



Understanding Replication

This chapter describes Cisco Access Registrar's configuration replication features, functions, limitations and operation.

Replication Overview

Cisco Access Registrar replication feature can maintain identical configurations on multiple machines simultaneously. When replication is properly configured, changes an administrator makes on the primary or *master* machine are propagated by Cisco Access Registrar to a secondary or *slave* machine.

Replication eliminates the need to have administrators with multiple Cisco Access Registrar installations make the same configuration changes at each of their installations. Instead, only the master's configuration need be changed and the slave is automatically configured eliminating the need to make repetitive, error-prone configuration changes for each individual installation. In addition to enhancing server configuration management, using replication eliminates the need for a hot-standby machine.

Using a hot-standby machine is a common practice to provide more fault-tolerance where a fully-installed and configured system stands ready to takeover should the primary machine fail. However, a system setup for hot-standby is essentially an idle machine only used when the primary system fails. Hot-standby or secondary servers are expensive resources. Employing Cisco Access Registrar's replication feature, both servers may perform RADIUS request processing simultaneously, eliminating wasted resources.

The replication feature focuses on configuration maintenance only, not session information or installation-specific information such as Administrator, Interface, Replication or Advanced machine-specific configuration changes. These configuration items are not replicated because they are specific to each installation and are not likely to be identical between master and slave. While changes to Session Managers, Resource Manager, and Remote Servers are replicated to the slave and stored in the slave's configuration database, they are not hot-configured on the slave (see Hot Configuration Detailed below for more information)

Changes should be made only on the master server. Making changes on a slave server will not be replicated and may result in an unstable configuration on the slave. Any changes made using replication will not be reflected in existing **aregcmd** sessions. **aregcmd** only loads its configuration at start up; it is not dynamically updated. For example, if **aregcmd** is running on the slave, and on the master **aregcmd** is used to add a client, the new client, while correctly replicated and hot-configured, will not be visible in the slave's **aregcmd** until **aregcmd** is exited and restarted.

How Replication Works

This section describes the flow of a simple replication as it occurs under normal conditions.

Replication Data Flow

The following sections describe data flow on the master server and the slave server.

Master Server

The following describes the data flow for the master server:

-
- Step 1** The administrator makes a change to the master server's configuration using the **aregcmd** command line interface (CLI) and issues a **save** command.
 - Step 2** After the changes are successfully validated, the changes are stored in the Access Registrar database.
 - Step 3** **aregcmd** then notifies the Access Registrar server executing on the master of the configuration change.
 - Step 4** The Access Registrar server then updates its version of the configuration stored in memory. (This is called *hot-config* because it happens while the server is running and processing requests.)
 - Step 5** The Access Registrar server first copies the changes pertaining to the **aregcmd save**, also known as a transaction to its replication archive, then transmits the transaction to the slave server for processing.
 - Step 6** In **aregcmd**, the prompt returns indicating that the **save** has completed successfully, the transaction has been archived, and the transaction has been transmitted to the slaves.
-

Slave Server

-
- Step 1** When the slave server receives the transaction, its contents are verified.
 - Step 2** Once verified, the changes are applied to the slave server's database
 - Step 3** The changes are then applied (hot-configured) in the slave server's in-memory configuration.
 - Step 4** The transaction is written to the slave server's replication archive.
-

Security

Replication has two primary security concerns:

- Security of the transactions transmitted to the slave server
- Storage of transactions in the replication archive

Both of these concerns use shared secret (MD5) encryption via the shared secret specified in the replication configuration on both master and slave servers. Replication data transmitted between master and slave is encrypted at the source and decrypted at the destination the same way as standard RADIUS

packets between Access Registrar's clients and the Access Registrar server. Transactions written to the replication archive are also encrypted in the same manner and decrypted when read from the replication archive.

Replication Archive

The replication archive serves two primary purposes:

1. To provide persistent, or saved, information regarding the last successful transaction
2. To persist transactions in case the slave server requires re synchronization (see Ensuring Data Integrity below for more information on re synchronization).

The replication archive is simply a directory located in `../CSCOar/data/archive`. Each transaction replicated by the master is written to this directory as a single file. The name of each transaction file is of the form `txn#####` where `#####` is the unique transaction number assigned by the master server. The replication archive size, that is the number of transaction files it may contain, is configured in the Replication configuration setting of `TransactionArchiveLimit`. When the `TransactionArchive` limit is exceeded, the oldest transaction file is deleted.

Ensuring Data Integrity

Access Registrar's configuration replication feature ensures data integrity through transaction data verification, transaction ordering, automatic resynchronization and manual full-resynchronization. With the single exception of a manual full-resynchronization, each of the following techniques help to automatically ensure that master and slave servers contain identical configurations. A detailed description of each technique follows.

Transaction Data Verification

When the master prepares a transaction for replication to a slave, the master calculates a 2's complement Cyclic Redundancy Check (CRC) for each element (individual configuration change) in the transaction and for the entire transaction and includes these CRC values in the transmitted transaction. When the slave receives the transaction, the slave calculates a CRC for each transaction element and for the entire transaction and compares its own calculated values with those sent with the message. If a discrepancy occurs from these comparisons, the transaction element or the entire transaction is discarded and a re-transmission of that particular transaction element or the entire transaction is requested by the slave from the master. This process is called automatic resynchronization. (described in more detail below)

Transaction Order

When the master prepares a transaction for replication, it assigns the transaction a unique transaction number. This number is used to ensure the transactions are processed by the slave in exactly the same order as they were processed on the master. Transactions are order dependent. Since the functionality of Access Registrar's configuration replication feature is to maintain identical configurations between master and slave, if transaction order were not retained, master and slave would not contain identical configurations. Consider where two transactions modify the same thing (a defined client's IP address for example). If the first transaction was a mistake and the second was the desired result, the client config on the master would contain the second setting; however, if the transactions were processed in the reverse order on the slave, the client config on the slave would contain the mistaken IP Address. This example illustrates the critical need for transaction ordering to ensure data integrity.

Automatic Resynchronization

Automatic Resynchronization is the most significant feature with respect to data integrity. This feature ensures the configurations on both the master and slave are identical. If they are not, this feature automatically corrects the problem.

When the master and slave start-up, they determine the transaction number of the last replication transaction from their respective replication archives. The master immediately begins periodic transmission of a TransactionSync message to the slave. This message informs the slave of the transaction number of the transaction that the master last replicated.

If the transaction number in the TransactionSync message does not match the transaction number of the last received transaction in the slave's archive, then the slave will request resynchronization from the master. The resynchronization request sent by the slave will include the slave's last received transaction number.

The master will respond by retransmitting each transaction since the last transaction number indicated by the slave in the resynchronization request. The master obtains these transactions from its replication archive.

Should the slave's last received transaction number be less than the lowest transaction number in the master's replication archive, then automatic resynchronization cannot occur as the master's replication archive does not contain enough history to synchronize the slave. In this case, the slave must be resynchronized with a full-resynchronization.

Full Resynchronization

Full Resynchronization means that the slave has missed more transactions than are stored in the master's replication archive and cannot be resynchronized automatically. There is no automatic full-resynchronization mechanism in Access Registrar's configuration replication feature. To perform a full resynchronization, refer to the *Cisco Access Registrar User's Guide*.

Understanding Hot-Configuration

Hot-Configuration is the process of reflecting configuration changes made to Access Registrar's internal configuration database in the in-memory configuration of the executing Access Registrar server. Hot-Configuration is accomplished without interruption of RADIUS request processing. For example, if an administrator uses **aregcmd** to configure a new client and issues a **save** command, when the prompt returns, the newly configured client may send requests to Access Registrar.

Hot-Configuration minimizes the down-time associated with having to restart an Access Registrar server to put configuration changes into effect. With the Hot-Configuration feature, a restart is only necessary when a Session Manager, Resource Manager or Remote Server configuration is modified. These configuration elements may not be hot-configured because they maintain state (an active session, for example) and cannot be modified without losing the state information they maintain. Changes to these configuration elements require a restart of Access Registrar to put them into effect.

Hot-Configuration is not associated with the replication feature. Hot-Configuration's only connection to the replication feature is that when a change is replicated to the slave, the slave is hot-configured to reflect the replicated change as if an administrator had used **aregcmd** to make the changes directly on the slave server.

Replication's Impact on Request Processing

The replication feature was designed to perform replication of transactions with minimal impact on RADIUS request processing. When a transaction is received by a slave, RADIUS requests are queued while the transaction is applied to the slave. Once the transaction is complete, RADIUS request processing resumes.

The impact on RADIUS request processing is a direct result of the size of a transaction. The smaller the transaction the lesser the impact, and the larger the transaction, the greater the impact. In other words, when making changes to the master, frequent saves are better than making lots of changes and then saving. Each change is one transaction element and all changes involved in a **save** comprise a single transaction with one element per change. Since the replication feature only impacts RADIUS request processing when changes are made, the impact under normal operation (when changes are not being made) is virtually unmeasurable.

Replication Configuration Settings

This section describes each replication configuration setting. In **aregcmd**, replication settings are found in `//localhost/Radius/Replication`.

RepType

RepType indicates the type of replication. The choices available are SMDBR and NONE.

When RepType is set to NONE, replication is disabled. To enable replication, set RepType to SMDBR for Single Master DataBase Replication. RepType must be set to SMDBR on both the master and slave servers.

RepTransactionSyncInterval

Master

On the master server, RepTransactionSyncInterval is the duration between periodic transmission of the TransactionSync message expressed in milliseconds. The default is 60000 or 1 minute.

The purpose of RepTransactionSyncInterval is to indicate how frequently to check for an out-of-sync condition between the master and slave servers. When the slave received the TransactionSync message, it uses its contents to determine if it needs to resynchronize with the master.

The larger the setting for RepTransactionSyncInterval, the longer the period of time between out-of-sync detection. However, if RepTransactionSyncInterval is set too small, the slave may frequently request resynchronization when it is not really out of sync. If the duration is too small, the slave cannot completely receive a transaction before it receives the TransactionSync message. In this case, the servers will remain synchronized, but there will be unnecessary excess traffic that could affect performance.

Slave

On the slave, `RepTransactionSyncInterval` is used to determine if the slave has lost contact with the master and to alert administrators of a possible loss of connectivity between the master and slave. If the elapsed time since the last received `TransactionSync` message exceeds the setting of `RepTransactionSyncInterval`, the slave writes a log message indicating that it may have lost contact with the master. This log message is repeated each `TransactionSyncInterval` until a `TransactionSync` message is received.

RepTransactionArchiveLimit

On both master and slave, the `RepTransactionArchiveLimit` setting determines how many transactions can be stored in the archive. The default setting is 100. When the limit is exceeded, the oldest transaction file is deleted. If a slave requires resynchronization and the last transaction it received is no longer in the archive, a full resynchronization will be necessary to bring the slave back in sync with the master.



Note

The value set for `RepTransactionArchiveLimit` should be the same on the master and the slave.

An appropriate value for `RepTransactionArchiveLimit` depends upon how much hard disk space an administrator can provide for resynchronization. If this value is large, say 10,000, then the last 10,000 transactions will be stored in the archive. This is like saying the last 10,000 saves from `aregcmd` will be stored in the archive. Large values are best. The size of each transaction depends upon how many configuration changes were included in the transaction, so hard disk space usage is difficult to estimate.

If the slave should go down or otherwise be taken off line, the value of `RepTransactionArchiveLimit` and the frequency of `aregcmd` saves will determine how long the slave may be off-line before a full-resynchronization will be required.

There are two reasons why a slave server should have an archive:

1. The slave must save the last received transaction for resynchronization purposes (at a minimum).
2. Should the master go down, the slave can then be configured as the master and provide resynchronization services to other slaves.

RepIPAddress

The `RepIPAddress` value is set to the IP Address of the machine containing the Access Registrar installation.

RepPort

The `RepPort` is the port used to receive of replication messages. In most cases, the default value (1645) is sufficient. If another port is to be used, the interfaces must exist in the machine.

RepSecret

RepSecret is the replication secret shared between the master and slave. The value of this setting must be identical on both the master and the slave.

RepIPMaster

The RepIPMaster setting indicates whether the machine is a master or a slave. On the master, set RepIPMaster to TRUE. On the slave set it to FALSE. Only the master may have this value set to TRUE and there can be only one master.

RepMasterIPAddress

RepMasterIPAddress specifies the IP Address of the master. On the master, set RepMasterIPAddress to the same value used in RepIPAddress above. On the slave, RepMasterIPAddress must be set to the IP Address of the master.

RepMasterPort

RepMasterPort is the port to use to send replication messages to the master. In most cases, the default value (1645) is sufficient; however, if another is to be used, the interfaces must exist in the machine.

Rep Members Subdirectory

The Rep **Members**\ subdirectory contains the list of slaves to which the master will replicate transactions.

Rep Members/Slave1

Each slave is added much like a client is added. Each slave must have a configuration in the Rep Members directory to be considered part of the *replication network* by the master. The master will not transmit any messages or replications to servers not in this list, and any communication received by a server not in this list will be ignored.

**Note**

Although it is possible to configure multiple slaves with the same master, we have only considered a single-master/single-slave configuration. This is the recommended configuration.

Name

This is the name of the slave. The name must be unique.

IPAddress

This is the IP Address of the slave.

Port

This is the port upon which the master will send replication messages to the slave.