Call Detail Records (CDRs) User Guide

Revised: 5/8/06

Table of Contents

Introduction

Call Detail Records (CDRs) describe the calls handled by a sipX PBX. They are primarily used for billing and cost accounting. This memo covers CDRs and the “Call Resolver” program used to create them.

The initial release offers these features:

- CDRs describing calls, including caller and callee addresses, call duration, and so forth.
- Storage in a relational database, namely PostgreSQL.
- Automated creation of CDRs on a daily basis.
- Automated purging of old CDRs on a daily basis. By default CDRs for the last 35 days are retained.

Limitations, to be addressed in subsequent releases:

- Only data is provided, no report generation. It's up to the user to process the data.
- One CDR per SIP call. For example, a call that comes in to an auto attendant and is then transferred to a person will be recorded as two CDRs. From a higher-level viewpoint there is a single “logical call” that the CDR user would like to see.
- Basic calling service can be made HA by setting up multiple servers, but CDR processing itself is not HA. There are single points of failure and data may be lost in the event of system or network outages.

Call Detail Records

CDRs are stored in a database named “SIPXCDR”. We recommend accessing CDRs through either of two database views:

- `view_cdrs`: holds commonly used CDR data
- `view_cdrs_with_call_direction`: a superset of `view_cdrs` that adds a “call direction” column

Call direction is described in the “Call Direction” section later in this document.

Using the views rather than the underlying tables protects you against possible future changes to the database schema. However, the schema is publicly available as part of the sipX install at `/etc/sipxpbx/cdr/schema.sql` if you wish to access the tables directly.

`view_cdrs` has the following columns:

- `id`: the database row ID.
- `caller_aor`: the address of record (AOR) for the caller. Example: “sip:100@pingtel.com”.
- `callee_aor`: the AOR for the callee.
- `start_time`: when the call started. Example: “2006-04-14 21:45:16”.
- `connect_time`: when the call was connected.
- `end_time`: when the call ended, either normally or because of a failure.
- `duration`: how long the call lasted.
- `termination`: a single character termination code (see below).
- `failure_status`: if the call failed, contains the SIP error code, e.g., “500”.
- `failure_reason`: if the call failed, contains the SIP error message. For example, the typical error message for a 500 error is “Server Internal Error”.

All times are stored in UTC (GMT) without time zones. Duration is computed as `end_time` minus `connect_time`, which is ordinarily close to the value `end_time` minus `start_time`. The advantage of subtracting `connect_time` from `end_time` is that in an HA configuration with distributed proxies, these two values are more likely to have been recorded on the same machine, so the difference is less likely to be affected by clock synchronization errors.

Termination codes are:

- `R`: call requested – the call was requested but not connected.
- `I`: call in progress – the call was connected, but we don't know when or how it ended.
- `C`: successful call completion – normal case.
- `F`: call failed – an error occurred.

A CDR with one of the first two termination codes, `R` and `I`, is called “incomplete”. Incomplete CDRs can result from running the Call Resolver on a call that starts in the time window being analyzed but ends outside of it. Such CDRs may be completed in a subsequent run.

Call Direction

“Call direction” is an optional feature where calls are identified as:

- Incoming (I): the call is coming from a PSTN gateway

- Outgoing (O): the call is going to a PSTN gateway
- Intranetwork (A): neither the caller nor callee is a PSTN gateway

The Call Resolver looks at the caller and callee contacts and compares these addresses with the addresses of all gateways configured in sipXconfig, the configuration server. It resolves domain names to IP addresses to get addresses in a standard form where they can be easily compared. As mentioned earlier, the database view `view_cdrs_with_call_direction` extends `view_cdrs` by adding a `call_direction` column.

Call direction is not computed by default so that users who don't need it don't incur the cost of computing it. See the Configuration section for instructions on how to turn it on.

Data Access and Export

psql

"psql" is a command-line tool included with PostgreSQL. Enter this command:

```
psql -d SIPXCDR -U postgres
```

to start an interactive psql session. To print out all the CDRs, enter:

```
select * from view_cdrs;
```

Here is an example of what that might look like for a single record, simplified to fit on one line:

Row	***id***	***caller_aor***	***callee_aor***	***connect_time***	***end_time***	***duration***
1	1	sip:300@pingtel.com	sip:301@pingtel.com	1:00:00 PM	1:05:00 PM	5:00

Date/time columns above show just the time. The `start_time`, `termination`, `failure_status` and `failure_reason` columns have been omitted.

To execute the same SQL command without an interactive session:

```
psql -d SIPXCDR -U postgres -c "select * from view_cdrs;"
```

To export data into a CSV file:

```
psql -At -F "," SIPXCDR -U postgres -c "select * from view_cdrs" > cdrs.csv
```

dumps the CDR data into a file `cdrs.csv` that can then be imported directly into Excel or another spreadsheet program. See http://www.varlena.com/GeneralBits/40.php for more info. Use `view_cdrs_with_call_direction` instead of `view_cdrs` with the above command line if you want to include call direction in the exported data. The CSV file includes just data, there are no column headers.

pgAdmin

"pgAdmin" is an open source graphical tool available from http://www.pgadmin.org/ that is easier to use than psql. See the sipX wiki (http://sipx-wiki.calivia.com) for more information – search on "pgadmin".

Other Choices

A quick Google search on “postgresql data export” leads to the product “EMS Data Export for PostgreSQL”: see http://sqlmanager.net/en/products/postgresql/dataexport . Caveat emptor: we haven't tested this product and are not endorsing it.

Call State Events

To create CDRs, Call Resolver analyzes call state events (CSEs) logged by the sipX proxies. CSEs are stored in the same database as CDRs. CSE logging must be turned on so that CSEs will be recorded, to provide the raw data for CDRs. See the section “Configuration” below for details.

Call Resolver

Automated Daily Runs

Users do not typically invoke Call Resolver directly, rather it runs automatically every day at a fixed time, if daily runs are enabled. Each run creates CDRs for the preceding 24-hour period and also purges old data, if purging is enabled. Daily runs are disabled by default since many customers don't use CDRs. Purging is enabled by default and happens independently of whether daily runs are enabled. Purging discards CDRs older than 35 days and CSEs older than 7 days, to keep the disk space consumption under control. The age thresholds are configurable. CSEs take up a lot more space than CDRs and are less useful, so the default purge age for CSEs is lower than that for CDRs.

Automated runs of the Call Resolver fire it up like so:

```
sipxcallresolver.sh --daily
```

where `--daily` tells it to perform daily processing:

- Create CDRs for the previous 24 hour period, if daily runs are enabled
- Purge CSEs and CDRs older than the respective age thresholds, if purging is enabled

See “Configuration” below for instructions on turning on daily runs and configuring purging.

Manual Runs

Open a terminal window and type:

```
sipxcallresolver.sh --start "2005-12-1T02:47" --end "2005-12-2T02:47"
```

This example runs the Call Resolver on CSEs collected between 2:47 AM on December 1, 2005 and 2:47 AM on December 2, 2005, recording CDRs. `--start` provides the time at which analysis begins and `--end` the time at which analysis ends. `--end` is optional and defaults to 24 hours after the start time.

The time format is ISO 8601, the same format used by sipX log files. If you don't specify a time zone, then the start and end times are interpreted in the local time zone. Add a “Z” prefix to pick the GMT time zone, for example, "`2005-12-1T02:47`**`Z`**".

Configuration

To configure CDRs, log in to the config server (sipXconfig) web UI as “superadmin”. Open the menu “Calling”, then “Configuration”, then “Call Detail Records (CDRs)”. You will see the following settings:

- **Create CDRs Daily**: Enable this setting to set up automated daily creation of CDRs, unless you plan to do this manually, or create your own cron job to run the Call Resolver.
- **Log Forking Proxy Events** and **Log Authorization Proxy Events**. Turn this on so that the proxies log call state events to the SIPXCDR database.
- **Purge the CDR Database Daily**. Leave this enabled, or the disk will fill up.
- **Purge Age for CDRs**. Defaults to 35 days. Adjust it according to disk space and your desire to keep data around.
- **Call Resolver Log Level**. Set to NOTICE by default. Set the log level to DEBUG to get detailed information on the operation of the Call Resolver, for troubleshooting. Be aware that debug logs can fill up the disk quickly.

Click on “Show Advanced Settings” in the upper right to display two more parameters:

- **Purge Age for CSEs**. Defaults to 7 days.
- **Call Direction**. Turn this on to compute call direction as part of the CDR. Disabled by default.

Troubleshooting

- Run `sipxcallresolver.sh --configtest` to check the configuration:
  - Can a connection to the local PostgreSQL database be established?
  - Are there multiple instances of the proxies running (there should be at most one instance of each proxy running on a server)?
  - Is call state event logging enabled?
  - Is the Call Resolver configured for daily runs and purging?
  - Did the daily cron job and the Ruby gems get installed correctly?

System Requirements

- Red Hat Enterprise Linux 4 or Fedora Core 4.
  - Other Linux distributions should work but have not been tested.
  - The Call Resolver is 100% portable Ruby code that should run just about anywhere, including Windows and MacOS. But we haven't tested that, and the proxies that generate call state events run only on Linux.
- PostgreSQL 7.4 or greater
- Ruby Libs 1.8.4
- Ruby 1.8.4
- Ruby Devel 1.8.4
- Irb 1.8.4
- RubyGems 0.8.11
- Rails 1.0.0 and its dependencies
- Postgres-pr 0.4

To check the Ruby installation, on Fedora Core 4:

```
$ yum list installed | grep ruby
ruby.i386                               1.8.4-1.fc4           installed
ruby-devel.i386                         1.8.4-1.fc4           installed
ruby-libs.i386                          1.8.4-1.fc4           installed
$ yum list installed | grep irb
irb.i386                                1.8.4-1.fc4           installed
```

Try running Ruby:

```
$ ruby –version
ruby 1.8.4 (2005-12-24) [i386-linux]
```

Use RubyGems to show which “gems” are installed. Most of these gems are Rails and its dependencies:

```
$ gem list

*** LOCAL GEMS ***
actionmailer (1.1.5)
    Service layer for easy email delivery and testing.
actionpack (1.11.2)
    Web-flow and rendering framework putting the VC in MVC.
actionwebservice (1.0.0)
    Web service support for Action Pack.
activerecord (1.13.2)
    Implements the ActiveRecord pattern for ORM.
activesupport (1.2.5)
    Support and utility classes used by the Rails framework.
postgres-pr (0.4.0)
    A pure Ruby interface to the PostgreSQL (>= 7.4) database
rails (1.0.0)
    Web-application framework with template engine, control-flow layer,
    and ORM.
rake (0.7.0)
    Ruby based make-like utility.
sources (0.0.1)
    This package provides download sources for remote gem installation
```

Call Resolver High Availability (HA) setup

In an HA environment, call state event logging to a database is done by proxies running on separate machines, with the Call Resolver located on the Master Server. The Call Resolver establishes SSL connections to these databases and resolves the call state events into a single CDR database on the Master Server.

There are a number of configuration steps that must be performed before the Call Resolver can operate on distributed databases. The first step is to follow all of the instructions given in the ‘High Availability Setup Guide’. Most important for the subsequent Call Resolver configuration is the creation and installation of SSL certificates on the Master Server and the Distributed Server. The name of the Certificate Authority file (*caname.crt* in the High Availability Setup Guide) is one of the configuration parameters needed by the Call Resolver.

Call Resolver setup on the Master Server

Two configuration parameters in /etc/sipxpbx/callresolver-config.in have to be added to enable the Call Resolver to establish SSL connections to distributed databases :

SIP_CALLRESOLVER_CSE_HOSTS : {comma-delimited list of host names and port numbers}

SIP_CALLRESOLVER_CSE_CA : {name of the Certificate authority file caname.crt}

The Call Resolver uses stunnel to establish SSL connections to remote databases. Stunnel is an SSL-encrypting socket wrapper that provides SSL support to applications by forwarding connections on local ports to another instance of stunnel running on a remote machine.

'localhost' must be specified as part of the list of host names if a CSE database is running on the local machine, otherwise the Call Resolver will not connect to the local CSE database. If no port is specified it will default to the standard PostgreSQL port 5432.

The following example has the Call Resolver running on the Master Server master.example.com and the Distributed Server running on distrib.example.com. The Call Resolver will connect to the database on the Distributed Server on port 5433. The name of the Certificate Authority file is 'ca.example.com.crt'.

SIP_CALLRESOLVER_CSE_HOSTS : localhost, distrib.example.com:5433

SIP_CALLRESOLVER_CSE_CA : ca.example.com.crt

More than one Distributed Server can be specified.

SIP_CALLRESOLVER_CSE_HOSTS : localhost, distrib1.example.com:5433, distrib2.example.com:5434

SIP_CALLRESOLVER_CSE_CA : ca.example.com.crt

Note that the ports specified can be any unused ports on the Master Server. The local PostgreSQL database is using its default port 5432, so 5433 and 5434 are unused but 'PostgreSQL-like' ports). The specified ports do not describe any real connections to distributed systems, they are only used for forwarding local connections on these ports to the distributed machines. The port numbers must be unique, no two host names can have the same port number.

Distributed Server setup

Because the web UI can only configure proxies on the Master Server the configuration files on the Distributed server need to be edited manually. To enable call state event logging manually edit the file /etc/sipxpbx/proxy-config.in to include the line

SIP_PROXY_CALL_STATE_DB : ENABLE

and the file /etc/sipxpbx/authproxy-config.in to include the line

SIP_AUTHPROXY_CALL_STATE_DB : ENABLE

in order to enable call state logging to the database.

Since the Call Resolver is not running on the Distributed Server there is no need to configure it. The Distributed Server still must be configured to accept SSL connections from the Call Resolver running on the Master Server to allow the Call Resolver to retrieve call state events.

The following example uses the Certificate Authority file 'ca.example.com.crt' to set up the Distributed Server. As root from the command line enter the command

sipxha-distrib.sh –setup ca.example.com.crt

and the Distributed Server should be set up to accept SSL connections to its database.

HA setup Troubleshooting

- On the Master Server run **`sipxcallresolver.sh –configtest`**
- On the Distributed Server run **`sipxha-distrib.sh –configtest`**
- Check if stunnel is installed on both machines:
  `rpm -qa | grep stunnel` should return an entry, otherwise run

  `up2date install stunnel` (on Red Hat Enterprise) or
  `yum install stunnel` (on Fedora Core)