Network Working Group
Internet-Draft
Expires: August 25, 2021

J. Rosenberg
Five9
C. Jennings
Cisco
T. Asveren
Ribbon Communications
February 21, 2021

SIP Extensions for High Availability and Load Balancing for Public Cloud
draft-rosenberg-dispatch-cloudsip-00

Abstract

Software making use of the Session Initiation Protocol (SIP) faces challenges in achieving high availability, especially for call stateful applications like softswitches, Session Border Controllers (SBCs), and IP-based call centers applications. The state maintained in the SIP, SDP and SRTP layers changes frequently, and is difficult to replicate. For this reason, commercial systems have often relied on complex active-standby configurations making use of IP address takeover. These solutions are also ill-suited for usage in modern public cloud environments. This document defines a SIP extension facilitating HA, including keeping calls active, which is optimized for server-to-server communication where one or both sides are in public cloud.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 25, 2021.

Table of Contents

1. Introduction

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14

[RFC2119] and indicate requirement levels for compliant CoAP implementations.

Software making use of the Session Initiation Protocol (SIP) [RFC3261] faces challenges in achieving high availability, especially for call stateful applications like softswitches, Session Border Controllers (SBCs), and IP-based call centers applications. The state maintained in the SIP, Session Description Protocol (SDP) Offer/Answer [RFC3264] and Secure Real Time Transport Protocol (SRTP) [RFC3711] layers changes frequently, and is difficult to replicate. For this reason, commercial systems have often relied on complex active-standby configurations making use of IP address takeover. These solutions are also ill-suited for usage in modern public cloud environments. SIP assumed server-side components would not maintain call state, and thus it never had built-in mechanisms to facilitate server side HA. In practice, the vast majority of server deployments are B2BUAs and maintain call state.

Besides the challenges in replicating call state, SIP also struggles in achieving HA in modern cloud deployments making use of elastic compute. In these environments, the underlying cloud platform (such as kubernetes), can automatically add and remove instances to a cluster based on usage. Similarly, they will remove elements from the cluster which fail health checks. This information needs to propagate quickly to upstream elements, in order to avoid sending calls to failed or overloaded instances. SIP envisioned that a DNS-based solution using SRV records, [RFC3263] would be sufficient. However, DNS changes are slow to propagate and unpredictable. Commercial implementations have made use of SIP OPTIONS probing to assess liveliness, without standardized behavior. There is also no standardized way to communicate or update the IP addresses used in a cluster of servers.

This specification seeks to remedy these gaps. It defines a simple SIP extension, which is largely a definition of mandatory behaviors for SIP elements, that enable rapid detection and recovery from a failed instance while ensuring that calls do not drop. It also defines a small protocol for retrieving and pushing the set of instances in a cluster so support elastic expansion and contraction of a cluster in a fully automated fashion.

2. Applicability

This extension is focused on server-to-server use cases, where one or both sides are a cluster of servers deployed in a public cloud environment. Examples of these situations include SIP trunks between a PSTN carrier and an enterprise, a PSTN carrier and a VOIP provider (such as a cloud contact center), or between VOIP providers providing

peering. The extension also assumes usage in bilateral peering arrangements, and as such, provides no mechanism for discovery. Rather, it assumes both sides have agreed to use this extension as part of configuration provided through techniques outside the scope of this specification.

3. Requirements

- The solution must enable a call to be recovered in less than 2 seconds. This time represents the amount of time before which a user would hangup because they cannot hear the other party.

- A recovered call means that media continues to flow, and future signaling for features or call hangup, can be performed

- The HA technique must not require servers in the cluster to replicate any SIP/SDP/RTP state beyond the dialog identifiers for calls

- The solution should minimize the changes required to the SIP and RTP protocols and their respective implementations

- The solution must support the case where the telco is using traditional SBCs and is not deploying kubernetes or using public cloud

- The solution must enable fully automated elastic expansion and contraction of clusters

- The solution must support availability, so that when an instance in a cluster fails, new calls are distributed across the remaining N instances

- The solution must support availability, so that when an instance of a cluster fails, all of the active calls that were being handled by that instance are spread across the remaining nodes in the cluster, within 2 seconds

- The solution must support clusters wherein each instance of a cluster has a differing amount of capacity for call handling

- The solution must support the ability for instances of a cluster to gracefully shut down without dropping calls

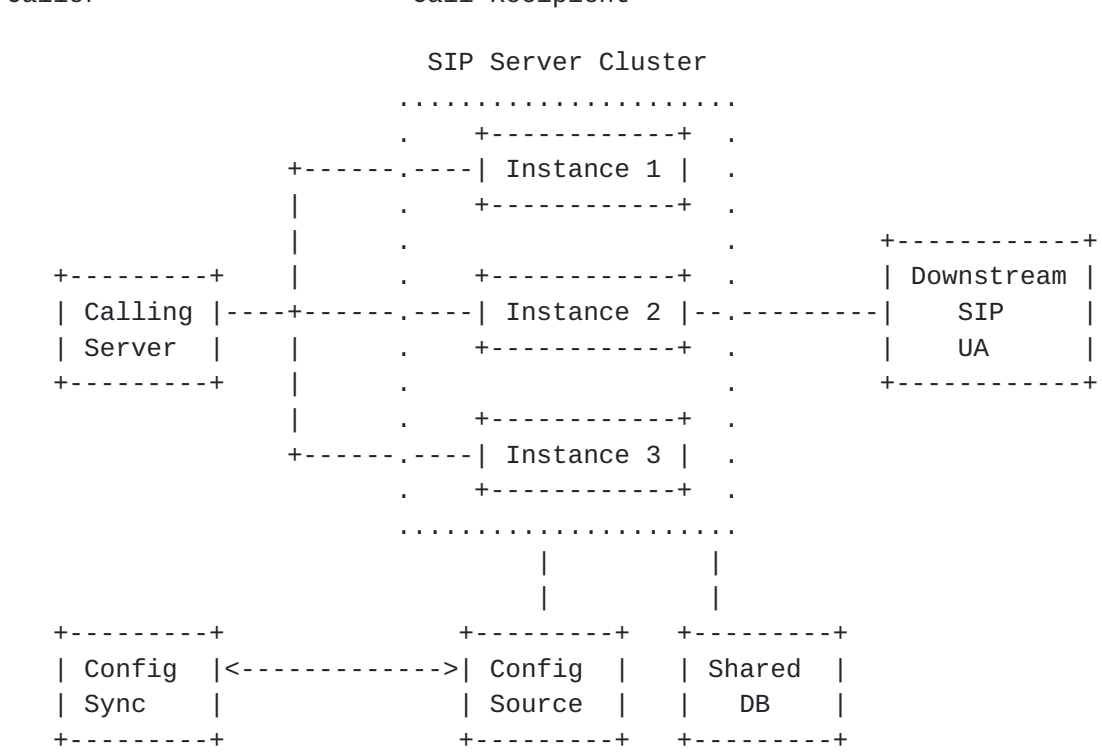
4. Relationship to RIPT

This protocol is similar in goals to RIPT - enabling SIP servers to run in public cloud environments, and achieve HA through techniques employed by web applications. RIPT attempted to solve this problem by utilizing HTTP/3 and fully redefining SIP, repairing many of its problems in the process. This specification is less ambitious, focusing on the minimum changes to SIP required to facilitate HA.

As such, this specification does not alleviate the value in a full-fledged replacement for SIP.

5. Reference Architecture

Cloud SIP uses an assymetric relationship between peers. One side acts as the caller, and the other as the call recipient. New SIP calls can only be placed by the caller, not by the call recipient. If a deployment requires calls to flow in both directions, each side acts as both caller and call recipient.



The calling server wishes to send calls to a cluster, which has a set of instances. The calling server is a B2BUA, and is capable of initiating calls, typically in response to an upstream INVITE it receives. The calling server itself may be a member of a cluster.

When the calling server wishes to generate a new INVITE for a new call, it load balances them amongst the instances in the cluster. Consider a specific call that was sent to instance 2, and was then forwarded through zero or more SIP proxies (not shown) before landing at a UA, referred to here as the downstream SIP UA. The downstream SIP UA may itself be another B2BUA that is a member of a cluster, or even be an end user client. This specification requires the downstream UA to implement the SIP Replaces header field [RFC3891].

When instance 2 fails, we wish to have the call taken over by one of the other instances in the cluster, which can then re-establish media with the downstream SIP UA using INVITE/Replaces.

There is a logical function associated with the cluster, called the config source, which is aware of the configuration of the cluster. Specifically, it knows the IP/port of each instance, and whether that instance is healthy. This config source learns this information through non-standardized means, unique to the cloud environment in which the cluster resides. The config source communicates that information to a config sync associated with the upstream calling server. This communication is bidirectional, using HTTP requests. The config sync distributes this information to the calling server (and any other calling servers should they themselves be a cluster).

There is a shared database of some sorts, accessible by all instances in the cluster. This is used to store the dialog state needed for operation of this extension.

6. Solution Applicability

This specification is applicable in two scenarios:

1. The calling server (and other members in its cluster) and the config sync service are controlled by one entity, and the cluster and the downstream UA are controlled by a second. A common example of this is where the calling server and config sync are part of a telecom carrier, and the cluster and downstream UA are part of an enterprise or SaaS provider that has purchased SIP trunking services from the carrier.

2. The calling server, cluster, and downstream UA are all controlled by a single administrative entity.

3. The instances which make up the cluster are assumed to be provided by the same vendor. This allows for vendor-specific solutions to replicate state and messaging as required by this specification.

This specification also requires that the calling server be a UA (including B2BUAs), and that the instances in the cluster are B2BUAs and the downstream UA is under the administrative control of the same entity that operates the cluster.

7. Overview of Solution

The solution is pretty straightforward.

The calling server will maintain, through the HTTP-based protocol described below, a list of instances in the cluster. These instances are identified by both IP and port. The inclusion of a port allows the instances to share a common IP but vary by port. Such a configuration is useful inside of public cloud environments which can be fronted by a network load balancer which allows each instance to actually have the same IP, but utilize different ports.

The calling server continuously validates that each instance in the cluster is alive, every 250ms. New calls are delivered only to instances which are healthy based on the algorithm defined here. It can ascertain health via reverse RTP traffic, rapid RTCP receiver reports, or via SIP OPTIONS. If SIP OPTIONS are used, these are performed at a rate of a new transaction every 250ms. This is very fast, but it is critical for rapid detection of failures.

If the calling server is itself a member of a cluster, the work of ascertaining the health of each instance can be distributed across the calling servers, in order to avoid a full-mesh of OPTION probing, and then the resulting state distributed through means outside of this specification.

When a call is initially established - to instance 2 in this case - instance 2 will place an entry into the database which contains three pieces of information - (1) the dialogID of the SIP leg from the caller to itself, (2) the dialog ID of the downstream SIP leg from itself to the downstream SIP UA, (3) the IP address and port of the downstream SIP UA.

If an instance transitions from healthy to unhealthy, the calling server 'moves' the existing instance 2 calls uniformly across to the remaining healthy instances in the cluster. To avoid a flood of instant traffic, it moves these calls over a window of at least 200ms and at most one second. To move the calls, the calling server sends an INVITE w. Replaces header field for each such call. Because this is a fresh SIP dialog, a new SDP offer/answer and SRTP is established. This is what avoids the need for replication of RTP state, SDP state and other lower-layer states across the instances in the cluster. When the INVITE/Replaces arrives at one of the

instances in the cluster (say, instance 3), the instance takes the dialogID in the Replaces header field, and looks it up in the shared DB. It will find that there is a matching dialog, and it will retrieve the outbound dialogID and downstream SIP UA. Instance 3 sends an INVITE/Replaces to the downstream UA, using the dialogID it retrieved from the database.

Establishment of a new SIP dialog between the calling server and instance 3 can take place in parallel with the establishment of the new dialog between instance 3 and the downstream UA. Thus the time required to failover the live call is equal to the time to detect instance failure, plus the time to establish a new SIP call.

8. Configuration

A cloud SIP "trunk" is configured in the config sync service through means outside of the scope of this specification. Each such trunk is defined by an HTTPS URI - the trunk config URI - which points to the config source service representing that cluster. This is the only configuration required to establish a cloud SIP trunk.

Once configured with this URI, the config sync MUST perform a GET against this URI. The config source MUST return a JSON document conformant to the schema defined in this specification. This document MUST provide the config sync with a list of instances, each with IP address and port. The JSON document MUST also contain a cluster name, formatted as a hostname, and a webhook registration URI.

Once retrieved, the config sync MUST perform a POST against the webhook registration URI. The POST MUST contain a JSON document conformant to the schema defined in this specification. That document MUST contain an HTTPS webhook URI used by the config sync to receive webhook callbacks that push an updated cluster configuration from the config source.

The config sync MUST refresh its webhook registration at least once a day to ensure that an up to date value for the webhook URI exists.

The config source MUST perform a POST against the webhook URI whenever the cluster configuration changes, including when it detects, on its own, that an instance is unhealthy, removing it from the list.

9. SIP Behavioral Requirements

9.1. Calling Server

9.1.1. Health Probing

The calling server MUST be capable of detecting the failure of any instance of the cluster within 1.5 + RTT seconds. The specific means for doing this detection can vary by implementation. It is also expected that some implementations may have failure detection computed from one instance of a calling server, and the resulting state shared with other instances of the calling server through some means outside the scope of this specification.

One suggested technique for detecting failure is to utilize a SIP OPTIONS probe. The OPTIONS request can be sent every 250ms, directed to each instance of the cluster. To facilitate high scale and determination of RTT, a single OPTIONS request can be sent for each transaction (since retransmits are largely useless due to the short timeout defined for this use case). With such an interval, the calling server can consider an instance unhealthy at time T if, at time T, zero OPTIONS responses have been received for a time equal to the RTT to the instance plus 6 * 250ms = 1.5s. The calling server can maintain the RTT in any fashion it desires. If the OPTIONS requests for a specific transaction are not retransmitted, the time between transmission of the request and receipt of the response can be used to measure RTT.

A MUST strength for 1.5s + RTT is specified to ensure that the cluster can count on consistent and predictable behavior from the upstream calling server. An instance is considered healthy if it is not unhealthy.

OPEN ISSUE: SHould we put a Require header field in the OPTIONS? Should we specify any other behaviors in the OPTIONS?

9.1.2. Utilization Measurement

The instances can place a SIP header, Instance-Utilization, into all responses sent to the calling server. These values indicate the utilization of that instance, as an integral value from 0 to 100. They are used by the calling server to weight the traffic in proportion to utilization.

The calling server MUST remove this header field before propagating it in any upstream responses, as they only have significance on the link between the calling server and cluster.

If this header field is present in a response, the calling server MUST remember the most recent value received from that instance (ordered by the wall clock time at which the response is received). The calling server MUST NOT utilize the source IP of the response to identify the instance. Instead, it MUST correlate the response to a request, and remember the instance to which the request was sent.

If no value has been received for 5 seconds, or no value was ever received, the default value of 50 MUST be used as the utilization.

9.1.3. New Call Initiation

The calling server MUST NOT place a new SIP call to an instance in the cluster which is unhealthy at the time the call is to be placed.

The calling server MUST select an instance for the call using a random function across the instances which are healthy. The calling server MUST weight the probability of selecting that instance in proportion to (100 - the utilization of that instance) . It MUST then direct the call to this instance, by sending the SIP INVITE to the IP address and port of this instance.

As an example, if a cluster has three instances with utilizations at 50, 75 and 100, and all three instances are healthy, no INVITEs are sent to the third instance, 66% are sent to instance one, and 33% are sent to instance 2. Note that, in this case, since instance 3 is not handling new calls, further utilization values can only be learned via responses to OPTION pings, which the calling server MUST send for instances with over 90% utilization.

9.1.4. Instance Failure

When the calling server detects the failure of an instance, it MUST identify all calls which are still active, which were sent to that instance. For each such call, it MUST select a new instance for that call, by choosing one using a uniformly distributed random function amongst the healthy instances. The calling server MUST generate a new INVITE (not a re-INVITE), establishing a new SIP dialog. This INVITE MUST contain a Replaces header field. The Replaces header field MUST contain the dialogID of the call which is being failed over. The INVITE requests MUST be sent uniformly across a 500ms window of time.

9.1.5. Instance to Inactive

If the config sync receives an updated configuration file, and one of the instances from the cluster has been marked as inactive, the calling server MUST NOT send new calls to that instance. However, it

MUST keep existing calls up, and MUST continue to send OPTIONS probes to that instance.

9.1.6. Receiving a REFER

If the calling server receives a REFER request, and the Refer-To URI has a domain portion equal to the IP address of a cluster instance or the FQDN of the cluster, and the Refer-To URI contains an embedded Replaces header field containing a dialogID of a call managed by the calling server, then this REFER is meant to trigger a movement of the call.

The calling server MUST authenticate that this request came from an instance in the cluster. The request is authorized if the domain portion of the Refer-To URI contains an IP address of a cluster instance, or the FQDN of the instance. Furthermore the dialogID in the embedded Refer-To header field matches a dialog that is in progress to that cluster.

If the domain portion of the URI contains an IP address, the calling server MUST perform the requested INVITE/Replaces to that cluster instance. If the domain portion contains the FQDN of the cluster, the calling server MUST send the INVITE/Replacs to one of the other cluster instances, besides the one to which the dialog is currently connected. It MUST select amongst the other instances as if the currently connected instance were inactive, and then round robin using the utilization measures for the remaining instances.

TODO: better explanation, more details

9.2. Cluster Instances

9.2.1. Sending Utilization Values

It is RECOMMENDED that if any one instance of a cluster send values for Cluster-Utilization, all instances do. If none send it, calls will be uniformly balanced across the cluster. Thus, the usage of this header field is only meant for cases where uniform load balancing will not produce uniform utilization.

If an instance is configured to send utilization, it MUST place an Instance-Utilization header field in all responses it sends to all transactions, and include its current measure of utilization. The utilization measure MUST be an integer between 0 and 100 inclusive. Since absolute ordering of responses cannot be guaranteed, the measure SHOULD NOT change more frequently than once a second.

9.2.2. Receiving INVITE w. Replaces

If an instance in the cluster receives an INVITE for a call, and that call has a Replaces header field containing a dialogID for a call that the instance knows is in progress within the cluster, it will know that this is a failover call. It may happen that the failover call is one being handled by the instance receiving the INVITE with Replaces. This is a race condition, but in this case the instance MUST still follow the procedures defined here.

If this is a failover call, the instance MUST authenticate that the INVITE came from the upstream calling server.

There may be cases where the cluster instance receives an INVITE with Replaces header field, but the dialogID does not match a dialog known to the cluster. In such a case, the INVITE MUST be treated as a normal INVITE with a Replaces header field as defined by [RFC3891]. In many cases this may be propagated downstream, or challenged for credentials, neither of which are done if the dialogID is a match for a dialog known to the cluster.

Any downstream SIP dialogs associated with the call MUST be sent an INVITE with Replaces, moving the call to this instance. This will necessarily require the cluster to store the dialogIDs for all dialogs in and out of the cluster, along with any application state needed to reconstruct the dialogs at a new instance.

9.2.3. Graceful Shutdown with Migration

In cases where an instance in the cluster wishes to shut down quickly (perhaps to facilitate a rolling upgrade across the cluster), it can do so by ceasing to respond to OPTIONS requests targeted to itself. The upstream caller will see this as a failure, and move all of the calls off of the instance, onto the remaining instances in the cluster. When the instance reboots, it will begin responding to the OPTIONS probes, enabling it to begin to receive new calls.

9.2.4. Graceful Shutdown without Migration

Another common use case for graceful restart is to cease accepting new calls, but to allow the calls in progress to complete. Once all of the calls have completed, the instance can shut down and restart if desired.

To accomplish this, the cluster config service will mark the instance as inactive in the config file, and pass the updated file to the config sync via webhook. This will cause the calling server to stop

sending new calls to the instance. However, calls in progress will not be dropped.

9.3. Moving a Dialog

Another common case is that an instance is overloaded and wishes to shed a few calls. To facilitate this, a cluster instance MAY send a REFER to the calling server, requesting it to send an INVITE with a Replaces header field. The Refer-To header field embedded in the Refer-To URI MUST contain the dialogID of the call from the calling server to that instance, which is to be moved. To move the call to a specific other instance in the cluster, the domain portion of the URI is set to be equal to the IP address of that instance. Note that the calling server will validate that this IP address is another member of the cluster before authorizing the REFER. Alternatively, the REFER can request the calling server to send the call to any one of the other instances in the cluster, not including itself. To do that, it sets the domain portion of the SIP URI equal to the cluster FQDN.

TODO: Probably need examples and some more details on in or out of dialog REFER

10. Cloud SIP Trunk Configuration File

Something like:

```
{
    "cloud-sip-trunk-name" : "trunk32.acme.com",
    "uri" : "https://configs.sip.acme.com/trunk32",
    "version": 23,
    "webhook-registration" : "https://webhooks.sip.acme.com/trunk32",

    "instances" : [
        {
            "IP" : "1.2.3.4",
            "port" : "5061",
            "status" : "active"
        },
        {
            "IP" : "1.2.3.7",
            "port" : "5061",
            "status" : "inactive"
        }

    ]
}
```

11. Webhook Registration Object

Something like:

```
{
    "webhook" : "https://webhook-receipt.sip.acme.com"
}
```

12. Instance-Utilization Header Field

Something like:

```
{
Instance-Utilization: 34
}
```

IANA registration and formal syntax TBD.

13. Why not DNS

The usage of DNS - and specifically [RFC3263] - might appear to be an alternative to the mechanism in this specification for communicating the IP addresses for the instances of the cluster. However, DNS does not meet the requirements outlined above.

Firstly, DNS is not fast enough to be responsive to the need to add or remove an instance from the cluster. Changes in DNS can take time to propagate. At the time [RFC3263] was conceived, the notion of elastic (and automated) expansion and contraction of clusters did not exist. Cluster instance IPs were extremely static and therefore DNS was sufficient. This is no longer the case.

Secondly, DNS cannot convey state - in particular, information about whether the cluster instances are active or inactive. This is needed to facilitate graceful shutdown of instances. [RFC3263] did not have to concern itself with this problem, because at the time it was believed SIP servers would not contain call state, and therefore, we would not need to worry about this problem.

In addition, because we need to failover extremely quickly - in under two seconds - the calling server needs to perform rapid health probing against all instances in the cluster. This requires the calling server to know all of the IP addresses of the all the instances in the cluster. Typically, DNS queries for an FQDN return one or perhaps a handful of A records, and not every single A record. We expect this specification to be used with clusters that have

instances counts in the hundreds, which is wholly inappropriate to convey via DNS.

14. TODO

Reconcile this with draft-kinamdar-dispatch-sip-audo-peer.

15. Informative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, DOI 10.17487/RFC3263, June 2002, <https://www.rfc-editor.org/info/rfc3263>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <https://www.rfc-editor.org/info/rfc3264>.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <https://www.rfc-editor.org/info/rfc3711>.

[RFC3891] Mahy, R., Biggs, B., and R. Dean, "The Session Initiation Protocol (SIP) "Replaces" Header", RFC 3891, DOI 10.17487/RFC3891, September 2004, <https://www.rfc-editor.org/info/rfc3891>.

Authors' Addresses

Jonathan Rosenberg
Five9

Email: jdrosen@jdrosen.net

Cullen Jennings
Cisco

Email: fluffy@cisco.com

Tolga Asveren
Ribbon Communications

Email: tasveren@rbbn.com