



Discover the Future of CORBA

Micro Focus IIOP Domain Boundary Controller (I-DBC) 4.1.0

Deployment Guide

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
<https://www.microfocus.com>

© Copyright 2010-2021 Micro Focus or one of its affiliates.

MICRO FOCUS, the Micro Focus logo and VisiBroker are trademarks or registered trademarks of Micro Focus or one of its affiliates.

All other marks are the property of their respective owners.

2022-02-11

Contents

Preface	1
Contacting Micro Focus	1
Further Information and Product Support	1
Information We Need	2
Contact information	2
High Availability and Scalability	3
High Availability and Scalability	3
Different Flavours of HA and Scalability	3
High Availability and Scalability at System Level.....	3
High Availability and Scalability at Application Level.....	4
High Availability and Scalability with a Traffic Redirector	4
High Availability & Scalability as provided by the DBC	5
Traffic Redirection: NAT versus Direct Routing.....	6
Connection Bundling	7
High Availability Provided by Hot Standby.....	9
Monitoring	10
DBC Built-In Monitoring	10
End-to-End Monitoring.....	11
Deployment Considerations	12
Planning the Installation	12
Calculate Application Throughput	12
Calculate DBC Requirements	12
Deployment Example.....	13
Deployment Requirements	13
Replication	15
Replication Technology.....	15
Shared Host.....	15
Resources.....	15
Maintenance.....	16
Reliability and Asynchronous Operation	16
Limitations and Restrictions.....	17
Communication	17
Security	17
IOR Timeout	17
Object Keys	18
Duplication of Calls	18
Delayed OBJECT_NOT_EXIST	18
Configuration	18
Replication Interface	20
Replication Message Properties	21
Performance.....	21
Multi Processor Machines	21
Estimated Throughput	21
Runtime Object Values.....	22
Installation Notes.....	25
Performance Monitoring	27
Setting up the Usage Data Collector	27
Activating the Usage Data Collector	27
SPS Client.....	29
SPS Client Commands	29

Administrative Rights for SPS Client Operations	31
Hardened System.....	33
Requirements and Recommendations for Linux	33
Operating System.....	33
Network Services	34
Kernel and Network Stack	34
I-DBC Authentication	37
I-DBC Authenticator Architecture	37
Caveats	38
Generic Interface	38
Generic Use	40
Authentication Methods.....	42
RSA/ACE SecurID Mapping	42
I-DBC Authenticator Events	45

Preface

The Micro Focus IIOP-Domain Boundary Controller (I-DBC) is an infrastructure building block that can be deployed in many ways, in diverse scenarios. This **Deployment Guide** takes a closer look at special deployment requirements like high availability and scalability and discusses various topics related to the deployment of DBCs, such as performance monitoring and hardening the operating system. Most topics presented in this guide require a basic understanding of DBC concepts as presented in the **I-DBC Administrator's Guide**.

The **Deployment Guide** contains the following parts:

- “[High Availability and Scalability](#)” discusses how the DBC can be deployed to provide linear scalability and unlimited support for various high-availability scenarios.
- “[Replication](#)” describes how the replication feature provides stateful failover. Stateful failover means that any hardware or software failures will go completely unnoticed to the client.
- “[Performance Monitoring](#)” describes how I-DBC performance can be monitored.
- “[SPS Client](#)” describes the SPS Client, a command line interface which can be used to configure the SPS and to obtain state information about the SPS.
- “[Hardened System](#)” gives recommendations on how to set up an operating system hardened against the risk of attack.
- “[I-DBC Authentication](#)” explains the I-DBC Authenticator plugin, an authentication framework via a dedicated CORBA interface which can be used when the client cannot do certificate based authentication.

Contacting Micro Focus

Our Web site gives up-to-date details of contact numbers and addresses.

Further Information and Product Support

Additional technical information or advice is available from several sources.

The product support pages contain a considerable amount of additional information, such as:

- The WebSync service, where you can download fixes and documentation updates.
- The Knowledge Base, a large collection of product tips and workarounds.
- Examples and Utilities, including demos and additional product documentation.

To connect, enter <http://www.microfocus.com> in your browser to go to the Micro Focus home page.

Note:

Some information may be available only to customers who have maintenance agreements.

If you obtained this product directly from Micro Focus, contact us as described on the Micro Focus Web site, <http://www.microfocus.com>. If you obtained the product from another source, such as an authorized distributor, contact them for help first. If they are unable to help, contact us.

Information We Need

However you contact us, please try to include the information below, if you have it. The more information you can give, the better Micro Focus SupportLine can help you. But if you don't know all the answers, or you think some are irrelevant to your problem, please give whatever information you have.

- The name and version number of all products that you think might be causing a problem.
- Your computer make and model.
- Your operating system version number and details of any networking software you are using.
- The amount of memory in your computer.
- The relevant page reference or section in the documentation.
- Your serial number. To find out these numbers, look in the subject line and body of your Electronic Product Delivery Notice email that you received from Micro Focus.

Contact information

Our Web site gives up-to-date details of contact numbers and addresses.

Additional technical information or advice is available from several sources.

The product support pages contain considerable additional information, including the WebSync service, where you can download fixes and documentation updates. To connect, enter <http://www.microfocus.com> in your browser to go to the Micro Focus home page.

If you are a Micro Focus SupportLine customer, please see your SupportLine Handbook for contact information. You can download it from our Web site or order it in printed form from your sales representative. Support from Micro Focus may be available only to customers who have maintenance agreements.

High Availability and Scalability

This chapter describes how to configure the DBC to scale in high throughput scenarios and how availability of DBC services can be ensured in case of hardware or software failures. Before reading this chapter you should be familiar with the standard DBC system. Stateful failover (replication) for I-DBC installations is explained in [“Replication”](#).

High Availability and Scalability

The following sections discuss the different mechanisms that the DBC architecture offers to achieve high availability and scalability. The ways in which high availability and scalability are tackled are closely related, therefore they are presented together.

High Availability (HA): The service of the DBC will still be provided even if a hardware or software component fails. This is achieved by replicating components of the DBC Proxy to eliminate single points of failure and providing health monitoring facilities. If a component fails, a failover mechanism will use a replica of the failed component.

Scalability: Adapt the service of the DBC Proxy to fit higher requirements in terms of number of clients, throughput, or latency. Scalability can be achieved in several ways. The type of scalability presented here is obtained by operating multiple DBC Proxies in a cluster. A traffic redirector is used to distribute requests among DBC Proxies so that the load is shared.

Different Flavours of HA and Scalability

High availability and scalability can be provided in two ways. Either the application takes care of failover and load-distribution itself (application level), or it relies on some external mechanism to provide the failover service (system level).

High Availability and Scalability at System Level

To provide high availability and scalability on the system level, an external mechanism (on protocol level) is required. Such a mechanism is usually called cluster management software. A central part of this cluster management software is the traffic redirector. A traffic redirector is a software add-on or dedicated device that employs a load-balancing algorithm to distribute client connections to a “cluster” of servers. Typically, this software presents the cluster host as a single virtual host and provides a single virtual IP-address to the client. The traffic redirector of the cluster management software simply redirects network traffic from a failed or overloaded component to another working and less busy one in a way possibly transparent to the client. The load of message processing is reduced on the individual DBC Proxy machines in the cluster, allowing the deployment of less expensive hardware. Examples for cluster management software are Sun Cluster 3 or Linux Virtual Server. Examples for traffic redirectors are Cisco CSS (Content Service Switch) or Cisco SLB (Server

Load-Balancer) which is a feature of Cisco's IOS software and can be run on Cisco's switches.

High Availability and Scalability at Application Level

The other possibility is to make the client aware of redundant components, thus providing high availability and scalability on the application level. This usually requires a higher development effort, but there are benefits: the application can be tailored more precisely to the requirements it has to fulfill. This includes but is not restricted to: faster failover, behaviour based on knowledge about the failure state of components, better dynamic load-balancing, improved stickiness of sessions. It also saves avoids the cost of cluster management software or traffic redirector.

High Availability and Scalability with a Traffic Redirector

To provide high availability and scalability several DBC Proxies can be operated in a cluster. Each DBC Proxy in a cluster shares its properties with any other DBC Proxy in the same cluster. In the standard case (as depicted in [figure 1](#)), a traffic redirector will distribute the traffic from the clients amongst the DBCs in this cluster. A typical cluster would consist of at least two DBC Proxies.

The clients reach the DBC service via the virtual IP address (VIP), that is, the address of the traffic redirector (or load-balancer). The traffic redirector receives all the traffic and distributes the IP packets among the active DBC Proxies, based on the result of regular monitoring checks. If a DBC Proxy is overloaded or fails, the traffic redirector removes this DBC Proxy from its distribution list and forwards packets to the remaining set of active DBC Proxies. The traffic redirector takes care that IP packets belonging to a TCP connection are always directed to the same DBC Proxy. If a DBC Proxy fails in such a scenario, high availability is provided by terminating all TCP connections associated with the failed DBC Proxy and re-routing all new TCP connections to another DBC Proxy. The clients will see that their connections to the cluster are broken and they will establish new TCP connections.

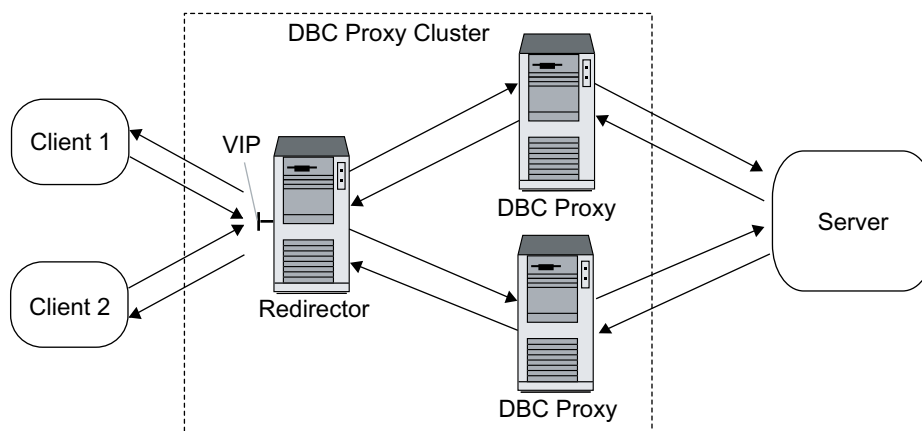


Figure 1 Multiple DBC Proxies with traffic redirector

High Availability & Scalability as provided by the DBC

The DBC Proxy offers several mechanisms to support high availability and scalability. In general, the recommended configuration uses at least the traffic redirector of a cluster management software at the domain boundary and does application level HA and scalability between DBC Proxies and Security Policy Servers (see [figure 2](#)). Therefore, a DBC installation can consist of multiple Security Policy Servers which constitute the **Security Policy Server Cluster**. All Security Policy Servers are configured the same way so that any of those Security Policy Servers can serve requests from any client. Thus, there can only be one Security Policy Server cluster belonging to a single DBC installation.

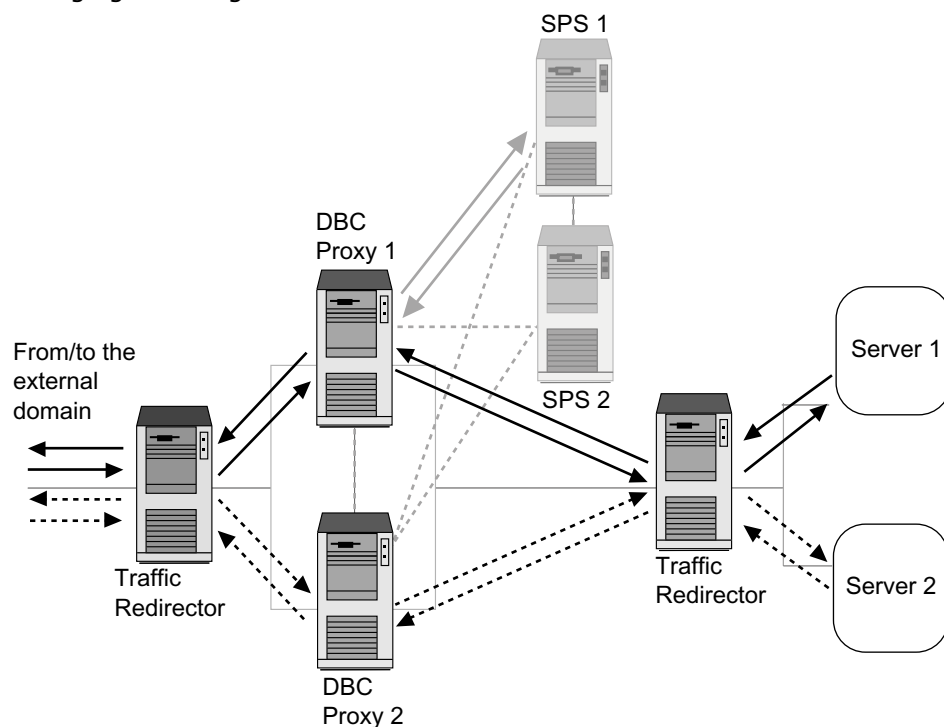


Figure 2 Recommended High Availability / Scalability configuration

Sensible configurations include at least two, but not more than ten Security Policy Servers. If operating only a single Security Policy Server, high availability and scalability are not provided.

Standard clients of these Security Policy Servers are the DBC Proxies. A DBC installation can have multiple clusters of DBC Proxies. Each DBC Proxy in a cluster shares its properties with any other DBC Proxy in the same cluster. In the standard case (as depicted in [figure 2](#)), a cluster management software distributes the traffic from the clients amongst the DBC Proxies in this cluster. A typical cluster would consist of at least two DBC Proxies.

The recommended configuration also requires the use of a cluster management software for the DBC Proxies. As a minimum, the traffic redirector is required. It is possible to operate without a traffic redirector, but then the distribution of clients must be achieved by other means, e.g., DNS round-robin. Doing so is not recommended.

If operating only a single DBC Proxy in a cluster, high availability and scalability are not provided for clients of this DBC Proxy.

The DBC Proxies are cluster-aware and interoperate with the cluster management, that is, they provide the cluster management with state information so that the cluster management can see if a DBC Proxy is still providing its service. Migration, as offered by some cluster management packages, is not supported by the DBC Proxy.

Towards the Security Policy Servers, the DBC Proxies provide application-level high availability and scalability themselves. DBC Proxies failover to another Security Policy Server autonomously. Therefore, no cluster management software is needed for the Security Policy Server. Multiple DBC Proxies statically distribute the load to the Security Policy Servers.

Traffic Redirection: NAT versus Direct Routing

There are various techniques for redirecting network traffic. The DBC can be used with *Network Address Translation (NAT)* and *Direct Routing (DR)*, as explained in the following sections. In both cases, the DBC software configurations on all cluster machines must be identical, except, of course, for local network addresses (the Administration Console takes care of this).

Network Address Translation (NAT)

The first redirection technique is *Network Address Translation (NAT)*. The redirector effectively is a NAT router, providing a virtual address (VIP) for the DBC service of the cluster, as shown in [figure 3](#). A client packet targeted at this virtual address is routed to one of the cluster DBC Proxies for processing, with the target address translated to the DBC Proxy's physical network address, the Real IP address (RIP). Replies from the DBC Proxy are routed back to the redirector, which translates the physical originator address back to the virtual DBC Proxy address before routing the reply to the client.

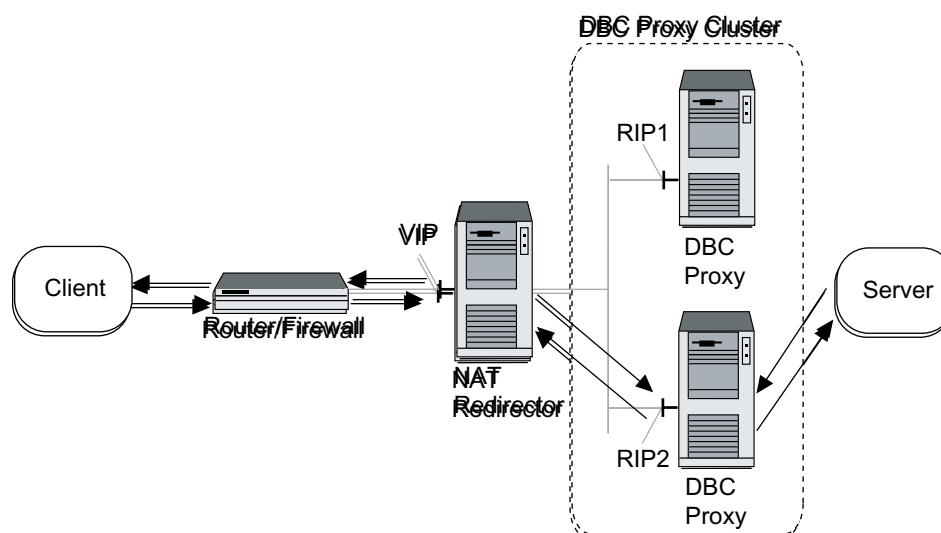


Figure 3 Traffic redirection using a NAT router

The individual DBC Proxy machines in the cluster must use the redirector as the default gateway for reply routing. The DBC software must be configured to use the virtual DBC Proxy address as NAT address on the external interface (assuming distribution is done for incoming client traffic). Apart from that, the configuration is the same as for a single DBC Proxy solution. An advantage of this redirection technique is that the DBC Proxies do not have to be located in the same physical network or on the same VLAN.

Direct Routing (DR)

The second redirection technique is Direct Routing (DR). Incoming and outgoing packets are routed on different paths (see [figure 4](#)).

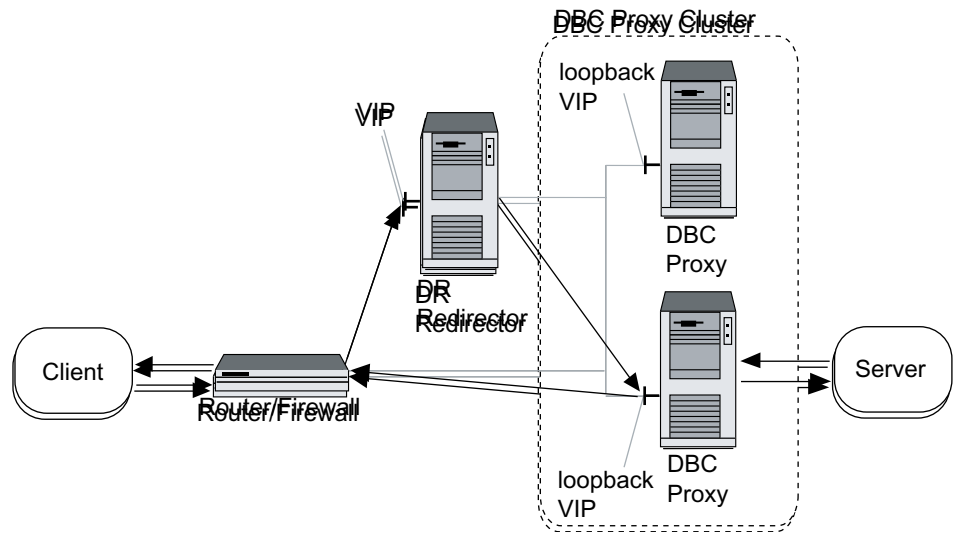


Figure 4 Traffic redirection using direct routing

All of the DBC Proxies in the cluster have the virtual DBC Proxy address (VIP) configured as an alias address typically on a loopback interface. The redirector forwards incoming client packets to one of the DBC Proxies for processing. Reply packets are routed directly to the client, bypassing the redirector.

This approach requires more complex configuration of the components. The router between the clients and the redirector must be configured to route all inbound client traffic to the redirector, but directly route outbound traffic. The individual DBC Proxy machines in the cluster must be capable of providing alias addresses on their loopback interface for configuring the virtual DBC Proxy address. Also, these addresses must be prevented from replying to ARP requests (see [“DBC Built-In Monitoring”](#)). The default gateway must be the router towards the client network. The DBC software must be configured to use the virtual DBC Proxy address (VIP) as external interface (assuming distribution is done for incoming client traffic). Apart from that, the configuration is the same as for a single DBC Proxy solution.

An advantage of this redirection technique is that it is faster than the NAT setup because replies are not routed via Traffic Redirector. Note that outgoing connections may not come from the VIP address. A disadvantage of the Direct Routing setup is that the DBC Proxies have to be located in the same physical network.

Connection Bundling

There is one problem with traffic redirection: CORBA IIOP is a multi-connection protocol, that is, a single set of application interactions between client and server may consist of multiple TCP connections. As the I-DBC is a stateful device with respect to exported IORs, all connections of a session must be routed to I-DBC Proxies which have this state available. The standard I-DBC edition has no provision for state replication between different I-DBC Proxy hosts. Accordingly, the redirector must recognize all connections of a session, and route them to the same I-DBC Proxy machine in the cluster. In other words, traffic redirection is restricted to complete sessions. This capability is usually called *bundling*, *persistence*, or *sticky*

mode. It is mandatory that this is enabled on the redirector, otherwise the I-DBC service will not work. As the director has no notion of what a session comprises for the I-DBC Proxy, all connections from the same source are routed to the same destination I-DBC Proxy in sticky mode.

Replication

The I-DBC enterprise edition provides a feature called “Replication”, enabling different I-DBC Proxies to share their IORs. When replication is active, operating the redirector in sticky mode is not necessary.

Usually, sessions are coupled with a timer. Once the last connection of a session is closed, its association to a particular machine remains active for a certain amount of time, so subsequent connections may continue the session. After this timer expires, new connections from the same source are considered to belong to a new session. A new association will be established for a different machine, based on the redirector’s load-balancing algorithm. The I-DBC software employs a similar timer, the Access Session termination timeout, that closes an Access Session once all client connections are closed. Both timeouts, on the I-DBC Proxy and on the redirector, must be configured to the same value.

Connection Bundling in tunneling scenarios

Although connection bundling is required for the IIOP protocol to function, it may cause a problem in tunneling scenarios, or in the case of clients hidden behind a masquerading firewall. In these cases all network traffic appears to come from a single IP address. Accordingly, the redirector has no means of distinguishing between the individual client sessions, in fact it will assume them all to be part of a single session. That way scalability is lost (in case the standard edition is used) but the architecture can still provide availability.

In tunneling scenarios, the bundling problem may be avoided by also using a cluster of DBC Proxies on the client side, where each DBC Proxy has an individual (possibly translated) address (see [figure 5](#)).

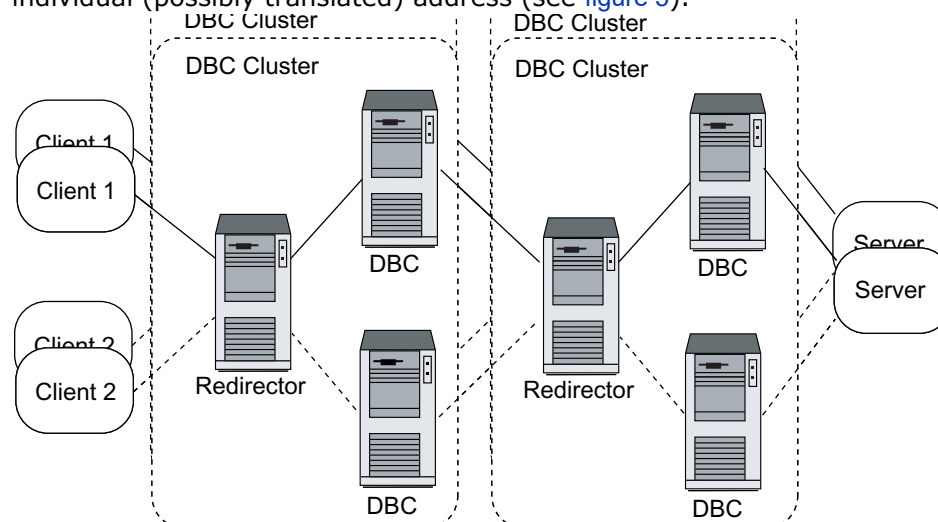


Figure 5 Load-balancing in a tunnel scenario

The server-side redirector may then use these addresses for load-balancing. This is especially useful if the tunnel runs over a high bandwidth network, as the client-side redirectors allow efficient use of the full bandwidth.

High Availability Provided by Hot Standby

The hot standby approach is based on a single machine hosting the *primary DBC installation*, which serves requests on a virtual IP address and performs normal message processing. In case of failures, a secondary (or “standby”) DBC machine takes over and guarantees uninterrupted service to clients.

This approach relies on a secondary DBC host *monitoring* the primary DBC host, and on network-level functionality to take over the virtual IP address used by the primary host. This functionality is offered by the separate *failover* package, which is included in the DBC distribution and combines with DBC specific monitoring and failover functions, as shown in [figure 6](#).

The DBC’s hot standby functionality is designed to mask two types of failures:

- failures of the entire machine, or the host’s network interface card (NIC), and
- failures of only the DBC Proxy process.

Failures of the Security Policy Server process are addressed by a different mechanism, viz. the failover functionality of the Security Policy Server Cluster, as explained in [“High Availability & Scalability as provided by the DBC”](#).

When the primary DBC host or just its network interface become unavailable, this is noticed by the *failover* daemon on the secondary DBC. This daemon will simply send out an ARP packet that announces the new NIC that now binds to the virtual IP address, so client requests will now arrive at the secondary DBC.

To integrate with the failover package Micro Focus provides additional monitoring components (see also the next section). These components are called *dbcmon* and *dbcfailover.sh*. *dbcmon* monitors the availability of the DBC Proxy and provides this monitoring information via HTTP. The script *dbcfailover.sh* regularly polls *dbcmon* and notifies the failover mechanism, which finally triggers the same ARP-based mechanism that was used to mask machine-level failures.

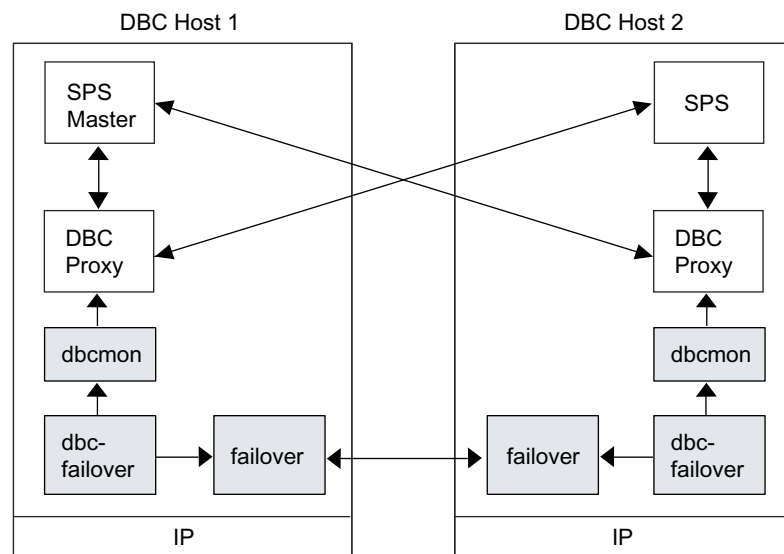


Figure 6 DBC Hot Standby

Note that to mask the failover process from clients, routers in the network where the DBC hosts are deployed must be configured to accept gratuitous ARP packets from the DBC hosts that announce that the virtual IP address is now to be mapped to the secondary DBC's network interface card (NIC). The primary DBC host can re-obtain this virtual address when it is back in operation.

Monitoring

To provide high availability all involved components must be constantly monitored to check their availability. There are many techniques that can be employed to monitor hardware, software, and network links. At a minimum, a simple `ping` test can be used to check the availability of a DBC Proxy machine. However, this does not verify the availability of the actual DBC software running on that machine.

The DBC supports two principal monitoring approaches. It can interwork with the monitoring mechanism of a traffic redirector (see next section), or it can use end-to-end monitoring, that is, access the Server across the DBC (this approach is explained in more detail in the section [“End-to-End Monitoring”](#)).

DBC Built-In Monitoring

The DBC software provides a facility for external monitoring, which is used by an additional monitoring agent software. This monitoring agent (DBC`Agent`) checks the availability of the DBC Proxy at regular intervals and provides this information to a monitoring agent by HTTP or port availability. When the master fails, the monitoring agent sends a gratuitous ARP which tells the routers and/or switches that the association between the VIP and the MAC address has changed. From then on IP packets destined for the VIP will be forwarded to a bystanding DBC Proxy. Note that sending the ARP is not part of the DBC`Agent`, and that setting up ARP sending tools is outside the scope of this document.

Interworking with Traffic Redirector Monitoring

Most traffic redirectors use a monitoring mechanism to determine the availability of individual cluster machines, that is, DBC Proxies, as well as the services running on them. The DBC's monitoring agent can be queried by the redirector monitor via a specific protocol. Currently an HTTP Agent is provided. Agents for other protocols or monitor products can be provided via professional services.

DBC`Agent` in Detail

The DBC monitoring mechanism (DBC`Agent`) works as follows. If the DBC Proxy is operational, it writes a single character to a FIFO queue roughly every second. The DBC`Agent` reads the FIFO queue. If it does not see a new character for five seconds (default), it will flag the DBC Proxy as down. The DBC`Agent` can be queried externally by opening a TCP connection to a port specified when starting the DBC`Agent`. If the DBC Proxy is up, the DBC`Agent` will send an HTTP reply with the state of 200 OK. If the DBC Proxy is down, DBC`Agent` will either send 503 Service unavailable or, if used with the `-a` option, it will refuse the connection. As most traffic redirectors are used for web servers, it is easy to configure the traffic redirectors to check the DBC`Agent` at regular intervals to find out if the DBC Proxy is up or down. The request actually sent to the DBC`Agent` is ignored by the DBC`Agent`. Thus, it does not matter which document is requested by the traffic

redirector's monitoring. For further reference, please refer to the man-page of the `DBCAgent`.

End-to-End Monitoring

The best way to ensure the proper operation of the I-DBC is to access a CORBA service across it. The monitoring interface of the CORBA server which provides information whether the service is working properly can be used for this purpose. If the service can be accessed from the DBC Proxy host across the DBC Proxy, all is well and the DBC Proxy is flagged up. Otherwise, it is flagged down. You simply need to implement a small CORBA client accessing the service at regular intervals and open a TCP listener if the test succeeded, and close the TCP listener again if it fails.

With this method, a very reliable monitoring can be achieved. The only remaining problem is that the failure of the test does not tell you whether the DBC Proxy or the original CORBA server is down or if something is misconfigured in between. For traffic redirectors capable of querying multiple sources, you can run the `DBCAgent` in parallel, and, if the service is considered unavailable by the traffic redirector, check the `DBCAgent`'s output to see if the DBC Proxy is causing the outage or the CORBA server.

Operating the DBC Proxy with Servers “In Line”

Sometimes, it is sensible to operate a DBC Proxy together with a bunch of servers as a failover group (see [figure 7](#)). In this case, the DBC Proxy and the servers are regarded as a unit. Whether the server(s) or the DBC Proxy fails does not matter – the whole unit is failed over to a hot-standby unit (or load-balanced onto the remaining units only).

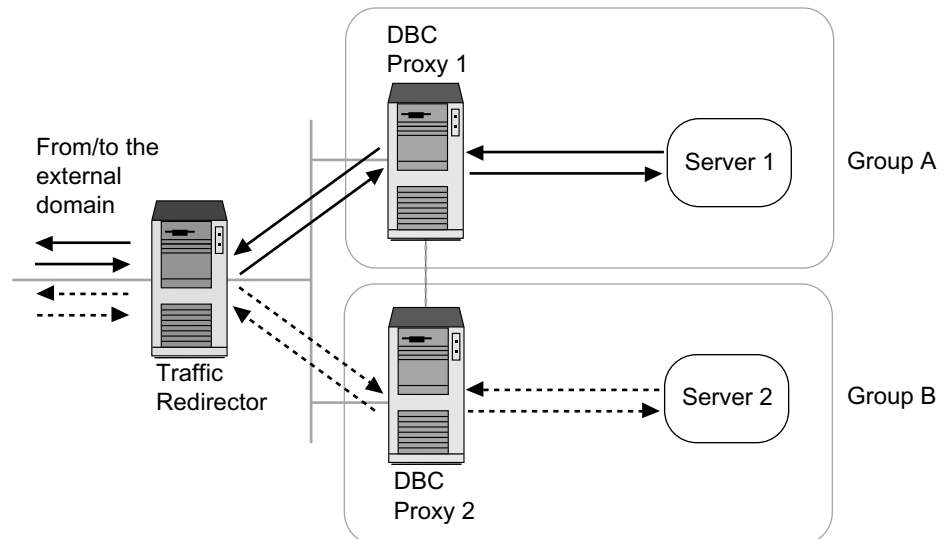


Figure 7 DBC Proxies with Servers “in line”

This setup saves the cost for an additional traffic redirector in front of the servers (see [“Recommended High Availability / Scalability configuration”](#)). It is a common scenario when the DBC is used for access control for all services offered by the servers and there is no other path of access to the servers than across the DBC Proxies. This deployment provides for similar availability compared to a deployment where the servers have their own traffic redirector in addition to the one in front of the DBC Proxies.

Deployment Considerations

The optimal deployment of a load-balanced DBC architecture depends on several interdependent variables that must be considered during the planning phase. The following procedure may help to determine the requirements for a given scenario.

If availability is an issue, remember that all involved components (server, redirector, networks) must be laid out in a redundant form.

Planning the Installation

For *high availability*, the DBC can be operated in a failover scenario (variant A). The actual failover operation is provided by a component which is not part of the DBC itself. For *scalability*, the traffic redirector is responsible for routing the clients to different DBC Proxies to distribute the load evenly among the available DBC Proxies (variant B).

From the DBC perspective, both installation variants are considered equal. For the DBC Proxy, it makes no difference whether it is steadily working in parallel to another DBC Proxy (in the load-balancing scenario, variant B) or whether it stands by until the primary DBC Proxy fails and it is assigned the virtual IP (VIP) in a failover scenario (variant A).

Hot-standby vs. load-balancing

When planning the installation, it is necessary to consider the sizing of the components to decide if a load-balancing solution is needed or if a hot-standby solution will suffice. If a single DBC Proxy can handle all the requests alone, a hot-standby solution is adequate, though it does not hurt to use a traffic redirector. The only reason not to use a traffic redirector would be the cost of the traffic redirector. For a recipe to calculate the sizing of a DBC installation from application throughput demands, see the next section.

Note that if high availability is an issue, remember that all involved components (server, redirector, networks) must be laid out in a redundant form.

Calculate Application Throughput

- 1 Measure the average throughput required by a typical interaction in your application, between a single client and the server, without the DBC Proxy.
- 2 Calculate the total required throughput from the anticipated number of concurrent sessions and the measured single-session value.
- 3 Make sure the network deployed between the client and the server is capable of handling the total throughput. If not, you will have to upgrade the network first.
- 4 Make sure your server is capable of handling this total throughput. If not you will have to find a load balancing solution for this problem first.

Calculate DBC Requirements

- 1 Select a traffic redirector product that is capable of handling the total application throughput. Make sure it has at least minimal monitoring capabilities; if not, add a compatible monitor product.

- 2 Select a hardware platform for the DBC Proxy cluster machines.
- 3 Measure the maximum throughput that a DBC Proxy on the selected platform can provide. You can do this by running an increasing number of concurrent sessions of your application through the DBC Proxy, and finding the strongest downward bend in the resulting performance graph.
- 4 Calculate the total number of DBC Proxies required from the total throughput and the maximum throughput of the single DBC Proxy. You may need to repeat these last three steps to optimize the cost-to-performance balance.

After finishing the process, you may want to estimate the performance of the scenario under peak load. For that purpose, repeat the application throughput calculation, but this time, measuring the *maximum* single session throughput. Compare the resulting throughput against the capacities of your server, network, and redirector. Divide the throughput by the number of planned DBC Proxy machines, and check their performance graphs with this load.

Deployment Example

Consider an application that requires an average throughput of 200kBit/sec and a maximum throughput of 300kBit/sec in each direction for a typical IIOP session, that is, between a single client and the server. The session consists of 6 request/reply round trips where each has a message size of 4 kByte (ca. 192kBit/sec in each direction).

We anticipate a requirement for 250 parallel sessions, so the required average throughput in each direction is 48MBit/sec with a peak of 75 MBit/sec and 1500 messages per second. The total required average throughput (in both directions) is 96MBit/sec with a peak of 150MBit/sec and 3000 messages per second.

We have an existing infrastructure built on 100MBit/sec FastEthernet, which will be capable of sustaining this load both in the average and maximum case, and thus is sufficient for this application. The server runs on high-end hardware and is also capable of handling this load.

For this example, we assume the traffic redirector deployed can handle the 100MBit/sec full duplex of the network without measurable performance impact.

Performance tests were done with standard PC hardware for the DBC platform (because it has a good price/performance ratio). Specifically, a Dual-Pentium III 866MHz system with 512 MB RAM and two quality network interface cards. This machine was capable of handling a peak throughput of 85MBit/sec at 4 KByte per IIOP message with 2700 IIOP messages per second.

Consequently, we need two of these machines to handle the average application throughput. Assuming a good load-balancing algorithm in the redirector, each machine would handle 48MBit/sec, running at 64% load, and handling 1500 IIOP messages per second.

The two machines will also be capable of handling the maximum application throughput. Each would handle 75MBit/sec, running at 88% load.

Deployment Requirements

When planning the system, take the following requirements into account: The Security Policy Servers must be able to contact each other directly. This

is necessary for the synchronization of configuration data and state information between the SPSs.

Each Security Policy Server in the cluster must be able to contact any DBC Proxy directly on the respective local or NAT address. This means that the connection is made directly to the respective DBC Proxy and that no redirector must be interfering with the connection. This is absolutely necessary to make sure every DBC Proxy will be configured in the startup process.

Without a Traffic Redirector for the SPS Cluster

If you are not using a cluster management software (that is, a traffic redirector) on the Security Policy Server cluster, the only requirement is that each DBC Proxy must be able to connect to at least one SPS directly on the respective local or NAT address.

With a Traffic Redirector for the SPS Cluster

If a traffic redirector is used for the Security Policy Server cluster, only the virtual IP address of the cluster mapped to the Control Connection port of all Security Policy Servers needs to be reachable. Combinations are supported, e.g., accessing the virtual IP-address from the user interfaces and letting the DBC Proxies connect directly to the real IP-addresses.

Replication

The I-DBC enterprise edition offers a feature called "Replication" which enables stateful failover. Stateful failover means that any hardware or software failures will go completely unnoticed to the client.

If a cluster component fails and replication is not enabled, a client must reconnect to the overtaking I-DBC Proxy, that is, the client needs to start over from the beginning. Replication enables multiple I-DBC Proxies to share their state. Newly proxified IORs will be multicasted to the other I-DBC Proxies. This enables stateful failover as any I-DBC Proxy will have all the necessary information available to serve any client request.

Replication also provides better scalability, because in contrast to the standard edition, it is no longer necessary to operate the traffic redirector in sticky mode. Load-balancing can be done based on the individual load of the I-DBC Proxies and is no longer restricted by connection bundling (see ["Connection Bundling"](#)).

The following sections describe Replication in detail, covering the different modes and configuration options.

Replication Technology

State replication between different Proxy Processes is done using UDP messages. A replication ADD request containing the Access Session identifier, the original and the proxified IOR is sent for each newly proxified IOR to a multicast address on which every Proxy Process is listening. This multicast address must be unique for each cluster using the same cluster interconnect network. Acknowledges and RESOLVE requests are sent and received from and to a unicast UDP socket.

Another state that is replicated is the termination of Access Sessions. An Access Session may only end after the last client has closed its last connection to any of the Proxy Processes. Thus, the Proxy Processes need to agree upon this moment. This is accomplished using a distributed termination detection algorithm.

Shared Host

State replication between Proxy Processes running on the same host is done using the same mechanism as for state replication across host boundaries. To make this work, the sockets are configured to receive multicast packets originating from the same host. This behaviour is called "loopback". It is not to be confused with the loopback interface. Thus, if there is more than one Proxy Process running on the same host, that is, if multiprocessor systems are employed, loopback is enabled automatically, so that Proxy Processes running on the same host can participate just as any other Proxy Process located on a different host.

Resources

Replication needs two UDP addresses per Proxy Process, which need to be configured: a unicast sender/receiver and a multicast receiver. The replication sender/receiver is bound to a configurable port on a unicast address on the cluster interconnect network. Each Proxy Process needs its own port. Thus, the configured port number is the start of a range of ports,

one for each Proxy Process on the host. The multicast address can be freely chosen from the range of multicast addresses. Sending to and receiving from the multicast group will only be done using the interface specified by the unicast address.

Maintenance

For maintenance, the I-DBC Proxy Cluster supports check out and check in. Check out of a cluster member happens automatically when the cluster host or the I-DBC Proxy running on it is shut down. For check in, the I-DBC Proxy coming up must have its state synchronized with the state of the other cluster members. This is achieved by a procedure called "ResolveAll". When an I-DBC Proxy is started, the replication system will automatically multicast a "ResolveAll" request first. This tells the other cluster members to send their state to the newly started Proxy Process, effectively copying the cluster state to the new cluster member. This supports maintenance of an I-DBC Proxy Cluster in the following scenario:

A cluster with two I-DBC Proxies (I-DBC Proxy A and I-DBC Proxy B) is running. I-DBC Proxy A is taken down for maintenance. Check out happens automatically. After completing the maintenance, A comes up again. Next, B shall be shut down for maintenance. To be able to safely shut down B without interrupting the CORBA service exported via the I-DBC Proxy Cluster, the state of B must be replicated onto A before shutting down B. This will be done automatically during startup of I-DBC Proxy A with the check in procedure "ResolveAll". An event "ReplicationIORTableCopySuccess" will be generated, when the state transfer is complete. It has an attribute "role" with the value "client" for the newly started Proxy Process and "server" for any other Proxy Process. You need to wait for this event to occur before shutting down I-DBC Proxy B otherwise the state of the cluster will be corrupted. Eventually, the service will be interrupted because of lost state.

If check in "ResolveAll" fails, it is retried after a timeout (which defaults to 60 seconds). It is retried again for as many times as specified in the field "number of retries" (which defaults to 3). This timeout can be changed by the user (see section "[Configuration](#)"). If even the last retry fails, an event "ReplicationIOR-TableCopyFailure" will be generated.

Reliability and Asynchronous Operation

The replication mechanism is reliable in the sense that it assures that state replication to at least one peer Proxy Process on a different host has been successful for every state change. The mechanism can be operated either asynchronously or synchronously.

Asynchronous mode

Asynchronous operation means that GIOP messages are forwarded even if the replication requests triggered by proxifications in this message have not been acknowledged yet. Synchronous mode means that the GIOP message is forwarded only after all replication requests triggered by this message have been acknowledged.

Synchronous mode

In the synchronous case, we can be sure that each IOR has been replicated within the cluster, that is, it is present on at least two different cluster hosts. In case of failure of any I-DBC Proxy host, the replicated state can be used

after the client has established a new connection to one of the other I-DBC Proxies. Synchronous operation is slower than asynchronous mode: It increases the latency for each call by about 1.5 milliseconds on a fast ethernet cluster interconnect, but does not limit the achievable throughput for independent parallel CORBA requests. Depending on your reliability and speed requirements, we recommend using asynchronous operation if you're ready to sacrifice a little reliability for speed. Even in asynchronous operation, requests are retried, so asynchronous operation is nearly as reliable as synchronous operation.

Asynchronous operation will lead to severe problems when the cluster interconnect is not redundant and fails. It might take the I-DBC a while to recognize the failure of the cluster interconnect, but by then it has no means to replicate the state changes to its peer I-DBC Proxies.

Making the cluster interconnect redundant will reduce the problem, but it can still not be guaranteed that the replication request has been sent before the GIOP request has been forwarded. If the I-DBC Proxy machine fails between sending the GIOP request and sending the replication request, state information will be lost. Loss of state would inadvertently make the Access Session unusable for the client. However, the chance of a replication request being delayed longer than the corresponding GIOP message is very small.

Limitations and Restrictions

This section lists some implementation details to clarify limitations and restrictions when using replication. For instructions on how to configure replication with the Administration Console, please refer to section [“Configuration”](#).

Communication

The Replication Module binds to the configured multicast address and port. It sets the socket option `REUSEADDR` to enable multiple receivers on the same port. It also needs to know its real interface address (supplied via the configuration), because it will only respond with `ACK` packets, if the address given in the request for selection of the `ACK` sender matches the interface address part of the configuration dictionary “localAddress”. Translated addresses (NAT) are not supported for the cluster interconnect.

Security

For performance reasons, there is no encryption and no authentication between hosts on the cluster interconnect network. Therefore the cluster interconnect network must be trustworthy and should be isolated from any other network. Either a dedicated cluster interconnect network is used or replication is done via one of the other network interfaces. In the latter case, firewalls need to be in place to prevent UDP traffic from entering into or leaving the cluster interconnect.

IOR Timeout

If IOR Invalidation Timeout Triggered (II-TT) is active, IORs are timed out autonomously. Currently, there is no provision for deleting IORs from the replicated state or to prevent deletion as described for the Access Session management. Because this prevents reliable failover, it can not be used in a replicated cluster.

Object Keys

Object keys are unique per Proxy Process to prevent duplicates. In addition to the object key scheme employed in the I-DBC standard edition, the process id of the Proxy Process and a fixed random number chosen at startup is prepended to every object key.

Duplication of Calls

There is a chance of a method being called multiple times on the server when failover occurs during a method call. The problem is caused by the fact that the client can not decide whether the method has already been called on the server when the connection brakes while waiting for the result from the I-DBC Proxy. The client has to retry the call. Method calls thus need to be idempotent to prevent inconsistencies or transaction semantics need to be used. This is true for any distributed application.

Consider a situation where the client sends a request to the I-DBC Proxy. The request gets forwarded to the server and the server sends a reply. The I-DBC Proxy host goes down before the reply reaches the client. What will the client do?

The client gets a timeout from the I-DBC Proxy, because it is no longer there. The client reconnects, gets switched over to another I-DBC Proxy. Then, the client re-sends the request and all goes well. The only drawback is, that the server has served the request twice without the client knowing. But that will happen in configurations without the I-DBC Proxies as well, if the network is interrupted and the client reconnects to re-issue the request. It is the responsibility of the application programmer to anticipate this behaviour.

To ensure transaction semantics you must use a transaction monitor. The use of such a monitor is highly recommended for any mission-critical business application.

Delayed OBJECT_NOT_EXIST

If faced with an unknown IOR or a request addressing an unknown object key, the Proxy Process needs to emit a resolve request to check if the IOR or object key is present in the state of any other Proxy Process and wait for a reply. If the IOR or object key can not be found, OBJECT_NOT_EXIST will be thrown eventually. If an application expects to see OBJECT_NOT_EXIST exceptions as part of its normal operation on a regular basis, the application will be slowed down a bit, because this exception will only be thrown after the I-DBC has verified that the key is not known to any host.

Configuration

This section explains how to configure the replication feature with the Administration Console. Go to the "I-DBC Proxy Cluster" panel and activate replication by checking the box "Replicate the state between DBCs". Now you can configure the replication interface on the "I-DBC Proxy" panel. The

replication interface is the physical interface the Multicast Address binds to (see below). It is used to exchange cluster interconnect messages.

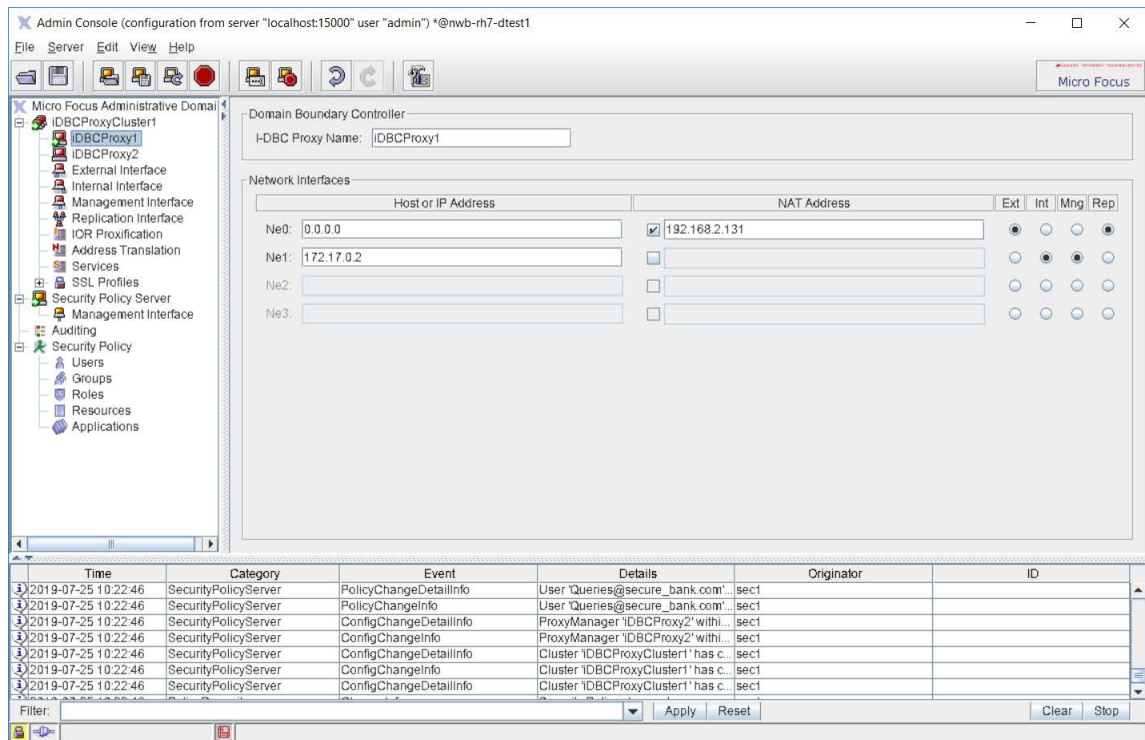


Figure 8 Network Interfaces – Replication Interface

The replication interface can be a separate interface, or it can be shared with any other interface, for example, the Internal or Management Interface. Note that the NAT Address of this interface does not apply the Replication Interface.

Replication Interface

After activating replication on the "I-DBC Proxy Cluster" panel, the panel "Replication Interface" will be available (see below).

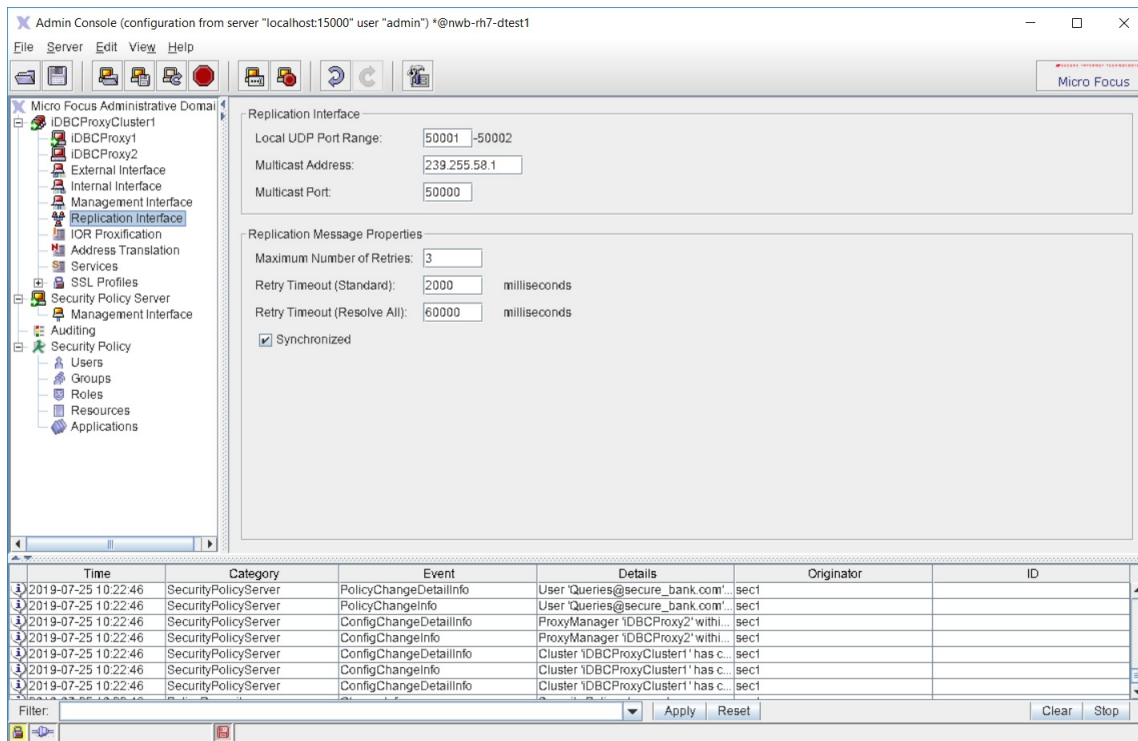


Figure 9 Replication Interface

On this panel you can configure replication properties:

- **Local UDP Port Range:** As stated before, the state information is exchanged between the different Proxy Processes using UDP messages. Here you configure the address of the unicast UDP sender and receiver. This address is used for sending ACKs and RESOLVES. The address must be on the cluster interconnect network. The UDP port given here is the base of a range of ports, one port for each Proxy Process on a host. The range must not overlap with the UDP port used in the multicast address (see below). The ports defined here will also be used as the target port for sending RESOLVES. Thus, all local ports of all I-DBC Proxies in a cluster are equal.
- **Multicast Address:** Defines the multicast address. The address must be between 224.0.0.2 and 239.255.255.255. As all addresses in the range 224.0.0.2 to 224.0.0.255 are link local addresses, these are recommended. The default multicast address is 239.255.58.1.
- **Multicast Port:** Defines the multicast UDP port. The UDP port can be chosen freely, but must be different from the ports specified for the local UDP port range.

Replication Message Properties

- *Maximum Number of Retries*: Defines the number of retries. This is valid for ADD, RESOLVE and RESOLVE_ALL.
- *Retry Timeout (Standard)*: The timeout for standard retries in milliseconds
- *Retry Timeout (Resolve All)*: The timeout for Resolve All retries in milliseconds
- *Synchronized*: If you check this box, GIOP messages will be forwarded only after the state replication has been confirmed. If not activated, state replication will be retried until number of retries is exceeded, GIOP messages will be forwarded before successful replication has been confirmed.

Performance

This section discusses how the performance of a cluster can be estimated.

Multi Processor Machines

All multicast requests need to be looped back to the same host if more than one Proxy Process is active on a host, which is usually only the case on a multiprocessor machine. This is expensive because the sending process will also see all of its own ADD requests.

“No looping” is selected automatically when the number of Proxy Processes is 1. If the number of Proxy Processes is larger than 1, “looping” is activated automatically.

When looping is selected, the sender can identify its own requests by the sender id contained in each ADD request. Thus, the sender can filter out its own packets before being processed further. This saves the effort for updating the IOR Table, but not the effort of receiving the packet and decoding at least part of it.

Estimated Throughput

The total capacity of the cluster is calculated by the capacity of single CPU multiplied by the number of CPUs, but limited by the time needed for replication. This is expressed by the following formula:

$$throughput = n \times \frac{1}{roundtriptime + overhead}$$

$$overhead = n \times p \times replicationtime,$$

where

- *n* is the number of CPUs in the cluster,
- *roundtriptime* the time for a method call including response,
- *replicationtime* the additional cost for the replication of a single proxified IOR.
- *p* is the proxification rate which gives the ratio of messages causing a proxification to the total number of messages. A message causing

proxification is a message transmitting an object reference (IOR) as parameter or return value.

The assumption here is that every *pth* requests a proxification including replication is done. Therefore, the time needed to accomplish this is the product of the time needed for a single replication multiplied by p and the number of CPUs.

If the rate of proxifications is low compared to the number of messages transmitted, scalability is expected to be very good, as can be seen in [Table 1](#), in the column for $p=0.1$. Please note that these numbers are synthesized. Measurements were made on two single processor Sun Ultra 10 running at 440 MHz. Measurements were taken to determine the average delay imposed on a method call without parameters returning a single IOR. The result was that a round-trip took no longer than 2000 microseconds without replication, and replication costs less than 500 microseconds per Proxy Process. The delays were verified to be linear by running 2 and 3 Proxy Processes on the same machines. For the table, we made the assumption that the capacity of the cluster simply scales linearly with the number of available CPUs. This is not true in general, but close enough for this demonstration.

Table 1 Predicted throughput on cluster with Sparc IIe / 440 MHz Processors

n	msg/sec for p=1	msg/sec for p=0.5	msg/sec for p=0.1
2	571	727	930
4	888	1230	1777
6	1090	1600	2553
8	1230	1882	3265
12	1411	2285	4528
16	1523	2560	5614

Runtime Object Values

During operation, the state of the Replication Service can be monitored using the SPS Client – a command line interface to the SPS to get state information from the SPS (for details on how to install and use the SPS Client, please refer to [“SPS Client”](#)).

Example: State Dump of Replication Service

The following example shows the state of a Replication Service after a test with a single client has been run. The I-DBC Proxy is configured with two Proxy Processes per host, which can be deduced from the listing of the peers containing two entries. The peer list includes only those Proxy Processes which are selectable as acknowledgers, so only peers on another host are listed. The name of the peer host is *dolphin*, while the host on which the state was dumped is named *mamba*.

```
{ "ObjectId" = "pid5966.ReplicationService"
  "Values" = {
    "active" = "true"
    "addMessagesSent" = "924"
    "averageRtt" = "0.294063"
    "currentProcesses" = "0"
```

```

"droppedLooped" = "887"
"identifier" = "1693381801"
"inhibited" = "false"
"maxRtt" = "1.166224"
"maxVirtualProcessId" = "2"
"messagesACK" = "1019"
"messagesReceived" = "3696"
"messagesRequest" = "2677"
"messagesResent" = "4"
"messagesSent" = "932"
"messagesTerminate" = "4"
"messagesVeto" = "2"
"minRtt" = "0.005208"
"outstandingReplicates" = "0"
"peers" = [
  { "address" = "hostname=dolphin.microfocus.com,
    address=192.168.1.33, port=50305"
    "state" = "up"
    "virtualProcessId" = "0" },
  { "address" = "hostname=dolphin.microfocus.com,
    address=192.168.1.33, port=50306"
    "state" = "up"
    "virtualProcessId" = "1" }
]
"resolveMessagesSent" = "2"
"sequenceNumber" = "928"
"singlePeer" = "false"
"synchronous" = "false"
"terminateMessagesSent" = "6"
"virtualProcessId" = "0" }

```

The fields defined here are as follows:

- The state of **active** is true or false. True means that replication is active. In a non-clustered environment, active is false.
- **addMessagesSent** counts the number of ADD messages.
- **currentProcesses** is the number of currently active replication processes.
- **minRtt**, **averageRtt** and **maxRtt** give the respective round-trip times in seconds.
- **droppedLooped** counts the number of packets which have been received by the sender itself and thus been dropped before being processed.
- **identifier** is the fixed random number which identifies this Proxy Process on the host. This is also used for object key disambiguation.
- **inhibited** is false, because ADD messages are sent while proxifying. For testing and performance evaluation, the Replication Service can be switched to **inhibited** mode where ADDs are sent only upon RESOLVE request.
- **maxVirtualProcessId** is the same as the number of Proxy Processes on one host.
- **messagesACK** is the number of acknowledges received.
- **messagesReceived** counts the total of messages received.
- **messagesRequest** is the count of request messages.
- **messagesResent** is the total of messages which timed out and needed to be resent.

- **messagesSent** counts the total number of messages sent.
- **messagesTerminate** is the number of requests for Access Session termination received.
- **messagesVeto** is the number of VETOs received.
- **outstandingReplicates** is the number of ACKs, which are expected to arrive but have not yet.
- **peers** is a vector of peers on other hosts.
- The **state** of each entry is either "up" or "down". "down" means, the peer has not responded to an ADD request within the timeout.
- The **virtualProcessId** just numbers the peers on one host.
- **resolveMessagesSent** is the number of RESOLVEs sent.
- **sequenceNumber** is the current sequence number.
- If there is only a single peer, then **singlePeer** is true.
- **synchronous** denotes the mode of operation.
- **terminateMessagesSent** is the number of requests for Access Session termination sent.
- **virtualProcessId** is the number this Proxy Process has on this host.

Installation Notes

When installing a cluster of I-DBC Proxies, there are several non-obvious things to consider. The following text covers some of these points.

Direct Routing

Direct routing is the recommended mode of operation for a traffic redirector for a small cluster, because it provides the best performance. When using direct routing, you need to do the following:

- Configure the virtual IP address of the traffic redirector on all I-DBC Proxy hosts. Make sure these virtual IP addresses are never advertised via ARP.
- Enter the virtual IP address as the local external and internal interface address in Administration Console. This will cause the Proxy Processes to actually bind to the virtual address, which is intended. Do not enter the virtual IP address in the field "virtual address".
- Make sure the virtual IP is routed from clients and servers to the virtual director.
- Check availability of the virtual address using `telnet vip <I-DBC port>`.

If you want to operate the I-DBC in dual homed mode, you must use either a second traffic redirector, or a second virtual address, on your traffic redirector.

NAT

When your traffic redirector is configured to use NAT for mapping the virtual IP address to the cluster hosts, do the following:

- Enter the real interface address of each I-DBC Proxy into the fields for the local address of external and/or internal interface of the Administration Console. Usually, you will not be filling out the NAT fields.
- Enter the virtual IP address in the field "virtual address" and check the box.
- Make sure the virtual IP is routed from clients and servers to the virtual director.
- Check availability of the virtual address using `telnet vip <I-DBC port>`.

Performance Monitoring

To support performance management tasks the DBC provides on-demand access to usage data, which may be taken into account for a single DBC Cluster or DBC host. The following performance indicators are used when operating the I-DBC:

- GIOP Message Bytes Received
- GIOP Message Bytes Sent
- Number of GIOP Messages Received
- Number of GIOP Messages Sent

The usage data collector has been designed to provide data at regular time intervals. Each time when usage data is retrieved the usage counters are reset to account usage within the next time interval. For the ease of integration with third party performance management solutions, the collected usage data is stored in a flat file with comma separated values (CSV). The usage data collector is provided by means of a shell script which can be adapted easily to your requirements.

Setting up the Usage Data Collector

The usage data collector script `collectperfddata.sh` is part of your SPS installation and it is located in directory `<INSTALLDIR>/sps/bin/`. By default, the collected usage data is stored on the Security Policy Server to the CSV file `<INSTALLDIR>/sps/adm/PerfData.csv`. Before using the script you need to adapt the script internal settings to your Security Policy Server (SPS) configuration. Use a text editor to open the shell script and modify the following settings:

- Check the settings for `SPS_PROTOCOL`, `SPS_HOST`, and `SPS_PORT`. These variables are set up during the installation of the SPS. `SPS_HOST` and `SPS_PORT` should contain the SPS host, IP address, and port provided for management access (that is, the endpoint to which you connect with the Administration Console). The `SPS_PROTOCOL` variable must be set to "sslsoap" if SSL protection is enabled for the SPS. Otherwise set the variable to "soap".
- The `CLUSTER_NAME` variable holds the name of the DBC Cluster. The default name used by the SPS configuration is "idbcCluster1" or "wsdbcCluster1" respectively. If you have assigned a different name you need to set the variable to the assigned name.
- The `ADMIN_USER` and `ADMIN_PWD` variables hold the user ID and password required for management access. If you changed the account settings you need to change the variables to the new settings.

Activating the Usage Data Collector

To activate the usage data collector you must execute the shell script at regular time intervals (that is, every hour). This can be achieved easily using the `cron` tool of the operating system. To define a cron job you can use the "`crontab -e`" command. This command will open a text editor to edit the current table of cron jobs. Each entry has the following format: "`<minute> <hour> <day of month> <month> <day of week> <command>`", where each time and date field may specify a single value or

a range of values. The asterisk "*" may be used for time and date fields to specify all possible values. To schedule the script for hourly execution you need to enter:

```
0 * * * * /opt/microfocus/sps/bin/collectperfddata.sh
```

SPS Client

The SPS Client is a command line interface to the Security Policy Server (SPS). The SPS Client can be used to configure the SPS or to obtain state information about the SPS.

For information on installing the SPS Client, see the chapter “Installing the SPS Client” in the documentation for the relevant **Micro Focus | CORBA Add-on** Guide.

SPS Client Commands

To start the SPS client type:

```
./spscli.sh
```

After start-up, the SPS Client will ask for a user name and password. You can use, for example, the default user `admin` with the password `admin` to log in.

The SPS Client provides several commands which are listed in the following table.

Command	Description
<code>collectUsageData</code> <code><clusterName></code> <code>[<dbcName>]</code>	Gathers usage data from the given cluster, accumulates the retrieved values, and prints a CSV record with the format: <code>timestamp, bytesReceived, bytesSent, messagesReceived, messagesSent</code> . Optionally, a DBC name may be specified to gather the data from a single DBC host only.
<code>dump <clusterName></code> <code><dbcName></code>	Dump all attributes from the named ProxyManager.
<code>get <clusterName></code> <code><dbcName> <objectId></code> <code><attribute name></code>	Retrieves the attribute value of the attribute named <code><attribute name></code> from the object with id <code><objectId></code> . For example the command: <code>get cluster1 dbc1 NodeManagerAdmin proxyProcesses</code> produces a list of all proxy process identifiers.
<code>getAll <clusterName></code> <code><dbcName> <objectId></code> <code><attribute name></code>	Retrieves the attribute values of the attribute named <code><attribute name></code> from all the objects with id <code><objectId></code> . For example the command: <code>getAll cluster1 dbc1 logging levels</code> returns a list of all logging options.
<code>getDescription</code>	Retrieves the description of the Security Policy Server.
<code>help</code>	Prints a list of all commands.
<code>ior ...</code>	The <code>ior</code> command has a number of sub-commands. See Table 3 for a list of these commands.
<code>login <username></code> <code><password></code>	Login to the Security Policy Server with username and password. If the password is omitted, it will be requested by a password prompt.
<code>mon</code>	Enter monitor mode.
<code>readConfig <filename></code>	Get the current configuration.

Table 2 SPS Client commands

Command	Description
restart <clusterName> [<dbcName>]	Restart the ProxyManager after configuration changes. Example: "restart dbcCluster1 dbcProxy1" restarts the specified ProxyManager Example: "restart dbcCluster1" restarts the whole cluster.
restartSPS <SPSName>	Restart SPS with the given name. Give an empty name to restart all SPSs.
sessionInfo <clusterName>	Prints the number of connections, IORs, and pending requests for all access sessions from the given cluster and calculates the total numbers. For example: sessionInfo iDBCProxyCluster1
set <clusterName> <dbcName> <objectId> <attribute name> <attribute value>	Sets the attribute with name <attribute name> of the object with id <objectId> to value <attribute value>. The attribute value must be a stringified dictionary, that is, string/int/bool literals must be quoted. Example: set dbcCluster1 dbcProxy1 pid1234.ADF enabled "false"
setAll <clusterName> <dbcName> <objectId> <attribute name> <attribute value>	Sets the attribute with name <attribute name> of the object with id <objectId> to value <attribute value> for all Proxies. Example: setAll dbcCluster1 dbcProxy1 ADF enabled "false"
smon	Enter StateMonitor mode.
writeConfig <filename>	Write the configuration from file <filename> to the Security Policy Server.
writeConfig <version> <filename>	Write the configuration diff from file <filename> to the Security Policy Server.
quit	Exit the program.

Table 2 SPS Client commands

Command	Description
ior deactivate <clustername> <stringified ior> [<match>]	Deactivate the given original IOR on all I-DBC's of the given cluster. This change is effective immediately. By default, the original IOR is matched. The other IOR is returned. [match] is a bitmask: 0 all, 1 host, 2 TCP port, 4 SSL port, 8 object key, 16 match will be done on the proxified IOR.
ior deploy <cluster name> <stringified original IOR> <proxification info dict>	Proxify the given original IOR and activate it on the given cluster. This command is effective immediately. <proxification info dict> is a dictionary containing additional proxification info. The dictionary must not contain any spaces.
ior deployTransient <cluster name> <stringified original IOR> <proxification info dict>	Proxify the given original IOR and activate it on the given cluster. This command is effective immediately, but the deployed IOR will be lost on the next config update or proxy restart. <proxification info dict> is a dictionary containing additional proxification info. The dictionary must not contain any spaces.

Table 3 Sub-commands for the ior command

Administrative Rights for SPS Client Operations

You can allow or deny administrative rights for the following SPS Client operations:

- Update the IOR table, that is the use of the commands `ior deactivate`, `ior deploy`, `ior deployTransient`
- Clear the ADF Cache , that is the use of the command `clearCache`
- Get/set attributes, that is the use of the commands `get`, `getAll`, `set`, `setAll`

Administrative rights can be configured on the "Roles - Administration" panel (see the "Role Properties – Administration" section in the ***Administrator's Guide***).

Hardened System

Requirements and Recommendations for Linux

The DBC software must be installed on machines that will become part of your firewall. All firewall machines are potential targets of attacks, so they require great care in the configuration of their operating system and network components. In this chapter, we describe the requirements for the machines you will use as DBC components, and provide recommendations for hardening your system for secure operation. If you are unsure about the topics described here, please consult a computer security or firewall expert for advice.

While it is a good idea to secure every machine in your network, it is mandatory for the firewall machines. Since these are located at the entry points of your network, they are the first line of defence that a potential attacker has to deal with.

Hardened system

On firewall machines you should make sure that there are as few handles for attacks as possible, both in terms of available network services and operating system features. This is called a *hardened system*.

When using DBC software, you should especially take care with the machine you will use as the DBC Proxy host, as it is part of your firewall. However, if your security policy requires protection against *inside attacks* too, the same applies to the Security Policy Server machines and possibly the DBC Proxy administration machines.

Operating System

Minimal operating system

In general, start with a minimum installation of the operating system. If you are more experienced with the installation procedures, use a custom installation. You should only install such components that are absolutely required for the operation of the security related software.

The DBC Proxy host is run solely as a server, so there is no need for any graphical user interface (GUI) components. This also applies to the Security Policy Server if you do not use the Administration Console locally.

File permissions and logon

Make sure you have very strict permissions on critical system files, especially in the directory `/etc`. Remove `SUID/SGID` flags from executables if possible. Restrict logon to the console or a dedicated management interface.

No NFS or YP/NIS

Configure the machine as a stand alone system. Do not use NFS or YP/NIS, since these components are very vulnerable to attacks. Instead, use only local file systems and the shadow password mechanism. If possible, deactivate any accounts on the machine, except those needed for administration and the services. There should be no need for your regular network users to access the firewall machines anyway.

Network Services

Pay special attention to network services. Most of them are unnecessary on a firewall machine, so you can disable them. Prevent standalone service daemons from starting, and also disable the `inetd` super daemon if you do not need the services it provides (check its configuration file `/etc/inetd.conf`). See your operating system manual for instructions.

Remote administration

There are a few exceptions to this general rule. If you need a means for remote administration, use a secure service such as SSH. This is usually provided by the standalone daemon `sshd`. Do not use traditional services such as `telnet` or `rsh/rexec`, since they are vulnerable to sniffing and hijacking attacks.

Use email only in forward mode

You may also want to use email notifications for problems or security alerts. Often, the sending of email is possible without a specific mail daemon, but in some cases you may have to use one. If so, you should disable all of its features, and use it only for forwarding mail to your mail gateway or administration system.

Restricted access to network services

If you use any network services, try to restrict their binding to network interfaces. If you have a dedicated management interface, configure the services to bind to that. Generally, the services should not be accessible from interfaces that are connected to a public or untrusted network. You may also use meta-daemons such as `tcpd` to restrict access, or use local packet filtering for that purpose.

After you have finished configuring the network services, use `netstat -a` and watch for any lines specifying `LISTEN` as state. Verify that there are only those services you need, and that they use the correct bind address (network interface).

Kernel and Network Stack

Make sure you have the latest patches installed for your kernel. For some operating systems, there may be special add-on patches available which can be used to enhance security.

Minimum kernel

On some systems, you can compile your own kernel. Apply the same rule as for the system components: Only include those features that you absolutely need, and omit anything else. Build a monolithic kernel if possible.

Check your operating system manual to see if your network stack provides basic protection against common low-level attacks, such as SYN flooding or fragmentation bombs. Enable these features or compile them in.

IP forwarding

If you use a multi-homed machine for the DBC Proxy host, make sure that IP forwarding/routing is disabled. This prevents an attacker bypassing the IIOP Proxy.

Strong end system model

Also, consult the documentation for your network stack to see whether it enforces a *strong end system model*. This means that each packet arriving at a network interface must have that interface's network address as its target address. If this is not the case, your system may be vulnerable to spoofing attacks. You can enforce a strong end system model by using a local packet filtering component (e.g., `ipchains` on Linux). Create a rule for each interface to only accept packets targeted at the interface address or its broadcast address, and set the default policy to discard or drop anything else.

I-DBC Authentication

The I-DBC Access Control is primarily designed to work with mutual authentication via SSL. However, in some scenarios the client cannot use certificate based authentication. For such cases the I-DBC provides an alternative authentication framework via a dedicated CORBA interface

This chapter is intended for developers who wish to use the I-DBC Authenticator in their applications. We assume you are familiar with the general I-DBC architecture, and the requirements of the specific native authentication method you wish to use.

I-DBC Authenticator Architecture

The I-DBC Authenticator interface is a framework for generic authentication. Specific authentication methods are provided as a mapping to this interface. We describe both the generic use and several specific mappings in the following sections.

The I-DBC Authenticator service is located on the I-DBC host (see [figure 10](#)). Clients can contact the I-DBC Authenticator interface at the same host and port address as any Initial Contact Point (ICP). It is an *inband service*, so the client must use the same connection for subsequent invocations on IIOP Proxies.

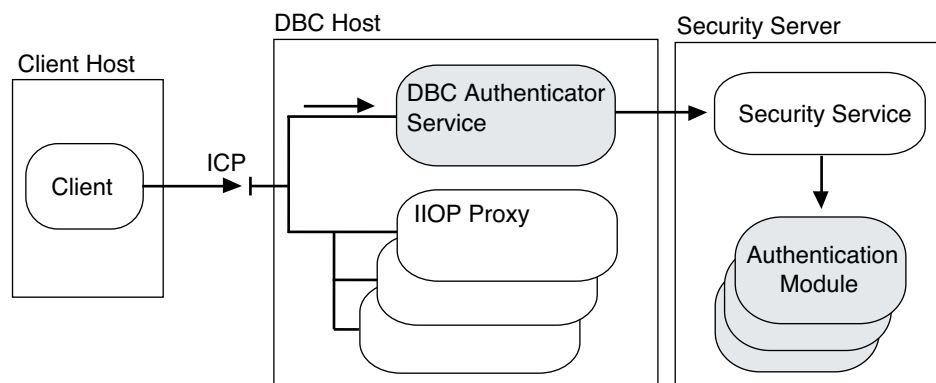


Figure 10 The I-DBC Authenticator Architecture

Inband authentication

The I-DBC Authenticator interface cannot authenticate clients by itself, since this process usually requires sensitive information. Accordingly, the authenticator forwards the request to the Security Policy Server, which in turn passes the request to the appropriate authentication module.

There may be multiple authentication modules, each of which can perform a specific authentication method. The module either performs the authentication by itself, or uses a local system service. Finally, the module sends a response to the I-DBC Authenticator service which in turn sends it back to the client.

Caveats

You may use the I-DBC Authenticator interface instead of SSL authentication, but there are some constraints.

If you do not use SSL, all data is sent unprotected over any intermediate networks between the client and the I-DBC host. Attackers may intercept this data on the way to get hold of passwords or other sensitive information.

You are strongly recommended to use at least client-side SSL

That way, your password is cryptographically protected as it travels through the network.

Authentication using the I-DBC-Authenticator Interface is also not as fine-grained as SSL. It only accepts a single user at a time from each client host, and denies access to other users from the same machine. (In some scenarios you may work around this restriction by using IP address based Access Sessions). The I-DBC Proxy has to enforce this restriction to securely separate Access Sessions. This may lead to problems if your client network is behind a masquerading firewall.

Since the I-DBC Authenticator is an inband service, a successful authentication is only valid for the TCP connection it was sent over. If your client ORB creates additional connections to the I-DBC Proxy, it must re-authenticate on the new connection. This is not a problem if your client ORB reuses connections to its peers, as is done in most ORBs. However, if your client ORB creates connections for each method invocation, you cannot use the I-DBC Authenticator. The I-DBC Proxy has to enforce this restriction, otherwise attackers could forge the source address of a connection to gain access to your authenticated session.

Generic Interface

The I-DBC Authenticator CORBA interface is sufficiently generic to allow mapping to all kinds of authentication services, including those that utilize query callbacks (for which I-DBC uses `continue_authentication`). For a detailed description of the methods and their parameters see the section [“Generic Use”](#).

There are similarities to the application view interfaces of the CORBA security specification. However I-DBC uses a different interface, to prevent clashes if your clients and servers use a compliant ORB security service.

```
// Generic I-DBC Authenticator Interface

#pragma prefix "xtradyne.com"

module Xtradyne {
    // return value of authentication functions
    // this is the same as Security::AuthenticationStatus
    enum AuthenticationStatus {
        SecAuthSuccess,
        SecAuthFailure,
        SecAuthContinue,
        SecAuthExpired
    };

    // this is the same as Security::Opaque
```

```

typedef sequence<octet> Opaque;

// this is the same as Security::AuthenticationMethod
typedef unsigned long AuthenticationMethod;

// we currently support these authentication methods:
// SSL, already verified by transport layer
const AuthenticationMethod authSSL = 0;

// RSA ACE (SecurID)
const AuthenticationMethod authACE = 2;

// Product-specific User ID/Password scheme
const AuthenticationMethod authUsernamePassword = 222;

// the methods of this interface are implemented
// or intercepted by the I-DBC
interface DBCAuthenticator {
    // authenticate
    AuthenticationStatus authenticate(
        in AuthenticationMethod method,
        in string                security_name,
        in Opaque                auth_data,
        out Opaque               session_data,
        out Opaque               continuation_data,
        inout Opaque             auth_specific_data
    );

    // respond to challenge
    AuthenticationStatus continue_authentication(
        inout Opaque             session_data,
        in Opaque                response_data,
        out Opaque               continuation_data,
        inout Opaque             auth_specific_data
    );

    // change authentication data
    AuthenticationStatus change_auth_data(
        in AuthenticationMethod method,
        in string                security_name,
        in Opaque                new_auth_data,
        out Opaque               session_data,
        out Opaque               continuation_data,
        inout Opaque             auth_specific_data
    );
};
};

```

Generic Use

In this section we describe the generic use of the I-DBC Authenticator interface. These are only general guidelines. See the description of a specific authentication method for additional information in the section [“Authentication Methods”](#).

Authenticate

The client starts an authentication session by calling `authenticate`, as shown in [figure 11](#). The desired authentication method prescribes the use and format of the other parameters. In all cases the client must provide a security name (e.g. user name) and authentication data (e.g. password).



Figure 11 Basic Authentication

The client may provide additional authentication specific data if this is needed for the authentication method. In that case the client may construct an appropriate sequence, for example, to indicate requested privileges. If no additional data is needed the client should pass an empty sequence.

Return values

The result of the call indicates the further course of action:

- If the I-DBC Proxy returns `SecAuthSuccess`, all is well. The user may still call `change_auth_data` if this is supported by the current authentication method (see [“change_auth_data”](#) for details).
- If the I-DBC Proxy returns `SecAuthFailure`, the user may retry authentication by calling `authenticate` again, usually with different parameters.
- If the I-DBC Proxy returns `SecAuthExpired`, the client should stop attempting to authenticate. Often the user must reestablish the account by some external means.
- If the I-DBC Proxy returns `SecAuthContinue`, the user must provide a response to the given challenge via `continue_authentication`.

In any case the I-DBC Proxy provides `session_data` (SD) that the client must use for further invocations. The client must consider it to be an opaque value, and thus pass it verbatim.

The I-DBC Proxy may also issue a challenge in `continuation_data`, specially when the return value is `SecAuthContinue`. See [figure 12](#).

Furthermore, the I-DBC Proxy may provide additional authentication specific data if needed for the current authentication method. In this case, the client is responsible for interpreting the contents of the sequence. The sequence may, for example, indicate which kind of response is expected to the challenge, or that the user is expected to change his authentication data.

continue_authentication

If the I-DBC Proxy issues a challenge, the client must respond by calling `continue_authentication` and must provide the returned session data verbatim (see [figure 12](#)). The client must give the response in `response_data`. Furthermore the client may provide authentication specific data if needed for the authentication method.

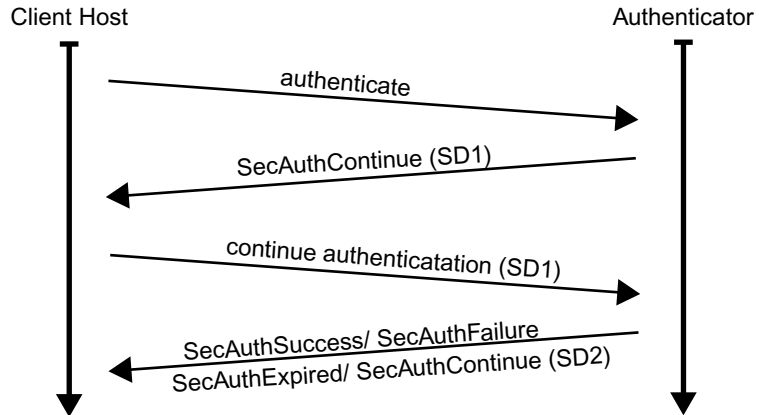


Figure 12 Authentication Challenge and Continuation

The I-DBC Proxy responds with the same return values as for `authenticate`, and also uses the out parameters in the same way. Note that the returned session data (SD2 in the figure above) may be different from the data the client provided (in the above example the client provided the session data SD1). The new data must be used for further invocations.

change_auth_data

After successful authentication, the client may change its authentication data via `change_auth_data` (see [figure 13](#)). It must provide the last returned session data along with the new authentication data. The I-DBC Proxy responds with the same return values and output parameters as with `authenticate`. But if it issues a challenge, `change_auth_data` must be called again instead of `continue_authentication`.

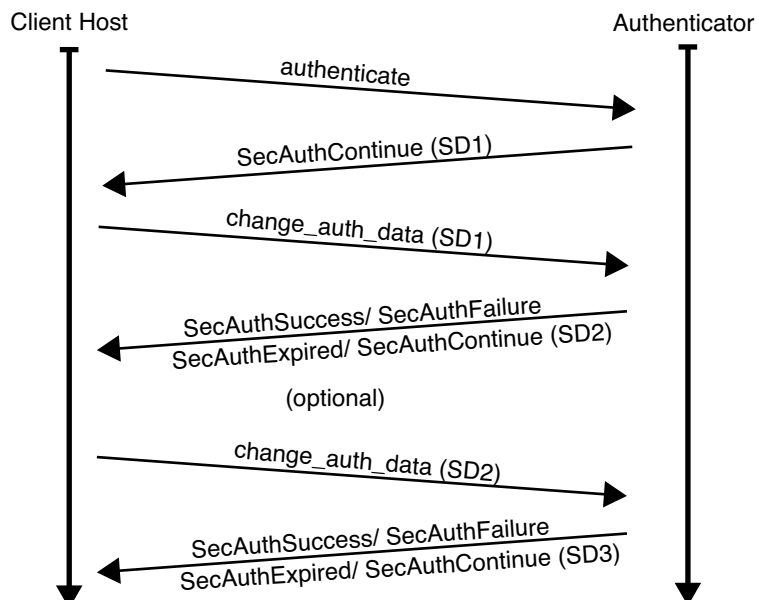


Figure 13 Change of Authentication Data

Exceptions

In some cases the I-DBC Authenticator may raise an exception. It uses the existing CORBA system exceptions, specifically:

- `BAD_PARAM` when the authentication method is unknown or not supported
- `MARSHAL` when the parameters provided by the client are not formed as expected
- `BAD_INV_ORDER` is used if `continue_authentication` or `change_auth_data` is called before `authenticate`

Authentication Methods

For each supported authentication method we provide a mapping to our generic I-DBC Authenticator scheme. Such a mapping describes how authentication data is encoded into the generic parameter sequences, and how the client must react to return values and output parameters. In some cases we provide convenience functions or classes which you can include into your client, these are described at the end of each mapping description.

RSA/ACE SecurID Mapping

This section describes how the RSA/ACE Agent API is mapped to the I-DBC Authenticator interface.

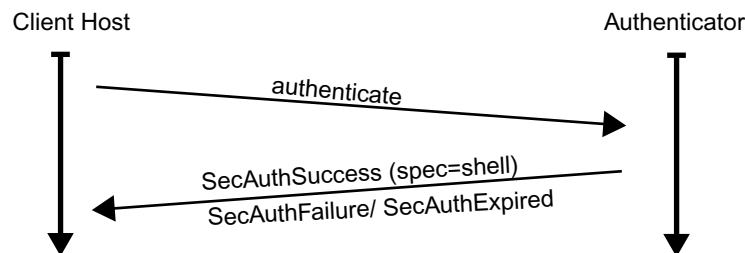


Figure 14 SecurID Authentication

authenticate

The client calls `authenticate` with the method constant `authACE` and specifies his user name as `security_name`, and his passcode (pin and token-code combination) in `auth_data` (see [figure 14](#)). The `auth_specific_data` parameter is not used in this mapping, the client must provide an empty sequence.

The return value of the call indicates the further course of action:

- `SecAuthSuccess`: Authentication succeeded. The `auth_specific_data` out parameter contains a string specifying the users default shell. This ends the authentication session. The client may now issue regular application calls.
- `SecAuthFailure`: Authentication failed. This ends the authentication session. The client may start a new session with different parameters. In most cases, the user made a mistake with the passcode.
- `SecAuthExpired`: The user's account does not exist or has expired. This ends the authentication session. The client may start a new session with different parameters. Usually, a different user name is given.

- `SecAuthContinue`: Further data is needed. The two continuation cases are indicated by the contents of the `auth_specific_data` output parameter. The sequence contains a single byte. If this byte has the value 0, the client must continue authentication with the next token code. If this byte is 1, the user must select a new pin, this is described below. In any case the I-DBC Proxy returns a session identifier, which must be passed verbatim for further invocations. The `continuation_data` output parameter contains values specific to each case.

continue_authentication: next token code required

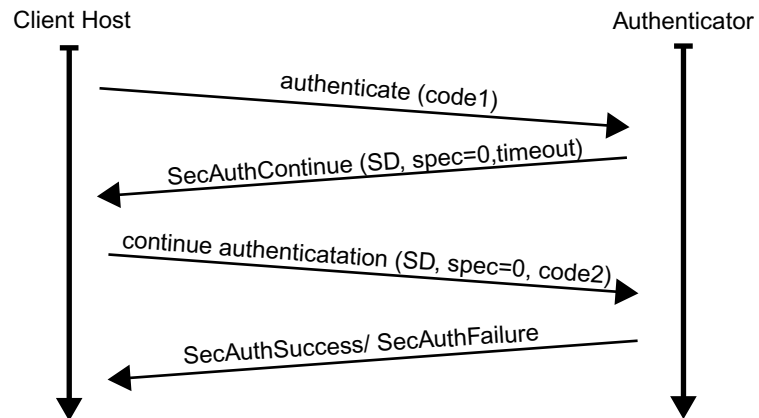


Figure 15 Next Token Code Challenge and Response

If the `authenticate` call results in the next token code situation, the client must respond with the next token within a certain time (see [figure 15](#)). The time-out value (in seconds) is given in the `continuation_data` out parameter in network byte order. To continue the authentication session, the client may call `continue_authentication`, giving the `session_data` as received before, and the next passcode as `response_data`. The `auth_specific_data` parameter must contain a single byte set to 0. The return value from this call is either `SecAuthSuccess` or `SecAuthFailure`, with the semantics as described above.

If the `authenticate` call results in the new pin situation, the client must respond in conformance with the domains policy (see [figure 16](#)). This is specified by the values encoded in the `continuation_data` out parameter. Its layout is as follows:

- octet 1 is the minimum pin length
- octet 2 is the maximum pin length
- octet 3 is the user-selectable flag
- octet 4 is the alphanumeric flag
- the next 16 octets contain a system generated pin.

See the ACE/Agent API documentation for a detailed description on how to use these values.

continue_authentication: new PIN selection

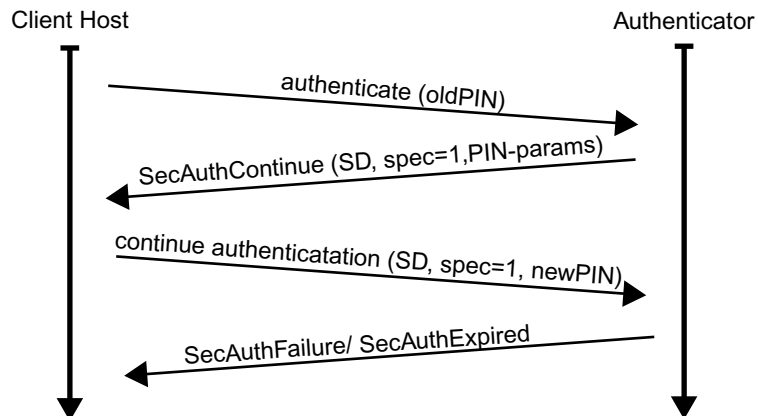


Figure 16 New PIN Challenge

The client must call `continue_authentication`, giving the `session_data` as received before, and the new pin as `response_data`. The user may decide to cancel the pin creation/selection, but then the client must still invoke `continue_authentication`, with an empty sequence as `response_data`. The `auth_specific_data` parameter must contain a single byte set to 1.

The I-DBC Proxy returns `SecAuthFailure` if the new pin is not accepted by the ACE system. This ends the authentication session. The client may start a new one if desired, but will have to supply a new pin.

The I-DBC Proxy indicates success by returning `SecAuthExpired`. This may seem a bit strange, but it means that the old pin has "expired". The ACE/Agent API requires that the user re-authenticates after a pin change, so the I-DBC Proxy does not return success here. This allows client authentication code to loop until `SecAuthSuccess` is returned.

This mapping does not use the `change_auth_data` method because changing the pin is handled by `continue_authentication`.

Exceptions

The I-DBC Proxy may raise an exception in certain cases:

- `INITIALIZE` means that the I-DBC Proxy could not start the ACE module for some reason.
- `BAD_PARAM` means that a parameter did not match the size requirements of the ACE/Agent API.
- `BAD_INV_ORDER` means that a next token code or pin change challenge were not answered with `continue_authentication`, or that the wrong type of response was sent.

See the ACE documentation for details.

I-DBC Authenticator Events

I-DBC Authenticator events can be used to check the behavior of the I-DBC Authenticator Interface. The following events are available and can be activated on the "Audit Policy" panel of the Administration Console:

- `DBCAuthenticatorAuthenticationFailure`
- `DBCAuthenticatorAuthenticationSuccess`
- `DBCAuthenticatorAuthenticationInfo`

Note that the Info event yields information about every step of the authentication process, whereas the Success and Failure events only state if the authentication as a whole was successful or not.

Index

A

Authentication Mechanisms
RSA SecurID 42

H

Hardened System
Kernel and Network Stack 34
Network Services 34
Operating System 33
Requirements and Recommendations 33
High Availability 3
as provided by the DBC 5
Calculate application throughput 12
Calculate DBC requirements 12
Deployment Considerations 12
Deployment Requirements 13
Types 3

I

I-DBC Authenticator 37
Architecture 37
Audit Events 45
Authenticator Caveats 38
Basic Authentication 40
Challenge and Continuation 41
Generic Interface 38

R

Replication 15
Configuration 18
Duplication of calls 18
Estimated Throughput 21
Installation Notes 25
IOR Timeout 17
Limitations and Restrictions 17
Loopback 15
Maintenance 16
Message Properties 21
Object Keys 18
Performance 21
Reliability and asynchronous operation 16
Runtime Object Values 22
Security Considerations 17
Technology 15
Replication Interface 20
RSA/ACE SecurID Mapping 42

S

Scalability 3, 6
Connection Bundling 7
Deployment Example 13
Direct Routing 7
Network Address Translation 6
Tunnel Scenario 8
Types 3
SecurID
RSA/ACE Mapping 42
SPS Client 29

Administrative Rights 31
Commands 29

T

Traffic Redirector 14

U

UserID/Password authentication
I-DBC Authenticator 37

