Functional Specification for Sip Session Replication

Author(s): erik.van.der.velden@ericsson.com

Version:2.3

1 Introduction

1.1 Scope

Although the title talks about Sip Session replication, the artifacts that are replicated do cover more then just the sessions. The replicated artifacts include:

- The SIP application Session (SAS)
- The SIP (protocol) sessions (SS)
- The dialog information; DialogFragments (DFes).
- The converged HTTP (protocol) sessions (CHS) (not covered directly in this document, but coordinated with the converged HTTP session FSD)

There are relations between these objects; they form a graph of related objects.

The name (Sip) Session Replication, abbreviated as SSR, will be used in this document to cover replication all of the above artifacts Also, we will often be talking about sessions, when the proper nomenclature would be (replication) artifact

1.2 Terminology [informative]

- **Session Availability**
  Also called **session retention.**
  The ability to serve traffic in ongoing sessions (even in the case of a failure).
- **SipSessionReplication (SSR)**
  The name or (Sip) Session replication is used in this document for all the activities surrounding the establishment of session availability.
- **SAS**
  Sip Application Session [jsr289]
- **SAS timer**
  The timer that controls the lifetime of the SAS. When the timer expires the application is informed and within this notification the application can request and extension of the lifetime of the SAS. The container is allowed to reject such an extension.
- **Application Timer**
  The application can create multiple timers which can be periodic or one-shot. These timers are associated with the SAS.
- **SS**
  Sip Session, approximately represents an ongoing SIP dialog [jsr289]
- **DF**
  Dialog fragment. One dialog can be associated with multiple applications via SIP chaining. Sip chaining can be done inside of the container or request can leave the container and re-enter it in the context of the same dialog For each (re-)entry of a dialog

a new DF is created. The DF refers to all the Sip Sessions of all the applications in an internal chain. A DF is uniquely identified by the callid, from-tag, to-tag and fragmentid.

- **DS**
  dialog. A collection of dialog fragments in one SIP session that are allocated to the same JVM, due to spiralling in SIP.
- **Converged HTTP session**
  A HTTP session that is related to a SAS.
- **Non-compromised state**
  The state in which session availability can be guaranteed in case of a single failure. In this state there are two copies of every SSR artifact and these artifacts are on different server instances.
- **Compromised state**
  After some events (e.g., a failure) some copies of artifacts are lost. This leaves the system vulnerable. When a failure occurs in the compromised state this might lead to loss of sessions (provided it happens on the wrong server instance).
- **Repair period**
  The period during which the system is in a compromised state.
- **Repair**
  The act of going from a compromised state to an non-compromised state.
- **Eager repair**
  A way of repairing where the system actively repairs, independent of external events. This will shorten the repair period at the cost of capacity.
- **Lazy repair**
  A way of repairing where the repair is driven by events on the session. These events can either be external events (e.g., a request is received) or internal events (e.g., a time-out). Such a mechanism will lead to a longer repair period, the length of which depends on the traffic and timer activity.
- **Replication**
  The act of making a replica copy, on the replication partner, of an artifact in the active cache.
- **Replication partner (replication destination)**
  All servers are ordered in a ring topology. The replication partner is the server instance that is a neighbor in a clock-wise direction. Replications flow from the active cache on the replication source to the replica cache on the replication destination (the replication partner).
- **Active partner (replication source)**
  All servers are ordered in a ring topology. The active partner is the server instance that is a neighbor in a counter clock-wise direction. Replications flow from the active partner to the replication partner.
- **Eager replication**
  Also called **repair-under-load or forward repair**
  The act of replication of all the artifacts in the active cache.
- **Lazy replication**
  Replication of an active session when the partner did not contain the previous version of the session.
  This is accomplished by normal replication, where the partner did not have the replica copy, e.g., due to a previous failure.

- **Reactivation**
  The act of re-creating an active copy of an artifact on an instance based on a replica copy.
- **Eager reactivation**
  Also called **reverse-repair.**
  The act of copying the all artifacts that are owned by the instance, from replica cache of the replication partner to the active cache of the owning instance.
- **Lazy reactivation**
  Reactivation based on activity in the session (e.g., a request or a time-out).
- **Home instance**
  The instance where the load balancer will route traffic that is targeted to this artifact This is based on the current shape of the cluster.
- **Owner**
  The instance that did the last replication of the artifact is said to be the artifact owner. Not to be confused with the home instance. The home and the owner can be different, due to cluster reshapes.
- **Migration**
  The act of re-creating an active copy of an artifact based on another active copy on another instance.
- **Load request**
  The request to load a session for which a sufficient version is not available in the active cache. This can entail a broadcast in the cluster.
- **AS Upgrade**
  The act of replacing the AS with a newer version of the AS. In case of compatible versions of the AS, this should be done without the loss of sessions.
- **Application Upgrade**
  The act of replacing an application with a newer version of the application. In case of compatible versions of the application, this should be done without the loss of sessions.
- **Rolling Upgrade**
  An upgrade where the instances in the cluster are upgraded one-by-one.
- **Failure**
  A failure of a server instance. This will result in the loss of both the active and the replica cache on the server instance.
- **Recovery**
  The server instance recovers after a failure.
- **Shutdown (downscale)**
  A planned shutdown of a server instance.
- **Restart**
  A planned recovery of a shutdown instance.
- **Start (Upscale)**
  An addition of one or more new server instances to the cluster.
- **Load balancer (LB)**
  An entity that routes the traffic to the instances based on some load balancing policy.
- **(Sticky) Round-Robin (RR) policy**
  This is only applicable to HTTP.
  Initial requests are routed independently of the content of the request. Subsequent requests in the session are routed to the same instance as the initial request.
  If a request can not be routed to the instance (e.g., during a failure), the request is re-

routed to another instance and the affinity is changed, i.e., from then on it remains sticky on the newly selected instance.

- **Data Centric (DC) policy**
  For historical reasons sometimes called User Centric (UC)
  This is applicable for both SIP and converged HTTP.
  Some parameters from the initial request are used to calculate a hash code. The request is routed according to a consistent hashing algorithm that allocates hash codes to the currently available server instances. The hashcode is included in any subsequent requests, all of which are routed based on the same consistent hashing algorithm. However, responses are routed to the same instance as the original request.
- **SIP transaction**
  As defined in [RFC3261]**Properly distributed**
  A situation where every artifact is active on its home instance.
  The advantage of such a situation (if it can be detected) is that every request arriving on an instance can be handled locally, without any load request in the cluster.

A short summary in a table:

	Eager	**Lazy**
Reactivation	Reverse repair From replica (partner) to active (restored) based on restore event (only used for restart in case of upgrade)	From replica (any) to active (home) based on traffic or timer.
Replication	Forward repair / repair-under-load From active (owner) to replica (partner) based on cluster reshape. (only used in restart in case of upgrade)	From active (owner) to replica (partner) based on traffic or timer.
Migration	Not used	From active (owner) to active (home). Based on traffic.

2 Design Overview

2.1 General [informative]

The main design is described in [SAD]. In this document only the high level design decisions are mentioned. For more details see [SAD].

2.2 Design principles [informative]

2.2.1 Lazy repair

After certain events (e.g., a failure or a shutdown) there will exist only one copy of some artifacts in the cluster. This puts the system in a compromised state, where it is vulnerable for events that lead to the loss of data (e.g., another failure, another shutdown). At least, this can happen if these events occur on the ‘wrong’ instance and remove the only remaining copy of the data.
After the repair period, the system again will have two copies of all the artifacts, and should be in a non-compromised state.

There are different strategies to handle such vulnerabilities. The opposite sides of the spectrum are:

- the **eager** strategy will try to ensure that the repair period (the period of vulnerability) is short by doing bulk copying of data to restore the state where there are two copies of every artifact

  The cost of this strategy is in the extra load on the system to repair the 'damage'. In a failure situation, these costs might be too high, this might lead to rejected traffic and to instability, something that should be avoided in such a situation, where the system is already vulnerable.
- The **lazy** strategy relies on activity in the session and only then ensures there are two copies of the data created.

  The disadvantage of this strategy is that the length of the repair period depends on the activities in the session (external traffic and/or time-outs). The period depends on the traffic mix can be quite long. Also, it is difficult to detect when the repair period is finished and we have restored a non-compromised state.
  However, the heuristics for mean time between such events (e.g., mean time between failures) and the average time between activity on the sessions are expected to ensure that the chance of session loss is acceptably low.

  Also, since the repair is done based on external traffic (e.g., requests), in the lazy scenario more requests will have a longer latency then in the eager scenario.

The decision is to normally do a lazy repair, based on the expected heuristics. However, in cases where this is not possible (specifically in the upgrade use case) the eager repair is used.

2.2.2 Healthy system

For an optimal functioning of the system we will try to enforce some state based on a best-effort basis.

- There will be zero or one active copy of a session (avoid duplicates). This is violated only in case of network segmentation or long term network failures.

There are also some nice-to-haves which can probably not be effectively achieved when using asynchronous and non-transactional replication.

- If the active copy is removed, the replica is also removed (no zombies).
- If a replica exists, it has the latest version (freshness).

FFS [informative]:
Because of some optimizations we might want to restore a properly distributed situation and be able to detect that this properly distributed situation is established.
A properly distributed system is characterized by the following:

- every active copy is on the home instance
- there is no replica without an active copy.

Effectively, this means that the load balancer will direct all requests to where the active copy is located, which in turn means that no data has to be fetched elsewhere in the cluster.

Note that a properly distributed situation is different from a non-compromised situation; the first allows active copies without replicas, the latter allows for active instances located at another instance than the home instance.

2.3 Requirements [normative]

The following requirements have been identified:

2.3.1 Configuration

2.3.1.1 Scope of sip session and application session replication

The scope of Sip session replication and Sip Application Session replication is ‘modified-session’ or ‘session’.

2.3.1.2 Scope of sip session and applciation session replication II

The scope 'modified-attribute' does not have to be supported.

If not supported, a configuration that indicates 'modified-attribute' will not be accepted. This means that a WARNING is logged and the configuration reverts to no replication for this application.

2.3.1.3 Frequency of sip session and application session replication

The frequency of Sip session replication and Sip Application Session replication is ‘sip-transaction’.

2.3.1.4 Type of sip session and sip application session replication

The type of Sip Session Replication and Sip Application session replication is 'memory' or 'replicated'.

2.3.1.5 Configuration of the SSR

The scope and frequency are defined in the domain.xml as attributes in the sip-container-availability section.
persistence-type: Specifies the SSR persistence type
persistence-scope: Specifies the SSR persistence scope
persistence-frequency: Specifies the SSR persistence frequency

These settings can be overridden for a specific application in the deployment descriptor of the application (sun-sip.xml)

2.3.1.6 Dynamic configuration of the Sip Session Replication

It is not possible to dynamically change the SIP session replication parameters, either in the domain.xml or in the sun-sip.xml. In order to change the SIP session replication parameters the application must be redeployed.

2.3.1.7 Converged session replication

The HTTP session replication configuration and the SIP session replication configuration must be the same per application.

2.3.1.8 Dialog replication configuration

There is no specific configuration of the dialog session replication; The dialog session replication will always be implicitly scoped as modified-session and have an implicit frequency sip-transaction.

In case of modified-session, the application must invoke setAttribute() after changing any attribute values to trigger replication.

2.3.1.9 Dialog replication activation

Dialog information will only be replicated if all the applications involved in the dialog have session replication enabled.

2.3.1.10Dialog replication disabled

Dialog information will not be replicated if none of the applications involved in the dialog have session replication enabled.

2.3.1.11Dialog replication misconfiguration

Dialog information will not be replicated if some of the applications involved in the dialog have session replication enabled and some do not.
In this case, additionally, a WARNING message will be issued.

The background for this is that replicating only some of the sessions involved in a dialog will most likely lead to errors (e.g,. 481) after failover anyway, so it is cheaper to not even try.

2.3.1.12repair-during-failure configuration

It will be possible to enable eager repair during upgrade, but disable it in other situations. This is implemented as an attribute in the sip-container-availability element of the domain.xml. The attribute will be called repair-during-failure.
The default value will be true, meaning that eager repair will always be done after a cluster reshape and after restarting an instance during an upgrade.
When set to false, eager repair will only be done after restarting an instance during an upgrade.

2.3.1.13Replication of ServletContext

There will be no replication of the ServletContext.
The servlet context is an object that can be accessed on any server instance and the application should not rely on the fact that the servlet context is shared between all instances according to the spec:

```
SRV.14.2.8 ("ServletContext"):

  There is one context per "web application" per Java Virtual
  Machine. (A "web application" is a collection of servlets and content
  installed under a specific subset of the server's URL namespace such
  as /catalog and possibly installed via a .war file.)

  In the case of a web application marked "distributed" in its
  deployment descriptor, there will be one context instance for each
  virtual machine.
  In this situation, the context cannot be used as a location to share
  global information (because the information won't be truly
  global). Use an external resource like a database instead. "
```

2.3.1.14Replication of transactions

The SIP transaction state is not replicated. This has impact on the transaction loss during failures (described below).

2.3.2 Deployment

2.3.2.1 Homogeneous deployment

The SSR requirements are only valid for an homogeneous cluster, where all the server instances are equal in their hardware and software configuration. More specifically, it requires that the application for which SSR is enabled is deployed and available on every instance in the cluster.

2.3.2.2 Memory fill grade

Enough memory headroom must be available; i.e., each instance should have enough room for (n-k)x/2n sessions. Where x is the number of sessions that can be stored in memory (memsize/session size), k is the cluster size, and k is the number of simultaneous faults we would like to handle.

Overload protection can be used to start rejecting incoming traffic in case these memory limits are reached. This will ensure reliability at the cost of throughput loss.

FFS [informative]:
There are some limits that can be set on the number of active sessions in the cache. If this number is reached, no new sessions can be created. This is some form of limitation. However, it would be more graceful to give priority to the active cache over the replica. Then in high memory situations, the capacity would not be influenced, but the reliability would be.

2.3.3 Normal situation

The normal situation is a non-compromised situation. I.e., the starting situation or the situation that is re-established after the repair period in case of failures etc.

2.3.3.1 Impact of asynchronous replication on latency

When asynchronous session replication is enabled the latency should still be in the lower than defined in [PerfReq]

2.3.3.2 Impact of asynchronous replication on throughput

The impact of asynchronous session replication on the throughput is 30% (target) or 50% (exit criteria) [PerfReq]

2.3.3.3 Failure rate during normal operation

The failure rate should be less then 0.01%. [PerfReq].

2.3.3.4 Accuracy of timers

Timers will not be completely accurate even when not in a failure situation.

Background; timer expiry can never be guaranteed exactly in a soft-realtime system. The accuracy depends on the current CPU load and thread usage.

2.3.4 Failure situation

We can distinguish three ’periods’;

- The failure event will have a short (seconds) impact on the traffic.
- After the failure there will be some repair period (we assume lazy re-activation, therefore the repair period depends on the average time between events in a session.

- After the repair period we are back in the normal situation.

2.3.4.1 Session loss during failure with asynchronous replication

It is acceptable to loose up to 0.01% (goal) or up to 0.05% (exit criteria) of the total amount of sessions during a failure. This applies to the testcases as defined in [PerfReq].

Actually it is more subtle then session loss.

There are three types of messages that can get lost:

1. a message that created a replica
   Either the session is really lost and any request directed to it will result in a 481 error response (in case of a SS; In case of SAS, it will ensure that the next SAS allocation using the @SAK will create the SAS)
   Or only a part of the tree is created and other parts are not, leading to inconsistent replica trees.
2. a message that modified a replica
   The replica is stale. See below.
3. a message that deleted a replica
   There is an zombie replica. There might be timer expiry on the zombie replica.

The first two count as session loss.

Because of the lack of version numbers in SIP we can not reliably detect a stale copy of the session, meaning that after a failure we might continue with an old session, without the client or the application being aware of this.

Also, the different objects (Dialogset, sipsession, sipapplicationsession, httpsession) have references to each other, but are replicated separately (i.e., not in one transaction). While the design tries to always keep the related objects co-located, the replication is best effort only. This means that it can happen that there are inconsistencies in the model, e.g., references to objects that no longer exist.

2.3.4.2 Transaction loss during failure

During a failure approximately 1/n-th of the ongoing transactions will be lost.
The reason for this is that the transaction state is not replicated.

2.3.4.3 Throughput loss during repair period after failure event

There will be extra load caused by the lazy repair.

Since the repair is only lazily, the throughput loss is not expected to be significant. The throughput loss (per server instance) should not be more than 5%.

2.3.4.4 Latency during repair period after failure

There will be extra latency (e.g., due to lazy reactivation any request addressing a session which has not yet been addressed after the failure will have an additional delay).

2.3.4.5 Call Failure during the failure event

There may be a traffic loss during the failure event corresponding to maximum 30 seconds out-of-service of one instance. This includes lost transactions and rejects.

After that time the sessions are handled by the remaining server instances in the cluster.

2.3.4.6 Call Failure during repair period after failure

The call failure rate will be below 0.01% [see PerfReq]

2.3.4.7 Timer firing during repair period after failure

After a failure some sessions will only exist on the replica. Timers on these will still expire.

2.3.4.8 Timer delay during repair period after failure

repair period Timers on replicas for which there is no active copy may fire with an additional delay.

2.3.4.9 Timer missed handling for repeating timers

When a repeating timer is fired with a delay, there will be one timer expiry for all the missed intervals. After this the normal schedule is restored.

2.3.4.10 Timer missed handling for single shot timers

When a one-shot timer is fired with a delay, it will fire only once.

2.3.4.11Duplicate timer firing

Barring network segmentation situations, timers will only fire once, even during the repair period.

2.3.4.12Duplicate timer firing in case of network segmentation

In case of network segmentation, timers might fire multiple times.

2.3.4.13Timer firing on invalidated sessions

During the repair period after failure, timers may expire on already invalidated sessions.

This can happen since the replication message indicating the removal from the replica is lost.

2.3.4.14Multiple simultaneous failures

In case of a failure of one of the replica partners during the repair period, there will be an additional loss of sessions. This is because during the recovery there are some sessions for which we have only one replica available.

The assumption is that there is no upper limit on the recovery time. The mean time between failures and the average time between messages in a session, combined with the chance the wrong instance fails (i.e., the replica partner) will guarantee that the risk of session loss is minimal.

2.3.5 Recovery situation

Assumption is that after recovery event (which can either be after a failure or when an instance is restarted after a planned shutdown) there is also a repair period after which the normal situation is restored. The recovery will be based on lazy migration (if the recovery happens after the repair period following the failure or as part of an upscale) . Or it will be based on lazy migration, lazy reactivation and lazy replication(if the recovery happens while we still are in the repair period after a failure).

2.3.5.1 Session loss during recovery event

There should be no session loss during recovery event.

2.3.5.2 Migrations during repair period after recovery event

Recovery or starting of a new instance can lead to the migration of ongoing active session in case of Data Centic load balancing.

In case of recovery, this happens when the session was reactivated during the failure. In case of a start, all traffic on ongoing sessions that are now directed to the started instance will cause a migration.

2.3.5.3 Transaction loss during repair period after recovery event

When the migration happens in during an ongoing transaction the transaction is lost, however, any attribute changes made during the lost transaction will have been replicated. This might lead to unexpected (from the application's point of view) results.

2.3.5.4 Rejections during repair period after recovery

If a migration (to the home instance) is requested when the session is still being accessed on the owner instance, then the migration is rejected with a 503.

2.3.5.5 Throughput during repair period after recovery

There will be a small loss of throughput during the repair period, due to the repair activities.

2.3.5.6 Latency loss during repair period after recovery

During the repair period, the latency of sessions that accessed for the first time since the recovery will increase.

2.3.5.7 Call failures during repair period after recovery

The call failure rate should be less than 0.01% [PerfReq]

2.3.5.8 Failure during repair period

If a failure occurs in the recovery time then there might be a loss of sessions.

Again, the length of the repair period is undefined and depends on the activity in the sessions (based on traffic or timer expiries).

2.3.6 Shutdown

Shutdown is handled the same as a failure. So all the requirements stated for failure apply. E.g., also in this case the repair period can be relatively long. So there is a risk of session loss if multiple servers are shutdown (if they are the wrong servers :-).

Only the requirements that deviate from the failure cases are mentioned explicitly here.

2.3.6.1 quiescence during shutdown

There is no quiescence during shutdown.

2.3.6.2 Unload during shutdown

The unload mechanism, which causes the eager replication to be done before the instance is shutdown, should be optional.

It can be disabled by setting the value max_session_unload_time_in_seconds to null.

2.3.6.3 Session loss during shutdown with async replication

See failure.

FFS [informative]:
We might want to provide some form of quiescence in the future. Ideally, the ongoing SIP transactions should be given a chance to finish. Barring that, at least the instance should get the chance to at least finish processing the ongoing messages and flush the replication buffers, so as to not loose any 'inflight' data.

2.3.6.4 call failure during shutdown

There may be a traffic loss during shutdown corresponding to maximum 15 seconds out-of-service of one instance. This includes lost transactions and rejects.

Planned shutdown activities are reload and disable/enable. After that time other instances must take over and continue all sessions that existed on the instance that was shutdown.

2.3.7 Upgrade

Upgrade will be modeled as a shutdown and a restart. with the exception that there will be no repair period after the shutdown. The repair procedure is started after the restart and the repair will be done eagerly in this case.

2.3.7.1 Session loss during upgrade

See shutdown and recovery. I.e., it is acceptable to lose up to 0.01% (goal) or 0.05% (exit criteria) sessions per instance during an upgrade.

2.3.7.2 Transaction loss during upgrade

15 seconds worth of SIP transactions for each server instance may be lost during an upgrade.

The 15 seconds is taken from ISP definitions where total traffic loss below 15s is not counted for planned activities. This is not total loss but still the most appropriate value. Seen from an individual subscriber this will give the same impact as a total loss for 15s. The 15 seconds must cover both when the instance to be upgraded is taken down and when it is taken back in operation, e.g. when taken back we must include lost transactions on the other instances that have taken over traffic for the upgraded instance.

2.3.7.3 Call failures during upgrade

Call failures should be less than 0.01% [see PerfReq]

2.3.7.4 Failures during upgrade

We will loose sessions when a failure occurs during the upgrade (see failure and recovery).

2.3.7.5 Time of upgrade

Goal: A 15 instance cluster should be able to upgrade in two hours.
Exit criteria: A 10 instance cluster should be able to upgrade in two hours**.**

This time includes the actual upgrade procedure. Upgrade preparations and data schema updates are not part of this time.

2.3.7.6 Latency impact of upgrade

There will be a latency impact during upgrade.

This is due to the migrations and activations occurring while the instance is being upgraded and the repair activities after the instance is restarted.

2.3.7.7 Throughput impact of upgrade

During an upgrade we should be able to handle 50% of the throughput compared to the normal situation with SSR enabled.

2.3.8 SipSessionsUtil

2.3.8.1 Out-of-band access to SipApplicationSession

When using the SipSessionsUtil.getApplicationSession(sessionId) method to request a SIP application session on another instance then the home instance, null is returned as a result.

Ideally, every request from which we access the SipApplicationSession is by the Loadbalancer directed to the instance where the SAS resides. This is the case for Sip Sessions that are associated with a SipApplicationSession via the @SipApplicationKey annotation, for Sip Sessions that are routed based on an encoded URI, for HTTP sessions that are routed based on an encoded URL or for HTTP sessions that are routed according to Data Centric routes (similar to the @SipApplicationKey).

However, the JSR-289 allows access to the SipApplicationSession via the SipSessionsUtil to the application also if the application got the id of the SAS via some out-of-band method. A badly written application could use some mechanism that is not controlled by Data Centric routing (e.g., MDB or remote RMI to a SLSB), ending up on a server instance which is not the home location of the SAS.

2.3.8.2 @SipApplicationKey and Data Centric routing

The @SAK mechanism will only work if it corresponds to the Data Centric routing allocation.

2.3.8.3 @SipApplicationKey and chaining

SSR will only work properly if all the applications in one application chain (I.e., as constructed by the Application Router) have a @SAK that ensures allocation to the same server instance by the Data Centric routing.

2.3.9 Other

2.3.9.1 Design time replication control

SIP session retainability must allow the application to “select at design time” which parts of an object that shall be replicated.

This can be accomplished by the serialization logic implemented by the application. Of the attributes stored in the session, only those parts that are serialized will be replicated.

2.3.9.2 Lazy creation

Sailfin must support lazy creation according to SSA 1.1. A lazy created SS and SAS will not be replicated unless the dialog reached confirmed state.

3 Quality and Availability

It is all about availability in the case of failure. No new failure modes are introduced.

Some of the test and failure scenarios that are of interest.

- normal replication, check for throughput and latency.
    - Use spiralling (to get multiple dialog fragments)
    - Use application chaining (to get multiple Sses in a DF)
    - Use @SipApplicationKey targeting (to get multiple Sses in a SAS)
    - check on create performance of @SAK mechanisms (I.e., use @SAK to create
    - Use application timers
    - Use SAS timer
- failure situation / shutdown
    - checks on zombies, inconsistent trees, cleanup
    - check on session loss
    - check on throughput and latency loss during repair
    - temporary network failures
    - permanent network failures
- recovery / restart / start
    - check on throughput and latency during repair
    - check on session loss
    - check on cleanup
- upgrade
    - check on time for upgrade
    - check on throughput and latency loss during upgrade
    - check on traffic failures during upgrade

4 Performance

The performance requirements document [perfreq] tests some of the requirements mentioned above.

5 Management and Monitoring

No dynamic configuration is available for this function.

FFS:
There is a need for more information regarding the condition of the caches. There is code already in place, but this has never been tested.
In addition to these it would be good to have some indicators of the amount of 'inflight' data.

6 Formal Interfaces

The configuration of the SSR is in-line with the HTTP replication configuration. See requirements.

A new configuration attributes is provided to configure the repair-under-load and reverse repair behaviour:

- *repair-during-failure* to enable to disable any eager repair actions during other situations than upgrade.

7 Packaging, Files, and Location

The code for SSR will be located in packages:

* org.jvnet.glassfish.comms.replication

8 Documentation Requirements

At least the following documentation should be provided:

- SIP Servlet Developers Guide
  Should contain a section describing how to program servlets taking into account SSR. E.g., rules to not store container objects in attributes etc.
- Upgrade guide
  Describing the procedures for upgrade, e.g., how to achieve some sort of quiescence using administrative procedures.

9 References

[sad]	*System Architecture Description for SSR*
[PerfReq]	*Performance requirements for Sailfin*

10 Open Issues