Cisco Policy Suite Geographical Redundancy Guide

Release 7.5.0

First Published: July 17, 2015
Last Updated: July 17, 2015

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies are considered un-Controlled copies and the original on-line version should be referred to for latest version.

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Preface

Welcome to Cisco Policy Suite (CPS) Geographical-Redundancy (GR) Overview Guide.

This document describes the Geo-Redundancy architecture. This document is intended as a starting point for learning about GR and how it works and does not contain details around system installation or configuration.

Readers

This guide is best used by the following readers:

- Deployment engineers
- System administrators
- Network administrators
- Network engineers
- Network operators
- Implementation engineers

This document assumes a general understanding of network architecture, configuration, and operations.

Additional Support

For further documentation and support:

- Contact your Cisco Systems, Inc. technical representative.
- Call the Cisco Systems, Inc. technical support number.
- Write to Cisco Systems, Inc. at support@cisco.com.
- Refer to support matrix at http://www.support.cisco.com and to other documents related to Cisco Policy Suite.

Terms and Definitions

This document uses certain terms and definitions specific to the CPS software application. For common Glossary of Terms, refer to http://wwwin.cisco.com/tech/EngCoE/cpdm/glossary.shtml.

Version Control Software

Cisco Policy Builder uses version control software to manage its various data repositories. The default installed version control software is Subversion, which is provided in your installation package.

Conventions (all documentation)

This document uses the following conventions.

Conventions	Indication
bold font	Commands and keywords and user-entered text appear in **bold** font.
italic font	Document titles, new or emphasized terms, and arguments for which you supply values are in *italic* font.
[ ]	Elements in square brackets are optional.
{x | y | z }	Required alternative keywords are grouped in braces and separated by vertical bars.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
courier font	Terminal sessions and information the system displays appear in courier font.
< >	Nonprinting characters such as passwords are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

Note: Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution: **Means *reader be careful.* In this situation, you might perform an action that could result in equipment damage or loss of data.**

Warning: **IMPORTANT SAFETY INSTRUCTIONS**

Means danger. You are in a situation that could cause bodily injury. Before you work on any equipment, be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents. Use the statement number provided at the end of each warning to locate its translation in the translated safety warnings that accompanied this device.

SAVE THESE INSTRUCTIONS

Regulatory: Provided for additional information and to comply with regulatory and customer requirements.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation* at: http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html.

Subscribe to *What's New in Cisco Product Documentation*, which lists all new and revised Cisco technical documentation as an RSS feed and delivers content directly to your desktop using a reader application. The RSS feeds are a free service.

Overview

First Published: July 17, 2015

Last Updated: July 17, 2015

This chapter covers the following sections:

CPS Architecture Overview

The Cisco Policy Suite (CPS) solution utilizes an innovative, three-tier virtual architecture that has been designed for scalability, system resilience and robustness to achieve 99.999% carrier grade system availability. Modules are deployed as Virtual Machines (VMs) using CentOS as the Operating System (OS) and the VMWare Hypervisor. Each module runs multiple virtual instances for performance and resiliency:

The main architectural layout of CPS is split into two main parts:

- Operations, Administration and Management (OAM)
- 3 Tier Processing Architecture

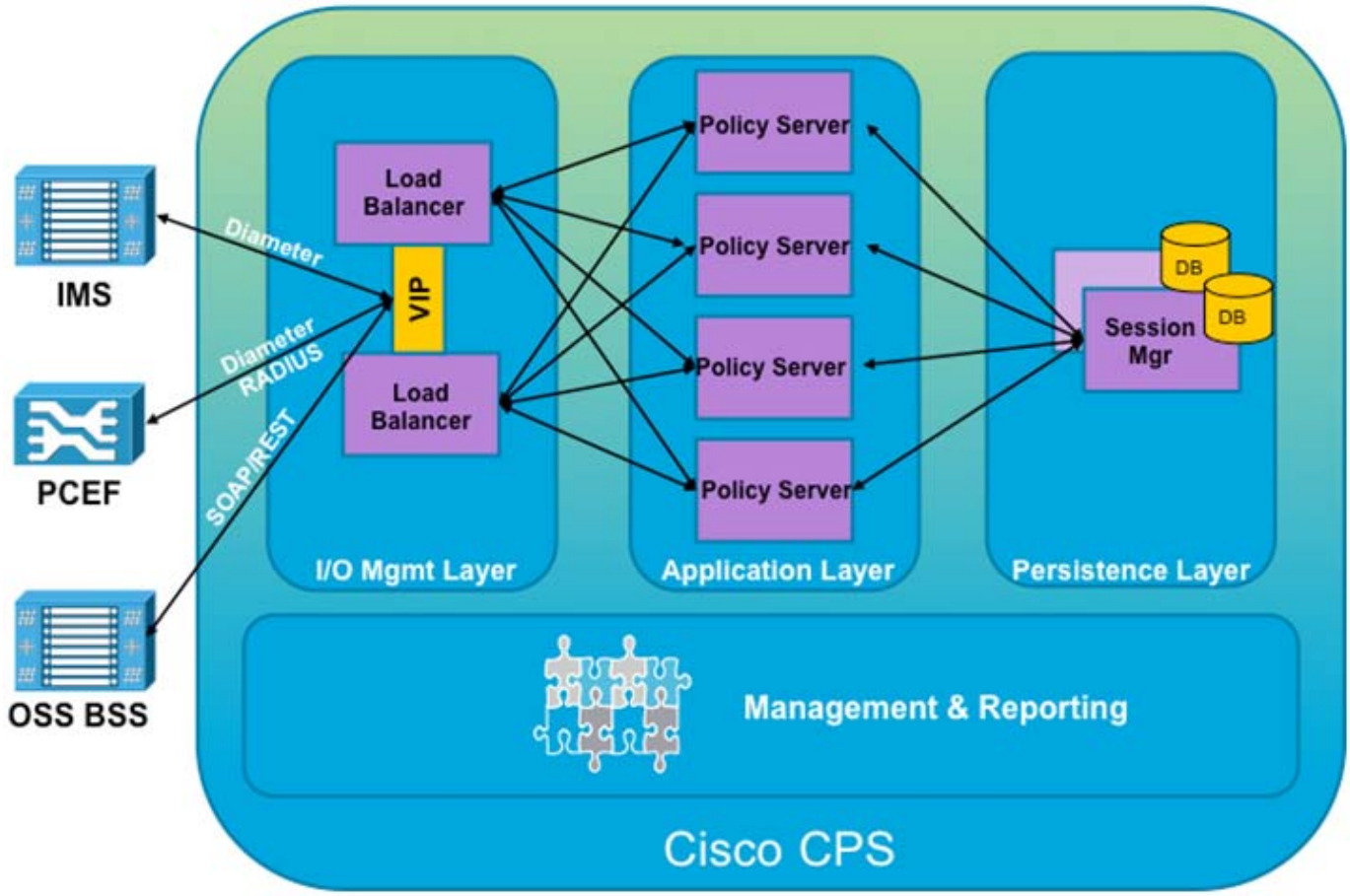
Operations, Administration and Management (OAM)

The OAM contains the CPS functions related to the initial configuration and administration of CPS. Operators use the Policy Builder GUI to define the initial network configuration and deploy customizations.

Operators use the Control Center GUI or functionality that is provided through the Unified API to monitor day-to-day operations and manage the network. One of the primary management functions of Control Center is to manage subscribers. This aspect of Control Center is also referred to as the Unified Subscriber Manager (USuM).

3 Tier Architecture

This document describes the 3 Tier architecture.



The 3 tier architecture defines how CPS handles network messages and gives CPS the ability to scale. The 3 processing tiers are:

- I/O Management Layer

  The I/O Management Layer handles I/O management and distribution within the platform. This functionality is implemented by the Load Balancer (LB) VMs, which are also sometimes referred to as Policy Director (PD) VMs. This layer supports internal load balancers to load balance requests to the relevant modules. One or more Virtual IP (VIP) addresses are allocated across load balancer instances to segregate the network as well as to handle failover.

  The Policy Directors terminate all external connections including Diameter, Web Services interfaces, etc.

- Application Layer

  The Application Layer handles the transaction workload and is stateless in that it does not maintain subscriber session state information. This functionality is implemented by the CPS VMs.

  The main module of the Application Layer is a high performance rules engine. The rules engine provides the foundation for enabling event triggers, conditions, and actions. This way, the solution allows for network, application, and subscriber events to trigger programmable business logic to modify the subscriber's network and application sessions in real time. The rules engine is optimized to process complex decisions and subscriber state machines to determine which actions to apply to a subscriber. It is exposed through a provisioning GUI for authoring granular business rules.

- Persistence Layer

  The persistence layer consists of the Session Manager - a document-oriented database used to store session, as well as subscriber and balance data (if/when applicable). This functionality is implemented by the Sessionmgr VMs. The database that Session Manager includes:

  - Admin
  - Audit
  - Customer Reference Data
  - Policy Reporting
  - Sessions
  - Balance
  - SPR

  For more information on Persistence Layer, refer to Persistence Layer, page 7.

Persistence Layer

The Session Manager application is responsible for storing session information, as well as subscriber profile and quota information if applicable.

The Session Manager is built using Mongo DB, a high performance, high availability document-oriented database. Unlike traditional databases which contains tables and columns/rows, Mongo contains collections (tables) and documents (rows). The documents are schema-less, but still can be indexed for high speed lookups.

One advantage of a schema-less database is the ability to enable plug-ins to add or augment existing data, indexes and key into the database. For example, CPS core keeps a minimum amount of information about a user's session such as IP address, MAC address, credential, etc. The Radius plug-in keeps additional data on the session document such as the network device's IP address and other radius specific information as well as providing a primary session key for accountSessionId (for call flows). These additional fields within the document can also be indexed if needed. This enables an extremely flexible data model without sacrificing performance.

The way that the Mongo DB obtains high performance is by using a 'file-backed in-memory database'. To achieve its high performance, the MongoDB stores as much of the database as possible in memory (and thus is very fast), but the data is mirrored and written out to disk to preserve the database information across restarts.

Access to the database is typically performed using the Unified API (SOAP/XML) interface. GUI access is typically limited to Lab environments for testing / troubleshooting, and can be used to perform the following tasks:

- Manage subscriber data (if SPR used), that is, find, create and/or edit subscriber information
- Stop database or check the availability of subscriber sessions
- Review and manage subscriber sessions
- Populate customer reference data tables: Customer reference data tables allow the service provider to create their own data structures that can then be used during policy evaluation. Examples of information that can be included in customer reference data tables include device parameters, location data mapping (e.g. to map network sites and cell sites into the subscriber's home network, roaming network, or preferred roaming network), IMEI data tagging for smart phone, Apple, or android device, etc.

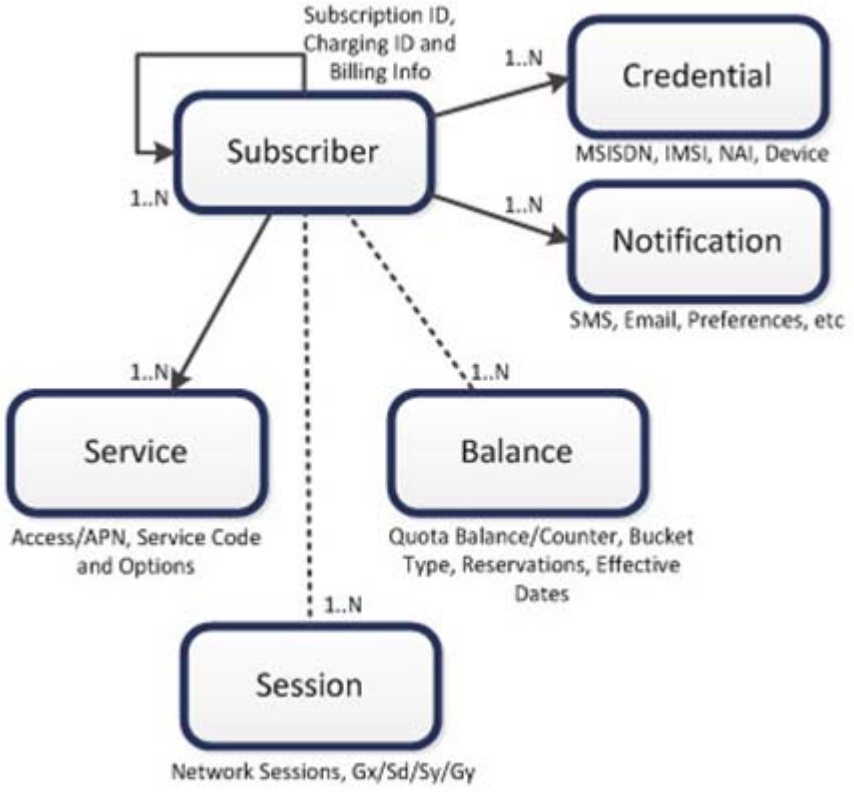
Unified Subscriber Manager (USuM)/Subscriber Profile Repository (SPR)

USuM manages subscriber data in a Subscriber Profile Repository (SPR). This includes the credentials a subscriber is able to log in with and what 'services' they are allocated. The details of what a 'service' means are stored in Policy Builder.

Each subscriber record that is stored in the USuM is a collection of the data that represents the real world end-subscriber to the system. Examples include which of the service provider's systems that the subscriber can access (mobile, broadband, wifi, etc.) or to identify specific plans and service offerings that the subscriber can utilize.

Additionally, USuM can correlate balance and session data to a subscriber. Balance data is owned by MsBM and is correlated by the Charging Id. Session data is correlated by the credential on the session which should match a USuM credential. Session data is managed by CPS core and can be extended by components.

In 3GPP terminology, the USuM is a Subscriber Profile Repository (SPR). The following is a symbolic representation of how the data portions of the Cisco SPR relates and depend on each other.



SPR mainly maintains the subscriber profile information such as Username, Password, Domain, devices configured, services (plans), etc. SPR database is updated by provisioning process and queried at start of session.

Session Manager (SM)

The session manager database contains all the state information for a call flow. Components such as diameter or custom components can add additional data to the session database without core code changes.

Multi-Service Balance Manager (MsBM)

MsBM is used to support any **Use Cases** that require balance e.g, volume monitoring over Gx also uses the Balance DB (without need for Gy, ...). It also handles the CPS implementation of an online charging server (OCS). It handles quota and manages the subscriber accounting balances for CPS. Quota information is stored separately from a subscriber so that it can be shared or combined among multiple subscribers.

MsBM defines the times, rates and balances (quota) which are used as part of CPS. In addition, it performs the following functions:

- Maintains the multiple balances that can be assigned for each subscriber. Balance types can include: Recurring balances (for example, reset daily, monthly, or per billing cycle); one-time balances such as an introductory offer might be assigned per subscriber; both recurring and one time balances can be topped-up (credited) or debited as appropriate.
- Balances can be shared among subscribers, as in family or corporate plans.
- Operators can set thresholds to ensure some pre-defined action is taken after a certain amount of quota is utilized. This can be used to prevent bill shock, set roaming caps and to implement restrictions around family or corporate plans.
- Operators can configure policies to take action when a given threshold is breached. Examples include: Sending a notification to the subscriber; redirecting the subscriber to a portal or a URL; downgrading the subscriber's balance.

NOTE: The decision whether to utilize quota is made on a per service basis. Users who don't have quota based services won't incur the overhead of querying/updating the MsBM database.

Geographic-Redundancy

Overview

CPS can be deployed in geographically redundant manner in order to provide service across a catastrophic failure, such as data center failure of a site hosting a Policy Suite cluster. In a Geographically-Redundant (GR) deployment, two Policy Suite clusters are linked together for redundancy purposes with the clusters located either locally or remote from each other in separate geographic sites.

Geo-redundancy is achieved through data synchronization between the two sites in a geo-redundant pair through a shared persistence layer. The specific subscriber profile, balance and/or session data replicated across sites is determined based on the deployment architecture, service requirements and the network environment.

CPS supports active/standby redundancy in which data is replicated from the active to standby cluster. The active site provides services in normal operation. If the active site fails, the standby site becomes the primary and takes over operation of the cluster. In order to achieve a geographically distributed system, two active/standby pairs can be setup where each site is actively processing traffic and acting as backup for the remote site.

Concepts

The following HA/GR concepts and terms are useful in understanding a GR implementation of CPS:

- Active/Standby, page 10
- Failure, page 10
- Failover, page 10
- Failover Time, page 10
- Heartbeat, page 10
- Split Brain, page 10
- Arbiter, page 10
- Data Redundancy, page 10
- Operations Log, page 11

Active/Standby

The Active site is one which is currently processing sessions. The Standby site is idle, waiting to begin processing sessions upon failure of one or more systems at the Active site.

Note: Active/Standby and Primary/Secondary are used interchangeably in the context of Active/Standby GR solutions.

Failure

Failure refers to the failure of a given part in functioning. The part may be hardware, software, networking, or other infrastructure (power).

Failover

Failover refers to termination of the application/system at one site and the initiation of the same application/system at another site at the same level. Failovers can be **manually triggered**, where the system is brought down at the direction of an administrator and restored at a different site, or **automatically triggered**, in scenarios like, if the Master DB is not available at primary site without the direction of an administrator.

In both cases, failovers proceed through a set of predefined steps. Manual failover differs from manual intervention wherein depending upon the situation, faults etc., there are some additional steps that are executed to make a system Up or Down. Such steps might include patch installations, cleanup etc.

Failover Time

This refers to the duration needed to bring down the system, and start the system at the other site, until it starts performing its tasks. This usually refers to automatic failover.

Heartbeat

This is a mechanism in which redundant systems monitor health of each other. If one system detects that the other system is not healthy/available, it can start failover process to start performing the tasks.

Split Brain

This situation arrives when the link between Primary and Secondary sites goes down, and due to unavailability of Heartbeat response, each site tries to become Primary. Depending upon technologies/solutions used for high availability, the behavior of each might differ (both becoming Primary or both becoming Secondary and so on.) In general, this is an undesirable situation, and is typically avoided using solutions like Arbiter.

Arbiter

Arbiter is a light weight 'observer' process that monitors the health of Primary and Secondary systems, and takes part in the election process of the Primary (active) system, breaking any ties between systems during the voting process and ensuring no split-brain occurs in the event of for example a network partition. To make sure that this process works smoothly, one must have odd number of participants in the system (e.g. Primary, Secondary, and Arbiter).

Data Redundancy

There are two ways to achieve Data Redundancy.

- Data Replication
- Shared Data

Data Replication

In this mechanism, data is replicated between the Primary and Secondary sites so that it is always available to the systems. There are various factors that matter on efficiency of replication.

Bandwidth

Bandwidth is important when the amount of data that is replicated is large. With higher bandwidth, more data can be sent simultaneously. Also, if the data is compressed, it helps further better utilization of bandwidth.

Latency

Latency is the time required to send a chunk of data from one system to another. The round-trip latency is an important factor that determines speed of replication. Lower latency equates to higher round-trip speed. Latency typically increases with the distance and number of hops.

Encryption

Since, during replication, the data might travel on public network, it is important to have encryption of data for protection. Encryption involves time, and slows the replication. Data needs to be encrypted before it is transmitted for replication which takes additional time.

Synchronous/Asynchronous

In an asynchronous write and asynchronous replication model, a write to local system is immediately replicated without first waiting for confirmation of the local write. With this form of replication there are chances of data loss if the replication could not take place due to some issue.

This risk can be mitigated by the replication system through maintenance of operations log (oplog) which can be used to reconfirm replication. In the combination of asynchronous write, synchronous replication, oplog play a vital role. The application is made efficient by responding fast to writes, and data synchronization can also be ensured.

Shared Data

This is mostly applicable in case of local high availability where the data can be stored on an external shared disk which is not part of the system. This disk is connected to both the systems. So, in case a system goes down, the data is still available to redundant host. Such a system is difficult to achieve in case of Geographical Redundancy as write time to disk would be significant due to latency.

Operations Log

In the context of MongoDB, the operations log (oplog) is a special capped collection that keeps a rolling record of all the operations that modify the data stored in a database. Operations are applied to the primary database instance which then records the operation in the primary's oplog. The secondary members then copy and apply these operations in an asynchronous process, which allows them to maintain the current state of the database. Whether applied once or multiple times to the target dataset, each operation in the oplog produces the same results.

Geographic-Redundancy

GR Reference Models

First Published: July 17, 2015

Last Updated: July 17, 2015

This chapter covers the following section:

GR Reference Models

The Cisco solution stores session data in a document-oriented database. This has the key advantage that the application layer responsible for transactional Session data is stored in Mongo DB (document-oriented DB). Data is replicated to help guarantee data integrity. Mongo DB calls its replication configuration as replica sets as opposed to Master/Slave terminology typically used in Relational Database Management Systems (RDBMS).

Replica sets create a group of database nodes that work together to provide the data resilience. There is a primary (the master) and 1..n secondaries (the slaves), distributed across multiple physical hosts.

Mongo DB has another concept called Sharding that helps redundancy and speed for a cluster. Shards separate the database into indexed sets, which allow for much greater speed for writes, thus improving overall database performance. Sharded databases are often setup so that each shard is a replica set.

The replica set model can be easily extended to a Geo-redundant location by stretching the set across two sites. In those scenarios, a so-called Arbiter node is required. The Arbiter is used as a non-data-processing node that helps decide which node becomes the primary in the case of failure. For example, if there are four nodes: primary, secondary1, secondary2 and the arbiter, and if the primary fails, the remaining nodes “vote” for which of the secondary nodes becomes the primary. Since there are only two secondaries, there would be a tie and failover would not occur. The arbiter solves that problem and “votes” for one node breaking the tie.

The following models are described in this section:

GR Reference Models

Without Session Replication

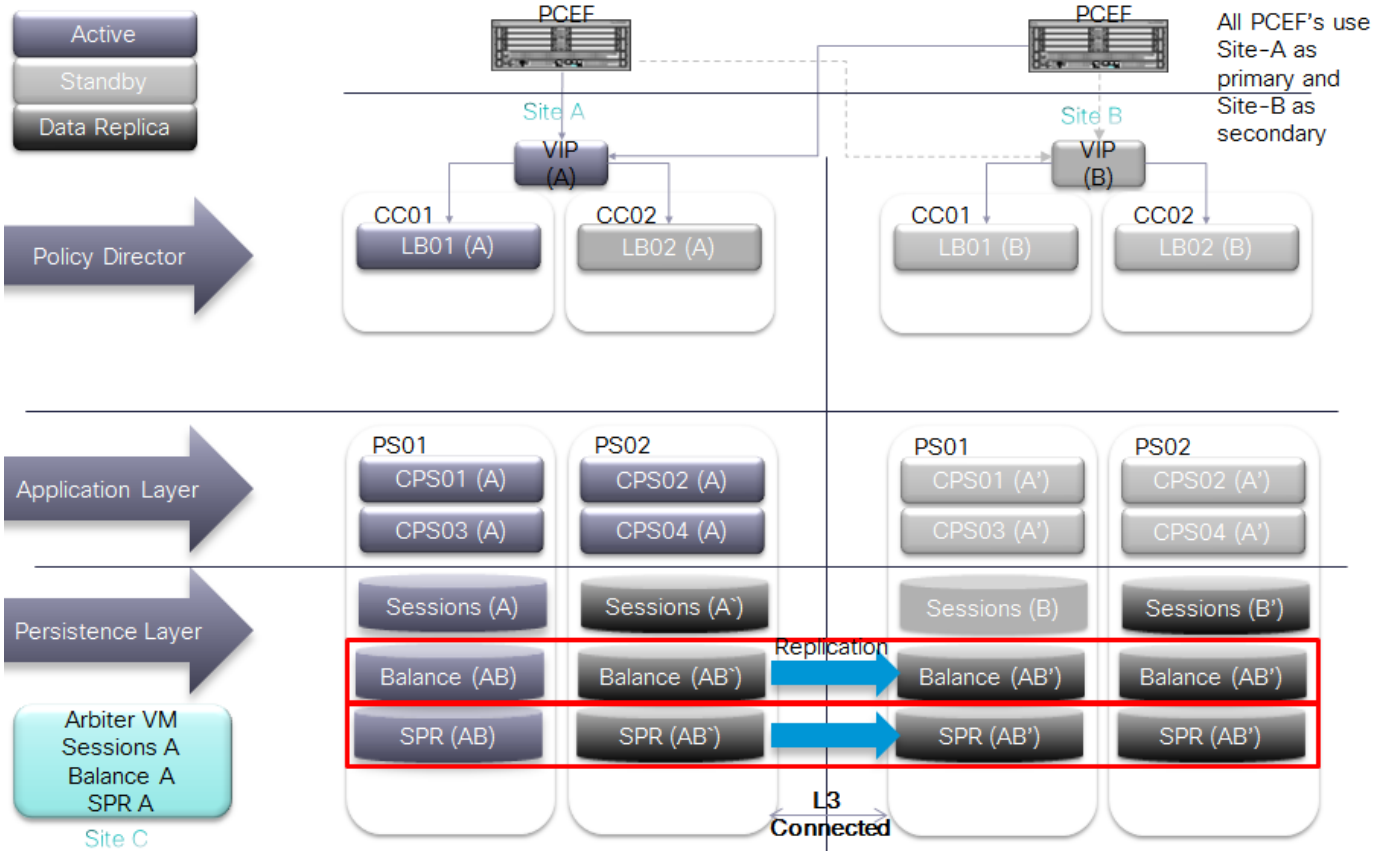
- If PCEF elements need to switch over clusters the current Diameter session between the PCEF and PCRF will be terminated and a new session will need to be reestablished.
- Simplifies architecture and reduces complexity.
- Quota data not reported. Currently, this is a limitation.

Active/Standby

In active-standby mode, one CPS system is active while the other CPS system, often referred to as the Disaster Recovery (DR) site, is in standby mode. In the event of a complete failure of the primary CPS cluster or the loss of the datacenter hosting the active CPS site, the standby site takes over as the active CPS cluster. All PCEFs use the active CPS system as primary, and have the standby CPS system configured as secondary.

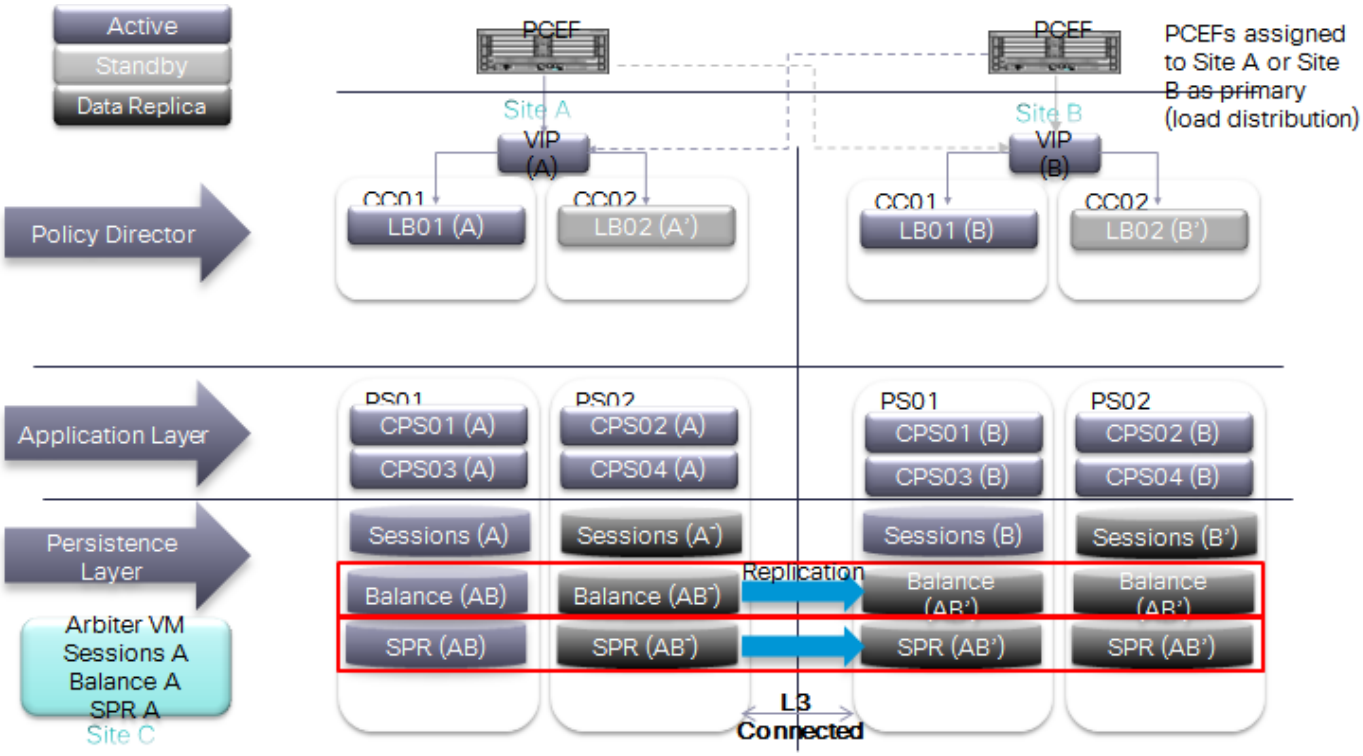
The backup CPS system is in standby mode; it does not receive any requests from connected PCEFs unless the primary CPS system fails, or in the event of a complete loss of the primary site.

If an external load balancer or Diameter Routing Agent (DRA) is used, the CPS in the active cluster is typically configured in one group and the CPS in the standby cluster is configured in a secondary group. The load balancer/DRA may then be configured to automatically fail over from active to passive cluster.



GR Reference Models

Active/Active

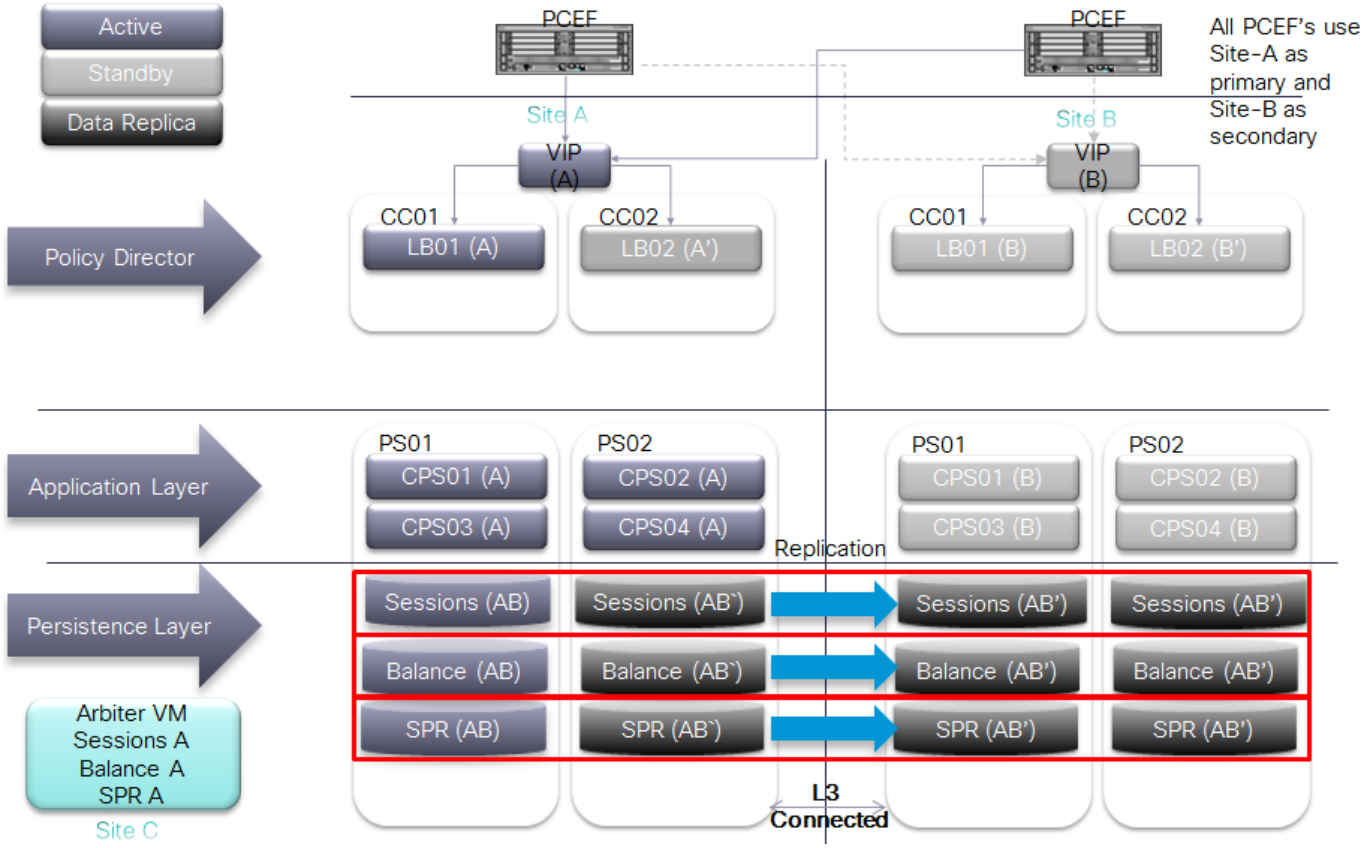


- Traffic from the network is distributed to two CPS clusters concurrently.
- PCEFs are divided within the Service Provider's network to have a 50/50% split based on traffic.
- Session data is not replicated across sites.
- SPR (subscriber information) data is replicated across Standby site.
- Balance data is replicated across Standby site.
- Diameter sessions need to be reestablished if a failover occurs. Outstanding balance reservations will time out and be released.
- In case of a failure all traffic is routed to the remaining CPS site.

GR Reference Models

With Session Replication

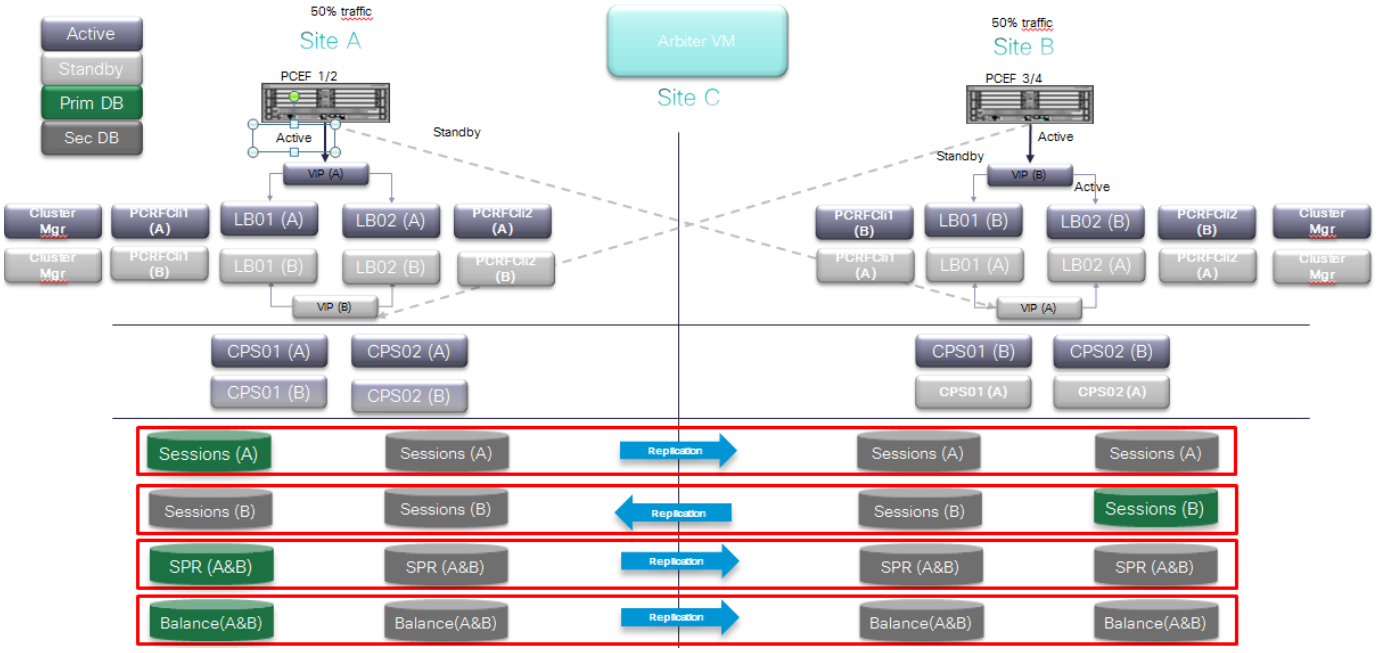
Active/Standby



- Solution protects against complete site outage as well as link failure towards one or more PCEF sites.
- If PCEF fails over to Secondary site while Primary site still active (e.g. link failure):
  - SPR data is retrieved from local SPR replica members at Secondary site.
  - Session and Balance data is read/written across from/to Primary site.
- Complete Outage of Policy Director Layer results in DB failover to Secondary site
- On recovery from a failure, a CPS node does not accept traffic until DBs are known to be in a good state

GR Reference Models

Active/Active



- Traffic from the network is distributed to two clusters concurrently.
- PCEFs are divided within the Service Provider's network to have a 50/50% split based on traffic.
- Session data is replicated across sites (two way replication).
- SPR (subscriber information) data is replicated across Standby site.
- Balance data is replicated across Standby site.
- Diameter session does not need to be reestablished if a failover occurs. No loss of profile or balance information.
- LB VMs use only local VMs for traffic processing.
- In case of a failure all traffic is routed to the remaining site.

GR Models Advantages and Disadvantages

The following table provides a comparison based on advantages and disadvantages for different GR models described in GR Reference Models, page 13.

Table 1 GR Models Advantages and Disadvantages

GR Model	Session	Other Databases	Advantages	Disadvantages
Active-Standby	Replicated	SPR and Balance replicated	Protection against complete site outage as well as link failure towards one or more PCEFs. Session Continuation, diameter sessions need not be re-established, hence VoLTE friendly.	Session replication demands bandwidth. In case there is network latency or high TPS, the hardware requirement increases as we are required split the incoming traffic across multiple virtual machines to achieve high speed replication and recovery.
Active-Standby	Not replicated	SPR and Balance replicated	Protection against complete site outage as well as link failure towards one or more PCEFs. Low bandwidth and low hardware requirements.	Sessions do not continue after failover, hence, they need to be re-established, NOT VoLTE friendly.
Active-Active	Replicated	SPR and Balance replicated, they are separate to each site	Protection against complete site outage as well as link failure towards one or more PCEFs. Session Continuation, diameter sessions need not be re-established, hence VoLTE friendly.	Session replication demands bandwidth. The hardware requirement increases significantly as we need additional load balancers and session cache virtual machines.
Active-Active	Not replicated	SPR and Balance replicated, they are separate to each site	Protection against complete site outage as well as link failure towards one or more PCEFs. Low bandwidth and significantly low hardware requirements.	Sessions do not continue after failover, hence, they need to be re-established, NOT VoLTE friendly.

SPR/Balance Considerations

SPR Considerations

- SPR data is read from secondary replica members
  - Mongo tag sets can be used to target read operations to local replica members, that way avoiding cross-site SPR reads
- SPR data is always written to primary DB

  - Profile updates are broadcast to other sites to trigger policy updates if/as required for sessions established through remote sites

- SPR updates that happen while primary site isolated are only enforced after session update (once primary site available again)

Balance Considerations

- Balance data is always read from primary DB unless primary DB not available (e.g. in the event of site isolation)
- Balance data is always written to primary DB
- Balance DB design options:
  - Single DB across two GR sites: cross-site balance read/writes from site without primary DB.
  - Sharded DB with each shard having primary DB at a different site.

    Requires that subscribers be assigned a home site (A or B). When user connects through P-GW connected to home site, balance read/writes are kept local

    Split can be based on e.g. IMSI range.
- CDR Reconciliation
  - During site isolation, debits are written to backup CDR database for reconciliation when connectivity is restored.
  - No thresholds / caps enforced during site isolation.
  - Policies associated with any threshold breaches during isolation are enforced at time of balance reconciliation.
  - Potential for balance leakage if balance consumed during isolation greater than user‘s remaining allowance.

Data Synchronization

Geo-redundancy is achieved by synchronizing data across the site(s) in the cluster. Three types of data are replicated across sites:

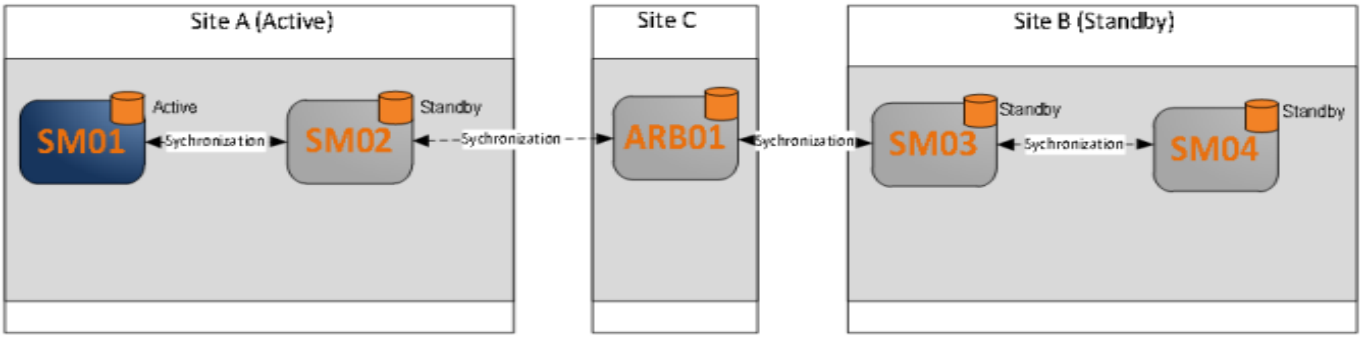
- Service and policy rule configuration
- Subscriber data that is stored in the SPR component
- Balance data stored in the MsBM component

In addition, active session data stored in the Session Manager component may also be synchronized across sites when network conditions permit. Active session data is the most volatile data in CPS and has the most stringent synchronization requirements.

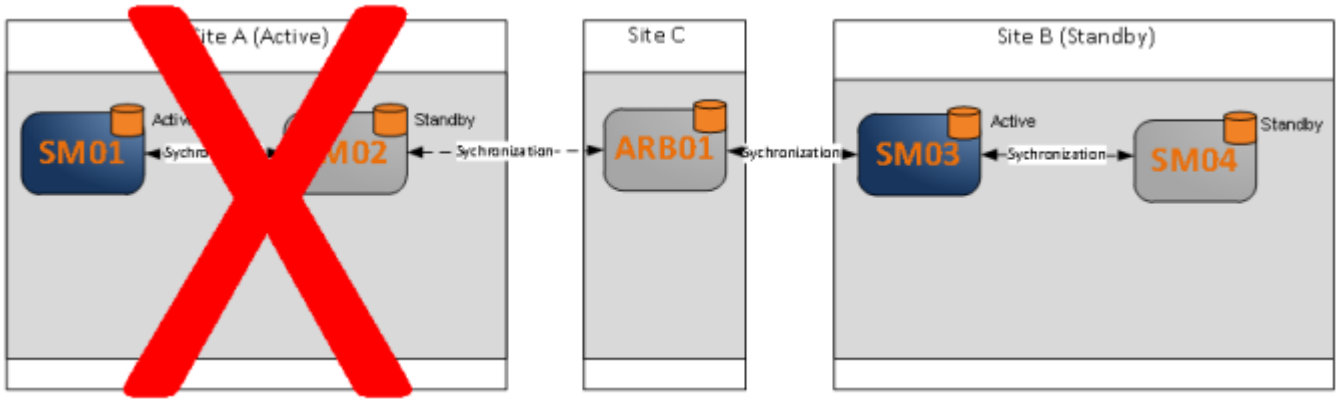
CPS utilizes a unicast heartbeat between sites in the geographic redundant solution. The heartbeat allows the session manager components to know which is the currently active component and protects against a split-brain scenario where data is accepted at more than one session manager component (possibly causing data corruption).

An additional external component called an “arbiter” provides a tie-breaking vote as to which of the session managers is the current master. This external component is required to reside on a separate site from the primary and secondary sites and must be routable from both sites. This is used to ensure that if one of the sites is lost, the arbiter has the ability to promote the standby sites session manager to be the master.

The following example shows a detailed architecture of the data synchronization for subscriber, balance and session data.



In the case of Site A failure, Site B's session manager will become master as shown in the following example:



Data Synchronization in mongoDB

In short, replication is achieved through a replica set where there are multiple members of a set. These members have one and only one primary member and others are secondary members. Write operations can happen to only primary, and read operations can happen from Primary and Secondary members. All the data written to Primary is stored in form of operation logs i.e. oplogs on Primary DB and secondaries fetch that to synchronize and remain up to date. In CPS, /etc/broadhop/mongoConfig.cfg defines replica members, replica sets and thus defines which data bases would be replicated and which not.

For more information on data synchronization in mongoDB, refer to http://docs.mongodb.org/manual/core/replica-set-sync/.

Recovery Procedures

When we expect a site to be down for a significant time, we must make sure that current available system is highly available i.e. if one of the databases instance fails, the system should still survive. To achieve this, one can remove failed members of the failed site from the replica set. Later during recovery, the removed members can be added again. The following example explains this scenario:

- Assume a replica set with five members, two on site-1 and two on site-2, and arbiter on 3rd site.
- Now, if site-2 fails, we have two members on site-1 and an arbiter available, and so system is up.

- If one of these members fails, we will have only two members available. Due to this, out of 5 members of replica set, we have only two members, and we do not have majority. Thus, the available members will become secondary (except arbiter) and the system will not be stable.
- To avoid this, and if we know our site-2 is not going to recover for a long time, we can remove members from site-2 from the replica set.
- Thus, there will be only three members in the replica set. So, even if one member from site-1 fails, we will have two members out of three, and still having majority and so will have a primary available. Hence, the system would be stable.

This section covers the following topics:

- Automatic Recovery, page 21
- Manual Recovery, page 21

Automatic Recovery

The system automatically recovers the database using the oplog to apply changes for failures which recover prior to the oplog rolling over.

Note: For databases that are mounted on tmpfs, automatic recovery is not recommended when virtual machines are rebooted as these are NOT local to Primary DB. In such cases, only manually recovery procedure is recommended.

Manual Recovery

If a failure impacts the system for a long period (e.g. data center, power or hardware failure) the database instance must be resynchronized manually as the oplog has rolled over. Full resynchronizations of the database are considered events that operation teams would like to execute during maintenance windows with personnel monitoring the status of the platform.

In Geo Redundancy, we have replica sets for different databases. The replication happens based on oplog. Oplog is a data structure that mongo maintains internally at Primary where the data operations logs are maintained. The secondaries fetch from the oplog on primary and apply those operations on themselves, thus getting synchronized. If secondary goes down for a long time, due to limited size of oplog, there are chances that some of the logs in oplog would be overwritten by new ones. In that case, when secondary comes up, it cannot synchronize with primary as it does not see a timestamp from where it had gone down.

Hence, we have to resort to manual resynchronization which is termed as initial-sync in mongodb. Here, on secondary, we delete the data contents, and ask to resynchronize. Secondary Session manager first copies all the data from primary and then copies the oplog.

Note: This procedure is only for manually recovering the databases. Based on the system status (all VMs down, traffic on other site, LBs down or up, all session cache down or only one down or so), some pre and post tests might be required to be executed. For example, if only one session manager is to be recovered, and we have primary DB and traffic on current site, we must not reset the DB priorities.

Similarly, if all our DBs and load balancers are down, one might want to stop **monit** and **qns** on all VMs before recovery. Similarly, after post recovery, one might want to run some sample calls on recovered site to make sure that system is stable, and then finally migrate traffic back to original **Primary**.

Before recovery, on all session manager virtual machines, perform the following steps:

1. vi /etc/ssh/sshd_config
2. Add the following entry at the end of `sshd_config` file.

```
#The below count - 130 - should be based on number of files that we have under secondary's data
directory. It should be close to the number of files there.
MaxStartups 130
```

3. Restart sshd service by executing the following command:

```
service sshd restart
```

4. Power ON pcrfclient01 on failed site.

5. Run `diagnostics.sh --get_replica` command to know which members are down.

Based on the status of system and subject to above note, check if you need to reset member priorities. For example, if site-1 is **Primary** and site-2 is **Secondary** and if site-1 has gone down, we need to login to new Primary and reset the replica members priorities in such a way that when site-1 comes UP again, it would not become Primary automatically.

To reset the replica members priorities in such a way that when site-1 comes UP again, it would not become Primary automatically, perform the following steps:

Note: If we have a Primary on site-1, and if we are recovering only a Secondary member on site-2, or site-1, we must not perform the following steps:

1. Go to current primary member and reset the priorities by executing the following commands:

```
ssh <primary member>
mongo --port <port>
conf=rs.conf()
conf.members[1].priority=2
conf.members[2].priority=1
conf.members[3].priority=4
conf.members[3].priority=3
rs.reconfig(conf)
exit
```

   Hereon, you need to maintain two consoles. One for recovering secondary member and another source secondary. The source Secondary is the Secondary that is nearest in terms of latency from the recovering Secondary. Also, note down the port and data directory for these members. Usually, they would be the same, only host name would be different.

2. Recover the member:

   a. Go to recovering Secondary and execute the following commands:

```
ssh <recovering Secondary>
ps -eaf | grep mongo
/etc/init.d/sessionmgr-<port> stop
cd <member data directory>
\rm -rf *
cp /var/qps/bin/support/gr_mon/fastcopy.sh .
```

   b. Go to nearest working available secondary and execute the following commands:

```
ssh <source Secondary>
mongo --port <mongo port>
#lock this secondary from writes
db.fsyncLock()
exit

ps -eaf | grep mongo
cd <data directory>
tar -cvf _tmp.tar _tmp
# You might see some errors here, those can be ignored.
tar -cvf rollback.tar rollback
# You might see some errors here, those can be ignored.
```

c. Go to recovering Secondary and execute the following commands:

```
cd <data directory>
./fastcopy.sh <nearest working secondary> <secondary data directory>
ps -eaf | grep scp | wc -l
```

d. Once the count is one, start secondary by executing the following commands:

```
tar -xvf _tmp.tar
tar -xvf rollback.tar
/etc/init.d/sessionmgr-<port> start
```

e. Go to nearest secondary from where we are recovering and execute the following commands:

```
mongo --port <port>
db.fsyncUnlock()
db.printSlaveReplicationInfo()
```

Check the lag for some time. Initially it would be small, later it will increase and then decrease. The member should be come secondary soon.

3. Now reset the sshd connection on all virtual machines. Just comment out the MaxStartup line in `/etc/ssh/sshd_conf` file.

```
#One member recovered
```

CPS GR Dimensions

CPS GR has the following multiple dimensions:

Out of the dimensions described in the following sections such as deployment style, databases, number of sites, arbiter considerations, and Shards, only deployment style does not dependent upon other factors. Rest of the attributes are inter-related. In that too,

- Arbiter typically decides models for shard.
- Number of sites impacts the decision to have database in common.
- Common database style impacts decision to have shard.

Different Databases

CPS has three databases that have subscriber critical data such as, Subscriber DB (SPR), Balance, and Sessions. Different deployment models exist depending upon how we want to have the databases configured. Some customers might want a DB common across different sites (typically this can happen for SPR), or individual instances at each site (most of the times this would be with sessions DB and balance DB). Typically, the databases that are updated more frequently (such as sessions and balance) would be maintained locally and replicated across sites whereas databases that are updated rarely can be kept common across sites (with some limitations).

Number of Sites

Typical deployment is expected to be two sites. However, there might be cases where multiple combinations might come up with respect to DB redundancy, common DB across multiple sites, general redundancy across multiple sites and so on. Since this is a highly variable factor, for each deployment model here, we need to understand various network requirements.

Arbiter Considerations

Typically the arbiter needs to be located at a third independent site. However, depending upon customers' needs and limitations, different deployment models come up where arbiter can be placed at one of the sites, creating limitations in the model.

The location of the Arbiter is an important factor in the design. Having the Arbiter located on the same site as that of Primary or Secondary poses various issues. The following table describes the issues:

Table 2 Arbiter Location Considerations

Distribution of Arbiter	Impact on System
Arbiter at Active site	When Active site goes down, DB on Secondary is supposed to become Primary. However, since it does not have required votes as Arbiter is also down at Primary site, role change cannot take place and we face downtime.
Arbiter on Secondary site	In this case, if Secondary site goes down, we do not have arbiter available. Due to this, DB on Primary site does not have majority of votes, and DB steps down. That way, we face downtime on system unless there is manual intervention. Additionally, if there is a split brain situation, since arbiter is on secondary site, DB role changeover starts from Primary to Secondary, which is unnecessary.
Arbiter on third site	This is the best and recommended way of placing an arbiter. In any case, either Primary failure or Secondary failure, a proper failover happens as there are always majority of votes available to select a Primary.

It is important to understand the placement of arbiter and its implications. In Geographical Redundancy, failover is expected when a site goes down completely. There are many possibilities for a site to go down and based on these possibilities, we can decide the location of arbiter.

DB Partitioning - Shards

When the DB size grows large, it is good to have it partitioned, in terms of MongoDB, this is done by creating shards for the DB. Mongo has some limitations for creating shards, and depending upon deployment model, shard considerations come in picture. When shards come in picture, we need to also consider the configuration servers for those shards. The configuration server decides which partition/shard contains what data. It has the keys based on which data distribution and lookup works.

Placement of these configuration servers also plays an important role in performance of databases. During site failures, if we have less number of configuration servers available, the performance of DB is degraded. Hence, it is important to place the configuration servers in such a way that maximum of them are available always. Typically, the configuration servers are placed in line with DB i.e. one at primary, another at secondary and third at the arbiter. Mongo supports maximum of three configuration servers.

Session Shard Considerations

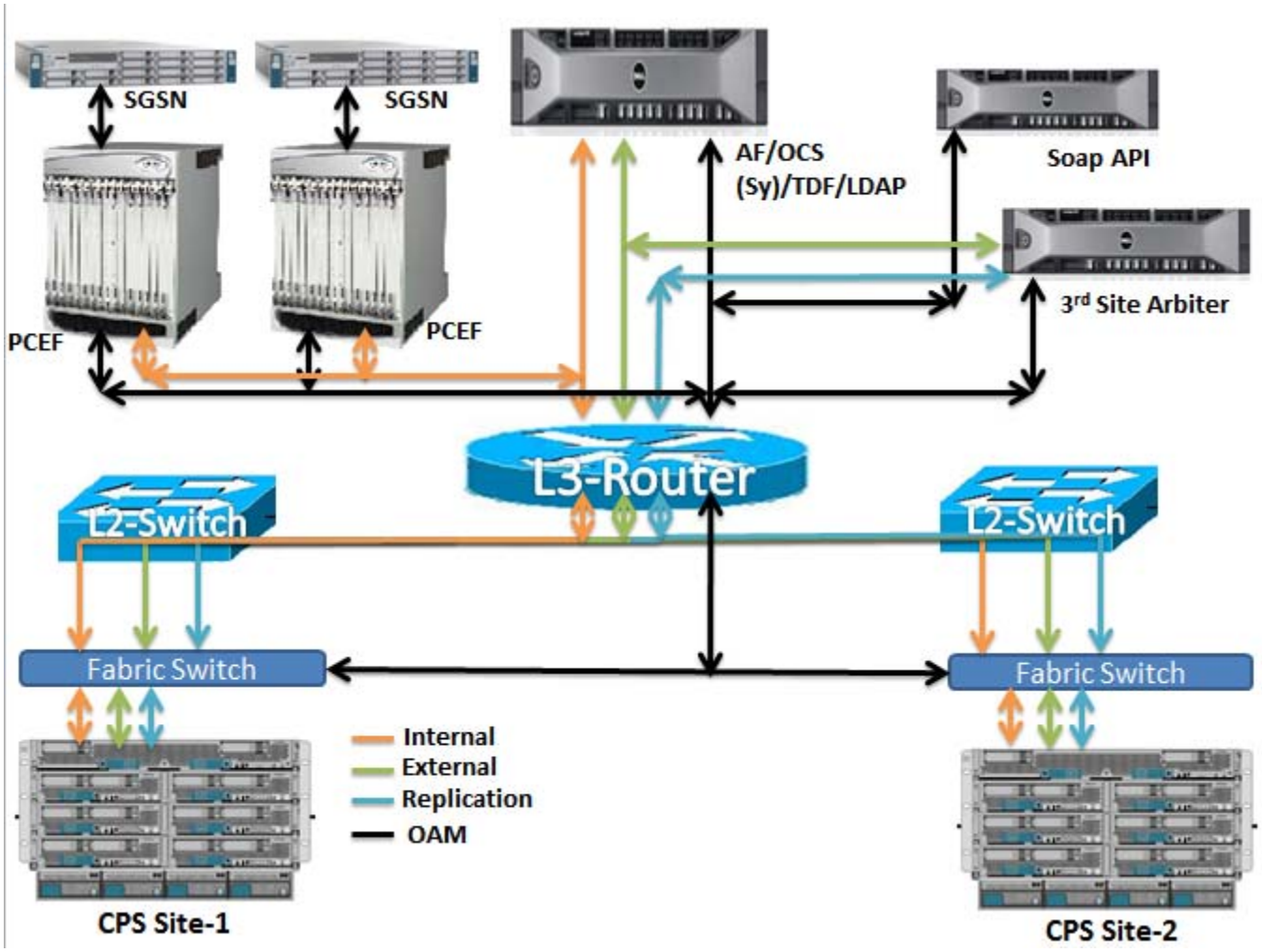
As against physical shards (equivalent to partitioned tables), for sessions, we define internal shards to make best of a sessions database. As of today, we create 4 internal shards per session database so that we see four internal databases. This helps to achieve parallel writes to same database thereby increasing write/read efficiency and achieve higher performance. Typically, for higher TPS, we might be required to create multiple such shards across different virtual

machines. In that case, an additional session replica set is created and that contains four more shards. The admin database contains information for all such shards so that qns processing engines route session calls to appropriate shards based on internal hashing algorithm. The actual number of shards required can be obtained from the dimensioning guide.

Network Diagrams

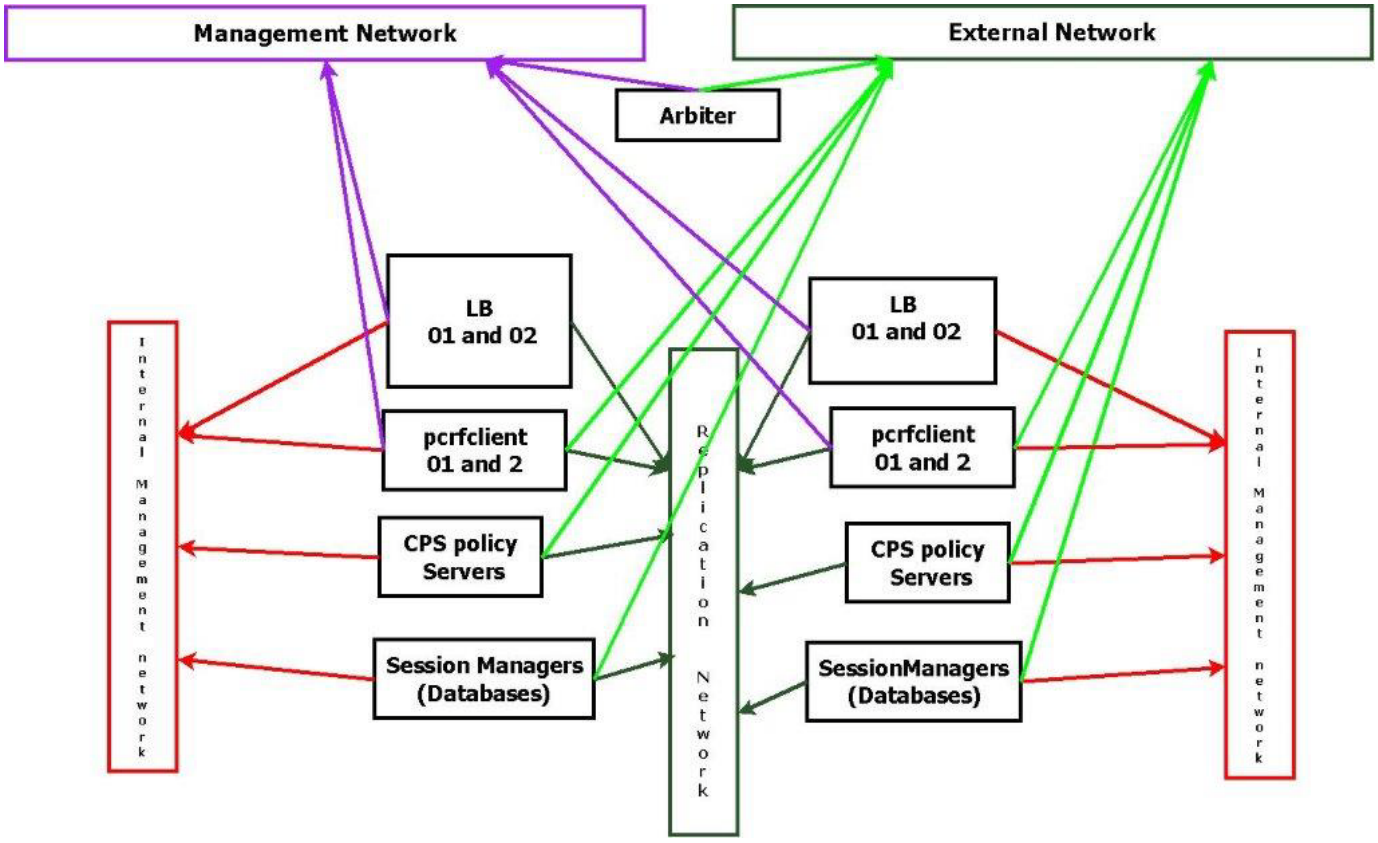
High Level Diagram including other Network Nodes

The following is an example of a high level diagram showing various network modules connected to CPS for GR setup. This diagram can be different for different deployment scenarios. Contact your Cisco Technical Representative for high level diagram specific to your deployment scenario.



CPS Level Network Diagram

The following network diagram explains various requirements for GR setup.



As shown in the above diagram, below are the interface requirements:

Management Network Interface

This interface is used for traffic to CPS, unified API, portal (not shown here), and for login to CPS machines through pcrfclient, and to access Policy Builder and Control Center web interfaces.

The following VMs need this network interface:

- Load balancers
- pcrfclient01
- Arbiter

External, Routable Network Interface

This interface is used for communication between DBs and arbiter. Since the mongo configuration servers reside on pcrfclient01 of both sites, a separate network is needed for both to communicate with each other over an interface other than replication network. If the replication network fails, communication would still be needed between the arbiter and session managers, and between arbiter and pcrfclient01 so that the arbiter is able to determine the appropriate primary for DBs, and make more than one configuration servers available. If this is not done, and if the arbiter is configured to communicate with DBs over the replication network, if the replication network fails, a split brain situation occurs since the arbiter would be disconnected from both sites.

When there are no shards configured for databases, no configuration servers are needed, the pcrfclient01 at both sites still needs external network connectivity with arbiter as scripts on pcrfclient need to communicate with the arbiter (such as get_replica_status.sh).

The following VMs need this network interface:

- pcrfclient01
- Arbiter
- Session managers

Replication Network Interface (Typically referred as Signaling Network)

This network carries the replication traffic between two sites. Also, qns policy servers on one site communicate with databases on another site using the same interface. The same network should be used to exchange messages between two sites.

The following VMs need this network interface:

- Load balancers (LBs)
- pcrfclient
- QNS policy servers (qns)
- Session managers (DBs).

Internal Network

This network is used for internal communication of virtual machines of the same site.

All the CPS VMs need this network interface.

Summary

The following table provides a summary of the different VM network requirements.

Table 3 VM Network Requirements

VM Name	Management IP	Signaling IP/Replication	Internal IP	External Non-Management IP
pcrfclient01/02	Yes	Yes	Yes	Yes
lb01/lbs01/02	Yes	Yes	Yes	No
portal01/portallb01/02	Yes	No	Yes	No
qns01-n/qnss01-n	No	Yes	Yes	No
sessoinmgrs	No	Yes	Yes	Yes

Network Requirements

Bandwidth and latency are to be obtained from Cisco Technical Representative depending upon your deployment model.

GR Failover Triggers and Scenarios

First Published: July 17, 2015

Last Updated: July 17, 2015

This chapter covers the following sections:

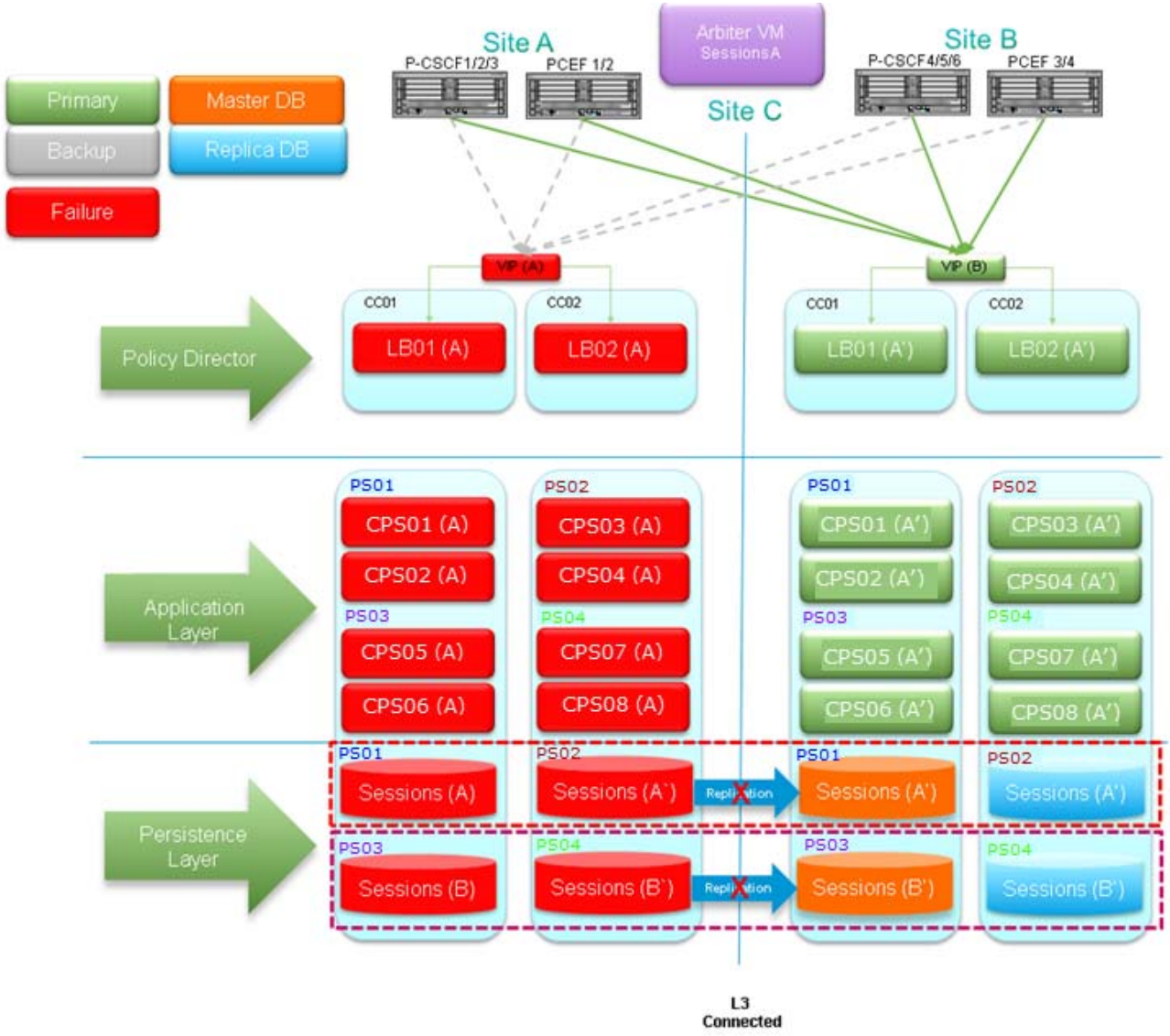
Failover Triggers and Scenarios

In Geographical Redundancy, there are multiple scenarios which could trigger a failover to another site.

This section covers the following topics:

Site Outage

As shown in the figure below, all P-GWs and P-CSCFs will direct traffic to the secondary site in the event of a complete outage of the primary site. Failover time will be dependent on failure detection timers on the P-GW and P-CSCF and the time it takes for the DB Replica set to elect a new Master DB at the secondary site.



In order for Site A to be considered “ready for service” after an outage, all 3x tiers (Policy Director, Application Layer and Persistence Layer) must be operational.

At the Persistence (DB Replica set) level, Mongo DB uses an operations log (oplog) to keep a rolling record of all operations that modify the data stored in the database. Any database operations applied on the Primary node are recorded on its oplog. Secondary members can then copy and apply those operations in an asynchronous process. All replica set members contain a copy of the oplog, which allows them to maintain the current state of the database. Any member can import oplog entries from any other member. Once the oplog is full, newer operations overwrite older ones.

When the replica members at Site A come back up after an outage and the connectivity between Sites A and B is restored, there are two possible recovery scenarios:

1. The oplog at Site B has enough history to fully resynchronize the whole replica set, for example the oplog did not get overwritten during the duration of the outage. In this scenario, the DB instances at Site A will go into “Recovering” state once connectivity to Site B is restored. By default, when one of those instances catches up to within 10 seconds of the latest oplog entry of the current primary at Site B, the set will hold an election in order to allow the higher-priority node at Site A to become primary again.
2. The oplog at Site B does not have enough history to fully resynchronize the whole replica set (the duration of the outage was longer than what the system can support without overwriting data in the oplog). In this scenario, the DB instances at Site A will go into “Startup2” state and stay in that state until we manually force a complete resynchronization (as they would be too stale to catch up with the current primary. A “too stale to catch up” message will appear in the mongodb.log or in the errmsg field when running rs.status(). For more information on manual resynchronization, refer to Manual Recovery, page 21.

   During a complete resynchronization, all the data is removed from the DB instances at Site A and restored from Site B by cloning the Site B session DB. All Read and Write operations will continue to use Site B during this operation.

Recovery time, holding time for auto recovery etc. depend upon TPS, latency, oplog size. For optimum values, contact your Cisco Technical Representative.

In CPS Release 7.5.0, at the Policy Director level, there is an automated mechanism to check availability of the Master DB within the local site. When the Master DB is not available, the policy director processes will be stopped and will not process with any incoming messages (Gx/Rx).

- This check runs at Site A (primary site).
- This check runs every 5 seconds (currently not configurable) and will determine whether the Master Sessions DB is at Site A.

  It is possible to configure which DBs the script will monitor (Sessions, SPR, Balance). By default, only the Sessions DB is monitored.
- If the Master DB is not available at Site A, the two Policy Director Processes (LoadBalancers) of site A will be stopped or remain stopped if recovering from a complete outage (as described in this section).
- In case of two replica sets, if one of the two Replica sets Master DB is not available at Site A, the two Policy Director Processes (LoadBalancers) of site A will be stopped or remain stopped if recovering from a complete outage and the second replica set Master DB will failover from Site A to Site B.

This above mentioned checks will prevent cross site communication for read/write operations. Once the site is recovered, P-GWs and P-CSCFs will start directing new sessions to Site A again.

For existing sessions, P-GWs will continue to send traffic to Site B until a message for the session (RAR) is received from Site A. That will happen, for example, when a new call is made and the Rx AAR for the new session is sent by the P-CSCF to Site A. Also, for existing Rx sessions, the P-CSCF will continue to send the traffic to Site B.

Gx Link Failure

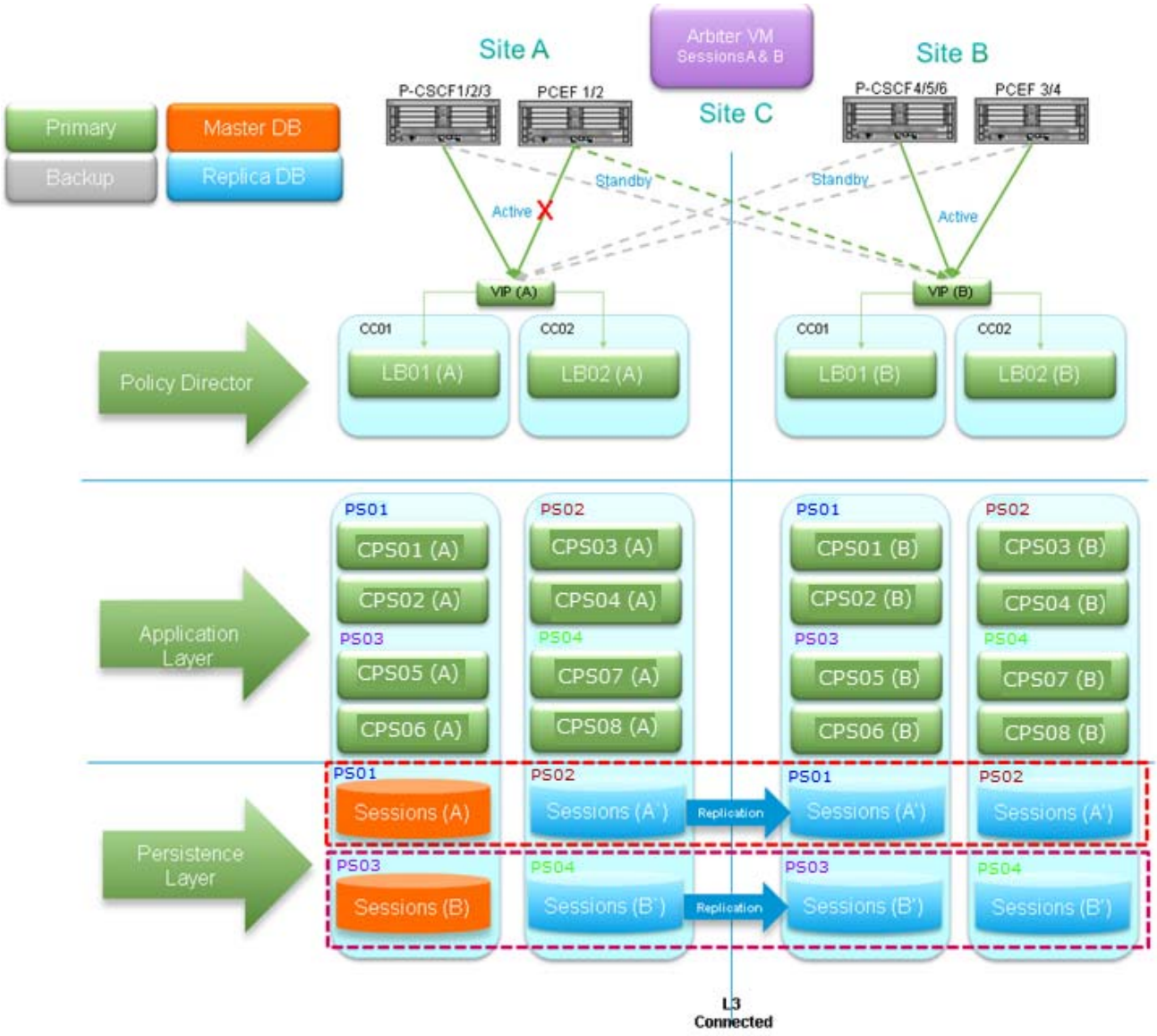
As shown in the figure below, failure of the Gx link between a P-GW and the primary CPS node (Site A) will result in the P-GW sending traffic to the secondary site (Site B). Failover time will be dependent on failure detection timers on the P-GW.

Gx transactions will be processed at Site B.

If a session already exists, the CPS0x(B) VM handling the transaction at Site B will retrieve the subscriber's session from the Master Sessions (A) DB at Site A. New sessions as well as session updates will also be written across to the Master DB at Site A.

Gx responses towards the P-GW (for example CCA), as well as Rx messages such as ASR that may be generated as a result of Gx transaction processing, will be sent from Site B.

After receiving an Rx AAR at Site A, the resulting Gx RAR will be proxied from the LB at Site A to the LB at Site B (as the P-GW is not reachable from Site A).

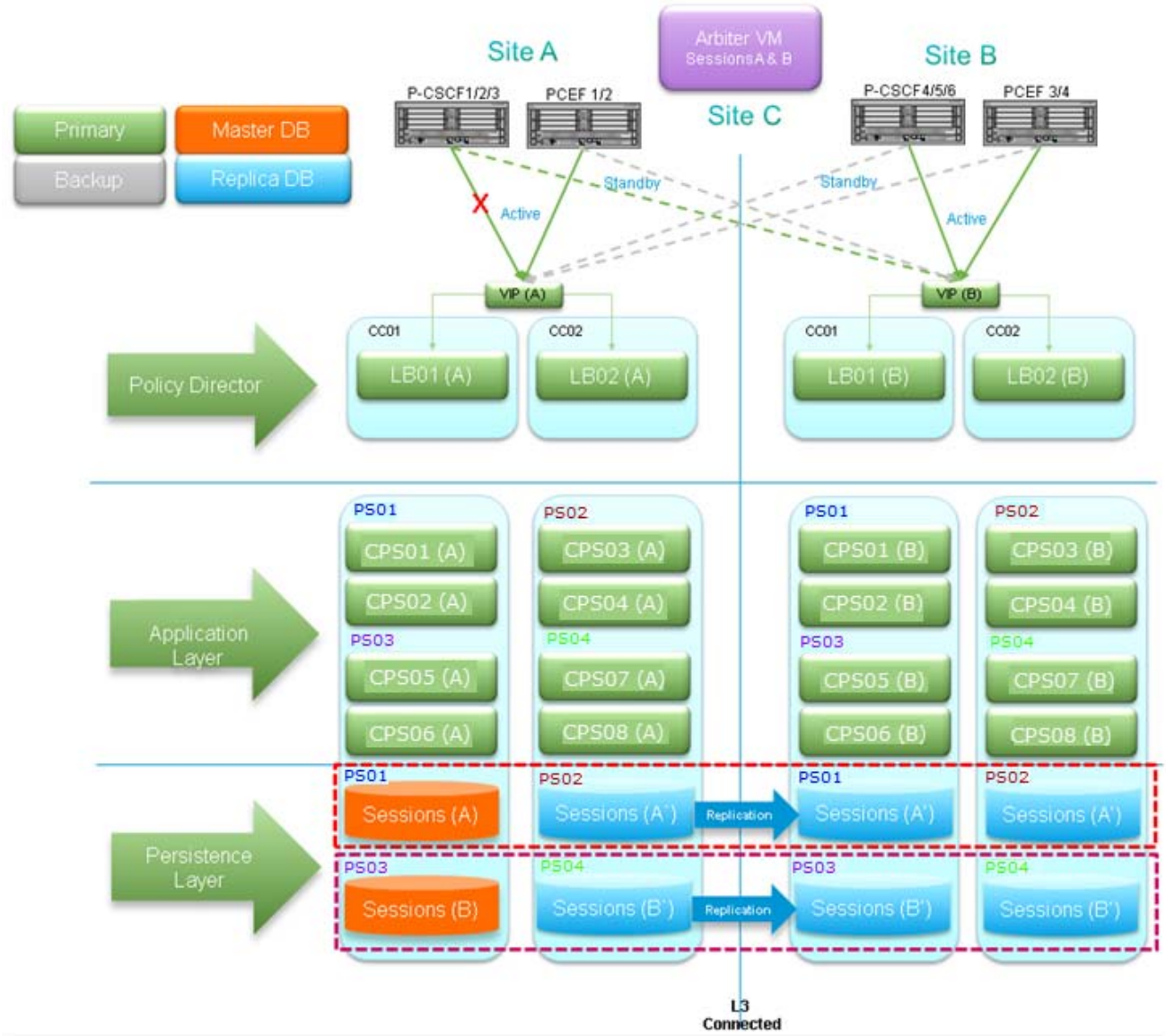


Rx Link Failure

As shown in the figure below, failure of the Rx link between a P-CSCF and the primary CPS node (Site A) will result in the P-CSCF sending traffic to the secondary site (Site B). Failover time will be dependent on failure detection timers on the P-CSCF.

Rx transactions will be processed at Site B. The CPS0x(B) VM handling the transaction at Site B will attempt to do the binding by retrieving the Gx session from the Master Sessions(A) DB at Site A. Session information will also be written across to the Master DB at Site A

The Rx AAA back to the P-CSCF as well as the corresponding Gx RAR to the P-GW will be sent from Site B.

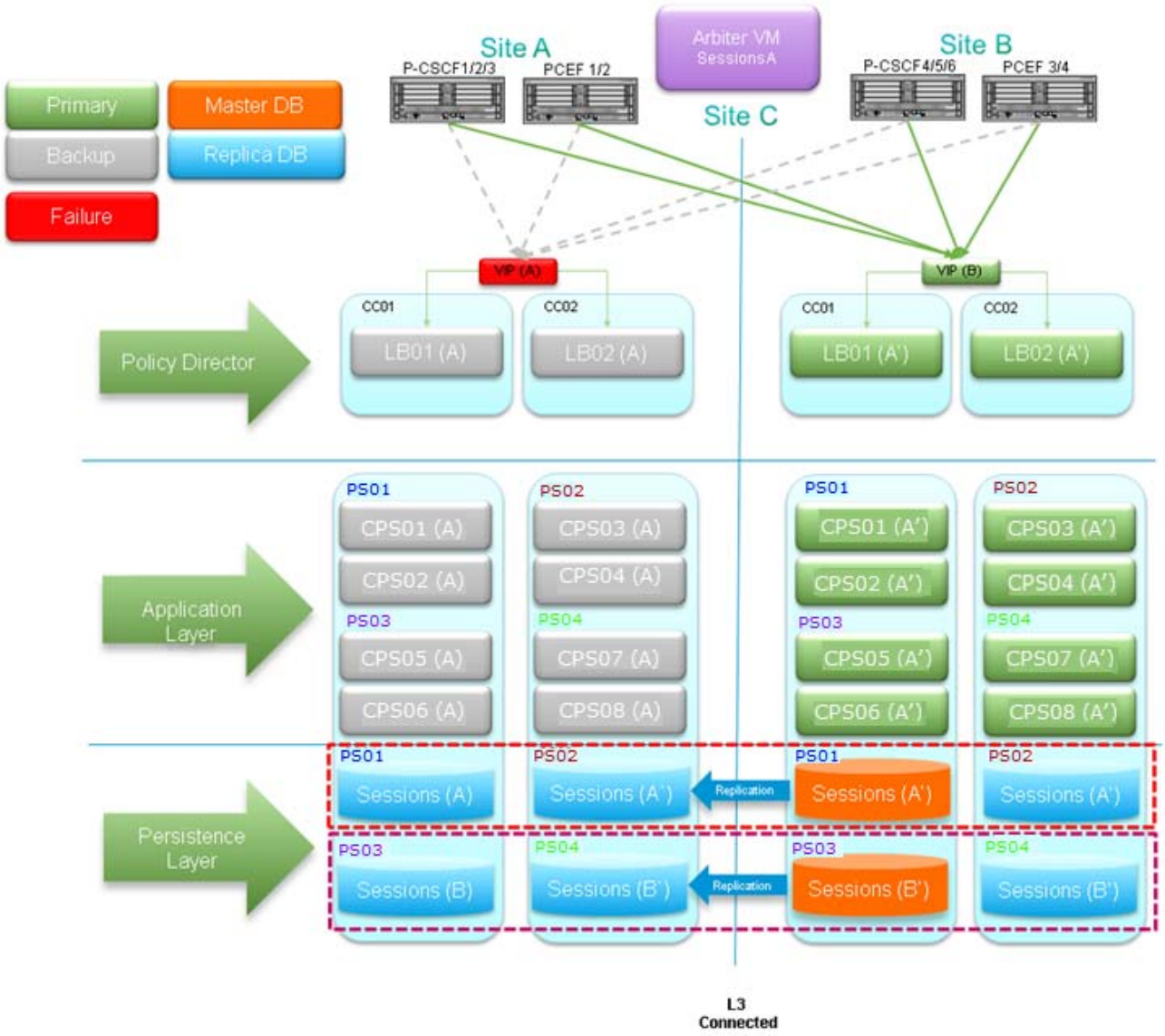


Load Balancer VIP Outage

As shown in the figure below, all P-GWs and P-CSCFs will direct traffic to the secondary site if both Load Balancer at the primary site is not available (which leads the VIP to be not available). Failover time will be dependent on failure detection timers on the P-GW and P-CSCF.

In order to avoid DB writes from Site B to Site A, the system can be configured to monitor VIP availability and, if VIP is not available, lower the priority of the DB instances at Site A to force the election of a new Master DB at Site B.

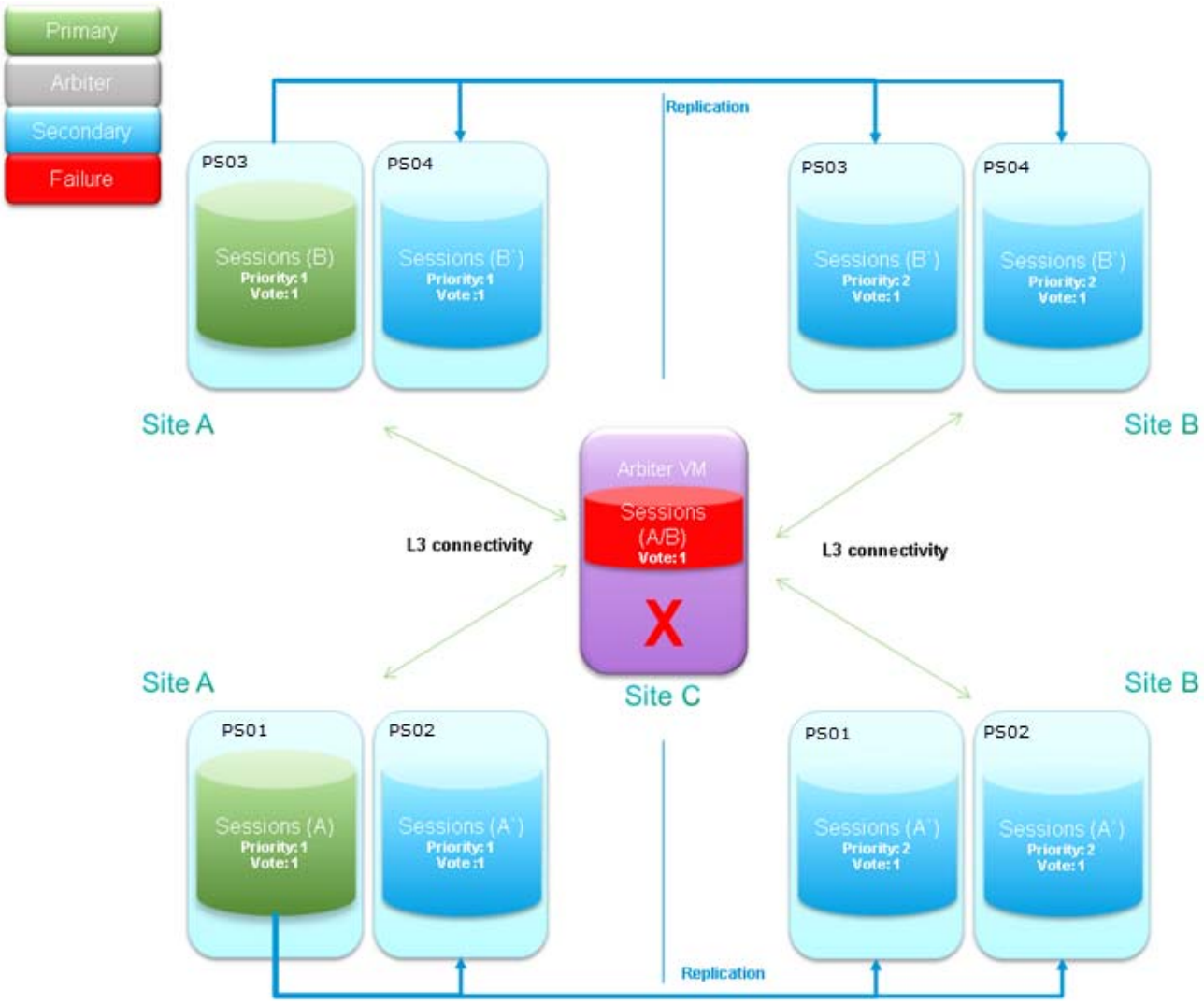
By default, VIP availability is monitored every 60s.



Arbiter Failure

As shown in the figure below, the Arbiter is deployed in a non-redundant manner, as failure of the Arbiter alone does not have any impact on the operation of the Replica Set.

However, a subsequent failure, for example a complete outage of Site A while the Arbiter is down, would result in service interruption as the remaining DB instances would not constitute a majority that would allow the election of a new Master DB..



Failover Triggers and Scenarios

Geo-Redundancy Configuration

First Published: July 17, 2015

Last Updated: July 17, 2015

This chapter covers the following sections:

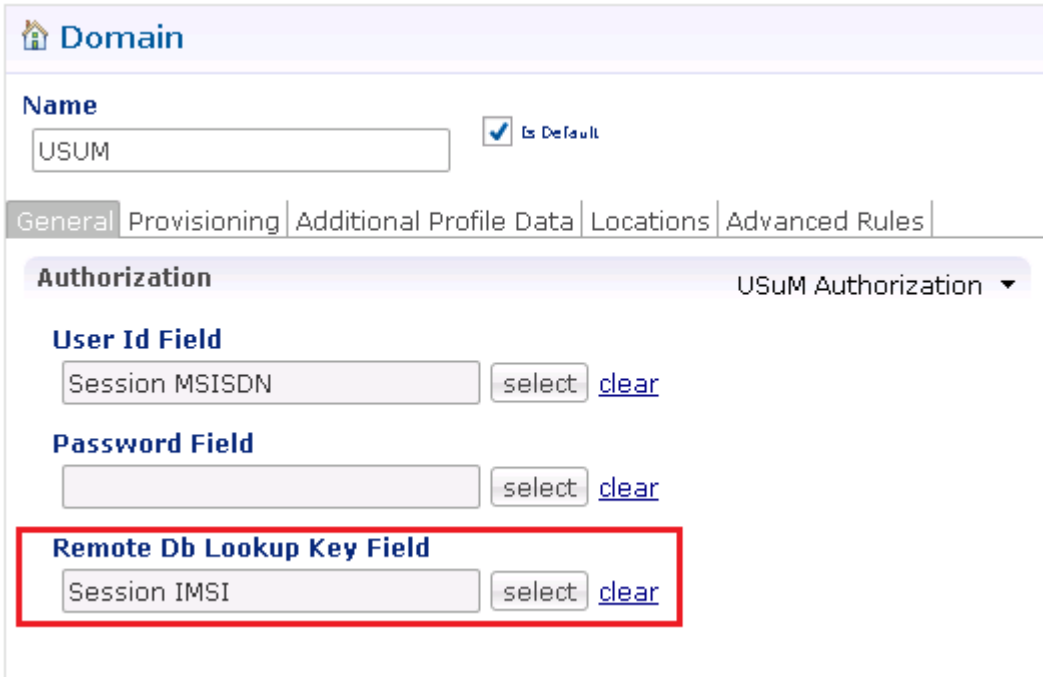
Remote SPR Lookup based on IMSI/MSISDN Prefix

Prerequisites

Policy builder configuration on both the sites should be same.

Configuration

1. Configure the Lookup key field in policy builder under 'Domain'. It can be IMSI, MSISDN, etc. An example configuration is given below:



2. Configure remote databases in policy builder under USuM Configuration.

Consider there are two sites: Site1 (Primary) and Site2 (Secondary). In Policy builder there will be two clusters for Site1 and Site2 in case of Active-Active model.

Under 'Cluster-Site2', create USuM Configuration and add remote databases to be accessed when Site1 is not available.

An example configuration is shown below:

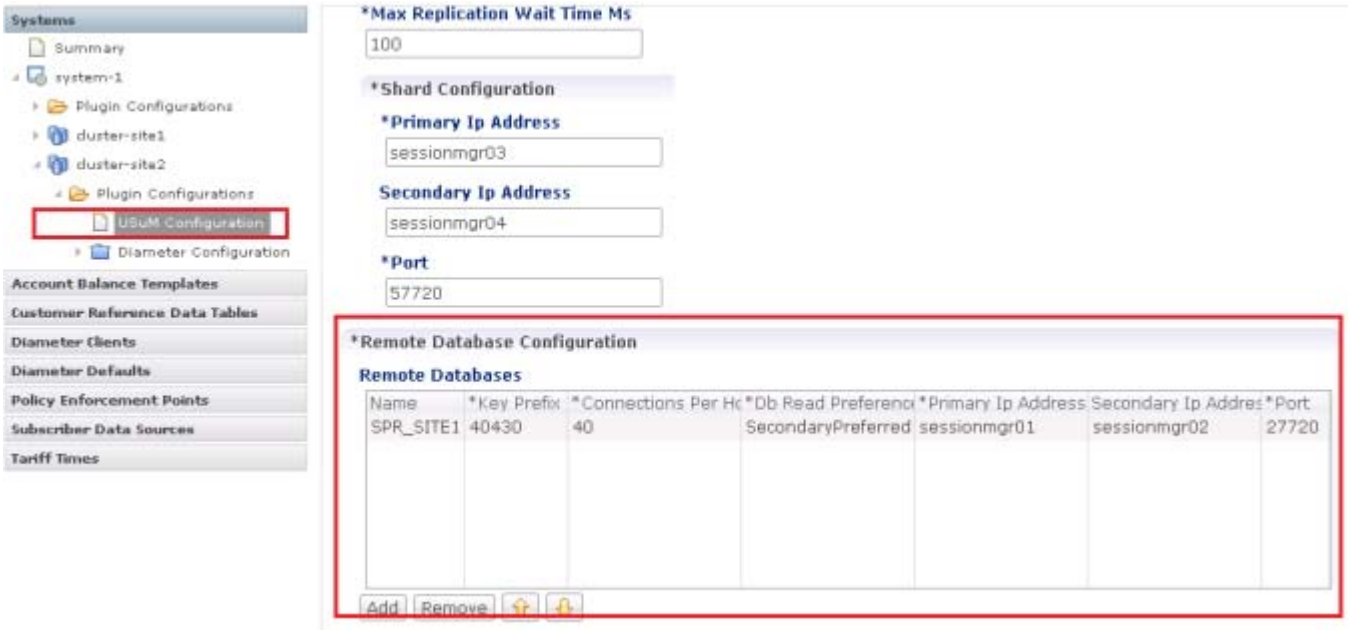


Table 1 Remote Database Configuration Parameters

Parameter	Description
Name	Unique name to identify the remote database.
KeyPrefix	Key prefix to be match for the remote database to be selected for lookup.
Connection per host	Number of connections that can be created per host.
DB Read Preference	Database read preferences.
Primary Ip Address, Secondary Ip Address, Port	Connection parameter to access database. This should be accessible from Site2 irrespective of Site1 is UP or DOWN.

For more information on Remote Database Configuration parameters, refer to *CPS Mobile Configuration Guide* for this release.

SPR Location Identification based on End Point/Listen Port

Prerequisites

Policy builder configuration on both the sites should be same.

Configuration

Consider there are two sites: Site1 (Primary) and Site2 (Secondary).

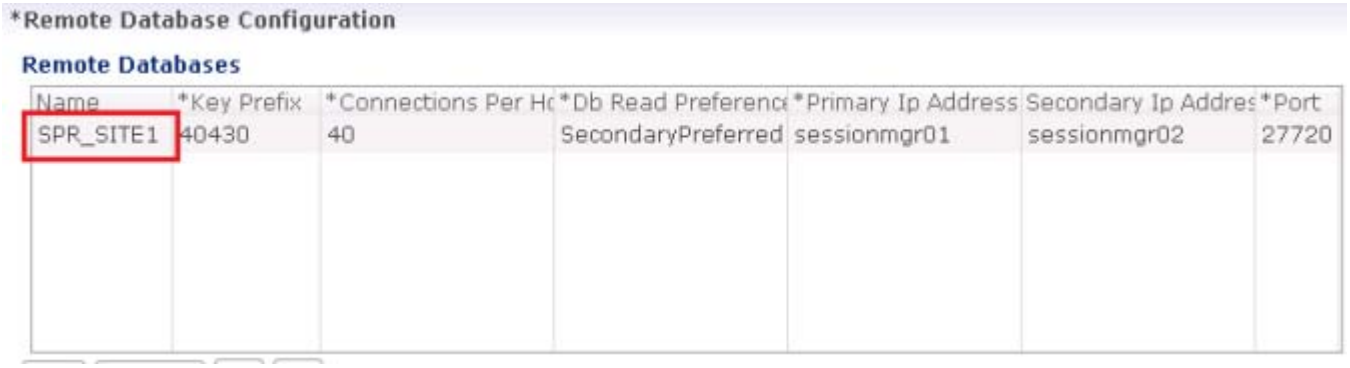
1. Add new entry on Site2 in haproxy.cfg (`/etc/haproxy/haproxy.cfg`) file listening on port 8081 (or any other free port can be used) with custom header “RemoteSprDbName”. Same configuration to be done on both lbs.

```
listen pcrf_a_proxy lbvip01:8081
        mode http
        reqadd RemoteSprDbName:\ SPR_SITE1
        balance roundrobin
        option httpclose
        option abortonclose
        option httpchk GET /ua/soap/KeepAlive
        server qns01_A qns01:8080 check inter 30s
        server qns02_A qns02:8080 check inter 30s
        server qns03_A qns03:8080 check inter 30s
        server qns04_A qns04:8080 check inter 30s
        # If there are more qns add all entries here
```

where,

RemoteSprDbName is the custom header.

SPR_SITE1 is the remote database name configured in 2. on page 38 of Remote SPR Lookup based on IMSI/MSISDN Prefix, page 37. (Refer to screen below):



SPR Query from Standby Restricted to Local Site only (Geo Aware Query)

1. Add new entry on Site1 and Site 2 qns.conf (/etc/broadhop/pcrf/qns.conf) file on pcrfclient01.

```
-DsprLocalGeoSiteTag=Site1
-DsprLocalGeoSiteTag=Site2
```

2. Execute `syncconfig.sh`. To reflect the above change, CPS needs to be restarted.
3. Add tag to spr mongo dbs.

a. First get the replica members by executing the following command:

Execute `rs.conf()` from any one member.

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:37720",
                        "arbiterOnly" : true
                },
```

```
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27720",
                        "priority" : 2
                },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27720",
                        "priority" : 2
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr05-site1:27720",
                        "priority" : 2
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr06-site1:27720",
                        "votes" : 0,
                        "priority" : 2
                },
                {
                        "_id" : 5,
                        "host" : "sessionmgr01-site2:27720"
                },
                {
                        "_id" : 6,
                        "host" : "sessionmgr02-site2:27720"
                },
                {
                        "_id" : 7,
                        "host" : "sessionmgr05-site2:27720"
                },
                {
                        "_id" : 8,
                        "host" : "sessionmgr06-site2:27720",
                        "votes" : 0
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
        }
}
```

a. Now from the list, find out the members of Site1 and Site2 to be tagged (excluding the arbiter). After finding member, execute the following command from Primary member to tag members.

conf = rs.conf();

```
conf.members[1].tags = { "sprLocalGeoSiteTag": "Site1" }
conf.members[2].tags = { "sprLocalGeoSiteTag": "Site1"}
conf.members[3].tags = { "sprLocalGeoSiteTag": "Site1"}
conf.members[4].tags = { "sprLocalGeoSiteTag": "Site1"}
conf.members[5].tags = { "sprLocalGeoSiteTag": "Site2"}
conf.members[6].tags = { "sprLocalGeoSiteTag": "Site2"}
conf.members[7].tags = { "sprLocalGeoSiteTag": "Site2"}
conf.members[8].tags = { "sprLocalGeoSiteTag": "Site2"}
rs.reconfig(conf);
```

THIS IS SAMPLE CONFIG, MEMBERS and TAG can be different as per your environment.

conf.members[1] means member with “_id” = “1” in output of rs.conf();

After executing this command, Primary member can be changed.

Verify tags are properly set by login on any member and executing the following command:

rs.conf();

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:37720",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27720",
                        "priority" : 2,
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27720",
                        "priority" : 2,
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr05-site1:27720",
                        "priority" : 2,
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr06-site1:27720",
                        "votes" : 0,
                        "priority" : 2,
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 5,
                        "host" : "sessionmgr01-site2:27720",
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site2"
                        }
                },
                {
                        "_id" : 6,
                        "host" : "sessionmgr02-site2:27720",
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site2"
                        }
                },
                {
                        "_id" : 7,
```

```
                        "host" : "sessionmgr05-site2:27720",
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site2"
                        }
                },
                {
                        "_id" : 8,
                        "host" : "sessionmgr06-site2:27720",
                        "votes" : 0,
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site2"
                        }
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
        }
}
```

4. Do same process for all other sites. Tag names should be unique for each site.

This change overrides the read preference configured in USuM Configuration in Policy builder.

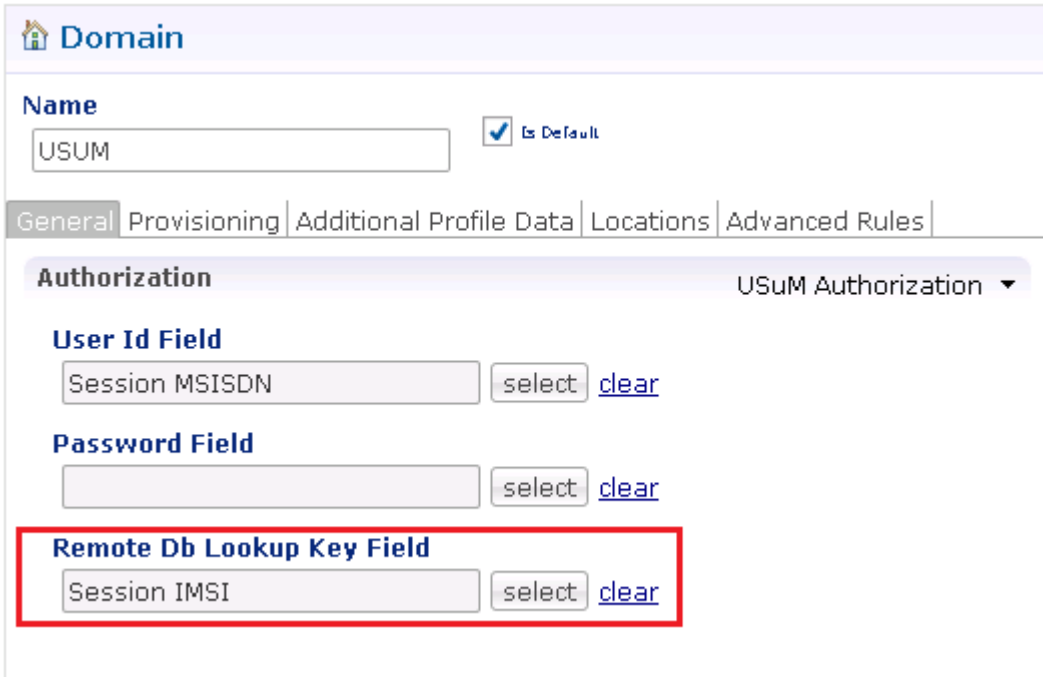
Remote Balance Lookup based on IMSI/MSISDN Prefix

Prerequisites

Policy builder configuration on both the sites should be same.

Configuration

1. Configure the Lookup key field in policy builder under 'Domain'. It can be IMSI, MSISDN, etc. An example configuration is given below:



2. Configure remote databases in policy builder under Balance Configuration.

Consider there are two sites: Site1 (Primary) and Site2 (Secondary). So in Policy builder there will be two clusters for Site1 and Site2.

Under 'Cluster-Site2', create Balance Configuration and add remote databases to be accessed when Site1 is not available.

An example configuration is shown below:

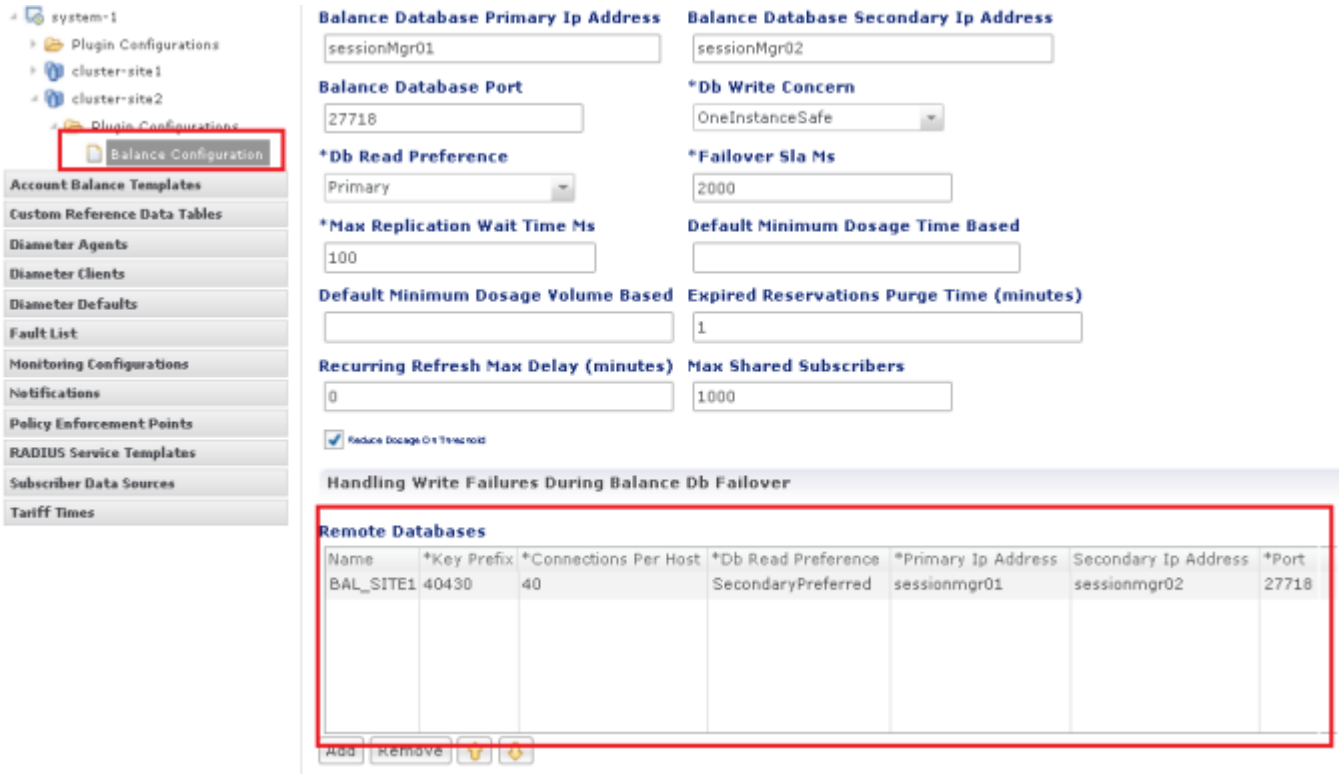


Table 2 Balance Configuration Parameter - GR

Parameter	Description
Name	Unique name to identify the remote database.
KeyPrefix	Key prefix to be match for the remote database to be selected for lookup.
Connection per host	Number of connections that can be created per host.
DB Read Preference	Database read preferences.
Primary Ip Address, Secondary Ip Address, Port	Connection parameter to access database. This should be accessible from Site2 irrespective of Site1 is UP or DOWN.

For more information on Balance Configuration parameters, refer to *CPS Mobile Configuration Guide* for this release.

Balance Location Identification based on End Point/Listen Port

Prerequisites

Policy builder configuration on both the sites should be same.

Configuration

Consider there are two sites: Site1 (Primary) and Site2 (Secondary).

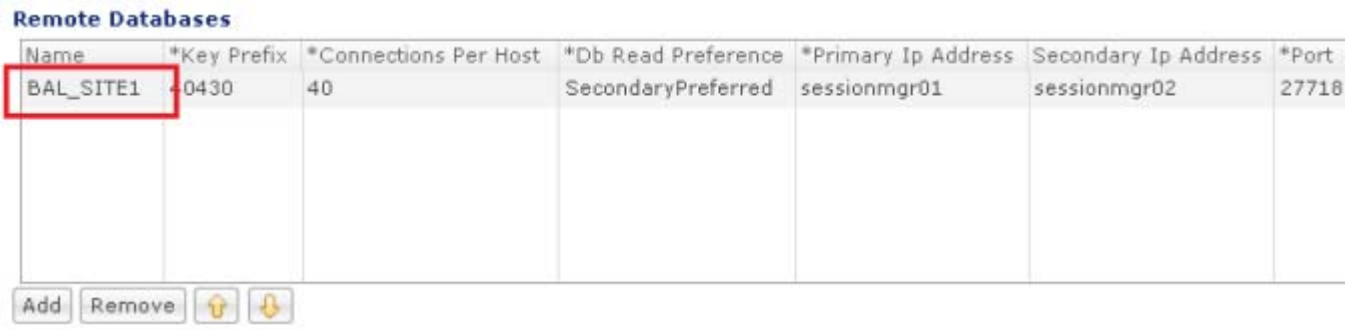
1. Add new entry on Site2 in haproxy.cfg (`/etc/haproxy/haproxy.cfg`) file listening on port 8081 (or any other free port can be used) with custom header “RemoteBalanceDbName”. Same configuration to be done on both lbs.

```
listen pcrf_a_proxy lbvip01:8081
        mode http
        reqadd RemoteBalanceDbName:\ BAL_SITE1
        balance roundrobin
        option httpclose
        option abortonclose
        option httpchk GET /ua/soap/KeepAlive
        server qns01_A qns01:8080 check inter 30s
        server qns02_A qns02:8080 check inter 30s
        server qns03_A qns03:8080 check inter 30s
        server qns04_A qns04:8080 check inter 30s
        # If there are more qns add all entries here
```

where,

RemoteBalanceDbName is the custom header.

BAL_SITE1 is the remote database name configured in 2. on page 44 of Remote Balance Lookup based on IMSI/MSISDN Prefix, page 43. (Refer to screen below):



Balance and Session Query from Standby Restricted Site

The following steps need to be performed for balance and session query from standby restricted to local site only (Geo aware query) during the database failover:

Note: This geo aware query will be in effect only when session/balance database is failovered and backup databases are configured.

In normal case when all databases are up, **Read** preference configured in Policy Builder will be effective.

1. Add new entry in Site1 `qns.conf` file (/etc/broadhop/qns.conf) on Cluster Manager.

```
-DbalanceLocalGeoSiteTag=Site1
-DsessionLocalGeoSiteTag=Site1
```

   a. Run `copytoall.sh` to restart the qns processes.

2. Add new entry on Site2 `qns.conf` file (/etc/broadhop/qns.conf) on Cluster Manager.

```
-DbalanceLocalGeoSiteTag=Site2
-DsessionLocalGeoSiteTag=Site2
```

a. Run `copytoall.sh` to restart the qns processes.

3. Add tag to mongo databases.

   a. First, get the balance replica members

      Execute `rs.conf()` from any one member.

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:27718",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27718",
                        "priority" : 4
   },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27718",
                        "priority" : 3
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr01-site2:27718",
                        "priority" : 2
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr02-site2:27718"
                        "priority" : 1
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
        }
}
```

   b. Now from the list, find out the members of Site1 and Site2 to be tagged (excluding the arbiter).

   c. After finding member, execute the following command from Primary member to tag members.

```
conf = rs.conf();
conf.members[1].tags = { "balanceLocalGeoSiteTag": "Site1" }
conf.members[2].tags = { "balanceLocalGeoSiteTag": "Site1"}
conf.members[3].tags = { "balanceLocalGeoSiteTag": "Site2"}
conf.members[4].tags = { "balanceLocalGeoSiteTag": "Site2"}
rs.reconfig(conf);
```

Note: THIS IS SAMPLE CONFIG, MEMBERS and TAG can be different as per your environment.

```
conf.members[1] means member with "_id" = "1" in output of rs.conf();
```

After executing this command, Primary member may get change if all members have same priority.

Verify that the tags are properly set by login on any member and executing rs.conf() command.

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:27718",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27718",
                        "priority" : 4,
                        "tags" : {
                                "balanceLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27718",
                        "priority" : 3,
                        "tags" : {
                                "balanceLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr01-site2:27718",
                        "priority" : 2,
                        "tags" : {
                                "balanceLocalGeoSiteTag" : "Site2"
                        }
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr01-site2:27718",
                        "priority" : 1,
                        "tags" : {
                                "balanceLocalGeoSiteTag" : "Site2"
                        }
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
        }
}
```

4. Repeat step 3. on page 47 for adding session tags. Tag name for session is “sessionLocalGeoSiteTag”. If session database is sharded, then tagging has to be done on all session replica sets.

Warning: Running `build_set.sh` script will delete all tags configured. Make sure to re-configure the tags, if `build_set.sh` script was executed.

qns.conf Parameters

The following setting should be present only for GR (multi-cluster) CPS deployments:

```
-DclusterFailureDetectionMS=1000
```

Note: In an HA or GR deployment with local chassis redundancy, the following setting should be set to **true**. By default, this is set to false.

```
-Dremote.locking.off
```