service Pairs** button.
Add	To add access to a network and service pair, type the specifications and click the Add button.

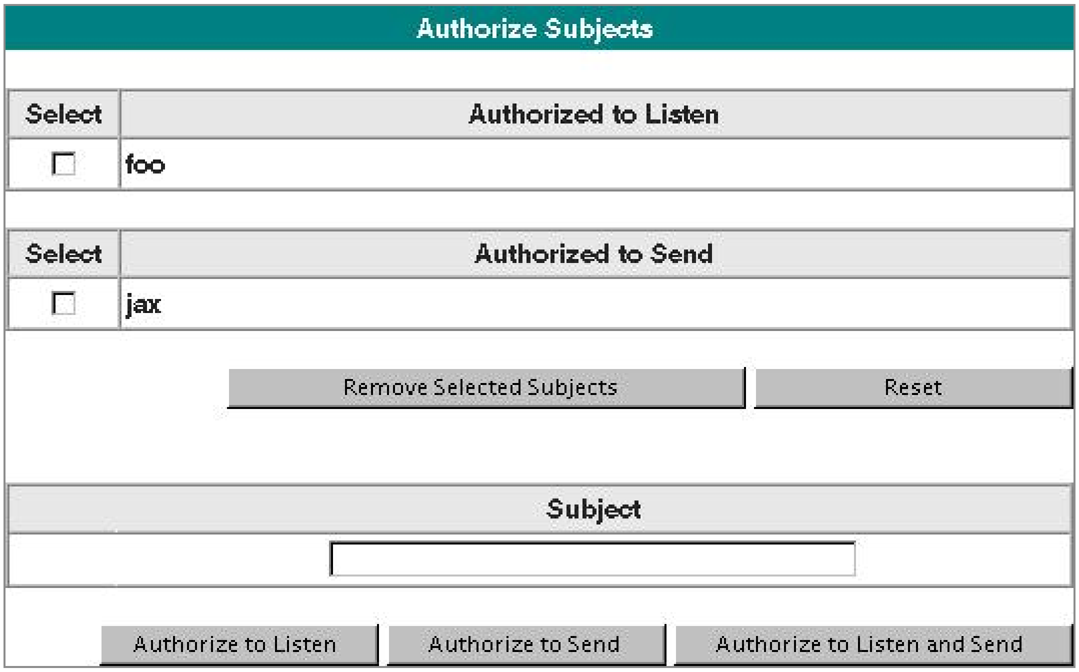
Authorize Subjects

This page lets you configure the Rendezvous subjects that users can access through the secure daemon.

To display this page, click **Subjects** in the left margin of any page of the secure daemon browser administration interface.

For background information, see Subject Authorization.

Figure 49: Secure Daemon Authorize Subjects Page



`_INBOX` subjects are implicitly authorized both for listening and sending. You do not need to authorize them explicitly on this page.

Item	Description
Authorized to Listen	This table lists subjects to which authenticated users may subscribe.
Authorized to Send	This table lists subjects to which authenticated users may send messages.
Remove Selected Subjects	To remove authorization for particular subjects, select the affected subjects and click this button.
Subject	To add access to a subject, type the subject here and click one of three buttons: • Authorize to Listen • Authorize to Send • Authorize to Listen and Send

Certificates

This page lets you configure the X.509 certificates that a secure daemon uses to identify itself.

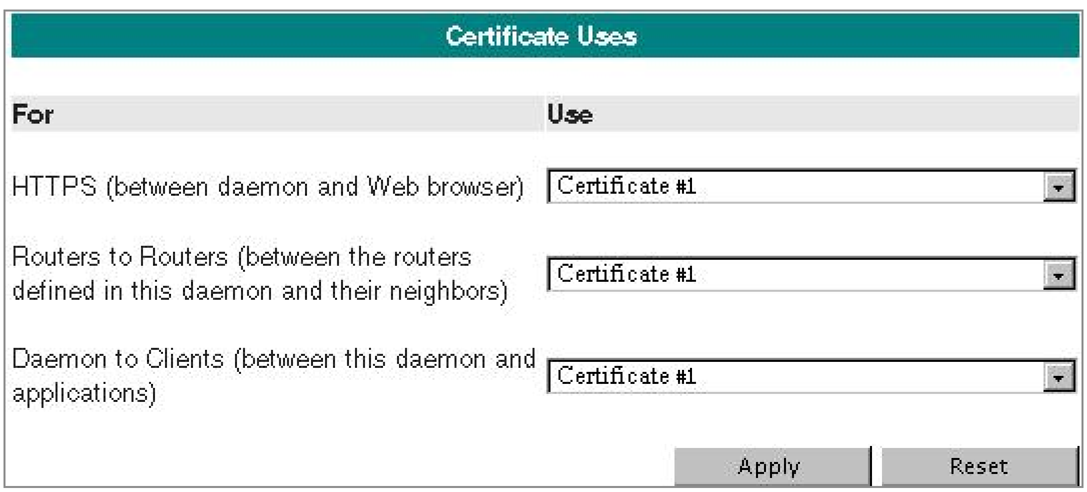
To display this page, click **Certificates** in the left margin of any page of the `rvsd` or `rvsrd` browser administration interface.

For background information, see Certificates and Security in TIBCO Rendezvous Concepts.

Each daemon process keeps a list of certificates it can use to identify itself. These certificates are numbered for easy reference. The first panel on this page determines which of these certificates the daemon uses for particular tasks. The remainder of the page lets you enter the certificates.

Certificate Uses

Figure 50: rvsrd Certificate Uses Form



Item	Description
HTTPS	Set the certificate for the secure browser administration interface. To avoid security warnings from the web browser, distribute the public portion of this certificate to authorized administrators.

Item	Description
Routers to Routers	Set the certificate for secure TLS neighbor connections. Distribute the public portion of this certificate to each applicable neighbor. (This item is included in routing daemons only; it is absent from `rvsd`.)
Daemon to Clients	Set the certificate for secure TLS client transport connections. Distribute the public portion of this certificate to each client program; see Secure Daemon in TIBCO Rendezvous Concepts.

Certificate List

Figure 51: rvsrd Certificate List

Item	Description
certificate number	Use this number to refer to the certificate in the Certificate Uses panel.
Add from File	Enter a file name and a private key password. When you click Add from File, the daemon reads the certificate with private key from the file. The file may be in either PEM encoding, or PKCS #12 format. See also Security Factors.
Add from Text	Paste the text of a certificate with private key. Enter a private key password. The certificate *must* be in PEM encoding. See also Security Factors.

Self-Signed Certificate

When the daemon creates its store file (the first time it starts), it also creates a self-signed certificate, and registers it in the list as certificate #1. You may use that certificate as is, add other certificates to the list, or delete it and enter other certificates.

The self-signed certificate expires one year after creation.

CA-Signed Certificate

You can also supply certificates signed by a certificate authority (CA). To use a CA-signed certificate, you must supply not only the certificate and private key, but also the CA's public certificate (or a chain of such certificates). Concatenate these items in one file or string. For more details, see CA-Signed Certificates.

CA-signed certificates expire at dates recorded within the certificate data.

Current Value Cache

In many distributed applications new processes can join the system at any time. Often these new processes need access to the current information state of the system in order to function properly. In many cases a straightforward cache program can fill that need.

The Rendezvous distribution includes a utility program called `rvcache`, which caches the data from messages sent to each subject name. Whenever a Rendezvous program begins listening to a subject name, it can query `rvcache` to send it the current data for that subject.

The data cached for a subject can be either the most recent whole message on that subject, or a composite set containing the most recent value of each field sent on that subject.

Although `rvcache` resembles a simple database program in some respects, it differs in an important way. Namely, updates are implicit; that is, `rvcache` monitors message activity and automatically caches the data. Application components query for data by subject.

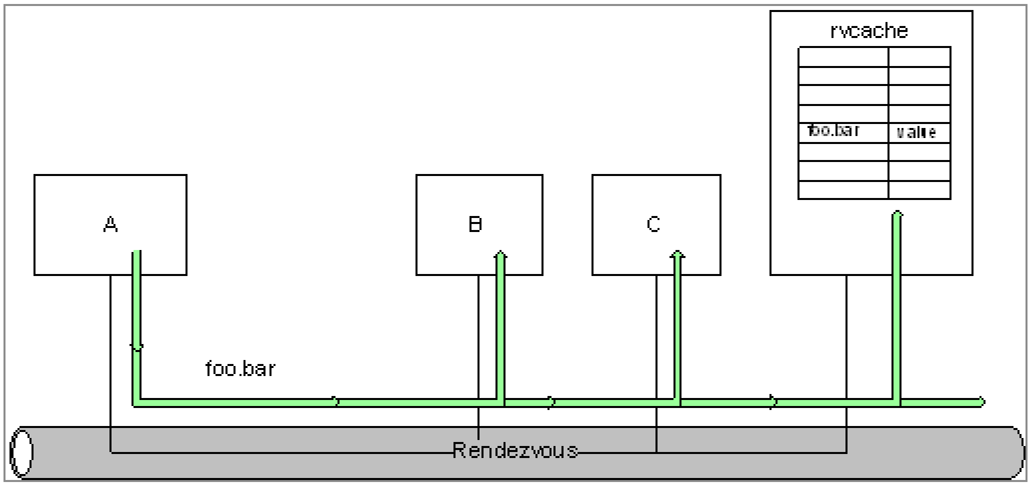
We recommend that administrators arrange for correct operation of `rvcache`. This section describes administrative considerations; for a command summary, see rvcache.

Operation

Although many distributed system components may depend upon `rvcache`, its caching operation remains transparent to them.

Other application programs do not send update messages specifically to `rvcache`. Instead, `rvcache` listens for a set of subjects, silently receiving messages and caching the most recent data on each subject as it arrives, as in Transparent Caching by rvcache.

Figure 52: Transparent Caching by rvcache

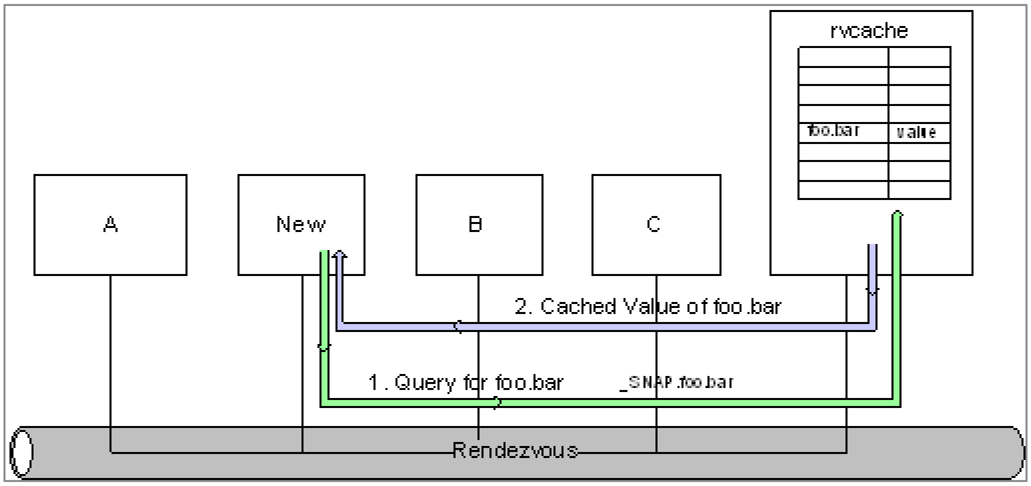


However, other application programs *do* send query requests to `rvcache`. Query and Response with rvcache illustrates this phase of `rvcache` operation:

Procedure

1. `rvcache` listens to the subject `_SNAP.>` for query messages.
2. A program submits a query for the cached value of `foo.bar` by sending an empty message to the subject `_SNAP.foo.bar` (more generally, build the query subject on the template `_SNAP.`*cached_subject*).
3. `rvcache` receives the query, and extracts the cache subject from the query subject name. It sends the cached value for that subject to the reply subject of the query message.

Figure 53: Query and Response with rvcache



Resource Requirements

Load

For fastest response, run `rvcache` on a computer with a light processing load.

Storage

The exact amount of required storage space varies with three factors—the storage mode, the number of subjects cached, and the size of the stored message values.

- In standard operation, `rvcache` keeps a table of cached subjects in memory, while it keeps message data for those subjects in a store file on disk. The computer running `rvcache` must have sufficient storage of each type.
- In memory-only mode, `rvcache` keeps both in memory (both the table of cached subjects and the message data). The computer running `rvcache` must have sufficient memory. (For background information, see Memory-Only Mode.)

Distributed Caches

In some cases, you may find it expedient to distribute the resource requirements and the processing load among several computers. To achieve this goal, you can run several process instances of `rvcache` on separate computers. However, it is important that various process instances of `rvcache` cache disjoint sets of subjects (see Avoid Duplicates).

Avoid Duplicates

Listening programs rarely profit from receiving duplicate copies of the current data. To prevent duplicates, consider one of these strategies:

- Run exactly one `rvcache` service.
- Ensure that `rvcache` services store disjoint subject sets.

  When two or more `rvcache` services store the same subjects, then duplicate messages can result. It the subject sets do not overlap, then duplication cannot occur.
- Segregate `rvcache` services by listening on different UDP services.

  You can use different UDP services to isolate groups of program processes so that members of each group receive Rendezvous messages exclusively from other members of the same group. In such cases, configure a separate instance of `rvcache` for each group by setting its service parameter to match the UDP service used by group members. For more detail about this parameter, see Service Selection.

Ensure Continuous Service

Interruptions in `rvcache` service can result in two undesirable consequences:

- Programs that query during the interruption do not receive the cached data.
- The cache does not record data from messages sent during the interruption. Gaps in the cache persist after the interruption.

To minimize these effects, we recommend that system administrators consider these strategies:

- Avoid interruptions whenever possible.

For example, shift rvcache (along with its disk file) to an alternate host computer before scheduled downtime. Since rvcache is independent of the computer on which it runs, switching hosts can be an effective remedy.

- Reduce the length of unavoidable interruptions.

  Monitor the health of the rvcache process and its host computer. If any problem prevents smooth operation, promptly correct the situation or shift rvcache to an alternate host computer.

- Run rvcache as a fault-tolerant service.

  For details, see Fault Tolerance.

Crossing Network Boundaries

When a network boundary separates rvcache from its client programs, and a routing daemon (rvrd) connects them across that boundary, you must configure rvrd to ensure correct operation of rvcache.

Cached Subjects

The routing daemons (on both sides of the neighbor link) must permit all the cached subjects to flow from all senders to all rvcache processes.

Query Subjects

The rvrd configuration for exchanging query subjects depends on the distribution of rvcache and its query clients.

- If each network runs one local cache process, with all the caches synchronized (so they all contain the same data), then it is crucial that only one rvcache process receive each query. The routing daemons *must not* import or export _SNAP.> (the query subject).

- If only one network runs a cache process, and programs on other networks query it across the network boundary, then the routing daemons *must* forward _SNAP.> (the query subject) into the rvcache network. That is, rvrd must import these query names into the `rvcache` network; `rvrd` must export these query names from each query client network.

Fault Tolerance

Multiple process instances of `rvcache` can cooperate for fault-tolerant service.

Fault tolerance protects `rvcache` service against hardware failures, process termination and segmentation of the local network.

In this configuration, two or more `rvcache` processes run on separate computers—usually on separate network segments. All cooperating processes listen for the same set of subjects, and store the current values of those subjects. Only one process (called the *primary active* process) actively sends the current values to new listeners. The remaining processes (called *inactive backup* processes) are inactive—unless they detect that the primary active process has failed. If the primary fails, one of the backup process activates in its place, restoring service automatically.

These sections describe fault tolerance concepts and parameters in detail:

- Fault Tolerance Concepts in TIBCO Rendezvous Concepts.
- Fault Tolerance Programming in TIBCO Rendezvous Concepts.
- Developing Fault-Tolerant Programs on in TIBCO Rendezvous Concepts.

For administrative details, see Fault Tolerance.

Usage

To run `rvcache` as a fault-tolerant service, start two or more `rvcache` processes. It is essential that all processes use identical parameters—with only one exception:

- The `-store` parameter specifies a file for persistent storage of the cache and configuration parameters. Member processes *must not* share this file. Each member must keep its own distinct cache file (we recommend storing it on a local disk).

Duplicating the Cache State

To duplicate the cache state, copy the cache file (so each process starts with an identical copy). Avoid file inconsistencies that can arise when copying the file while `rvcache` is running.

Replace and Merge

Replace & Merge

`rvcache` can store message data in two ways. For each subject, it can either *replace* all previously stored data with the contents of each new message, or it can *merge* information from the fields of the message into the stored data, overwriting only those fields specified in the new message.

Select one of these two storage methods each time you add a subject.

Shallow & Deep Merge

Furthermore, `rvcache` can merge data in either of two ways. The command line parameter `-merge` selects either shallow merge or deep merge as the behavior of the cache, consistent for all merged subjects. Consider a new message that contains nested messages as field values.

- *Shallow merge* replaces the old field value with the new nested message as an indivisible unit, without special treatment for the individual fields of the nested message. That is, merging occurs at only one level of nesting; level-one fields replace level-one fields.
- *Deep merge* inspects the fields of nested messages, and recursively merges the fields of a new nested message into fields of a stored nested message. Merging continues recursively to any depth of nesting.

Example

rvcache Replace and Merge presents an example, contrasting the different behavior of replacement, shallow merge and deep merge.

rvcache Replace and Merge

Stored Message	Replace	Shallow Merge	Deep Merge

`{`
` Field1 1.0`
` Field2 "a str"`
` Field4 {`
`      Sub1 2.0`
`      Sub2 "c str"`
`        }`
`}`
	`{`
	` Field3 "b str"`
	` Field4 {`
	`      Sub1 3.0`
	`         }`
	`}`
	
	The new message (in this column) replaces the stored message as an indivisible unit. After replacement, the stored message is an identical copy of the new message.
		`{`
		` Field1 1.0`
		` Field2 "a str"`
		` Field3 "b str"`
		` Field4 {`
		`      Sub1 3.0`
		`         }`
		`}`
		
		Fields of the stored message remain in place, except where superseded by the fields of the new message.
		
		Existing `Field1` and `Field2` remain unchanged.
		
		The new `Field3` is added to the stored message.
		
		The new value of `Field4` *replaces* the stored value.
			`{`
			` Field1 1.0`
			` Field2 "a str"`
			` Field3 "b str"`
			` Field4 {`
			`      Sub1 3.0`
			`      Sub2 "c str"`
			`         }`
			`}`
			
			Existing `Field1` and `Field2` remain unchanged.
			
			The new `Field3` is added to the stored message.
			
			The new value of `Field4` *recursively merges* into the stored value, replacing the value of `Sub1`, but leaving `Sub2` unchanged.

Data stored in `rvcache` never expires. It remains in the cache until superseded or augmented by data from a new message on the same subject.

Memory-Only Mode

Store File

In standard operation, `rvcache` uses the store file to record parameter configuration and for persistent cache storage. Parameter configuration includes the list of cached subjects. Persistent cache storage includes the cached values of those subjects. The required command line parameter `-store` specifies the pathname of the store file.

Memory-Only

In some high-volume situations, writing cached values to the store file can result in an I/O bottleneck. If rvcache I/O presents a problem, you can disable persistent cache storage. The command line parameter `-memory-only` starts `rvcache` in memory-only mode (otherwise, `rvcache` runs in *store mode*).

Memory-only mode changes the operation of `rvcache` as follows:

- When `rvcache` starts, it does *not* read initial values from the store file. Consequently, `rvcache` starts with an empty cache for all subjects.
- `rvcache` keeps cached values only in process memory; it does not write values to the store file.
- However, `rvcache` still reads its configuration from the store file at start time, and writes configuration changes to the store file.

Memory-only mode increases the operating speed of `rvcache` at the cost of data persistence. If the process exits, all cached values are lost.

In memory-only mode, you can expect `rvcache` to consume more process memory than in store mode. Since it cannot retrieve infrequently accessed values from the store file, `rvcache` must keep all values in process memory. You must ensure that adequate memory is available.

Note: Switching from store mode to memory-only mode does not automatically erase all initial values from the store file. Even though `rvcache` does not read them, they remain in the store file (which as a result might be larger because of these unnecessary values).

Tip	Since a larger store file can slow `rvcache` initialization, we recommend starting with an empty store file when you begin using memory-only mode. You can use the `dumpXML` command of `tibrvcfg` to dump the configuration of `rvcache` as an XML document. Then use the `matchXML` command to create an empty store file with the required configuration. For details, see Command Line Tool—tibrvcfg in TIBCO Rendezvous Configuration Tools.
Note	After switching from store mode to memory-only mode and back again, you cannot rely on cached values in the store file. Some subject configuration changes can erase the value of a subject as a side effect. Before switching modes, we recommend saving a backup copy of the store file. Alternatively, you can use the `dumpXML` command of `tibrvcfg` to save a backup copy of configuration data before you switch modes. See Command Line Tool—tibrvcfg in TIBCO Rendezvous Configuration Tools.
Tip	When configuring large or complex sets of subjects, the configuration API is more convenient than the browser administration interface; see the Current Value Cache - rvcache section in TIBCO Rendezvous Configuration Tools.

rvcache

Command

Syntax

```
rvcache   -store filename
         [-http [ip_address:]http_port]
         [-https [ip_address:]https_port]
         [-http-only]
         [-https-only]
         [-no-http]
         [-idle]
         [-sync interval]
         [-merge shallow | deep]
```

```
          [-memory-only]
          [-tls-min-proto-version version]
          [-tls-max-proto-version version]
          [-tls-ciphers string1:string2:stringN]
          [-tls-ciphersuites name1:name2:nameN]
```

Purpose

The program `rvcache` stores data from recent messages, indexed by subject name, and automatically sends the cached data to new listeners.

Remarks

Given a set of one or more subject names, `rvcache` listens for messages addressed to those subjects. Each time it receives such a message, it stores the message's data content.

When a client program queries for a cached subject, `rvcache` sends a reply message with the current cached value.

Browser Administration Interface

To administer or configure `rvcache`, view `http://`*rvcache_host*`:`*http_port* with a web browser. When the program starts, it prints the actual HTTP administration port.

State

`rvcache` can run in either of two states—running or idle.

When *running*, `rvcache` listens to subjects, caches message values, and responds to queries.

When *idle*, `rvcache` does not operate; however, the browser administration interface is available for configuring parameters.

Initial Subject Configuration

The first time you run `rvcache`, you must configure its subjects and change its state to running. After that, `rvcache` reads the subject list from its file.

Storage

`rvcache` stores the data in process memory and in a disk file. The command line parameter `-store` specifies the name of the disk file; if the file exists when `rvcache` starts, then `rvcache` reads the file to initialize its configuration parameters and to populate its cache in process memory.

The command line parameter `-sync` specifies the interval at which to synchronize the file-based store with process-based store.

Command Line Parameters

Parameter	Description
`-store` *filename*	Use *filename* to record parameter configuration and for persistent cache storage. For best performance, use a local file system (remote file servers can cause delays and synchronization difficulties). For more information, see Storage. See also Store Files.
`-http` *ip_address*`:`*http_port* `-http` *http_port* `-https` *ip_address*`:`*https_port* `-https` *https_port*	The browser administration interface accepts connections on this HTTP or HTTPS port. Permit administration access only through the network interface specified by this IP address. To limit access to a browser on the `rvcache` host computer, specify `127.0.0.1` (the local host address). When the IP address is absent, the daemon accepts connections through any network interface on the specified HTTP or HTTPS port. If the explicitly specified HTTP port is already occupied, the program exits. If the explicitly specified HTTPS port is already occupied, the program selects an ephemeral port. When the `-http` parameter is entirely absent, the default

Parameter	Description
	behavior is to accept connections from any computer on HTTP port `7581`; If this default port is unavailable, the operating system assigns an ephemeral port number. When the `-https` parameter is entirely absent, the default behavior is to accept secure connections from any computer on an ephemeral HTTPS port. In all cases, the program prints (in its start banner and log file) the actual HTTP and HTTPS ports where it accepts browser administration interface connections.
`-http-only`	Disable HTTPS (secure) connections, leaving only an HTTP (non-secure) connection.
`-https-only`	Disable HTTP (non-secure) connections, leaving only an HTTPS (secure) connection.
`-no-http`	Disable *all* HTTP and HTTPS connections, overriding `-http` and `-https`.
`-idle`	When present, start `rvcache` in its *idle* state. When absent, start `rvcache` in its *running* state—caching values and responding to queries. However, if subjects are not configured, `rvcache` begins in its *idle* state. You can toggle the state at any time using the browser administration interface.
`-merge shallow | deep`	For subjects that cache by merging the new value into the stored value, two types of merge behavior are available. When present, this parameter selects that behavior for *all* merged subjects. For complete details, see Replace and Merge. When absent, the default behavior is shallow merging.
`-memory-only`	When present, `rvcache` keeps cached values in process

Parameter	Description
	memory only; it does not write values to the store file. Nor does it read initial values from the store file at start time. For details, see Memory-Only Mode. When absent, `rvcache` writes cached values to the store file.
`-sync` *interval*	The synchronization behavior of operating systems varies. You can use this parameter to balance message processing speed against disk synchronization guarantees. When absent, `rvcache` relies on the operating system for all synchronization. `rvcache` opens the store file in *synchronous write mode*—that is, the operating system writes every message to the file system before the write call returns. When present, `rvcache` explicitly synchronizes data to the file system at this interval. `rvcache` opens the store file in *asynchronous write mode*—that is, write calls return independently of disk operations, and the operating system completes disk writes on its own schedule. Explicitly synchronizing at a fixed interval limits exposure to loss by enforcing an upper bound—though the operating system might also synchronize between intervals (reducing exposure even further).
`-tls-min-proto-version` *version* `-tls-max-proto-version` *version*	Set the minimum or maximum supported protocol versions for the ctx using OpenSSL calls `SSL_CTX_set_min_proto_version and SSL_CTX_set_max_proto_version`.
`-tls-ciphers` *string1:string2:stringN*	Set the list of available ciphers (TLSv1.2 and earlier) using OpenSSL call `SSL_CTX_set_cipher_list`.
`-tls-ciphersuites` *name1:name2:nameN*	Configure the available TLSv1.3 ciphersuites using OpenSSL call `SSL_CTX_set_ciphersuites`.

Browser Administration Interface

Parameter	Description
information	
This page displays general information about the `rvcache` process.	
change state	
`State`	Toggle between `idle` and `running`. When running, `rvcache` listens to its cache subjects, caches values, and responds to queries. When idle, `rvcache` does not operate; however, the browser administration interface is available for configuring parameters. The program does not store this parameter.
certificates	
This page lets you configure the certificates that `rvcache` uses to identify itself to web browsers. For more information, see the analogous section for secure daemons, Certificates.	
security	
These parameters control access to the configuration pages of the `rvcache` browser administration interface.	
connection	
`rvcache` uses these parameters to create its network transport object. For general explanations, see Network Details.	
`Service`	See Service Selection.
`Network`	See Network Selection.
`Daemon`	See Daemon Client Socket—Establishing Connections.

Parameter	Description
fault tolerance	
`Enable`	Enable or disable fault-tolerant operation. The remaining parameters in this group apply only when fault tolerance is enabled.
`Service`	Use this UDP service for fault tolerance control messages between `rvcache` member processes. The default value is `rendezvous-ft`; if the operating system cannot interpret that service name, then the secondary default is UDP port `7504`.
`Network`	Use this network for fault tolerance control messages between `rvcache` member processes. The default value is the computer's primary network interface.
`Group`	Use this string as the name of the `rvcache` fault tolerance group. Processes with the same group name cooperate to provide fault-tolerant service. The default value is `RVCACHE`.
`Weight`	Set the weight of *this* `rvcache` process. Weight specifies relative precedence among fault-tolerant processes. A process with greater weight takes precedence over a process with lesser weight. The default value is `10`.
`Heartbeat`	Use this floating point value (in seconds) as the fault tolerance heartbeat interval. Members of a fault tolerance group send status reports at this interval. We recommend that this value be slightly less than one third of the activation interval. The default value is `3` seconds.

Parameter	Description
`Activation`	Use this floating point value (in seconds) as the fault tolerance activation interval. This value represents the longest interruption in service before the partner process activates. It must be the same for all members of a fault tolerance group. The default value is `10` seconds.
subjects	
`Subjects`	To see information about a specific subject, click that subject in the current subject list. You can add new subjects or remove current subjects at any time. For more information, see Replace and Merge.

XML Configure

View the current configuration as an XML document, and reconfigure the component by submitting an edited XML document.

Performance Assessment (rvperf)

Performance assessment software can help you gauge and improve Rendezvous network performance, plan hardware purchases and software deployment, and test network configurations.

Overview

Rendezvous performance assessment software is a tool for evaluators, reviewers, and network administrators. It measures the potential performance of Rendezvous software in an actual network situation, and outputs a detailed report.

This performance assessment software helps you compare various equipment options and network configurations, using the performance of Rendezvous software as a gauge.

Remember that speed benchmarks are relevant only in the context of a specific network with specific computers. Networks can differ widely in their performance. Adding or removing a problematic computer can dramatically alter performance.

Rendezvous benchmark data is not a guarantee of application performance. It demonstrates the potential maximum performance that your network can achieve. Although it can model common application behaviors regarding message sending and receiving, it cannot exactly mimic the actual performance of your Rendezvous applications. The performance assessment tool stresses message transport capabilities, but does not engage in other common application behaviors, such as calculations or managing a graphics display.

The performance assessment tool can establish an upper bound on application message transport performance, and help gauge some of the secondary effects of network usage patterns, but it cannot prove that an application as a whole will operate properly.

Components

Performance assessment software consists of two executable programs:

- `rvperfm` (master) sends messages, gathers performance data, and outputs the report.

  See rvperfm.

- `rvperfs` (slave) subscribes to messages from `rvperfm`, and sends back data about its own speed and effectiveness.

  See rvperfs.

You can run `rvperfm` alone, or with any number of `rvperfs` processes in the network.

Principles of Operation

In our experience, Rendezvous distributed applications achieve optimal network performance when senders transmit messages in short batches, pausing briefly between batches. This observation is the foundation for the performance assessment tool.

Performance assessment software measures network performance by sending runs of messages, and compiling statistics. You can experiment by varying parameters such as message size, run length, batch size, pause interval—which affect the network data rate.

Each message contains *s* bytes of payload data (plus message headers and packet overhead).

`rvperfm` sends one or more sequences (or runs) of *m* messages to the network.

Instead of sending a continuous stream of messages, `rvperfm` groups them into batches of *b* messages, pausing for an interval of *i* seconds between the end of one batch and the start of the next batch.

While `rvperfm` is sending messages, zero or more process instances of `rvperfs` are listening for those messages and compiling statistics. At the end of each run, the `rvperfs` processes report back to `rvperfm`, which outputs the statistics.

Listeners

`rvperfm` can send messages to the network whether or not any `rvperfs` processes are listening to receive those messages.

- When `rvperfs` listeners are present, the performance assessment tool measures their capacity to receive messages as part of overall network performance.

- In the absence of `rvperfs` listeners, the performance assessment tool measures the network performance of the sending computer only.

Single Mode and Automatic Mode

Two modes characterize the operation of `rvperfm`:

- In **single mode**, `rvperfm` sends a single run of messages, governed by its command parameters. At the end of the run, it outputs a report and exits.

  You can use single mode as a modeling tool to answer questions about network behavior under sustained load conditions. For example:

  — What happens to network performance when an application sends a batch of ten thousand messages without pausing?

  — Which computers in my network can send messages the fastest? Which can receive fastest?

  — How does introducing a router affect network throughput under normal network load conditions? How do peak loads affect network throughput?

- In **automatic mode**, `rvperfm` sends several runs of messages, modifying the parameters for each run until it finds the batch size and interval parameters that yield maximum sustainable network throughput. At the end of each run, it outputs a report. Then it adjusts the parameters and starts the next run.

  The overall effect is that `rvperfm` tunes its send rate to match the maximum receive rate of the slowest `rvperfs` process. The last report before the process exits displays the parameters that yield the maximum throughput. (In the *absence* of `rvperfs` processes, `rvperfm` determines the maximum send rate that the network can support. Note that the parameter tuning algorithm is the same, only the significance of the result differs.)

  You can use the results of testing in automatic mode to tune applications for maximum performance in a specific network configuration.

Automatic Mode—Binary Search

In automatic mode, `rvperfm` uses a binary search algorithm to adjust its batch size and interval parameters between runs. These rules control `rvperfm` as it empirically determines the maximum throughput:

- For the first run, `rvperfm` determines the batch size and interval from command parameters or default values.
- If `rvperfm` loses data, it aborts the run immediately, and adjusts parameters to decrease the send rate for the next run.
- If even one of the `rvperfs` processes lost data during the run, then the send rate exceeds the maximum network throughput. In this case, `rvperfm` aborts the run, records the upper bound on maximum throughput, and adjusts the parameters to decrease the send rate for the next run.
- If all active `rvperfs` processes keep pace without lagging behind, and receive all the messages without losing data—then the send rate is lower than the maximum throughput. In this case, `rvperfm` records the lower bound on maximum throughput, and adjusts the parameters to increase the send rate for the next run.
- After a finite number of runs, the upper and lower bounds converge at the maximum throughput. When `rvperfm` exits, the report of the last run indicates the batch size and interval parameters that yield the maximum network throughput.

  If no `rvperfs` processes are active, the parameters of the last run yield the maximum send rate for `rvperfm` on its host computer (with the prevailing network conditions).

Dataloss Advisory

`rvperfm` and `rvperfs` both subscribe to `DATALOSS` advisories. At the end of each complete run, both programs report the number of advisories they received during the run.

If `rvperfm` receives a `DATALOSS` advisory, it aborts the run immediately. (This paragraph applies only to automatic mode; if `rvperfs` receives a `DATALOSS` advisory while `rvperfm` is in single mode, the run does *not* abort.)

If `rvperfs` receives a `DATALOSS` advisory, while receiving messages from `rvperfm` in *automatic mode*, then `rvperfs` informs `rvperfm` in its response to the next auto window poll, and `rvperfm` aborts the run. (This paragraph applies only to automatic mode; if `rvperfs` receives a `DATALOSS` advisory while `rvperfm` is in single mode, the run does *not* abort.)

See also, `DATALOSS`, TIBCO Rendezvous Concepts.

Multicast, Broadcast, Point-to-Point and Direct

Rendezvous can transport messages among application programs using several mechanisms. The performance assessment tool can model the performance of an application sending messages in any of these ways:

Transport	Modeling	Notes
Multicast	Specify multicast addressing in the `-network` parameter. Omit the `-inbox` parameter.	
Broadcast	Do *not* specify multicast addressing in the `-network` parameter. Omit the `-inbox` parameter.	TRDP only
Point-to-Point	Include the `-inbox` parameter.	
Direct Point-to-Point	Specify a two part `-service` parameter to enable direct communication (bypassing `rvd`). Include the `-inbox` parameter.	

Performance Characteristics

The various transport mechanisms can display dramatically different performance profiles, which group into two broad classes.

Transport	Description and Performance Characteristics
Multicast; Broadcast	Multicast and broadcast messages use an unmetered protocol with negative acknowledgment. A sender transmits packets as fast as possible. Receivers are responsible for requesting retransmission of missed packets. Many applications can gain efficiency by dividing very large multicast or broadcast messages into smaller pieces, sending them in batches, and pausing between batches to avoid overloading slow receivers (that is,

Transport	Description and Performance Characteristics
	receivers running on relatively slow computers).
Point-to-Point; Direct Point-to-Point	Point-to-point messages use a metered protocol with positive acknowledgment. A sender requires positive acknowledgment from the receiver before it transmits additional point-to-point packets. As a result, point-to-point packets (from a single sender) rarely arrive faster than a receiver can process them. Applications generally do not gain efficiency by dividing very large point-to-point messages into smaller pieces (since the protocol itself already meters delivery).

Before You Test

Warning	Before running Rendezvous performance assessment software, read this section carefully.

Test in an Insulated Environment

We strongly recommend that you run all performance tests in a network environment that is insulated from other Rendezvous applications and other network traffic.

Consider these two important benefits of an insulated environment:

- Insulation prevents performance assessment message traffic from disrupting deployed applications.
- Insulation ensures that traffic from other applications does not skew performance measurements.

Testing in a physically isolated network yields the most accurate measurements.

When physical isolation is impractical, you can still obtain valid measurements by insulating tests within unused multicast addresses. However, in this arrangement, `rvperfm` traffic can still affect the performance of other deployed network applications.

rvd Reliability

Warning

To ensure accurate and efficient testing, it is critical that you first *disable* the reliable message storage feature of `rvd`.

To disable reliability fordaemons, manually start `rvd` (or `rvrd`) with the command line parameter `-reliability 0`.

This zero value instructs `rvd` *not* to retain outbound messages in case they are needed for retransmission. For more information, see Reliability and Message Retention Time.

`rvperfm` attempts to determine the carrying capacity of the network. To do so, it tests whether the network and a set of receivers can absorb a run of messages without missing any packets. The reliable delivery feature of `rvd` defeats this purpose; it compensates for transient network problems by retaining and retransmitting packets. This behavior is often beneficial in a production environment, but in a performance testing situation it is counterproductive—by compensating for network problems, it delays detection of those problems. This delay falsifies the results of performance testing, and unnecessarily prolongs the testing period.

rvperfm

Command

Syntax

```
rvperfm  [-service service ]
         [-network network ]
         [-daemon daemon ]
         [-subject subject ]
         [-inbox]
         [-auto]
         [-non-vectored]
         [-terse]
         [-messages m ]
         [-size size ]
         [-interval interval ]
         [-batch batch_size ]
```

```
        [-cm]
        [-cm-name name ]
        [-cm-ledger filename ]
        [-cm-sync]
        [-h]
```

Purpose

`rvperfm` coordinates the tasks of measuring network performance. It sends messages to the network, and reports statistics to `stdout`.

Remarks

In single mode (without the flag `-auto`), `rvperfm` sends one run of messages, and then exits.

In automatic mode (with the flag `-auto`), `rvperfm` sends several runs of messages, adjusting the batch size and interval parameters to empirically determine the combination that yields maximum network throughput. After it finds the optimal settings, it exits; the parameters and report of the final run reflect optimal network performance. For details, see Automatic Mode—Binary Search.

Outline

Each run consists of these steps:

Procedure

1. Dynamically discover the available `rvperfs` processes; output a list of participating instances. In the discovery step, `rvperfm` polls for listeners, and waits 3 seconds for ready signals from `rvperfs` processes; then it continues to the next step.
2. Send the run of messages.
3. Output statistics that measure the performance of the *sender*.
4. Output statistics that measure the performance of each *receiver* (if any).
5. Output a summary of error advisories pertaining to the *sender*.

Collision

When two instances of `rvperfm` (simultaneously) attempt to use the same subject, service and network, at least one of them detects the collision and exits immediately.

Simultaneous instances that differ in subject or service (or both) do not constitute a collision. Such processes can coexist.

Parameter	Description
`-service` *service*	*service* is the service name or UDP port number that defines the service group. See Service Selection. If you do not specify the `-service` parameter, the default value is `7599`.
`-network` *network*	*network* narrows the service group by selecting a local network by network name or IP network number (when the host computer has multiple network interfaces). It can also specify multicast addresses. See Network Selection. If you do not specify the `-network` parameter, the default value is the multicast address `";225.9.9.9"`. On operating systems that do not support multicast addressing, you must supply a valid broadcast network address.
`-daemon` *daemon*	The `-daemon` parameter instructs the program about how and where to find `rvd` and establish communication. See Daemon Client Socket—Establishing Connections. You can specify a daemon on a remote computer. For details, see Remote Daemon. However, the program cannot start a remote daemon automatically—you must start it manually on the remote computer. If you do not specify the `-daemon` parameter, the program finds the local daemon on TCP socket `7500`.

Parameter	Description
`-subject` *subject*	`rvperfm` sends messages to this subject name. If you specify neither `-subject` nor `-inbox`, then the program uses `_perf` as a prefix to construct broadcast subjects.
`-inbox`	`rvperfm` sends point-to-point messages `rvperfm` probes the network to discover available instances of `rvperfs`. The first instance to respond becomes the sole receiver—`rvperfm` sends point-to-point messages only to an inbox in that process instance. (Since `rvperfm` uses broadcast subjects for the initial discovery phase, it is not a contradiction to specify both `-inbox` and `-subject` parameters. When both parameters are present, `-inbox` determines the sending behavior.)
`-auto`	When present, `rvperfm` operates in *automatic mode,* sending several runs of messages to automatically determine the optimal batch size and interval parameters for the network. When absent, `rvperfm` operates in *single mode,* sending only one run of messages. See also, Automatic Mode—Binary Search.
`-non-vectored`	When present, `rvperfm` sends individual messages. When absent, `rvperfm` sends each batch of messages using a single vector call.
`-terse`	When present, suppress most reporting and simplify the final report. The terse final report contains one line per receiver (`rvperfs`). Each line is a comma-separated list of the following values: send message rate, received message rate, send byte rate, received byte rate, elapsed send time, elapsed receiver time, receiver SlowConsumer, receiver DataLoss,

Parameter	Description
	messages sent, send size, batch interval, batch size, reply name For descriptions of these values, see -reliability time.
`-messages` *m*	`rvperfm` sends *m* messages per run. If not present, the default is 10000 messages per run.
`-size` *size*	`rvperfm` sends messages with *size* bytes of payload data. Use this size to model application data rates. This size does not include message header data nor packet overhead, so computing the network byte transfer rate from this size results in an slight underestimate of the actual throughput. If not present, the default is 256 payload bytes in each message.
`-interval` *pause*	`rvperfm` sends messages in batches, waiting for *pause* seconds between the end of one batch and the start of the next batch. When absent, the default pause is zero seconds. In *single mode*, `rvperfm` sends the run with this interval. In *automatic mode*, `rvperfm` sends the *first* run with this interval, adjusting the parameters in subsequent runs. **Warning** **Change of Units**: In earlier releases the value of this parameter was interpreted as milliseconds—now it is a floating point value interpreted as seconds.
`-batch` *batch_size*	`rvperfm` sends messages in batches, with *batch_size* messages in each batch. When absent, the default is 128 messages per batch. To send messages individually, specify 1 as the *batch_size*.

Parameter	Description
	In *single mode*, `rvperfm` sends the run with this batch size. In *automatic mode*, `rvperfm` sends the *first* run with this batch size, adjusting the parameters in subsequent runs.
`-cm`	When present, `rvperfm` sends messages with certified delivery features. If `rvperfs` also specifies `-cm`, then the programs establish a certified delivery agreement.
`-cm-name` *name*	When present, `rvperfm` specifies this reusable correspondent name when it enables certified delivery. When `-cm` is present, but `-cm-name` is not, `rvperfm` operates with a non-reusable correspondent name.
`-cm-ledger` *filename*	When present, `rvperfm` uses this ledger file. You must also supply `-cm-name`.
`-cm-sync`	When present, then operations that update the ledger file do not return until the changes are written to the storage medium. You must also supply `-cm-ledger` and `-cm-name`. When absent, the operating system writes ledger file changes to the storage medium asynchronously.
`-h`	When present, output a parameter usage list to `stdout`, and exit immediately.

rvperfs

Command

Syntax

```
rvperfs  [-service service ]
         [-network network ]
         [-daemon daemon ]
         [-subject subject ]
         [-non-vectored]
         [-cm]
         [-cm-name name ]
         [-cm-ledger filename ]
         [-cm-sync]
         [-h]
```

Purpose

`rvperfs` listens for messages from `rvperfm`, gathers and reports statistics to `rvperfm` at the end of each run.

Remarks

`rvperfs` operates passively; it sends messages only in response to requests from `rvperfm`.

You can leave process instances of `rvperfs` running idle. Each instance of `rvperfs` can report statistics from several consecutive process instances of `rvperfm`—as long as only one `rvperfm` executes at a time. You can relocate the `rvperfm` process from one host computer to another without restarting the `rvperfs` processes.

Unlike `rvperfm`, an `rvperfs` process never exits by itself. You must explicitly terminate each `rvperfs` process.

In addition to sending its statistics to `rvperfm`, `rvperfs` also prints its report to `stdout`.

Parameter	Description
`-service` *service*	*service* is the service name or UDP port number that defines the service group. See Service Selection. If you do not specify the `-service` parameter, the default value is `7599`.

Parameter	Description
`-network` *network*	*network* narrows the service group by selecting a local network by network name or IP network number (when the host computer has multiple network interfaces). It can also specify multicast addresses. See Network Selection. If you do not specify the `-network` parameter, the default value is the multicast address `";225.9.9.9"`.
`-daemon` *daemon*	The `-daemon` parameter instructs the program about how and where to find `rvd` and establish communication. See Daemon Client Socket—Establishing Connections. You can specify a daemon on a remote computer. For details, see Remote Daemon. However, the program cannot start a remote daemon automatically—you must start it manually on the remote computer. If you do not specify the `-daemon` parameter, the program finds the local daemon on TCP socket `7500`.
`-subject` *subject*	`rvperfs` listens for messages with this subject name. If this parameter is absent, then `rvperfs` uses `_perf` as a prefix to construct broadcast subjects. (When you specify the `-inbox` flag to `rvperfm`, you need not specify this `rvperfs` parameter.)
`-non-vectored`	When present, `rvperfs` receives messages individually, using an ordinary listener. When absent, `rvperfs` receives messages using a vector listener.
`-cm`	When present, `rvperfs` listens for messages using certified delivery features. If `rvperfm` also specifies `-cm`, then the programs establish a certified delivery agreement.
`-cm-name` *name*	When present, `rvperfs` specifies this reusable correspondent

Parameter	Description
	name when it enables certified delivery. When `-cm` is present, but `-cm-name` is not, `rvperfs` operates with a non-reusable correspondent name.
`-cm-ledger` *filename*	When present, `rvperfs` uses this ledger file. You must also supply `-cm-name`.
`-cm-sync`	When present, then operations that update the ledger file do not return until the changes are written to the storage medium. You must also supply `-cm-ledger` and `-cm-name`. When absent, the operating system writes ledger file changes to the storage medium asynchronously.
`-h`	When present, output a parameter usage list to `stdout`, and exit immediately.

Interpreting the Report

This section describes the output from `rvperfm`.

First, `rvperfm` outputs a header, version information, and a summary of its configuration parameters.

Next it polls the network to discover existing `rvperfs` processes. Each `rvperfs` process resets itself and signals its readiness to participate in a new run. When `rvperfm` receives the ready signals, it prints an identifier for each participating `rvperfs`.

`rvperfm` prints a brief string as it begins sending the run of messages, and another when it finishes sending the run. Then it outputs its run report:

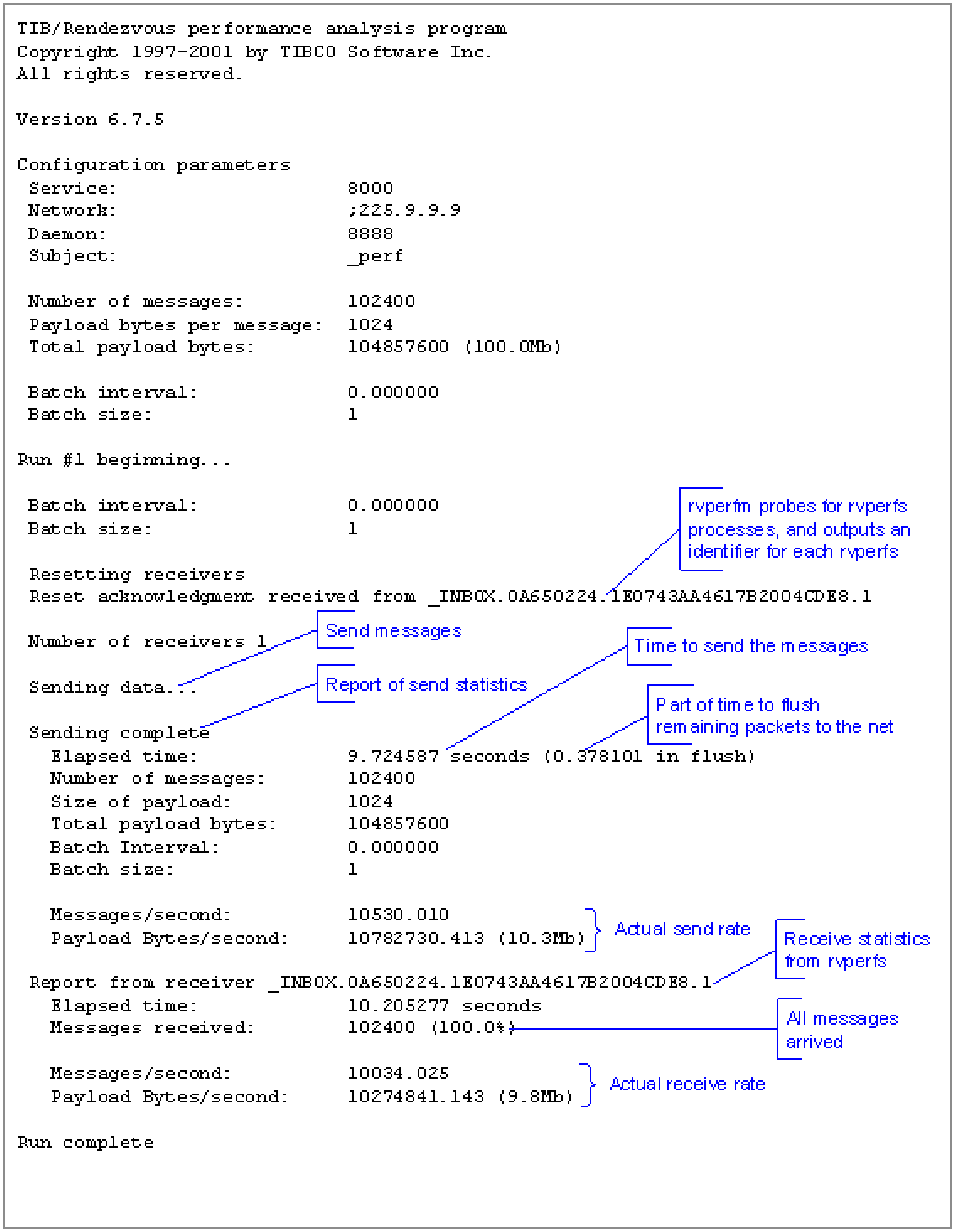
Procedure

1. Statistics that `rvperfm` collects while sending the messages.
2. Statistics that each `rvperfs` process collects while receiving messages. Each group of statistics represents the performance of one `rvperfs` process.

rvperfm Example Report

The annotated `rvperfm` transcript in Report from rvperfm shows the result of a run with one `rvperfs` process receiving the messages.

Figure 54: Report from rvperfm



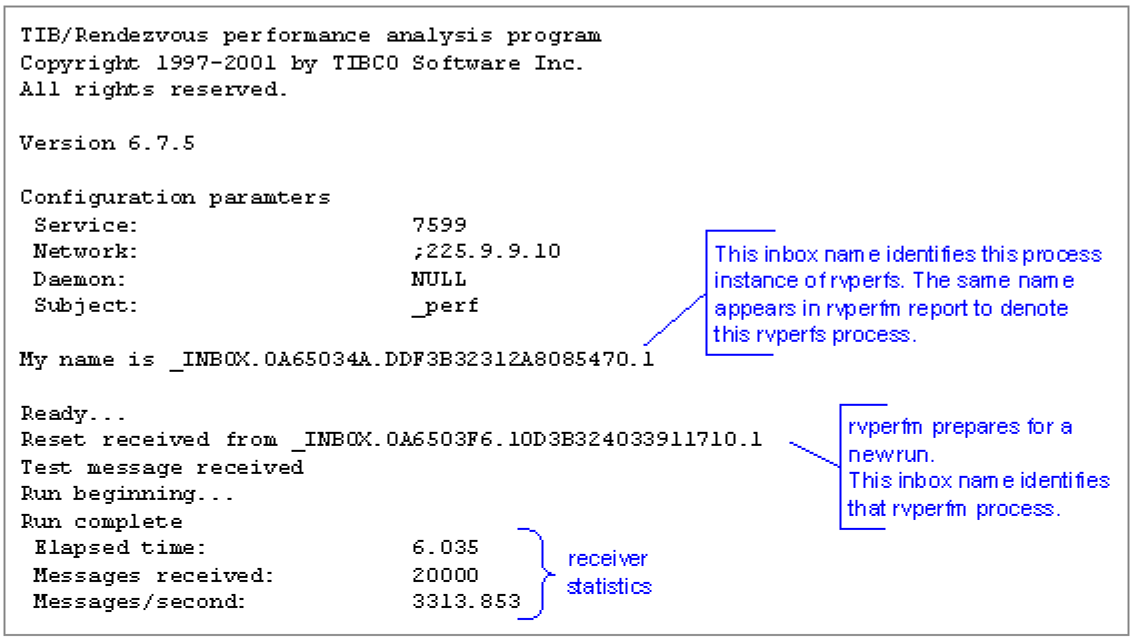
```
TIB/Rendezvous performance analysis program
Copyright 1997-2001 by TIBCO Software Inc.
All rights reserved.

Version 6.7.5

Configuration parameters
 Service:                     8000
 Network:                     ;225.9.9.9
 Daemon:                      8888
 Subject:                     _perf

 Number of messages:          102400
 Payload bytes per message:   1024
 Total payload bytes:         104857600 (100.0Mb)

 Batch interval:              0.000000
 Batch size:                  1

Run #1 beginning...

 Batch interval:              0.000000
 Batch size:                  1

 Resetting receivers
 Reset acknowledgment received from _INBOX.0A650224.1E0743AA4617B2004CDE8.1

 Number of receivers 1

 Sending data...

 Sending complete
   Elapsed time:              9.724587 seconds (0.378101 in flush)
   Number of messages:        102400
   Size of payload:           1024
   Total payload bytes:       104857600
   Batch Interval:            0.000000
   Batch size:                1

   Messages/second:           10530.010
   Payload Bytes/second:      10782730.413 (10.3Mb)

 Report from receiver _INBOX.0A650224.1E0743AA4617B2004CDE8.1
   Elapsed time:              10.205277 seconds
   Messages received:         102400 (100.0%)

   Messages/second:           10034.025
   Payload Bytes/second:      10274841.143 (9.8Mb)

Run complete
```

rvperfs Example Report

The excerpt in Report from rvperfs illustrates the output of `rvperfs`. First, the `rvperfs` process reports its configuration parameters. Next it outputs its globally unique inbox name, by which you can identify it in `rvperfm` reports.

Then `rvperfs` reports a reset message from `rvperfm`, signaling the start of a run. At the end of the run, `rvperfs` reports its statistics.

Figure 55: Report from rvperfs



```
TIB/Rendezvous performance analysis program
Copyright 1997-2001 by TIBCO Software Inc.
All rights reserved.

Version 6.7.5

Configuration paramters
 Service:                    7599
 Network:                    ;225.9.9.10
 Daemon:                     NULL
 Subject:                    _perf

My name is _INBOX.0A65034A.DDF3B32312A8085470.1

Ready...
Reset received from _INBOX.0A6503F6.10D3B324033911710.1
Test message received
Run beginning...
Run complete
 Elapsed time:               6.035
 Messages received:          20000
 Messages/second:            3313.853
```


Certified Delivery Agreements

When the performance programs use certified message delivery, `rvperfm` prints information about registration requests in its output; for example:

rvperfm

```
Resetting receivers
 Reset acknowledgment received from _INBOX.0A65034A.DEA3B32324F8085470.2
 CM registration request received from rvperfs
 Number of receivers 1
```

Similarly, `rvperfs` prints information about certified delivery registration in its output:

rvperfs

```
Ready...
Reset received from _INBOX.0A6503F6.11D3B3241239116F0.2
CM registration received from rvperfm
Test message received
Run beginning...
```

Elapsed Time

Both programs in the performance tool report the *total* time that elapsed in each complete run. The speed at which the Rendezvous daemon can deliver messages to the network depends on the network itself, the network interface card (and other hardware parameters), and the host operating system. If `rvperfm` sends at a faster rate than the network can accept, `rvperfm` retains messages in its outbound queue until the network can accept them.

```
Sending complete
 Elapsed time:               9.737 seconds
```

In this example, 9.737 seconds elapsed from the time that `rvperfm` sent the first message of the run, until the time that the daemon transmitted the last message of the run to the network.

Usage and Examples

The Rendezvous performance assessment tool is extremely flexible; it can measure performance in many different configurations of hardware, operating systems, network topologies, and operating conditions. The results can guide hardware purchases, tune for optimal network performance, or validate the suitability of Rendezvous software in specific situations. The remaining parts of this section present selected examples.

Remember, it is crucial to gauge performance in actual deployment networks, rather than in a laboratory. Benchmarks produced in one network rarely apply to other networks.

Changing the network (by adding or removing computers, routers, or other elements) can change performance dramatically.

Network Stress

Warning	On a fast computer Rendezvous software can overwhelm the capacity of the network. Other programs operating during this kind of performance test can display symptoms of network stress.

Hardware Capabilities

In this group of examples, the performance assessment tool measures the speed of specific computers—individually or in a group.

Optimal Sustained Receive Rate

What is the maximum rate at which a specific computer can *receive* a stream of messages?

To answer this question, run `rvperfs` on the receiving computer. Then run `rvperfm` in automatic mode on another computer; select a message size that reflects the messages that your application will send when deployed.

```
receiver> rvperfs
sender> rvperfm -auto -size 1000
```

After `rvperfm` experiments with its parameters, the final run indicates the values that yield the optimal receive rate for the receiving computer under prevailing network conditions. (However, you must validate this measurement; see below.)

A similar test with several receivers determines the optimal rate of the slowest receiver.

Validate against Max Transfer Rate

To validate an optimal receive rate, check that it is strictly less than the maximum transfer rate from `rvperfm` to `rvd` (see below).

- If `rvperfm` on the sending computer has successfully transferred a run of messages at a rate strictly greater than the optimal receive rate, then that receive rate is valid.
- If the measured receive rate is approximately equal to the maximum transfer rate, it might be because some limitation on the sending host is causing an artificially low result for the receive test.

Finding the Max Transfer Rate

To obtain the sender's maximum transfer rate, run `rvperfm` in automatic mode on the sending computer, without any `rvperfs` processes to receive the messages; use the same message size as in the receive test.

```
sender> rvperfm -auto -size 1000
```

After `rvperfm` experiments with its parameters, the final run indicates the values that yield the maximum transfer rate to the daemon. This result is not a useful measure of network performance; its only legitimate use is to validate measurements of receive rates.

Fixed Receive Rate

Can all computers on this network receive 2000-byte messages at a sustained rate of approximately 5 batches per second, with 10 messages per batch?

To answer this question, run `rvperfs` on each of the receiving computers. Then run `rvperfm` in single mode on another computer.

```
receiver1> rvperfs
receiver2> rvperfs
...
receiver42> rvperfs
sender> rvperfm -size 2000 -batch 10 -interval .2
```

The run report indicates whether the receivers keep pace with the sender under prevailing network conditions.

Wide Area Networks

In a wide area network (WAN) the transit time between sites can limit throughput. To keep information flowing smoothly, it is essential to measure the optimal throughput rates for the entire WAN, and limit sending rates to avoid exceeding overall network capacity.

Consider a global network connected using the Rendezvous routing daemon, `rvrd`. You can use the performance assessment tool for these tasks:

- Measure optimal sustainable throughput rates for the entire WAN.
- Compare actual speed and throughput of available WAN carrier links.
- Compare different neighbor configurations between `rvrd` components.
- Select `rvrd` host hardware.
- Demonstrate the effects of exceeding network capacity.
- Discover optimal locations from which to send messages to the rest of the network.

Certified Message Delivery

Certified message delivery introduces additional complexity. The performance assessment tool can help you measure its effects on application performance.

When the `-cm` parameter is present, `rvperfm` sends messages using Rendezvous certified delivery features. Before each run, it clears its ledger (whether file-based or process-based), and sends a test message to provoke `-cm` receivers to register for certified delivery. After the registration period, it sends the run of messages.

Number of Certified Receivers

When a certified sender process is operating near maximum capacity (either the capacity of its host computer, or the network capacity), then the number of certified receivers can dramatically affect the timing results.

Throughput of certified messages decreases as the number of registered receivers increases. This decrease is a direct result of confirmation messages flowing back from certified receivers to the sender. You can use the performance tool to measure the network capacity for certified messages with varying numbers of registered receivers.

Ledger

Certified delivery depends on a ledger to track messages and confirmations. Two types of ledger are available; each has a different effect on performance:

- A file-based ledger with asynchronous I/O offers persistence at the cost of disk operations. With asynchronous file I/O, some information could be lost in the event of sudden termination.
- A file-based ledger with synchronous I/O offers greater certainty at the cost of additional speed because the disk write operations block. Synchronous file I/O dramatically reduces the probability of lost information in the event of sudden termination.

You can use the performance tool to compare the effect of these options on certified message throughput.

Very Large Messages

Rendezvous software can transport very large messages; it divides them into small packets, and places them on the network as quickly as the network can accept them. In some situations, this behavior can overwhelm network capacity; applications can achieve higher throughput by dividing large messages into smaller chunks and regulating the rate at which it sends those chunks. You can use the performance tool to evaluate chunk sizes and send rates for optimal throughput.

This example, sends one message consisting of ten million bytes. Rendezvous software automatically divides the message into packets and sends them. However, this burst of packets might exceed network capacity, resulting in poor throughput:

```
sender> rvperfm -size 10000000 -messages 1
```

In this second example, the application divides the ten million bytes into one thousand smaller messages of ten thousand bytes each, and automatically determines the batch size and interval to regulate the flow for optimal throughput:

```
sender> rvperfm -size 10000 -messages 1000 -auto
```

By varying the `-messages` and `-size` parameters, you can determine the optimal message size for your applications in a specific network. Application developers can use this information to regulate sending rates for improved performance.

Sufficiency and Effects

Designers of distributed applications need to assess the effect of a proposed application on the network—long before deployment, and often before any code exists. The performance assessment tool can help answer questions such as these:

- Can Rendezvous software *in this network* transfer data at the rate projected for this application?
- Increased message traffic affects the operation of network infrastructure and elements such as routers, WAN links, individual computers, and previously deployed network applications (including Rendezvous applications, as well as mounted remote file systems, `telnet`, and others). What are the secondary effects of deploying an application that sends messages at the projected data rate?

Limits of Performance Assessment

Warning

Although the performance assessment tool can measure sufficiency of network transport, and the secondary effects of projected message traffic, its measurements are an idealized abstract. It *cannot* measure the *total* effect of a proposed, and as yet unimplemented, application.

Generating data to send in messages, processing inbound messages, displaying data from inbound messages to the user—all of these activities and their affect on the application's host computer are beyond the scope of the performance assessment tool. For example, this tool can determine that the *network* can absorb 300 query messages per second, but this figure does not indicate whether a database application can actually process queries and return results at that rate.

The performance assessment tool can establish an upper bound on application message transport performance, and help gauge some of the secondary effects, but it cannot prove an application as a whole will operate properly.

Locating Performance Obstacles

An application that performs more poorly than expected could be sending messages faster than the network can accept them. The performance assessment tool can help in two ways:

- Use `rvperfm` in automatic mode to determine the optimal send rate for the network. Then adjust the application to send messages at that rate.

  For a specific example of this method, see Very Large Messages.

- Set `rvperfm` parameters to mimic the sending behavior of the application. Then adjust `rvperfm` parameters to improve performance. Finally, adjust the application's behavior.

  A network protocol monitor (such as `rvtrace`) can help you diagnose performance obstacles using this method. For more information, see Protocol Monitor (rvtrace).

Latency Assessment (rvlat)

Latency assessment software can help you gauge message latency in your network.

Overview

In some application domains, response time is critically important. Several factors affect network latency, including network bandwidth conditions, hardware capabilities, multitasking, and messaging throughput patterns.

The latency assessment tool, `rvlat`, can help you understand the latency characteristics of your network. `rvlat` measures latency statistics and produces reports.

Message latency (as measured by `rvlat`) is the round-trip time between the client call that sends a request message to a server, and the message callback when the client receives a response from the server.

`rvlat` is an executable program that runs in two modes—as a requesting client or as a responding server. To use `rvlat`, you must run one instance of each mode.

Principles of Operation

The basic operation of `rvlat` is similar to Rendezvous performance assessment software, even though it measures a different property of the network. An `rvlat` client process sends a run of messages to a server; the server replies to those messages; the client receives the replies and measures latency statistics. You can vary parameters such as message size, run length, batch size, pause interval—which affect the network latency. (For descriptions of these quantities, see Performance Assessment (rvperf).)

`rvlat` can measure multicast or broadcast latency. It does not measure point-to-point latency.

You can use `rvlat` to measure latency while communicating through Rendezvous local daemons and remote daemons.

Measuring Technique

`rvlat` measures the round-trip time for a request-reply message pair:

Procedure

1. The client timestamps its outbound request message.
2. The server responds to a request by immediately returning the same message to the client.
3. The client timestamps the inbound reply, and measures the difference between the two timestamps to obtain the round-trip time.

Many applications that require low latency send messages in only one direction. However, clock synchronization between two computers is not precise enough to accurately measure one-way travel time. To avoid this difficulty `rvlat` measures round-trip time using a single clock.

Nonetheless, measuring round-trip time can also distort the results in several ways. For example, doubling the number of messages doubles the network bandwidth usage, and the effect on Rendezvous can be different for one-way versus two-way communication. The two computers might have different throughput capabilities. Timestamps, data computations and data output at the client add overhead. Turn-around time at the server adds a small overhead.

Serial & Batch Modes

`rvlat` can measure round-trip time in two modes. To get a full understanding of your network's latency characteristics, we recommend measuring latency in both modes.

- In *serial* mode (the default behavior), the client sends one request at a time. When the reply arrives, the client records the round-trip time. Only after processing the reply does the client send the next request message.

  Serial mode can help you understand patterns of latency variation over time.
- In *batch* mode, the client sends a batch of messages as rapidly as possible, then pauses for a specified interval while gathering replies and measuring their round-trip times. When the interval elapses, the client sends another batch. The `–batch` parameter specifies batch mode, and requires a *size* argument (that is, the number of requests per batch).

Batch mode simulates high-throughput network conditions, which can produce different latency characteristics than low-throughput conditions.

When you specify batch mode, you may optionally specify vectored mode as well—which sends each batch as a vector of messages, using a single send call.

Random Sampling

In serial mode and with small batches, the distorting factors are minimal. However, when the batch size is large, the distortion can be more noticeable.

You can reduce these distorting factors large batch sizes by reducing the number of round-trip messages. The `-sample` parameter instructs the server to respond to a subset of the request messages that it receives, using a probability-based sampling method.

You can use sampling to create high-throughput network conditions, while dramatically reducing the volume of data collected.

For example, consider a run of 1,000,000 requests with a message payload of 100 bytes each. Sending 1,000,000 requests but only 5000 replies (a 0.5% sample) represents a network bandwidth load of approximately 100,500,000 bytes. The 0.5% sample distorts the results less than a 100% sample, and collects far less data, yet the client still has enough data points to measure latency under high-throughput conditions.

Caveat

However, this sampling technique can also miss important patterns in the data. For example, if latency spikes occur with regular periodicity, random sampling might miss some or all of those spikes.

Using rvlat

`rvlat` runs as a client-server pair. You cannot run several clients with one server, nor vice versa.

We recommend that you write a script to run `rvlat`, redirecting its output streams to capture files (see Output Streams).

Start the server before starting the client.

Subjects

`rvlat` uses subjects that match `RVLAT.>` to carry its requests, replies, control signals and data. If the network interposes a routing daemon between the client and server, you must configure it to forward these subjects in both directions.

Test Conditions

Measure latency in a controlled environment. That is, ensure that no other applications are consuming CPU or I/O resources on the two test computers, and that no other applications are consuming network bandwidth. For example:

- Terminate all other user applications.
- Terminate scheduled jobs (UNIX `cron`), and anything else that uses CPU cycles.
- Use a network analyzer to detect other bandwidth usage, and eliminate it.
- To reduce variability that can distort latency measurements, you can set the reliability interval to zero.

Test Strategies

Run a series of trials lasting between 30 seconds and a few minutes.

Set message size to approximate expected data payloads for your applications.

Use serial mode to establish baseline round-trip statistics under uniform low-throughput conditions, and to understand network behavior. Then use batch mode to understand how latency degrades under high-throughput conditions. Vary batch size to simulate the actual data rates of your applications.

Data Strategies

Capture summaries from several trials in a spreadsheet, so you can easily analyze and manipulate the results.

Large datasets are unwieldy. When it is crucial to capture raw data (with `-datapoints`), use the `-sample` parameter to reduce dataset size.

Use the `-spikes` parameter to capture only the outliers.

Interpreting Data

Use plotting and statistical tools to graph latency data, visualize patterns, and correlate those patterns with network behavior.

rvlat

Command

Syntax

```
rvlat  { cli | srv }
       [-service service ]
       [-network network ]
       [-daemon daemon ]
       [-sample response_rate]
       [-messages m ]
       [-time t ]
       [-size size ]
       [-batch batch_size ]
       [-interval interval ]
       [-inbox]
       [-vectored]
       [-terse]
       [-datapoints]
       [-spikes threshold ]
       [-w filename]
       [-h]
```

Purpose

`rvlat` measures network latency. The client sends request messages to the server, and reports statistics to `stdout` and `stderr`. The server responds to client requests immediately.

Outline

Each run consists of these steps:

Procedure

1. Send a run of messages.
2. Output statistics that measure the performance of the *sender*.
3. Output statistics that measure the performance of each *receiver* (if any).
4. Output a summary of error advisories pertaining to the *sender*.

Parameter	Description
Mode Parameters	
`cli`	Run a client instance.
`srv`	Run a server instance.
Parameters that Apply to Both Client and Server	
`-service` *service*	*service* is the service name or UDP port number that defines the service group. See Service Selection. When absent, the default value is `12486`.
`-network` *network*	*network* narrows the service group by selecting a local network by network name or IP network number (when the host computer has multiple network interfaces). It can also specify multicast addresses. See Network Selection. When absent, the default value is the multicast address `";224.1.1.5"`. On operating systems that do not support multicast addressing, you must supply a valid broadcast network address.
`-daemon` *daemon*	The `-daemon` parameter instructs the program about how and where to find `rvd` and establish communication. See Daemon Client Socket—Establishing Connections. You can specify a daemon on a remote computer. For details, see

Parameter	Description
	Remote Daemon. However, the program cannot start a remote daemon automatically—you must start it manually on the remote computer. Note that using a remote daemon could increase latency. When absent, the program finds the local daemon on TCP socket `50000`.
`-vectored`	**Sending** When `rvlat` sends messages in batches, it can send either individual messages (with `Send`) or message vectors (with `Sendv`). The `Send` and `Sendv` calls have different latency characteristics. You can use this parameter to test either call. When present, `rvlat` sends each batch of messages as a single vector (using the `Sendv` API call). When absent, `rvlat` sends each message using a separate `Send` call. **Receiving** When `rvlat` receives messages, it can dispatch them as individual messages, or as message vectors. You can use this parameter to test either paradigm. When present, `rvlat` receives messages with a vector listener. When absent, `rvlat` receives messages individually with an ordinary listener.

`-h`
 	
`-help`
	When present, output a parameter usage list to `stdout`, and exit immediately.

Server-Only Parameters	
`-sample` *response_rate*	For background information, see Measuring Technique. When present, the server uses a random number generator to select a subset of requests to which it responds, while ignoring all

Parameter	Description
	the rest. The value of *response_rate* specifies the probability of a response (as a percentage) for each message. When absent, the server responds to 100% of the request messages it receives. We do *not* recommend using `-sample` when measuring in serial mode.
Client-Only Parameters that Control Measuring	
`-messages` *m*	When present, `rvlat` sends a run of *m* messages. When absent, the default is a run of 10,000 messages.
`-time` *t*	When present, `rvlat` sends a run of messages that continues for *t* seconds. When `-messages` is also present, `-time` overrides `-messages`. When absent, the default behavior sends a specific number of messages (rather than running for a specific time).
`-size` *size*	`rvlat` sends request messages with *size* bytes of payload data. Use this size to model application data rates. This size does not include message header data nor packet overhead. When absent, the default is 0 payload bytes in each message.
`-batch` *batch_size*	When present, `rvlat` sends messages in batches, with *batch_size* messages in each batch. When absent, `rvlat` sends messages serially, sending each request immediately after receiving a response to the previous request.
`-interval` *pause*	When `rvlat` sends messages in batches, it waits for *pause* seconds between the end of one batch and the start of the next batch. When absent, the default pause is 1 second.

Parameter	Description
	In serial mode (that is, when `-batch` is absent), `rvlat` ignores this parameter.
`-inbox`	When present, the client sends request messages to an inbox on the server (using point-to-point protocols). The server responds to an inbox on the client. When absent, the client sends request messages to a multicast subject (using either multicast or broadcast protocols, as specified in the `-network` parameter). The server responds to a multicast subject.
Client-Only Parameters that Control Output	
`-terse`	The client can output two types of reports: • A terse report with limited information, for import to spreadsheets • A verbose, human-readable report The two types of report include the same information, and both are in CSV format. When this option is present, `rvlat` outputs only a terse report to `stdout`. When absent, `rvlat` outputs two reports—a terse report to `stdout`, and a human-readable verbose report to `stderr`.
`-datapoints`	When present, `rvlat` outputs each round-trip data point (in milliseconds) to `stdout`. Caution: This option can generate an unwieldy volume of data.
`-spikes` *threshold*	When present, `rvlat` outputs data points greater than *threshold* to `stderr`. Each data point includes the round-trip time and the sequence number of the request message (within the run).
`-w` *filename*	When present, `rvlat` takes output data that would otherwise go

Parameter	Description
	to `stdout`, and instead writes it to the specified file for later analysis. If the file is not empty, `rvlat` appends the data at the end of the file. Output to `stderr` is not affected; see Output Streams.

Output

`rvlat` produces several kinds of output, including summary, raw data points, and outliers; human-readable and spreadsheet-ready; as well as error messages (which indicate problems with the command line parameters).

Output Streams

`rvlat` directs its output to two streams:

- `stderr` for human-readable output

  This category includes the human-readable summary, the spike data points (when requested), and error messages (if any).

- `stdout` for spreadsheet output

  This category includes the terse spreadsheet summary, and the raw data points (when requested).

Summaries

After a run, `rvlat` produces a summary of its data. The summary includes 13 comma-separated values (see rvlat Summary Output). The summary is available in two forms:

- Human-readable summary, including units and abbreviated labels for each value
- Terse spreadsheet summary, including only the numeric values (with neither units nor labels)

rvlat Summary Output

#	Label	Description
1	`max`	Maximum latency (in milliseconds)
2	`min`	Minimum latency (in milliseconds)
3	`mean`	Average latency (in milliseconds)
4	`stddev`	Standard deviation (in milliseconds)
5	`>1ms`	Spikes—number of messages with latency greater than 1 millisecond. (Notice that this 1 millisecond summary threshold is independent of any threshold you might specify using the `-spikes` parameter.)
6	`sampled`	Number of responses actually sent by the server.
7	`discarded`	Number of data points discarded by the client; for background information, see Discarded Data Points.
8	`received`	Number of message received by the server.
9	`sent`	Number of request messages sent by the client.
10	`total time`	Total processing time for the trial (from the time the client sends the first request, to the time it receives the last reply).
11	`msgs/s`	Approximate message rate, computed as inverse of the average processing time per message.

Raw Data Points

With the `-datapoints` option, `rvlat` outputs a column of raw data points, one value per line to `stdout`. Each value is the time (in milliseconds) for one round-trip message exchange. The values are purely numeric, suitable for spreadsheets.

After all the data points, `rvlat` outputs the terse summary line to `stdout`.

Spikes

With the `-spikes` option, `rvlat` outputs high-latency data points (those with round-trip time greater than a threshold). This output to `stderr` is in two columns. Each row is one data point, which consists of two comma-separated values—the sequence number of the request message and its round-trip time (in milliseconds).

After all the spike data, `rvlat` outputs the human-readable summary line to `stderr` (unless `-terse` suppresses it).

Discarded Data Points

If the round-trip time is faster than the client computer's clock resolution (measuring time of day), the client can record the round-trip time as zero. Zero data points would invalidate the statistical measurements, so the client discards them.

The occurrence of *any* discards indicates that measurements are less precise than they could be. A computer with finer-grained time of day clock could produce more precise measurements.

If any discards occur, the client outputs a warning message. If this warning appears, we recommend selecting a different computer for the client.

Measuring Tools for IPM

This section describes executable tools that measure performance and latency while communicating using IPM.

IPM Tools

The product distribution includes executable tools that measure performance and latency while communicating using IPM; see Measuring Tools.

Each IPM tool corresponds to a regular tool that communicates through a Rendezvous daemon. For a description of the regular tool, see TIBCO Rendezvous Administration.

Measuring Tools

IPM Tool	Corresponding Regular Tool
rvperfm_ipm	rvperfm
rvperfs_ipm	rvperfs
rvlat_ipm	rvlat

Command Line Parameters

IPM Tools—Command Line Parameters presents the differences in command line parameters between IPM tools and their corresponding regular tools.

IPM Tools—Command Line Parameters

Parameter	Description
IPM Parameters	

Parameter	Description
`-reliability` *time*	You can control the reliability window of IPM measuring tools using this parameter; see Reliable Delivery & Latency in TIBCO Rendezvous Concepts. When present, IPM retains messages for *time* (in seconds). The value must be a non-negative integer. For least distortion of latency measurements, we recommend zero. When absent, IPM uses its own default (5 seconds).
`-tb` *size*	Enable outbound batching of data from IPM, and set the batch size (in bytes). This option can increase throughput, at the cost of higher latency.
Invalid Parameters	
`-daemon` *daemon*	IPM tools ignore this parameter (which is available with the corresponding regular tools).

Protocol Monitor (rvtrace)

This section describes `rvtrace`, the Rendezvous protocol monitoring tool, which is distributed with Rendezvous *Software Release 8.8.0*.

Overview

Rendezvous protocol monitor, `rvtrace`, is a tool for network administrators. It monitors network packets, categorizes them, and reports statistics at regular intervals.

Network administrators can use `rvtrace` to diagnose network difficulties in real time, answering questions such as these:

- Which computers are inundating the network?
- Which computers are sending or receiving an inordinate number of retransmission requests?

Snapshots

`rvtrace` operates by capturing network packets, extracting information from packet headers, and gathering statistics about those packets. At the end of each interval, it compiles a statistical snapshot, and resets its counters for the next interval.

`rvtrace` can output those statistics in table format.

Prerequisites

`rvtrace` is a tool for experienced network administrators.

- You must already understand IP protocols and addressing conventions.
- You must already understand Rendezvous software from an administrator’s perspective.
- To use `rvtrace` effectively, you must understand the topology of your network.

Limitations

Range Limitations

`rvtrace` cannot examine packets unless they traverse the immediate network segment in which `rvtrace` is running. For example, point-to-point conversations within or between other network segments are invisible to `rvtrace`. Most saliently, retransmission requests and retransmission rejections are point-to-point packets, so they are visible to `rvtrace` only when they either originate or terminate in the local network segment. Consequently, in some situations `rvtrace` can detect retransmission broadcasts, even though it cannot detect the point-to-point packets that request retransmissions.

Switched network environments (such as switched Ethernet, or ATM) further limit the usefulness of `rvtrace` as a diagnostic tool. Since switching hardware forwards every point-to-point packet directly to its destination host, `rvtrace` detects point-to-point packets only when they either originate or terminate in the computer running `rvtrace`. In some switched networks, you can ameliorate this situation by disabling switching behavior—for example, by setting one port to diagnostic mode, or by using a diagnostic utility. This tactic can yield the full stream of point-to-point packets in a limited portion of the network; run `rvtrace` in that portion.

In addition, some network switching hardware can route multicast packets to a network segment only when a host in the segment is actually listening to the corresponding multicast group. Such high specificity further limits the range of `rvtrace`.

Interface Limitation

`rvtrace` supports only Ethernet interfaces.

`rvtrace` does not support these (or any other) non-Ethernet interfaces: Token Ring, FDDI, ATM.

Platform Support and Limitations

`rvtrace` operates on all platforms that Rendezvous supports—except z/OS.

TIBCO supports `rvtrace` on Microsoft Windows platforms, but requires that you first obtain and install `WinPcap` (see Obtaining pcap).

Warning	However, we do *not* support `rvtrace` on Windows SMP (Symmetric Multi-Processor) platforms at this time. `WinPcap` does not support SMP platforms, and might not operate correctly in multiprocessor environments. For more information, see www.winpcap.org/misc/faq.htm.
Note	On Windows platforms, we strongly recommend that you upgrade to the most recent version of `WinPcap`.

Note	UnixWare places security restrictions on programs that open interfaces in promiscuous mode (such as `pcap`). To run `rvtrace` on UnixWare platforms, you must dedicate a separate physical interface for that purpose.

Note	Mac OS X does not support `rvtrace`.

Passive Monitor

`rvtrace` is a *passive* monitor. It neither uses nor interferes with Rendezvous daemon processes. `rvtrace` does not add any packet traffic to the network.

`rvtrace` does not require a Rendezvous daemon for its operation. Instead, it collects network packets using the `pcap` facility.

Performance Effects

Warning	Although `rvtrace` is a passive monitor, it opens the network device in *promiscuous mode,* which consumes CPU and network resources on its host computer (in proportion to total network traffic). Running `rvtrace` on the same computer as any Rendezvous daemon (`rvd` or routing daemon) indirectly affects the operation of the local daemon by consuming these resources. Running `rvd` and `rvtrace` together on the same computer changes the timing and loading profiles of the host computer. Avoid this situation whenever possible. Instead, run `rvtrace` on a computer that is otherwise free of Rendezvous activity. We further recommend that `rvtrace` run on a computer that is fast enough to process every Ethernet packet that appears at its network interface.

The pcap Facility

`rvtrace` uses the `pcap` facility to capture network packets.

Obtaining pcap

Before using `rvtrace`, you must first ensure that the `pcap` facility is properly installed.

On most UNIX platforms, `pcap` is ready to use.

For Windows, you can download the `WinPcap` NDIS packet capture driver from this URL:

- http://www.winpcap.org/install/default.htm

For Windows platforms with multiple network interfaces, see also Selecting the Network Interface.

Packet Filtering

`pcap` has a flexible filtering language for selecting the set of packets to capture. `rvtrace` inherits this language through its `-filter` parameter.

You can select packets based on source, destination, host, network interface, port, packet length, and protocol. Packets that match the filter appear in `rvtrace` output; packets that do not match are ignored.

See Also

-filter expr

Filtering

Selecting the Network Interface

UNIX

On UNIX platforms with more than one network interface, use `iniftst` to determine the correct interface name.

Windows

On Windows platforms with more than one network interface, it is sometimes difficult to determine the correct interface name. The remainder of this section presents a method to determine it:

Procedure

1. If data capture appears correct, then the remaining steps are not needed. However, if the captured data is all zeroes, then specify a different network interface (using `rvtrace -i`). Only one of the interfaces carries the data that `rvtrace` requires.
2. Install the `WinDump` utility from this URL:

   http://www.winpcap.org/windump/install/default.htm
3. Use this command to obtain a list of interface names:

   ```
   windump -D
   ```

4. Try each interface until the data appears reasonable.

Consider this example:

```
C:\>windump -D
1.\Device\Packet_{D7308399-80B2-4CA1-A9E6-C90DD74511A8} (EL574ND4
Ethernet Adapter)
2.\Device\Packet_NdisWanIp (NdisWan Adapter)
C:\>rvtrace -i \Device\Packet_{D7308399-80B2-4CA1-A9E6-C90DD74511A8}
```

Data Capture Files

`rvtrace` can write packets into a capture file, and read a stream of packets from a file (as if from the network).

Motivation

Packet capture files are an important tool for problem diagnosis. Several techniques are useful:

- Capture packet data for later analysis.
- Capture packet data for further analysis at a remote location.
- Capture packets at high speed, then replay later when I/O delays are acceptable.

  In general, `rvtrace` can capture packets to a file faster than it can display statistics. Large amounts of display data can create I/O delays, which could cause `rvtrace` to miss packets. For example, in a heavily loaded network, displaying subject statistics for many subjects could have this undesirable result.

  You can use data capture files to side-step this difficulty. For example, capture a five-minute snapshot of packets (capturing suppresses display); then replay packets from the file, displaying statistics when the consequences of I/O delays are no longer problematic.

Output File Rotation

The rotation regimen for data capture output is *almost* identical to the rotation regiment for log files; see Log Rotation.

The only difference between them, is that `rvtrace` *always* deletes an older existing file before opening a file for writing packet data. (That is, it *never* appends to the end of an existing data capture file.)

Output Rotation Backward Compatibility

The command line option `-w-rotate` *total_size* is deprecated in release 7.5, and will become obsolete in a subsequent release. We recommend migrating to the new rotation parameters at your earliest convenience.

In the meantime, we preserve backward compatibility by converting the value of this deprecated parameter to corresponding values of the new parameters:

- `-w-rotate` *total_size* retains its old meaning—specifying the total size for all data capture files. The maximum size for each individual file in the rotation is *total_size*`/10`.
- If both old and new parameters are present, the new parameters take precedence (overriding the old parameter).

rvtrace

Command

Syntax

```
rvtrace  [-i interface ]
         [-r input_file ]
         [-addr expr ]
         [-src expr ]
         [-dst expr ]
         [-port expr ]
         [-filter expr ]
         [-w output_file ]
         [-w-max-size size ]
         [-w-max-rotations n ]
         [-no-display]
         [-addrinfo]
         [-u update_interval ]
         [-no-mcast]
         [-ptp]
         [-no-subjects]
         [-hostmsgs]
         [-rate]
         [-logfile log_file ]
         [-log-max-size size ]
         [-log-max-rotations n]
         [-foreground]
         [-h]
```

Purpose

`rvtrace` is a network protocol monitor that specializes in Rendezvous protocols. It collects and prints statistics about network packets.

Remarks

`rvtrace` runs in a loop—capturing packets, analyzing them, categorizing them, and periodically printing a summary to standard output.

An `rvtrace` process never exits by itself (except as a consequence of a command syntax error); you must explicitly terminate each process.

Delimit all parameters and arguments with a space character.

Parameter	Description
Data Source	
`-i` *interface*	The program monitors packets on the network interface with this name. If absent, the default value varies, depending on operating system and network hardware. For Windows platforms, see also Selecting the Network Interface.
`-r` *input_file*	When present, read recorded packets from *input_file* instead of a network interface. This option overrides the `-i` parameter. For more information, see Data Capture Files.
Data Filtering	
`-addr` *expr*	Filter the set of packets to process only those with source or destination in the set of hosts or networks specified in *expr*. For filter expression syntax and semantics, see Filtering. Enclose filter expressions in quotation marks ("). The parameter `-addr` *filter* is equivalent to: `-filter udp and (src` *filter* `or dst` *filter*`)` When any of the parameters `-src`, `-dst`, `-addr`, or `-port` are present, `rvtrace` concatenates them into a single effective filter. However, when the `-filter` parameter is present, `rvtrace` ignores all four of these parameters, and `-filter` overrides them.
`-src` *expr*	Filter the set of packets to process only those that originate from the set of hosts or networks specified in *expr*. For filter expression syntax and semantics, see Filtering. Enclose filter expressions in quotation marks ("). The parameter `-src` *expr* is equivalent to:

Parameter	Description
	`-filter udp and src` *expr* When any of the parameters `-src`, `-dst`, `-addr`, or `-port` are present, `rvtrace` concatenates them into a single effective filter. However, when the `-filter` parameter is present, `rvtrace` ignores all four of these parameters, and `-filter` overrides them.
`-dst` *expr*	Filter the set of packets to process only those with destination in the set of hosts or networks specified in *expr*. For filter expression syntax and semantics, see Filtering. Enclose filter expressions in quotation marks ("). The parameter `-dst` *filter* is equivalent to: `-filter udp and dst` *filter* When any of the parameters `-src`, `-dst`, `-addr`, or `-port` are present, `rvtrace` concatenates them into a single effective filter. However, when the `-filter` parameter is present, `rvtrace` ignores all four of these parameters, and `-filter` overrides them.
`-port` *expr*	Filter the set of packets to process only those with destination port in the set of ports specified in *expr*. For filter expression syntax and semantics, see Filtering. Enclose filter expressions in quotation marks ("). The parameter `-port` *port* is equivalent to: `-filter udp and dst port` *port* When any of the parameters `-src`, `-dst`, `-addr`, or `-port` are present, `rvtrace` concatenates them into a single effective filter. However, when the `-filter` parameter is present, `rvtrace` ignores all four of these parameters, and `-filter`

Parameter	Description
	overrides them.
`-filter` *expr*	Filter the set of packets to process only those that match *expr*. For filter expression syntax and semantics, see Filtering. Enclose filter expressions in quotation marks ("). When present, this parameter overrides the `-src`, `-dst`, `-addr`, and `-port` parameters.
Data Capture	
`-w` *output_file*	When present, write packet information to *output_file* for later replay or analysis. When absent, do not record packet information to a file. For more information, see Data Capture Files. When `-w` is present, `rvtrace` does not display statistics. To see statistics, use `-r` to read the packet capture file. When both `-r` and `-w` are present, `rvtrace` reads packets from *input_file*, filters them, and then recaptures the filtered packets to *output_file*. You can use this technique to prune an existing capture file by reducing information or filtering extraneous traffic.
`-w-max-size` *size* -w-max-rotations *n*	When present, activate the capture-file rotation regimen (see Data Capture Files and Log Rotation). When you specify these options, you must also specify `-w`. *size* is in megabytes. If *size* is non-zero, it must be in the range [100, 2097152]. Values outside this range are automatically adjusted to the nearest acceptable value. Zero is a special value, which disables rotation. When `-w-max-size` is zero or absent, a single capture file may grow without limit (other than the limit of available storage). *n* indicates the maximum number of files in the rotation.

Parameter	Description
	When `-w-max-rotations` is absent, the default value is 10.
Statistics	
`-no-display`	When present, do not output statistics. Nonetheless, `rvtrace` continues to compile statistics, which are available through Prometheus queries. When absent, `rvtrace` outputs statistics (either to `stdout`, or to a log file).
`-u` *update_interval*	Summarize network packet at this time interval (in seconds). If absent, the default value is 10 seconds.
`-addrinfo`	When present, display network totals, subtotals, and detail rows. When absent, display only network totals and subtotal rows. For example output, see rvtrace Output with -addrinfo, and rvtrace Output without -addrinfo.
`-no-mcast`	When present, *omit* the multicast table. When absent, display the multicast table; see Multicast Data Statistics.
`-ptp`	When present, display the point-to-point table; see Point-to-Point Statistics; see also Range Limitations. When absent, *omit* the point-to-point table.
`-no-subjects`	When present, *omit* the subject table. When absent, display the subject table; see Subject Statistics.
`-hostmsgs`	When present, display Rendezvous `HOST` messages at the conclusion of each interval. TIBCO personnel might request that you supply `rvtrace` output transcript that includes these messages. These messages useful *only* to TIBCO personnel.

Parameter	Description
`-rate`	When present, display packet counts as *per-second* rates. When absent, display the actual number of packets in the update interval.
Log Output	
`-logfile` *log_file*	Send log output to this file. When absent, the default is `stdout`.
`-log-max-size` *size* -log-max-rotations *n*	When present, activate the log rotation regimen (see Log Rotation). When you specify these options, you must also specify –`logfile`. *size* is in kilobytes. If *size* is non-zero, it must be in the range [100, 2097152]. Values outside this range are automatically adjusted to the nearest acceptable value. Zero is a special value, which disables log rotation. When `-log-max-size` is zero or absent, a single log file may grow without limit (other than the limit of available storage). *n* indicates the maximum number of files in the rotation. When `-log-max-rotations` is absent, the default value is 10.
Other	
`-no-prometheus`	Disables the Prometheus HTTP server.
`-prometheus`	HTTP interface to start the Prometheus server ([[ip:]port]).
`-foreground`	Available only on UNIX platforms. When present, `rvtrace` runs as a foreground process. When absent, `rvtrace` runs as a background process.
`-h`	When present, output a parameter usage list to standard output, and exit immediately.

Errors

- `rvtrace` uses the `pcap` facility, which requires root privileges (because it must open the raw Ethernet device in *promiscuous mode*). Without appropriate privileges, `pcap` denies permission to initialize, and `rvtrace` exits immediately.
- The `pcap` library calls reject improperly formed filter expressions. It reports them with messages such as this:

  ```
  pcap_compile: syntax error
  ```

  This error causes `rvtrace` to exit.

Filtering

You can modify the set of packets that `rvtrace` processes by supplying either the `-filter` parameter, or a combination of the `-src`, `-dst`, `-addr`, and `-port` parameters. Filter expressions specify the set of packets to process.

`rvtrace` uses the `pcap` facility to capture and filter packets. The `tcpdump` utility also uses `pcap`, so the syntax and semantics for `rvtrace` and `tcpdump` filter expressions are identical. Filter Expressions summarizes the subset of filter expressions that are relevant to `rvtrace`; for additional options, see documentation for `tcpdump`. (Disclaimer: `pcap` and `tcpdump` are not TIBCO products; we do not sell, support or document them.)

Each row of Filter Expressions constitutes an *expr,* and can be used in place of the syntax marker *expr* elsewhere in Filter Expressions, and in the parameter table for rvtrace.

When specifying a filter expression to an `rvtrace` parameter, enclose the expression in quotation marks (").

Filter Expressions

Element	Description
Host Expressions	
`host` *host*	Process a packet if either the IP source or destination of the packet is *host*. Specify *host* as a name or an IP address.
`dst host` *host*	Process a packet if its IP destination is *host*.

Element	Description
`src host` *host*	Process a packet if its IP source is *host*.
Network Expressions	
`net` *net*	Process a packet if either the IP source or destination of the packet is *net*. Specify *net* as a name or an IP network number.
`dst net` *net*	Process a packet if its IP destination is *net*.
`src net` *net*	Process a packet if its IP source is *net*.
Port or Service Expressions	
`port` *port*	Process a packet if either the IP source or destination port of the packet is *port*. Specify *port* as a service name or a UDP port number.
`dst port` *port*	Process a packet if its IP destination port is *port*.
`src port` *port*	Process a packet if its IP source port is *port*.
Broadcast or Multicast Expressions	
`ip broadcast` `ip broadcast` *expr*	Process a packet if it is an IP broadcast packet. If *expr* is present, then process the packet only if it also meets the criteria of *expr*.
`ip multicast` `ip multicast` *expr*	Process a packet if it is an IP multicast packet. If *expr* is present, then process the packet only if it also meets the criteria of *expr*.
Protocol Expressions	
`udp` `udp` *expr*	Process a packet if it is an IP packet with protocol type `udp`. (All Rendezvous packets are UDP packets.)

Element	Description
	If *expr* is present, then process the packet only if it also meets the criteria of *expr*.
`ip` `ip` *expr*	Process a packet if it is an IP packet. If *expr* is present, then process the packet only if it also meets the criteria of *expr*.
Boolean Operators Use parentheses to group boolean expressions; use appropriate escape characters to override shell-specific semantics of parentheses.	
expr1 `and` *expr2* *expr1* *expr2*	Process a packet if it meets both criteria.
expr1 `or` *expr2*	Process a packet if it meets either criterion.
`not` *expr*	Process a packet if it does not meet the criterion.

Interpreting the Report

The remaining areas of this section describe the output from `rvtrace`.

rvtrace Output with -addrinfo shows a sample of the output that `rvtrace` prints at the conclusion of each interval, when the `-addrinfo` flag is present:

- Time stamp—identifies the interval
- Multicast Data Statistics—summarizes Rendezvous *multicast and broadcast* packets during the interval, organized by UDP port (service) and destination address (see Multicast Data Statistics)
- Multicast Retrans Statistics—summarizes requests to retransmit packets of multicast and broadcast data (this table does not appear in rvtrace Output with -addrinfo; see Multicast Retransmit Statistics)

- PTP Statistics—summarizes Rendezvous *point-to-point* packets during the interval, organized by UDP port (service) and destination address (see Point-to-Point Statistics)
- Subject Statistics—recapitulates Rendezvous multicast and broadcast *message* activity during the interval, featuring information about *subject names* (see Subject Statistics)

Notice that each table begins with a network total, and then breaks down the total into subtotals and fine-grained categories.

rvtrace Output without -addrinfo shows a sample of the less verbose output that `rvtrace` prints when the `-addrinfo` flag is absent. Notice that tables omit the fine-grained categories—displaying only the network total and subtotals

General Network Load

To assess network load, inspect the Data and Bytes columns of the Multicast Data Statistics table, and the Data and Bytes columns of the Point-to-Point Statistics table.

Number of Senders

To determine the number of Rendezvous daemons that sent data messages during an interval, count the number of distinct source addresses in all source rows of the Multicast Data Statistics table and the Point-to-Point Statistics table.

Scanning for Problems

To quickly review `rvtrace` output for problems, scan down the right side of the page, looking for non-zero values in the Bad, Gaps, and Rbytes columns of the multicast data tables. Non-zero values in these columns indicate a problem; look more closely at statistics in other columns in that interval and subsequent intervals.

Figure 56: rvtrace Output with -addrinfo

```
Snapshot  2010-06-09 10:41:51  (10.0 elapsed seconds)

                                      Multicast Packet Statistics
  Port   Address          SCid    Data      Bytes  Null   Rdata    Rbytes   Gaps   Bad    R
 -----   --------------- ----- -------- --------- ----- ------- --------- ------ ----- ----
         Totals                      22      3978    24       0         0      0     6    -
  5039 * 10.97.128.255               11      1989    12       0         0      0     3    -
         10.97.128.30    37523        1       529     2       0         0      0     3   60
         10.97.128.31    49671       10      1460    10       0         0      0     0   60
  5662 * 10.97.128.255               11      1989    12       0         0      0     3    -
         10.97.128.30    54305        1       529     2       0         0      0     3   60
         10.97.128.31    55363       10      1460    10       0         0      0     0   60

                          Multicast Subject Statistics
  Port   Address          SCid    Msgs      Bytes  Subject
 -----   --------------- ----- -------- ---------  -------
         Totals                      22      3978
  5039 * 10.97.128.255                1       529  _RV.INFO.SYSTEM.HOST.STATUS.0A61801E
         10.97.128.30    37523        1       529
  5039 * 10.97.128.255               10      1460  subject.1
         10.97.128.31    49671       10      1460
  5662 * 10.97.128.255                1       529  _RV.INFO.SYSTEM.HOST.STATUS.0A61801E
         10.97.128.30    54305        1       529
  5662 * 10.97.128.255               10      1460  subject.1
         10.97.128.31    55363       10      1460
```

Figure 57: rvtrace Output without -addrinfo

```
Snapshot  2010-06-09 10:46:16  (10.0 elapsed seconds)

                                      Multicast Packet Statistics
  Port   Address          SCid    Data      Bytes  Null   Rdata    Rbytes   Gaps   Bad    R
 -----   --------------- ----- -------- --------- ----- ------- --------- ------ ----- ----
         Totals                      21      3449    23       0         0      0     4    -
  5039 * 10.97.128.255               10      1460    11       0         0      0     1    -
  5662 * 10.97.128.255               11      1989    12       0         0      0     3    -

                          Multicast Subject Statistics
  Port   Address          SCid    Msgs      Bytes  Subject
 -----   --------------- ----- -------- ---------  -------
         Totals                      21      3449
  5039 * 10.97.128.255               10      1460  subject.1
  5662 * 10.97.128.255                1       529  _RV.INFO.SYSTEM.HOST.STATUS.0A61801E
  5662 * 10.97.128.255               10      1460  subject.1
```

Bad Packets

Bad packets lack UDP checksums, or are corrupt in some other way.

Warning

Bad packets usually indicate a severe misconfiguration or network problem. Remedy the situation immediately.

> **Note**
>
> Checksums are crucial to correct operation of Rendezvous software; see Enable Packet Checksums.

False Bad Packets

In some situations, `rvtrace` can incorrectly report bad packets.

When a sending host computer enables checksum off-loading features, the network interface card (rather than the CPU) adds checksums to outbound packets. If `rvtrace` is running on the same host as the sender, it captures outbound packets before the checksums have been added. `rvtrace` detects the missing checksums, and reports bad packets. However, by the time these packets actually reach the network, they might not be bad packets.

Multicast Data Statistics

Multicast Packet Statistics shows a multicast data table (from `rvtrace -addrinfo`). The text below introduces important concepts. Multicast Packet Statistics—Column Headings describes the columns in detail.

Figure 58: Multicast Packet Statistics

Multicast Packet Statistics

Port		Address	SCid	Data	Bytes	Null	Rdata	Rbytes	Gaps	Bad	R
		Totals		79	23758	47	4	1032	0	0	-
5863	*	10.101.2.255		15	4298	7	0	0	0	0	-
		10.101.2.41	0	15	4298	7	0	0	0	0	60
5864	*	10.101.2.255		8	2225	4	0	0	0	0	-
		10.101.2.160	0	8	2225	4	0	0	0	0	60
5865	*	10.101.2.255		14	4199	7	0	0	0	0	-
		10.101.2.56	0	14	4199	7	0	0	0	0	60
5866	*	10.101.2.255		12	4086	5	0	0	0	0	-
		10.101.2.36	0	12	4086	5	0	0	0	0	60
5867	*	10.101.2.255		10	3790	4	0	0	0	0	-
		10.101.2.57	0	10	3790	4	0	0	0	0	60
20000	*	224.1.1.12		20	5160	20	4	1032	0	0	-
		10.101.2.102	39365	10	2580	10	2	516	0	0	60
		10.101.2.102	39364	10	2580	10	2	516	0	0	60

Notice that the rows divide visually into six groups, as indicated by a number in the `Port` `column and an asterisk (*).`

Network Total Row

The first row (immediately after the table and column headings, and before the four groups) is a *network total row;* the word `Totals` in the `Address` column is a visual cue. This row shows the grand total of multicast and broadcast packets on the network during the interval. For example, the Data column shows the total number of data packets that `rvtrace` detected on the network during the interval.

The remaining rows display more fine-grained information about those packets—grouping them by UDP service, destination address, and source address.

Subtotal Groups

A number in the `Port column indicates the UDP service for its row, and the group of rows that follow it. A blank in this column means that the row has the same port as the row above, and is part of the same subtotal group. Notice how the pattern of numbers and blanks in the Port column visually indicates the subtotal groups.`

Destination Row

`*` flags a row as a *destination subtotal row*. A blank (space character) in this column flags a row as a *source* row. Each group begins with a destination subtotal row, followed by one or more source rows.

Each destination subtotal row is the heading and subtotal for the source rows that follow it. For example, consider the destination row with `20000` in the `Port` column, and `224.1.1.12` in the Address column. The Data column indicates 20 packets on UDP service `20000` sent to the multicast group `224.1.1.12`. The two subsequent source rows indicate that those 20 packets came from two sources—the daemon or IPM with SCid `39365` at `10.101.2.102` sent 10 packets, while SCid `39364` at the same host sent another 10 packets. The subtotal 20 in turn contributes to the grand total 51 in the first row.

A destination subtotal row *governs* the source rows below it (until the next destination subtotal row). That is, the UDP service (port) and address in the *governing row* apply to those source rows. Similarly, the governing row address implies either multicast or broadcast protocol, and this protocol also applies to the statistics in the source rows that it governs. (Naturally, all of this information also applies to the governing row itself.)

Source Row

Each *source row* shows a very narrow subset of packet activity during the interval—packets on a specific UDP service (port), with a specific destination address, and *originating* at a specific source (IP address). The Address column shows the source; the UDP service and destination address are specified in the governing row (that is, the nearest preceding destination subtotal row).

Statistics

In *destination rows* numbers in statistics columns count packets with the destination specified in the Address column.

In *source rows* numbers in statistics columns count packets originating from the IP address in the Address column.

In *network total rows*, numbers in statistics columns represent the packet totals for the network during the interval.

Multicast Packet Statistics—Column Headings

Column	Description
`Port`	In *destination subtotal rows,* this column contains a UDP port number indicating the Rendezvous service for the group of rows that it begins. In *source rows* this column is blank; the service in the nearest preceding destination row also applies to the source row.
`*`	Asterisk (`*`) in this unlabeled column indicates a destination subtotal row. Blank in this column indicates a source row.
`Address`	In *destination rows* this column shows the destination address shared among a group of packets. It can be an IP address or a multicast group. In *source rows* this column shows the IP address from which group of packets originate. In *network total rows*, this column contains the word `Totals`.
SCid	Service communication ID. In source rows, this value differentiates the source of packets when several daemons or IPM instances on the same host computer reuse the same service port. Zero in this column indicates that the source is the only sender on that service

Column	Description
	and host. In destination rows this column is blank.
`Data`	Data packets. This column shows the number of multicast or broadcast data packets.
`Bytes`	Data bytes. This column sums the number of payload bytes over the data packets (as counted in the Data column).
`Null`	Null packets. When a Rendezvous daemon has no data packets to transmit, it periodically sends *null packets* to maintain continuity. This column displays the number of null packets that `rvtrace` detected.
`Rdata`	Retransmitted data packets. `rvtrace` counts retransmitted packets separately from first-time data packets. This column displays the number *retransmitted* data packets during the interval. Semantics of this column are otherwise analogous to the Data column. For statistics concerning retransmission requests and rejections, see Multicast Retransmit Statistics.
`Rbytes`	Retransmitted bytes. This column sums the number of payload bytes over the retransmitted data packets (as counted in the Rdata column).
`Gaps`	Sequence gaps. `rvtrace` tracks the serial numbers of Rendezvous packets. The Gaps column counts the missing packets in each sequence of multicast or broadcast data packets. For more information, see Gaps Diagnoses.

Column	Description
Bad	Bad packets. This column shows the number of packets that lack UDP checksums, or are corrupt in some other way. **Warning** See Bad Packets.
R	Reliability. A numeric value indicates the reliability of a specific service on a specific host. Hyphen (–) is a place holder in rows that don’t represent a host (that is, in port rows and total rows).

Gaps Diagnoses

A sequence gap can occur in two situations:

- `rvtrace` misses one or more packets; that is, the hardware or operating system on which `rvtrace` is running drops one or more packets.
- The network infrastructure drops one or more packets between their source and `rvtrace`.

To determine which of these two situations has actually occurred, check the Rdata values within the interval and in subsequent intervals. If `Rdata` remains at zero, then it is likely that `rvtrace` alone is missing packets. If `Rdata` is non-zero, then it is likely that the network infrastructure is dropping packets (`Rdata` is non-zero because other daemons on the network are requesting retransmission of the missing packets).

Multicast Retransmit Statistics

Sending Rendezvous daemon process instances retransmit missed packets upon request from receiving Rendezvous daemons. This table displays statistics related to those retransmission requests. For statistics concerning the actual retransmitted packets, see Multicast Data Statistics—in particular, the Rdata and Rbytes columns.

The Rreq column of this table counts point-to-point packets. In contrast, the actual retransmitted data packets use the same protocol (multicast or broadcast) as the original data packets that they recapitulate (as do the rejection notices in the Rrej column).

Warning

In switched Ethernet environments point-to-point packets remain invisible to `rvtrace`—except for packets addressed specifically to the `rvtrace` host computer. This fact severely limits the usefulness of retransmit statistics in switched networks.

In some switched networks, you can disable switching behavior—for example, by setting one port to diagnostic mode, or by using a diagnostic utility. This tactic can yield the full stream of point-to-point packets in a limited portion of the network; run `rvtrace` in that portion.

Multicast Retransmit Packet Statistics shows a multicast retransmit table (from `rvtrace -addrinfo`). The text below introduces important concepts. Multicast Retransmit Packet Statistics—Column Headings describes the columns in detail.

Figure 59: Multicast Retransmit Packet Statistics

```
                          Multicast Retrans Packet Statistics
 Port    Address           SCid      Rreq        Rseq  Rrej      Bad
-----    ---------------- ----- ------- ---------- ----- -------
         Totals                        12          54     0       12
 5662                                   4          20     0        4
       * 10.97.128.30          -        0           0     0        0
                                        4          20     0        4
       * 10.97.128.31     50416         4          20     0        4
                                        0           0     0        0
 5039                                   8          34     0        8
       * 10.97.128.30          -        0           0     0        0
                                        8          34     0        8
       * 10.97.128.31     58026         8          34     0        8
                                        0           0     0        0
```

Network Total Row

The first row (immediately after the table and column headings) is a *network total row;* the word `Totals` in the `Address` column is a visual cue. This row shows the grand total of packets related to retransmission detected on the network during the interval.

The remaining rows display more fine-grained information about those packets—grouping them by UDP service, and destination or source IP address.

Port Subtotal Row

The second row in Multicast Retransmit Packet Statistics is a *port subtotal row*—its columns subtotal the statistics over the subsequent destination and source rows which it governs (until the next port subtotal row).

A number in the `Port` column indicates the UDP service for its row, and the group of rows that follow it. A blank in this column means that the row has the same port as the row above, and is part of the same subtotal group. Notice how the pattern of numbers and blanks in the `Port` column visually indicates the subtotal groups.

Destination and Source Rows

For each IP address with retransmission request activity, this table contains a destination row and a source row—always paired in that order. An `*` and an IP address (in the Address column) flags a row as a *destination row*. A blank (space characters) flags a row as a *source* row. The address in the destination row also applies to the source row that immediately follows it.

Counting Packets

This table displays each packet twice—once in a destination row, and once in a source row.

In each statistical column, the number in the port subtotal row is equal to the sum of the values in the destination rows, which is also equal to the sum of the values in the source rows.

In many networks it is possible to match the numbers in the source row for one IP address against the numbers in the destination row for another IP address. From this information you can deduce which Rendezvous daemons are missing packets and requesting retransmissions.

Multicast Retransmit Packet Statistics—Column Headings

Column	Description
`Port`	In *port subtotal rows,* this column contains a UDP port number indicating the Rendezvous service for the group of rows that it begins.

Column	Description
	In destination and source rows this column is blank; the service in the nearest preceding port subtotal row governs the destination and source rows below it.
*	Asterisk (`*`) in this unlabeled column indicates a destination row. Blank in this column indicates a source row.
`Address`	In *destination rows* this column shows the destination IP address of retransmission request packets (that is, the Rendezvous daemon that originally sent data packets). In *source rows* this column shows the IP address from which retransmission request or rejection packets originate (that is, the Rendezvous daemon that missed receiving data packets). In *network total rows*, this column contains the word `Totals`.
SCid	Service communication ID. In *destination rows* this column differentiates the destination ID of retransmission request packets (that is, the Rendezvous daemon that originally sent data packets). In *source rows* this column is blank.
`Rreq`	Retransmission requests. This column displays the number of packets that contain retransmission requests. Each request packet counts separately, even if several request packets specify the same data packet numbers for retransmission. For example, if two daemons each request retransmission of the data packets numbered 121–125, and a third daemon requests retransmission of the data packets numbered 100–144, then `Rreq` is 3.
`Rseq`	Retransmission sequence numbers. Each retransmission request packet can solicit one or more data packets for retransmission. This column sums the number of data packets for which retransmission is requested over the request packets (as counted in the Rreq

Column	Description
	column). If some data packets are requested several times, each data packet counts separately each time it is requested. For example, if three daemons request retransmission of data packets numbered 121–125, then the `Rseq` sum is 15. For more information, see Diagnoses.
`Rrej`	Retransmission rejection notices. Although Rendezvous daemons comply with retransmission requests whenever possible, sometimes the requested packets are no longer available. This column displays the number of packets that contain retransmission rejections. (Daemons send these rejection notices in multicast packets.)
`Bad`	Bad packets. This column shows the number of packets that lack UDP checksums, or are corrupt in some other way. **Warning** See Bad Packets.

Diagnoses

Scanning for Problems described a quick scanning technique for locating problems in `rvtrace` output, looking for non-zero values in the Bad, Gaps, and Rbytes columns of the multicast data tables. When such a scan indicates problems, look more closely at the retransmit statistics in nearby intervals.

Rseq measures retransmission requests for missed multicast or broadcast packets. Non-zero Rseq values generally indicate a problem. The ratio Rdata/Data measures the severity of the problem. Small ratios indicate low-level problems, which you can investigate as time permits. Ratios of 2% or greater indicate potentially serious network problems; investigate further. High ratios that last for only one interval, could indicate an intermittent problem, which could become more serious in other situations.

Notice that Rseq tabulates packets that serve a feedback mechanism within the protocol. A data receiver becomes a feedback sender when it detects that it has missed data packets. So the Rseq value in source rows indicates a data receiver *sending* retransmission requests. Conversely, the Rseq value in destination rows indicates a data sender *receiving* retransmission requests.

Consider the following two examples.

Difficulty at One Specific Receiver

Rseq Reveals Difficulty at a Receiver shows `rvtrace` output for three intervals, which indicate a difficulty at one specific receiver. The administrator must investigate that receiver, its network hardware, and its load.

Several situations could cause this pattern in `rvtrace` display output. For example:

- One slow computer is flooded by too much data from a network of faster senders; the receiver cannot process inbound data as fast as the rest of the network.
- One receiver with intermittent network interface failures or a loose network cable.

Figure 60: Rseq Reveals Difficulty at a Receiver

```
Snapshot 2010-06-02 10:14:38 (10.0 elapsed seconds)
                                    Multicast Packet Statistics
 Port   Address          SCid    Data     Bytes  Null   Rdata    Rbytes   Gaps   Bad    R
-----   ---------------- ----- ------- --------- ----- ------- --------- ------ ----- ----
        Totals                    989    311535     1     641    201959      9     0    -
 7599 * 225.9.9.10                989    311535     1     641    201959      9     0    -
        10.101.3.237     49039    989    311535     0     640    201600      9     0   60
        10.101.3.251     34523      0         0     1       1       359      0     0   60

                            Multicast Retrans Packet Statistics
 Port   Address          SCid    Rreq      Rseq  Rrej    Bad
-----   ---------------- ----- ------- --------- ----- ------
        Totals                     20      2222     0      0
 7599                              20         7     0      0
      *  10.101.3.74         -      0         0     0      0
                                    4         7     0      0
      *  10.101.3.237    49039     20      2222     0      0
                                    0         0     0      0
      *  10.101.3.246        -      0         0     0      0
                                   14      2211     0      0
      *  10.101.3.251    34523      0         0     0      0
                                    2         4     0      0

Snapshot 2010-06-02 10:14:48 (10.0 elapsed seconds)
                                    Multicast Packet Statistics
 Port   Address          SCid    Data     Bytes  Null   Rdata    Rbytes   Gaps   Bad    R
-----   ---------------- ----- ------- --------- ----- ------- --------- ------ ----- ----
        Totals                   1000    315044     2      62     19530      0     0    -
 7599 * 225.9.9.10               1000    315044     2      62     19530      0     0    -
        10.101.3.237     49039    999    314685     0      62     19530      0     0   60
        10.101.3.246     34523      1       359     2       0         0      0     0   60

Snapshot 2010-06-02 10:14:58 (10.0 elapsed seconds)
                                    Multicast Packet Statistics
 Port   Address          SCid    Data     Bytes  Null   Rdata    Rbytes   Gaps   Bad    R
-----   ---------------- ----- ------- --------- ----- ------- --------- ------ ----- ----
        Totals                    999    314685     2       0         0      0     0    -
 7599 * 225.9.9.10                999    314685     2       0         0      0     0    -
        10.101.3.237     49039    999    314685     0       0         0      0     0   60
        10.101.3.246     33201      0         0     1       0         0      0     0   60
        10.101.3.251     34523      0         0     1       0         0      0     0   60
```

In Rseq Reveals Difficulty at a Receiver, the first interval shows 9 sequence gaps in the multicast statistics table—that is, 9 gaps in the stream of multicast packets. The Rseq column of the multicast retransmit table contains further details; the host at address `10.101.3.246` requested 2211 packets for retransmission, while the other hosts requested a total of 11 packets. Conclude that the locus of the problem is at `10.101.3.246`, and that retransmit requests from the other receivers are side effects.

The second interval of Rseq Reveals Difficulty at a Receiver shows zero sequence gaps—the problem has abated. Nonetheless, the Rdata and Rbytes columns indicate that retransmissions continue as Rendezvous daemons recover from the problem by resending stored data.

By the third interval of Rseq Reveals Difficulty at a Receiver, everything has returned to normal.

Difficulty at One Specific Sender

Rseq Reveals Difficulty at a Sender shows output indicating a difficulty at one specific sender. The administrator must investigate that sender, its sending applications, and its network hardware.

Several situations could cause this pattern in `rvtrace` display output. For example:

- The sender is flooding the network—that is, it is sending packets faster than most other daemons on the network can receive them.
- The sender has intermittent network interface failures or a loose network cable.

In Rseq Reveals Difficulty at a Sender, the multicast statistics table shows 411 sequence gaps—that is, 411 gaps in the stream of multicast packets. Moreover, all the missing packets originate at one sender, `10.101.3.246`. The Rseq column of the multicast retransmit table contains further details; *both* of the receivers in the network requested those packets for retransmission—that is `10.101.3.74` and `10.101.3.237` both sent retransmit requests to `10.101.3.246`. Conclude that the locus of the problem is at `10.101.3.246`.

The Rdata column of the multicast statistics table shows that before the end of the interval, the sender had retransmitted all 411 missing packets. The problem was a brief glitch—the Rendezvous reliable transport mechanisms easily smoothed over this temporary rough spot. Nonetheless, if such behavior recurs, the administrator must investigate the problem.

Figure 61: Rseq Reveals Difficulty at a Sender

```
Snapshot 2010-06-12 08:49:37 (10.0 elapsed seconds)
                                     Multicast Packet Statistics
 Port    Address          SCid    Data      Bytes  Null   Rdata    Rbytes   Gaps   Bad    R
-----    ---------------- ----- ------- --------- ----- ------- --------- ------ ----- ----
         Totals                     588    185220     0     411    129465    411     0    -
 7599 * 225.9.9.10                  588    185220     0     411    129465    411     0    -
         10.101.3.246     49039     588    185220     0     411    129465    411     0   60

                                 Multicast Retrans Packet Statistics
 Port    Address          SCid    Rreq       Rseq  Rrej    Bad
-----    ---------------- ----- ------- --------- ----- ------
         Totals                       6       822     0      0
 7599                                 6       822     0      0
       * 10.101.3.74          -       0         0     0      0
                                      3       411     0      0
       * 10.101.3.237     33201       0         0     0      0
                                      3       411     0      0
       * 10.101.3.246     49039       6       822     0      0
                                      0         0     0      0
```

Point-to-Point Statistics

Point-to-Point Statistics shows a point-to-point (PTP) table (from `rvtrace -addrinfo -ptp`). The text below introduces important concepts. Point-to-Point Statistics—Column Headings describes the columns in detail.

Warning

In switched Ethernet environments point-to-point packets remain invisible to `rvtrace`—except for packets addressed specifically to the `rvtrace` host computer. Since this fact severely limits the usefulness of reporting point-to-point statistics, `rvtrace` *omits* them from its output unless you specify the `-ptp` command line option.

In some switched networks, you can disable switching behavior—for example, by setting one port to diagnostic mode, or by using a diagnostic utility. This tactic can yield the full stream of point-to-point packets in a limited portion of the network; run `rvtrace` in that portion.

Figure 62: Point-to-Point Statistics

PTP Packet Statistics

Port	Address	SCid	Data	Bytes	AckR	Ack	Nak	Bad
-----	----------------	-----	--------	----------	-----	------	-------	------
	Totals		18	4158	18	18	0	36
20000			0	0	0	18	0	0
	* 10.101.2.102	23001	0	0	0	0	0	0
			0	0	0	10	0	0
	* 10.101.2.102	23002	0	0	0	0	0	0
			0	0	0	8	0	0
	* 10.101.2.249	20000	0	0	0	18	0	0
			0	0	0	0	0	0
23001			10	2310	10	0	0	20
	* 10.101.2.102	23001	10	2310	10	0	0	20
			0	0	0	0	0	0
	* 10.101.2.249	0	0	0	0	0	0	0
			10	2310	10	0	0	20
23002			8	1848	8	0	0	16
	* 10.101.2.102	23002	8	1848	8	0	0	16
			0	0	0	0	0	0
	* 10.101.2.249	0	0	0	0	0	0	0
			8	1848	8	0	0	16

Network Total Row

The first row (immediately after the table and column headings) is a *network total row;* the word `Totals` in the `Address` column is a visual cue. This row shows the grand total of packets related to retransmission detected on the network during the interval.

The remaining rows display more fine-grained information about those packets—grouping them by UDP service, and destination or source IP address.

Port Subtotal Row

The second row in Point-to-Point Statistics is a *port subtotal row*—its columns subtotal the statistics over the subsequent destination and source rows which it governs (until the next port subtotal row).

A number in the `Port` column indicates the UDP service for its row, and the group of rows that follow it. A blank in this column means that the row has the same port as the row above, and is part of the same subtotal group. Notice how the pattern of numbers and blanks in the `Port` column visually indicates the subtotal groups.

Destination and Source Rows

For each IP address with point-to-point data packet activity, this table contains a destination row and a source row—always paired in that order. An `*` and an IP address (in the Address column) flags a row as a *destination row*. A blank (space characters) flags a row as a *source* row. The address in the destination row also applies to the source row that immediately follows it.

Counting Packets

This table displays each packet twice—once in a destination row, and once in a source row.

In each statistical column, the number in the port subtotal row is equal to the sum of the values in the destination rows, which is also equal to the sum of the values in the source rows.

In many networks it is possible to match the numbers in the source row for one IP address against the numbers in the destination row for another IP address. From this information you can deduce which Rendezvous daemons are exchanging point-to-point data packets and requesting retransmissions.

Point-to-Point Statistics—Column Headings

Column	Description
`Port`	In *port subtotal rows,* this column contains a UDP port number indicating the Rendezvous service for the group of rows that it begins. In destination and source rows this column is blank; the service in the nearest preceding port subtotal row governs the destination and source rows below it.
`*`	Asterisk (`*`) in this unlabeled column indicates a destination row. Blank in this column indicates a source row.
`Address`	In *destination rows* this column shows the destination IP address of point-to-point data packets. In *source rows* this column shows the IP address from which point-to-point data packets originate. In *network total rows*, this column contains the word `Totals`.

Column	Description
`SCid`	Service communication ID. In *destination rows* this column differentiates the destination ID of point-to-point data packets. In *source rows* this column is blank.
`Data`	Point-to-point data packets. This column shows the number of point-to-point data packets.
`Bytes`	Point-to-point data bytes. This column sums the number of payload bytes over the point-to-point data packets (as counted in the Data column).
`AckR`	Acknowledgement request packets. Sending Rendezvous daemons explicitly request positive acknowledgment for groups of point-to-point data packets. This column shows the number of packets containing acknowledgment requests for point-to-point data packets.
`Ack`	Acknowledgement packets. Receiving Rendezvous daemons explicitly acknowledge groups of point-to-point data packets upon request from sending daemons. This column shows the number of packets containing acknowledgments for point-to-point data packets.
`Nak`	Negative acknowledgement packets. Receiving Rendezvous daemons use negative acknowledgments to request retransmission of missing data point-to-point packets. This column displays the number of packets containing retransmission requests for point-to-point data packets. For more information, see Nak Diagnoses.
`Bad`	Bad packets. This column shows the number of packets that lack UDP checksums, or are

Column	Description
	corrupt in some other way. ⚠ **Warning** See Bad Packets.

Nak Diagnoses

Nak measures the number of point-to-point packets that request retransmission of point-to-point data.

Non-zero Nak values to or from a specific address usually indicates one of these problems:

- A faulty network interface card at a specific computer.
- A faulty or overloaded network infrastructure component (for example, switching or router hardware).
- A fast sender is overwhelming a slower receiver with point-to-point packets.
- A sender on a fast network is overwhelming a network infrastructure component by sending point-to-point packets to a receiver on a slower network.

Begin by checking the specific interface card, and widen the investigation to other components until you can resolve the difficulty.

Nak Indicates Faulty Network Card or Infrastructure Component displays example output with this pattern.

- SCid `35` at address `10.101.2.102` is sending point-to-point data to SCid `16` at `10.101.3.249`.
- The AckR column shows that `10.101.2.249` received 68 requests for acknowledgment to `10.101.2.102`.
- The Nak column shows that SCid `16` at `10.101.2.249` did not receive all the packets correctly, and sent 23 NAKs back to SCid `35` at `10.101.2.102`. These NAKs constitute retransmission requests for the missed point-to-point packets.
- The Ack column shows that eventually, `10.101.2.249` did receive all 68 retransmitted packets correctly, recovering from the problem.

- This particular example report does not contain sufficient information to determine the locus of the problem—it could be either at the sender or the receiver.

Figure 63: Nak Indicates Faulty Network Card or Infrastructure Component

PTP Packet Statistics

Port	Address	SCid	Data	Bytes	AckR	Ack	Nak	Bad
	Totals		1716	597168	68	68	23	0
20000			1716	597168	68	68	23	0
	* 10.101.2.102	22	0	0	0	0	0	0
			0	0	0	0	0	0
	* 10.101.2.102	35	0	0	0	68	23	0
			1716	597168	68	0	0	0
	* 10.101.2.249	16	1716	597168	68	0	0	0
			0	0	0	68	23	0

Subject Statistics

The subject table counts multicast and broadcast *messages* (not packets) and organizes statistics by Rendezvous subject name (in addition to UDP service and destination address).

Multicast Subject Statistics shows a subject table (from `rvtrace -addrinfo`). The text below introduces important concepts. Subjects Statistics—Column Headings describes the columns in detail.

Figure 64: Multicast Subject Statistics

Multicast Subject Statistics

Port	Address	SCid	Msgs	Bytes	Subject
	Totals		21	5446	
5863	* 10.101.2.255		1	286	_RV.RVRD.P
	10.101.2.41	0	1	286	
20000	* 224.1.1.12		20	5160	foo.1
	10.101.2.102	39365	10	2580	
	10.101.2.102	39364	10	2580	

Network Total Row

The first row (immediately after the table and column headings) is a *network total row;* the word `Totals` in the `Address` column is a visual cue. This row shows the grand total of messages that `rvtrace` detected on the network during the interval.

The remaining rows display more fine-grained information about those messages—grouping them by UDP service, destination address, subject name, and source address.

Subject Subtotal Groups

A character string in the Subject column indicates the Rendezvous subject name for its row, and the group of rows that follow it. A blank in this column means that the row has the same subject as the row above, and is part of the same subtotal group. Notice how the pattern of subject names and blanks in the Subject column visually indicates the subtotal groups. The visual pattern of numbers in the Port column echoes this division.

Each subject subtotal group begins with a subject row (which is also a destination row) followed by one or more source rows.

Destination and Source Rows

`*` flags a row as a *destination row*. A blank (space character) in this column flags a row as a *source* row.

Each destination row is the heading and subtotal for the source rows that follow it. For example, consider the destination row with `foo.1` in the Subject column. The Msgs column indicates 20 multicast messages. The two subsequent source rows indicate that those 20 messages came from two sources on the host `10.101.2.102`—that is, SCid `39365` sent 10 messages, while `39364` sent another 10 messages. The subtotal 20 in turn contributes to the grand total 21 in the network total row.

A subject row *governs* the source rows below it (until the next subject row). That is, the subject, UDP service (port), and address in the *governing row* apply to those source rows. Similarly, the governing row address implies either multicast or broadcast protocol, and this protocol also applies to the statistics in the source rows that it governs. (Naturally, all of this information also applies to the governing row itself.)

Statistics

In *destination rows* numbers in statistics columns count messages with the destination specified in the Address column.

In *source rows* numbers in statistics columns count messages originating from the IP address in the Address column.

In *network total rows*, numbers in statistics columns represent the message totals for the network during the interval.

Subjects Statistics—Column Headings

Column	Description
`Port`	In *destination subtotal rows,* this column contains a UDP port number indicating the Rendezvous service for the group of rows that it begins. In *source rows* this column is blank; the service in the nearest preceding destination row also applies to the source row.
`*`	Asterisk (`*`) in this unlabeled column indicates a destination subtotal row. Blank in this column indicates a source row.
`Address`	In *destination rows* this column shows the destination address shared among a group of messages. It can be an IP address or a multicast group. In *source rows* this column shows the IP address from which group of messages originate. In *network total rows*, this column contains the word `Totals`.
SCid	Service communication ID. In source rows, this value differentiates the source of packets when several daemons or IPM instances on the same host computer reuse the same service port. Zero in this column indicates that the source is the only sender on that service and host. In destination rows this column is blank.
`Msgs`	Rendezvous messages. This column shows the number of messages that use multicast or broadcast protocols.
`Bytes`	Data bytes. This column sums the number of payload bytes over the messages (as counted in the Msgs column).
`Subject`	This column shows the Rendezvous subject name shared among the messages in a subtotal group.

Subject Table Diagnoses

The subject table reveals interesting information about the subject name space, and its use within the network:

- Programs that send messages in violation of the subject usage guidelines for your enterprise
- Duplicate process instances of a sending program
- Subjects that consume large portions of network capacity

Certified Message Delivery

Forward RVCM Administrative Messages across Network Boundaries

Rendezvous certified message delivery software depends on administrative announcements (as well as point-to-point messages) between CM transports.

These messages travel freely within a single network segment. However, if your network consists of several segments connected by Rendezvous routing daemons, then you must instruct the routing daemons to forward the subjects in Critical Subjects for Certified Delivery.

Routing daemons must forward these subjects in both directions—import and export.

Similarly, if clients in your network use TLS to connect to `rvsd` or `rvsrd`, you must configure the secure daemon to authorize the subjects in Critical Subjects for Certified Delivery.

Critical Subjects for Certified Delivery

Subject	Description
`_RVCM.>`	Rendezvous certified delivery software uses administrative messages with these subjects. Routing daemons must forward these subjects in both directions.

Ledger File Location

The ledger file must reside on the same host computer as the program that uses it. Do *not* use network-mounted storage for ledger files.

Remember that certified message delivery protects against component or network failure. Placing ledger files across a network (for example, on a separate file server) introduces a new dependency on the network, leaving components vulnerable to network failures.

Prometheus Endpoints

Daemons

Rendezvous Prometheus endpoints provide a familiar interface for basic metrics. The following Prometheus endpoints are added to all the dameon HTTP/HTTPS interfaces:

- `/metrics`
- `/metrics/subscriptions`
- `/metrics/clients`
- `/metrics/neighbors`
- `/metrics/profiling`

rvtrace

By default, when `rvtrace` starts, an HTTP server is started on the `7595` port. The server presents the Prometheus metrics at the `/metrics` endpoint. The metrics are presented on the basis of the command-line parameters passed to `rvtrace`. For example, by default, `rvtrace` gathers point-to-point statistics only when you pass the `-ptp` parameter. Therefore, the Prometheus metrics present point-to-point statistics when you pass the `-ptp` parameter.

The `/metrics` endpoint uses data from the last snapshot created by `rvtrace`. `rvtrace` creates snapshots every 10 seconds unless you change the update interval by passing the `-u` flag. As a result, before the first snapshot is created, `rvtrace` returns the `404 NOT FOUND` error.

Fault Tolerance

Network Placement

When you deploy a fault-tolerant application, it is important to distribute the member processes appropriately across computers and across network segments. Independence increases the effectiveness of redundant processes. For details, see Distribute Members in TIBCO Rendezvous Concepts.

It is also important to use files in a way that does not jeopardize fault tolerance. For guidelines, see Member File Access in TIBCO Rendezvous Concepts.

Forward Fault Tolerance Messages across Network Boundaries

Rendezvous fault tolerance software depends on messages between group members, and on some Rendezvous system advisory messages.

These messages travel freely within a single network segment. However, if your network consists of several segments connected by Rendezvous routing daemons, then you must instruct the routing daemons to forward the subjects in Critical Subjects for Fault Tolerance.

Routing daemons must forward all these subjects in both directions—import and export.

Similarly, if clients in your network use TLS to connect to `rvsd` or `rvsrd`, you must configure the secure daemon to authorize the subjects in Critical Subjects for Fault Tolerance.

Critical Subjects for Fault Tolerance

Subject	Description
`_RVFT.*.`*group_name*	Rendezvous fault tolerance software uses messages with these subjects to communicate among group members. Routing daemons must forward these subjects in both directions.

Distributed Queues

Forward Administrative Messages across Network Boundaries

Rendezvous distributed queue software depends on administrative announcements (as well as point-to-point messages) between distributed member transports.

These messages travel freely within a single network segment. However, if your network consists of several segments connected by Rendezvous routing daemons, then you must instruct the routing daemons to forward the subjects in Critical Subjects for Distributed Queues.

Warning	We do *not recommend* sending messages across network boundaries to a distributed queue, nor distributing queue members across network boundaries. However, when crossing network boundaries in either of these ways, you *must* configure the Rendezvous routing daemons to exchange `_RVCM` and `_RVCMQ` administrative messages.

Routing daemons must forward the subjects in Critical Subjects for Distributed Queues in *both* directions—import and export.

Similarly, if clients in your network use TLS to connect to `rvsd` or `rvsrd`, you must configure the secure daemon to authorize the subjects in Critical Subjects for Distributed Queues.

Critical Subjects for Distributed Queues

Subject	Description
`_RVCMQ.>`	Rendezvous distributed queue software uses

Subject	Description
	administrative messages with these subjects. Whenever *potential scheduler members* run in one network, and *potential listener members* of the same distributed queue run in a second network, then routing daemons must forward these subjects in both directions between the two networks. Similarly, whenever *potential listener members* of the same distributed queue run in two separate networks, then routing daemons must forward these subjects in both directions between the two networks.
`_RVCM.>`	Rendezvous distributed queue software uses administrative messages with these subjects. Whenever a process in one network *sends* task messages to *potential scheduler members* in a second network, then routing daemons must forward these subjects in both directions between the two networks.
`_RVFT.>`	Rendezvous distributed queue software uses administrative messages with these subjects. Whenever *potential scheduler members* of the same distributed queue run in two separate networks, then routing daemons must forward these subjects in both directions between the two networks.

Store Files

Many daemons associated with TIBCO Rendezvous use store files to maintain configuration or state. This appendix presents details about store files that are common to all daemons.

Locking

Daemons access store files using a cooperative file locking regimen, which guarantees unique sequential access when two or more daemon processes attempt to share the same store file. This guarantee depends on the daemon processes' adherence to the locking regimen.

Warning

Since non-Rendezvous processes do not adhere to the locking regimen, they can cause store file corruption—for example, by replacing the store file at inappropriate times.

To prevent store file corruption, ensure that non-Rendezvous processes do not replace or modify a store file while any instance of its daemon is running. For example, avoid replacing a store file with a saved backup version while the daemon is running.

See Also

rvrd

rvsd

rvsrd

rvcache

Upgrading rvrd to a New Release

This appendix presents general guidelines and recommendations for the tasks associated with upgrading routing daemons to a new Rendezvous release. Many of these guidelines also apply when reconfiguring a routing daemon (without upgrading to a new product version).

Note Every enterprise is unique in the way it deploys routing Rendezvous daemons. Local factors such as network hardware, scripts and automated process restart can require modifications and adaptations to the instructions in this appendix. Before starting to upgrade, review your enterprise architecture and document your upgrade plan.

Preliminary Information

Before upgrading `rvrd` on a production server, we recommend upgrading in a test or development environment using the same procedure.

We recommend that you upgrade or reconfigure any `rvrd` only during scheduled downtime (not during production hours).

When upgrading routing daemons in an enterprise, we recommend upgrading them one at a time. Verify that each upgraded daemon is properly forwarding messages and cooperating with any redundant routing daemons (for fault tolerance). Only then upgrade the next routing daemon.

Backup

Before upgrading *any* `rvrd`, make backup copies of *all* `rvrd` store files and log files (throughout your network).

Configuration Changes

Determine whether you must modify the routing daemon's configuration. If so, prepare the new configuration before starting an upgrade. For more information, see Reconfiguring an Upgraded Routing DaemonReconfiguring an Upgraded Routing DaemonReconfiguring an Upgraded Routing Daemon.

Message Flow

Upgrading a routing daemon involves stopping the daemon, making changes, and starting the new daemon. While the daemon is stopped, it cannot forward messages. Two scenarios are possible:

- When redundant routing daemons cooperate for fault-tolerant service, you can upgrade one while the others continue to forward messages. This technique lets you upgrade without disrupting message flow.
- Otherwise, you might need to stop the flow of messages that require forwarding through the stopped daemon. (see Stopping Messages that Require Routing).

Clients

Routing daemon executables can also serve as ordinary daemons. If any client applications connect specifically to the daemon you are upgrading, you must stop those client processes before upgrading. (You may restart them such that they connect to a different daemon process during the upgrade; but see Message Flow.)

General Outline for Upgrading a Routing Daemon

Upgrading a routing daemon generally involves these steps:

Procedure

1. Stop the routing daemon process.
2. Install the upgraded routing daemon executable.
3. If you modified the routing daemon's configuration, install the new store file.
4. Restart the routing daemon.
5. If you stopped any application processes, restart them.

Reconfiguring an Upgraded Routing Daemon

When upgrading all routing daemons *in place* (that is, on the same host computers, without any changes to the physical configurations of those computers), you can configure the new daemons by copying the old daemons' store files, and after installing the new executable, restoring their respective store files.

In contrast, changing any physical detail of the hardware of any routing daemon can require changes to the store file configurations of several routing daemons. For example, if one routing daemon moves to a new host computer, you must modify its configuration and the configurations of all its neighbors. (At minimum, the local network interfaces configurations and neighbor interfaces configurations must reflect the change.) Similarly, changes to the network interfaces on one of the host computers requires changes to the configurations of several routing daemons. In such situations, you must modify the configurations.

Manually Changing the Configuration in a Store File

These steps outline one possible method for modifying the store file configuration.

Procedure

1. Copy the store file to another computer where `rvrd` is installed.
2. Start `rvrd -idle` on that computer. (This command line option lets you view and modify the configuration in the store file without routing any messages, binding any resources, or making any network connections.)
3. Use the browser administration interface to modify the router's network specifications and neighbor declarations as needed.
4. Stop the idle `rvrd` process.

5. Backup the modified store file, then install it on the appropriate host computer.

See Also

Store Files

Stopping Messages that Require Routing

When upgrading a non-redundant (and so, non-fault-tolerant) routing daemon, stopping that daemon completely stops message forwarding. If correct operation of the application processes depends upon continuous message forwarding, then stop all dependent applications before upgrading that routing daemon.

If any dependent applications use the Rendezvous certified message facility (RVCM), first ensure that all certified messages have been delivered to all certified listeners, *before* you stop these application processes. (For more information, see the `DELIVERY.COMPLETE` advisory in *TIBCO Rendezvous Concepts*.) After cleanly stopping all the processes that use RVCM, we recommend that you make backup copies of all RVCM ledger files.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The documentation for this product is available on the TIBCO Rendezvous® Product Documentation page.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.