Network Working Group
Internet-Draft
Updates: 3261,3327
(if approved)
Intended status: Standards Track
Expires: December 27, 2007

C. Jennings, Ed.
Cisco Systems
R. Mahy, Ed.
Plantronics

June 25, 2007

Managing Client Initiated Connections in the Session Initiation Protocol (SIP)

draft-ietf-sip-outbound-09

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 27, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

The Session Initiation Protocol (SIP) allows proxy servers to initiate TCP connections and send asynchronous UDP datagrams to User Agents in order to deliver requests. However, many practical considerations, such as the existence of firewalls and Network Address Translators (NATs), prevent servers from connecting to User

Agents in this way. This specification defines behaviors for User Agents, registrars and proxy servers that allow requests to be delivered on existing connections established by the User Agent. It also defines keep alive behaviors needed to keep NAT bindings open and specifies the usage of multiple connections.

Table of Contents

1. Introduction 4
2. Conventions and Terminology 4
2.1. Definitions 5
3. Overview 5
3.1. Summary of Mechanism 5
3.2. Single Registrar and UA 6
3.3. Multiple Connections from a User Agent 7
3.4. Edge Proxies 9
3.5. Keepalive Technique 11
3.5.1. CRLF Keepalive Technique 11
3.5.2. TCP Keepalive Technique 12
3.5.3. STUN Keepalive Technique 12
4. User Agent Mechanisms 13
4.1. Instance ID Creation 13
4.2. Registrations 14
4.2.1. Registration by Other Instances 16
4.3. Sending Requests 16
4.4. Detecting Flow Failure 17
4.4.1. Keepalive with TCP KEEPALIVE 18
4.4.2. Keepalive with CRLF 18
4.4.3. Keepalive with STUN 18
4.5. Flow Recovery 19
5. Edge Proxy Mechanisms 20
5.1. Processing Register Requests 20
5.2. Generating Flow Tokens 20
5.3. Forwarding Requests 21
5.4. Edge Proxy Keepalive Handling 22
6. Registrar Mechanisms: Processing REGISTER Requests 22
7. Authoritative Proxy Mechanisms: Forwarding Requests 24
8. STUN Keepalive Processing 24
8.1. Use with Sigcomp 26
9. Example Message Flow 26
10. Grammar 30
11. Definition of 430 Flow Failed response code 30
12. IANA Considerations 30
12.1. Contact Header Field 30
12.2. SIP/SIPS URI Parameters 31
12.3. SIP Option Tag 31
12.4. Response Code 31

12.5. Media Feature Tag 31
13. Security Considerations 32
14. Operational Notes on Transports 33
15. Requirements 34
16. Changes 34
16.1. Changes from 08 Version 34
16.2. Changes from 07 Version 34
16.3. Changes from 06 Version 35
16.4. Changes from 05 Version 35
16.5. Changes from 04 Version 35
16.6. Changes from 03 Version 36
16.7. Changes from 02 Version 37
16.8. Changes from 01 Version 37
16.9. Changes from 00 Version 38
17. Acknowledgments 38
Appendix A. Default Flow Registration Backoff Times 38
18. References 39
18.1. Normative References 39
18.2. Informative References 40
Authors' Addresses 41
Intellectual Property and Copyright Statements 42

1. Introduction

There are many environments for SIP [1] deployments in which the User Agent (UA) can form a connection to a Registrar or Proxy but in which connections in the reverse direction to the UA are not possible. This can happen for several reasons. Connections to the UA can be blocked by a firewall device between the UA and the proxy or registrar, which will only allow new connections in the direction of the UA to the Proxy. Similarly a NAT could be present, which is only capable of allowing new connections from the private address side to the public side. This specification allows SIP registration when the UA is behind such a firewall or NAT.

Most IP phones and personal computers get their network configurations dynamically via a protocol such as DHCP (Dynamic Host Configuration Protocol). These systems typically do not have a useful name in the Domain Name System (DNS), and they almost never have a long-term, stable DNS name that is appropriate for use in the subjectAltName of a certificate, as required by [1]. However, these systems can still act as a TLS client and form connections to a proxy or registrar which authenticates with a server certificate. The server can authenticate the UA using a shared secret in a digest challenge over that TLS connection.

The key idea of this specification is that when a UA sends a REGISTER request, the proxy can later use this same network "flow"--whether this is a bidirectional stream of UDP datagrams, a TCP connection, or an analogous concept of another transport protocol--to forward any requests that need to go to this UA. For a UA to receive incoming requests, the UA has to connect to a server. Since the server can't connect to the UA, the UA has to make sure that a flow is always active. This requires the UA to detect when a flow fails. Since such detection takes time and leaves a window of opportunity for missed incoming requests, this mechanism allows the UA to use multiple flows to the proxy or registrar. This specification also defines multiple keepalive schemes. The keepalive mechanism is used to keep NAT bindings fresh, and to allow the UA to detect when a flow has failed.

2. Conventions and Terminology

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 [2].

2.1. Definitions

Authoritative Proxy: A proxy that handles non-REGISTER requests for a specific Address-of-Record (AOR), performs the logical Location Server lookup described in RFC 3261, and forwards those requests to specific Contact URIs.

Edge Proxy: An Edge Proxy is any proxy that is located topologically between the registering User Agent and the Authoritative Proxy.

Flow: A Flow is a network protocol layer (layer 4) association between two hosts that is represented by the network address and port number of both ends and by the protocol. For TCP, a flow is equivalent to a TCP connection. For UDP a flow is a bidirectional stream of datagrams between a single pair of IP addresses and ports of both peers. With TCP, a flow often has a one to one correspondence with a single file descriptor in the operating system.

reg-id: This refers to the value of a new header field parameter value for the Contact header field. When a UA registers multiple times, each simultaneous registration gets a unique reg-id value.

instance-id: This specification uses the word instance-id to refer to the value of the "sip.instance" media feature tag in the Contact header field. This is a Uniform Resource Name (URN) that uniquely identifies this specific UA instance.

outbound-proxy-set A set of SIP URIs (Uniform Resource Identifiers) that represents each of the outbound proxies (often Edge Proxies) with which the UA will attempt to maintain a direct flow. The first URI in the set is often referred to as the primary outbound proxy and the second as the secondary outbound proxy. There is no difference between any of the URIs in this set, nor does the primary/secondary terminology imply that one is preferred over the other.

3. Overview

Several scenarios in which this technique is useful are discussed below, including the simple co-located registrar and proxy, a User Agent desiring multiple connections to a resource (for redundancy, for example), and a system that uses Edge Proxies.

3.1. Summary of Mechanism

The overall approach is fairly simple. Each UA has a unique instance-id that stays the same for this UA even if the UA reboots or is power cycled. Each UA can register multiple times over different connections for the same SIP Address of Record (AOR) to achieve high reliability. Each registration includes the instance-id for the UA and a reg-id label that is different for each flow. The registrar

can use the instance-id to recognize that two different registrations both reach the same UA. The registrar can use the reg-id label to recognize that a UA is registering after a reboot or a network failure.

When a proxy goes to route a message to a UA for which it has a binding, it can use any one of the flows on which a successful registration has been completed. A failure on a particular flow can be tried again on an alternate flow. Proxies can determine which flows go to the same UA by comparing the instance-id. Proxies can tell that a flow replaces a previously abandoned flow by looking at the reg-id.

UAs can use a simple periodic message as a keepalive mechanism to keep their flow to the proxy or registrar alive. For connection oriented transports such as TCP this is based on CRLF or a transport specific keepalive while for transports that are not connection oriented this is accomplished by using the keepalive usage profile of STUN (Session Traversal Utilities for NAT) [3].

3.2. Single Registrar and UA

In the topology shown below, a single server is acting as both a registrar and proxy.



User Agents which form only a single flow continue to register normally but include the instance-id as described in Section 4.1. The UA can also include a reg-id parameter which is used to allow the registrar to detect and avoid keeping invalid contacts when a UA reboots or reconnects after its old connection has failed for some reason.

For clarity, here is an example. Bob's UA creates a new TCP flow to the registrar and sends the following REGISTER request.

```
REGISTER sip:example.com;keep-crlf SIP/2.0
Via: SIP/2.0/TCP 192.0.2.1;branch=z9hG4bK-bad0ce-11-1036
Max-Forwards: 70
From: Bob <sip:bob@example.com>;tag=d879h76
To: Bob <sip:bob@example.com>
Call-ID: 8921348ju72je840.204
CSeq: 1 REGISTER
Supported: path
Contact: <sip:line1@192.168.0.2>; reg-id=1;
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-000A95A0E128>"
Content-Length: 0
```

The registrar challenges this registration to authenticate Bob. When the registrar adds an entry for this contact under the AOR for Bob, the registrar also keeps track of the connection over which it received this registration.

The registrar saves the instance-id ("urn:uuid:00000000-0000-1000-8000-000A95A0E128") and reg-id ("1") along with the rest of the Contact header field. If the instance-id and reg-id are the same as a previous registration for the same AOR, the registrar replaces the old Contact URI and flow information. This allows a UA that has rebooted to replace its previous registration for each flow with minimal impact on overall system load.

When Alice sends a request to Bob, his authoritative proxy selects the target set. The proxy forwards the request to elements in the target set based on the proxy's policy. The proxy looks at the target set and uses the instance-id to understand if two targets both end up routing to the same UA. When the proxy goes to forward a request to a given target, it looks and finds the flows over which it received the registration. The proxy then forwards the request on that flow, instead of resolving the Contact URI using the procedures in RFC 3263 [4] and trying to form a new flow to that contact. This allows the proxy to forward a request to a particular contact over the same flow that the UA used to register this AOR. If the proxy has multiple flows that all go to this UA, it can choose any one of the registration bindings for this AOR that has the same instance-id as the selected UA.

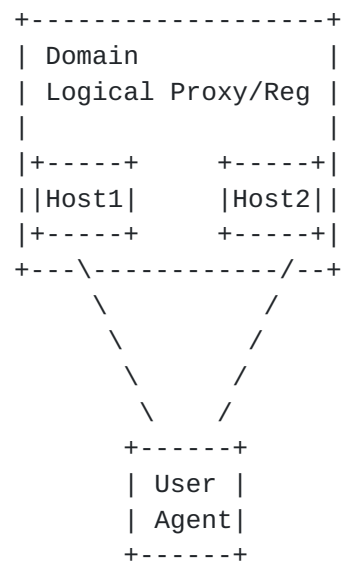
3.3. Multiple Connections from a User Agent

There are various ways to deploy SIP to build a reliable and scalable system. This section discusses one such design that is possible with the mechanisms in this specification. Other designs are also possible.

In the example system below, the logical outbound proxy/registrar for the domain is running on two hosts that share the appropriate state and can both provide registrar and outbound proxy functionality for the domain. The UA will form connections to two of the physical hosts that can perform the outbound proxy/registrar function for the domain. Reliability is achieved by having the UA form two TCP connections to the domain.

Scalability is achieved by using DNS SRV to load balance the primary connection across a set of machines that can service the primary connection, and also using DNS SRV to load balance across a separate set of machines that can service the secondary connection. The deployment here requires that DNS is configured with one entry that resolves to all the primary hosts and another entry that resolves to all the secondary hosts. While this introduces additional DNS configuration, the approach works and requires no additional SIP extensions.

> Note: Approaches which select multiple connections from a single DNS SRV set were also considered, but cannot prevent two connections from accidentally resolving to the same host. The approach in this document does not prevent future extensions, such as the SIP UA configuration framework [19], from adding other ways for a User Agent to discover its outbound-proxy-set.



The UA is configured with multiple outbound proxy registration URIs. These URIs are configured into the UA through whatever the normal mechanism is to configure the proxy or registrar address in the UA. If the AOR is Alice@example.com, the outbound-proxy-set might look something like "sip:primary.example.com;keep-stun" and "sip:secondary.example.com;keep-stun". The "keep-stun" tag indicates that

a SIP server supports STUN and SIP multiplexed over the same flow, as described later in this specification. Note that each URI in the outbound-proxy-set could resolve to several different physical hosts. The administrative domain that created these URIs should ensure that the two URIs resolve to separate hosts. These URIs are handled according to normal SIP processing rules, so mechanisms like SRV can be used to do load balancing across a proxy farm.

The domain also needs to ensure that a request for the UA sent to host1 or host2 is then sent across the appropriate flow to the UA. The domain might choose to use the Path header approach (as described in the next section) to store this internal routing information on host1 or host2.

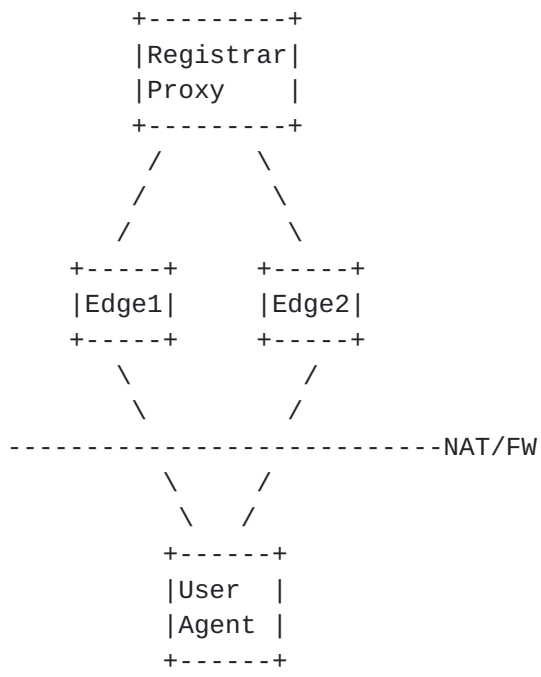
When a single server fails, all the UAs that have a flow through it will detect a flow failure and try to reconnect. This can cause large loads on the server. When large numbers of hosts reconnect nearly simultaneously, this is referred to as the avalanche restart problem, and is further discussed in Section 4.5. The multiple flows to many servers help reduce the load caused by the avalanche restart. If a UA has multiple flows, and one of the servers fails, the UA delays the specified time before trying to form a new connection to replace the flow to the server that failed. By spreading out the time used for all the UAs to reconnect to a server, the load on the server farm is reduced.

When used in this fashion to achieve high reliability, the operator will need to configure DNS such that the various URIs in the outbound proxy set do not resolve to the same host.

Another motivation for maintaining multiple flows between the UA and its registrar is related to multihomed UAs. Such UAs can benefit from multiple connections from different interfaces to protect against the failure of an individual access link.

3.4. Edge Proxies

Some SIP deployments use edge proxies such that the UA sends the REGISTER to an Edge Proxy that then forwards the REGISTER to the Registrar. The Edge Proxy includes a Path header [5] so that when the registrar later forwards a request to this UA, the request is routed through the Edge Proxy. There could be a NAT or firewall between the UA and the Edge Proxy.



These systems can use effectively the same mechanism as described in the previous sections but need to use the Path header. When the Edge Proxy receives a registration, it needs to create an identifier value that is unique to this flow (and not a subsequent flow with the same addresses) and put this identifier in the Path header URI. This identifier has two purposes. First, it allows the Edge Proxy to map future requests back to the correct flow. Second, because the identifier will only be returned if the user authentication with the registrar succeeds, it allows the Edge Proxy to indirectly check the user's authentication information via the registrar. The identifier is placed in the user portion of a loose route in the Path header. If the registration succeeds, the Edge Proxy needs to map future requests that are routed to the identifier value from the Path header, to the associated flow.

The term Edge Proxy is often used to refer to deployments where the Edge Proxy is in the same administrative domain as the Registrar. However, in this specification we use the term to refer to any proxy between the UA and the Registrar. For example the Edge Proxy may be inside an enterprise that requires its use and the registrar could be from a service provider with no relationship to the enterprise. Regardless if they are in the same administrative domain, this specification requires that Registrars and Edge proxies support the Path header mechanism in RFC 3327 [5].

3.5. Keepalive Technique

This document describes three keepalive mechanisms. Each of these mechanisms uses a client-to-server "ping" keepalive and a corresponding server-to-client "pong" message. This ping-pong sequence allows the client, and optionally the server, to tell if its flow is still active and useful for SIP traffic. The server responds to pings by sending pongs. If the client does not receive a pong in response to its ping, it declares the flow dead and opens a new flow in its place.

This document also suggests timer values for two of these client keepalive mechanisms. These timer values were chosen to keep most NAT and firewall bindings open, to detect unresponsive servers within 2 minutes, and to prevent the avalanche restart problem. However, the client may choose different timer values to suit its needs, for example to optimize battery life. In some environments, the server can also keep track of the time since a ping was received over a flow to guess the likelihood that the flow is still useful for delivering SIP messages. In this case, the server provides an indicator (the 'timed-keepalives' parameter) that the server requires the client to use the suggested timer values.

When the UA detects that a flow has failed or that the flow definition has changed, the UA needs to re-register and will use the back-off mechanism described in Section 4 to provide congestion relief when a large number of agents simultaneously reboot.

A keepalive mechanism needs to keep NAT bindings refreshed; for connections, it also needs to detect failure of a connection; and for connectionless transports, it needs to detect flow failures including changes to the NAT public mapping. For connection oriented transports such as TCP and SCTP, this specification describes a keepalive approach based on sending CRLFs, and for TCP, a usage of TCP transport-layer keepalives. For connectionless transport, such as UDP, this specification describes using STUN [3] over the same flow as the SIP traffic to perform the keepalive.

3.5.1. CRLF Keepalive Technique

This approach can only be used with connection-oriented transports such as TCP or SCTP. The client periodically sends a double-CRLF (the "ping") then waits to receive a single CRLF (the "pong"). If the client does not receive a "pong" within an appropriate amount of time, it considers the flow failed.

3.5.2. TCP Keepalive Technique

This approach can only be used when the transport protocol is TCP.

User Agents that are capable of generating per-connection TCP keepalives can use TCP keepalives. When using this approach the values of the keepalive timer are left to the client. Servers cannot make any assumption about what values are used.

> Note: when TCP is being used, it's natural to think of using TCP KEEPALIVE. Unfortunately, many operating systems and programming environments do not allow the keepalive time to be set on a per-connection basis. Thus, applications may not be able to set an appropriate time.

3.5.3. STUN Keepalive Technique

This technique can only be used for transports, such as UDP, that are not connection oriented.

For connection-less transports, a flow definition could change because a NAT device in the network path reboots and the resulting public IP address or port mapping for the UA changes. To detect this, STUN requests are sent over the same flow that is being used for the SIP traffic. The proxy or registrar acts as a Session Traversal Utilities for NAT (STUN) [3] server on the SIP signaling port.

> Note: The STUN mechanism is very robust and allows the detection of a changed IP address. Many other options were considered, but the SIP Working Group selected the STUN-based approach. Approaches using SIP requests were abandoned because to achieve the required performance, the server needs to deviate from the SIP specification in significant ways. This would result in many undesirable and non-deterministic behaviors in some environments. Another approach considered to detect a changed flow was using OPTIONS messages and the rport parameter. Although the OPTIONS approach has the advantage of being backwards compatible, it also significantly increases the load on the proxy or registrar server. Related to this idea was an idea of creating a new SIP PING method that was like OPTIONS but faster. It would be critical that this PING method did not violate the processing requirements of a proxies and UAS so it was never clear how it would be significantly faster than OPTIONS given it would still have to obey things like checking the Proxy-Require header. After considerable consideration the working group came to some consensus that the STUN approach was a better solution than these alternative designs.

4. User Agent Mechanisms

4.1. Instance ID Creation

Each UA MUST have an Instance Identifier URN that uniquely identifies the device. Usage of a URN provides a persistent and unique name for the UA instance. It also provides an easy way to guarantee uniqueness within the AOR. This URN MUST be persistent across power cycles of the device. The Instance ID MUST NOT change as the device moves from one network to another.

A UA SHOULD create a UUID URN [6] as its instance-id. The UUID URN allows for non-centralized computation of a URN based on time, unique names (such as a MAC address), or a random number generator.

> A device like a soft-phone, when first installed, can generate a UUID [6] and then save this in persistent storage for all future use. For a device such as a hard phone, which will only ever have a single SIP UA present, the UUID can include the MAC address and be generated at any time because it is guaranteed that no other UUID is being generated at the same time on that physical device. This means the value of the time component of the UUID can be arbitrarily selected to be any time less than the time when the device was manufactured. A time of 0 (as shown in the example in Section 3.2) is perfectly legal as long as the device knows no other UUIDs were generated at this time.

If a URN scheme other than UUID is used, the URN MUST be selected such that the instance can be certain that no other instance registering against the same AOR would choose the same URN value. An example of a URN that would not meet the requirements of this specification is the national bibliographic number [20]. Since there is no clear relationship between a SIP UA instance and a URN in this namespace, there is no way a selection of a value can be performed that guarantees that another UA instance doesn't choose the same value.

To convey its instance-id in both requests and responses, the UA includes a "sip.instance" media feature tag as a UA characteristic [7] . As described in [7], this media feature tag will be encoded in the Contact header field as the "+sip.instance" Contact header field parameter. The value of this parameter MUST be a URN [8]. One case where a UA may not want to include the URN in the sip.instance media feature tag is when it is making an anonymous request or some other privacy concern requires that the UA not reveal its identity.

RFC 3840 [7] defines equality rules for callee capabilities parameters, and according to that specification, the "sip.instance" media feature tag will be compared by case-sensitive string comparison. This means that the URN will be encapsulated by angle brackets ("<" and ">") when it is placed within the quoted string value of the +sip.instance Contact header field parameter. The case-sensitive matching rules apply only to the generic usages defined in RFC 3840 [7] and in the caller preferences specification [9]. When the instance ID is used in this specification, it is effectively "extracted" from the value in the "sip.instance" media feature tag. Thus, equality comparisons are performed using the rules for URN equality that are specific to the scheme in the URN. If the element performing the comparisons does not understand the URN scheme, it performs the comparisons using the lexical equality rules defined in RFC 2141 [8]. Lexical equality could result in two URNs being considered unequal when they are actually equal. In this specific usage of URNs, the only element which provides the URN is the SIP UA instance identified by that URN. As a result, the UA instance SHOULD provide lexically equivalent URNs in each registration it generates. This is likely to be normal behavior in any case; clients are not likely to modify the value of the instance ID so that it remains functionally equivalent yet lexicographically different from previous registrations.

4.2. Registrations

At configuration time UAs obtain one or more SIP URIs representing the default outbound-proxy-set. This specification assumes the set is determined via any of a number of configuration mechanisms, and future specifications can define additional mechanisms such as using DNS to discover this set. How the UA is configured is outside the scope of this specification. However, a UA MUST support sets with at least two outbound proxy URIs and SHOULD support sets with up to four URIs. For each outbound proxy URI in the set, the UA SHOULD send a REGISTER in the normal way using this URI as the default outbound proxy. Forming the route set for the request is outside the scope of this document, but typically results in sending the REGISTER such that the topmost Route header field contains a loose route to the outbound proxy URI. Other issues related to outbound route construction are discussed in [21].

Registration requests, other than those described in Section 4.2.1, MUST include an instance-id media feature tag as specified in Section 4.1.

These ordinary registration requests include a distinct reg-id parameter in the Contact header field. Each one of these

registrations will form a new flow from the UA to the proxy. The sequence of reg-id values does not have to be sequential but MUST be exactly the same sequence of reg-id values each time the UA instance power cycles or reboots so that the reg-id values will collide with the previously used reg-id values. This is so the registrar can replace the older registration.

> The UAC can situationally decide whether to request outbound behavior by including or omitting the 'reg-id' parameter. For example, imagine the outbound-proxy-set contains two proxies in different domains, EP1 and EP2. If an outbound-style registration succeeded for a flow through EP1, the UA might decide to include 'outbound' in its Require header field when registering with EP2, in order to insure consistency. Similarly, if the registration through EP1 did not support outbound, the UA might decide to omit the 'reg-id' parameter when registering with EP2.

The UAC MUST indicate that it supports the Path header [5] mechanism, by including the 'path' option-tag in a Supported header field value in its REGISTER requests. Other than optionally examining the Path vector in the response, this is all that is required of the UAC to support Path.

The UAC MAY examine successful registrations for the presence of an 'outbound' option-tag in a Supported header field value. Presence of this option-tag indicates that the registrar is compliant with this specification, and that any edge proxies which need to participate are also compliant.

Note that the UA needs to honor 503 (Service Unavailable) responses to registrations as described in RFC 3261 and RFC 3263 [4]. In particular, implementors should note that when receiving a 503 (Service Unavailable) response with a Retry-After header field, the UA is expected to wait the indicated amount of time and retry the registration. A Retry-After header field value of 0 is valid and indicates the UA is expected to retry the REGISTER immediately. Implementations need to ensure that when retrying the REGISTER, they revisit the DNS resolution results such that the UA can select an alternate host from the one chosen the previous time the URI was resolved.

Finally, re-registrations which merely refresh an existing valid registration SHOULD be sent over the same flow as the original registration.

4.2.1. Registration by Other Instances

A User Agent MUST NOT include a reg-id header parameter in the Contact header field of a registration if the registering UA is not the same instance as the UA referred to by the target Contact header field. (This practice is occasionally used to install forwarding policy into registrars.)

Note that a UAC also MUST NOT include an instance-id or reg-id parameter in a request to unregister all Contacts (a single Contact header field value with the value of "*").

4.3. Sending Requests

When a UA is about to send a request, it first performs normal processing to select the next hop URI. The UA can use a variety of techniques to compute the route set and accordingly the next hop URI. Discussion of these techniques is outside the scope of this document but could include mechanisms specified in RFC 3608 [22] (Service Route) and [21].

The UA performs normal DNS resolution on the next hop URI (as described in RFC 3263 [4]) to find a protocol, IP address, and port. For protocols that don't use TLS, if the UA has an existing flow to this IP address, and port with the correct protocol, then the UA MUST use the existing connection. For TLS protocols, there MUST also be a match between the host production in the next hop and one of the URIs contained in the subjectAltName in the peer certificate. If the UA cannot use one of the existing flows, then it SHOULD form a new flow by sending a datagram or opening a new connection to the next hop, as appropriate for the transport protocol.

The contact is formed normally in that the UAC uses the IP address of the device (even if the device is behind a NAT). Unless there are privacy reason not to include an instance-id, the contact SHOULD include the instance-id media feature tag as specified in Section 4.1. The UAC MUST also include an "ob" parameter in the Contact URI if, and only if, the UAC is sending the request over a flow for which the Registrar applied outbound processing.

> Note that if the UA wants its flow to work through NATs or firewalls it still needs to put the 'rport' parameter [10] in its Via header field value, and send from the port it is prepared to receive on. More general information about NAT traversal in SIP is described in [23].

4.4. Detecting Flow Failure

The UA needs to detect when a specific flow fails. The UA actively tries to detect failure by periodically sending keepalive messages using one of the techniques described in Section 4.4.1, Section 4.4.2, or Section 4.4.3. If a flow has failed, the UA follows the procedures in Section 4.2 to form a new flow to replace the failed one.

When the outbound-proxy-set contains the "timed-keepalives" parameter, the UA MUST send its keepalives according to the time periods described in this section. The server can specify this so the server can detect liveness of the client within a predictable time scale. If the parameter is not present, the UA can send keepalives at its discretion.

The time between each keepalive request when using non connection based transports such as UDP SHOULD be a random number between 24 and 29 seconds while for connection based transports such as TCP it SHOULD be a random number between 95 and 120 seconds. These times MAY be configurable. To clarify, the random number will be different for each request. Issues such as battery consumption might motivate longer keepalive intervals. If the 'timed-keepalives' parameter is set on the outbound-proxy-set, the UA MUST use these recommended timer values.

> Note on selection of time values: For UDP, the upper bound of 29 seconds was selected so that multiple STUN packets could be sent before 30 seconds based on information that many NATs have UDP timeouts as low as 30 seconds. The 24 second lower bound was selected so that after 10 minutes the jitter introduced by different timers will make the keepalive requests unsynchronized to evenly spread the load on the servers. For TCP, the 120 seconds upper bound was chosen based on the idea that for a good user experience, failures normally will be detected in this amount of time and a new connection set up. Operators that wish to change the relationship between load on servers and the expected time that a user might not receive inbound communications will probably adjust this time. The 95 seconds lower bound was chosen so that the jitter introduced will result in a relatively even load on the servers after 30 minutes.

The client needs to perform normal RFC 3263 [4] SIP DNS resolution on the URI from the outbound-proxy-set to pick a transport. Once a transport is selected, the UA selects a keepalive approach that is allowed for that transport and that is allowed by the server based on the tags in the URI from the outbound-proxy-set.

4.4.1. Keepalive with TCP KEEPALIVE

This approach MUST only be used with TCP.

User Agents that form flows, check if the configured URI they are connecting to has a 'timed-keepalives' URI parameter (defined in Section 12). If the parameter is not present and the UA is not already performing keepalives using another supported mechanism, the UA needs to periodically perform keepalive checks by using the TCP Keepalive mechanism. Not all environments can use this approach and it is not mandatory to implement. Deployments that use it should also include keep-crlf so that clients that do not implement this option but are using TCP have an alternative approach to use.

4.4.2. Keepalive with CRLF

This approach MUST only be used with connection oriented transports such as TCP or SCTP.

User Agents that form flows check if the configured URI they are connecting to has a 'keep-crlf' URI parameter (defined in Section 12). If the parameter is present and the UA is not already performing keepalives using another supported mechanism, the UA can send keep alives as described in this section.

For this mechanism, the client "ping" is a double-CRLF sequence, and the server "pong" is a single CRLF, as defined in the ABNF below:

```
CRLF = CR LF
double-CRLF = CR LF CR LF
CR = 0x0d
LF = 0x0a
```

The ping and pong need to be sent between SIP messages and cannot be sent in the middle of a SIP message. If sending over a TLS protected channel, the CRLFs are sent inside the TLS record layer. If a pong is not received within 10 seconds then the client MUST treat the flow as failed. Clients MUST support this CRLF keepalive.

4.4.3. Keepalive with STUN

This approach MUST only be used with transports, such as UDP, that are not connection oriented.

User Agents that form flows, check if the configured URI they are connecting to has a 'keep-stun' URI parameter (defined in Section 12). If the parameter is present and the UA is not already performing keepalives using another supported mechanism, the UA can

periodically perform keepalive checks by sending STUN [3] Binding Requests over the flow as described in Section 8. Clients MUST support STUN based keepalives.

If the XOR-MAPPED-ADDRESS in the STUN Binding Response changes, the UA MUST treat this event as a failure on the flow.

4.5. Flow Recovery

When a flow to a particular URI in the outbound-proxy-set fails, the UA needs to form a new flow to replace the old flow and replace any registrations that were previously sent over this flow. Each new registration MUST have the same reg-id as the registration it replaces. This is done in much the same way as forming a brand new flow as described in Section 4.2; however, if there is a failure in forming this flow, the UA needs to wait a certain amount of time before retrying to form a flow to this particular next hop.

The amount of time to wait depends if the previous attempt at establishing a flow was successful. For the purposes of this section, a flow is considered successful if outbound registration succeeded, and if keepalives are in use on this flow, at least one consecutive keepalive response was received.

The number of seconds to wait is computed in the following way. If all of the flows to every URI in the outbound proxy set have failed, the base time is set to 30 seconds; otherwise, in the case where at least one of the flows has not failed, the base time is set to 90 seconds. The wait time is computed by taking two raised to the power of the number of consecutive registration failures for that URI, and multiplying this by the base time, up to a maximum of 1800 seconds.

```
wait-time = min( max-time, (base-time * (2 ^ consecutive-failures)))
```

These times MAY be configurable in the UA. The three times are:

- max-time with a default of 1800 seconds
- base-time-all-fail with a default of 30 seconds
- base-time-not-failed with a default of 90 seconds

For example, if the base time is 30 seconds, and there were three failures, then the wait time is min(1800,30*(2^3)) or 240 seconds. The delay time is computed by selecting a uniform random time between 50 and 100 percent of the wait time. The UA MUST wait for the value of the delay time before trying another registration to form a new flow for that URI.

To be explicitly clear on the boundary conditions: when the UA boots it immediately tries to register. If this fails and no registration on other flows succeed, the first retry happens somewhere between 30

and 60 seconds after the failure of the first registration request. If the number of consecutive-failures is large enough that the maximum of 1800 seconds is reached, the UA will keep trying indefinitely with a random time of 15 to 30 minutes between each attempt.

5. Edge Proxy Mechanisms

5.1. Processing Register Requests

When an Edge Proxy receives a registration request with a reg-id header parameter in the Contact header field, it needs to determine if it (the edge proxy) will have to be visited for any subsequent requests sent to the user agent identified in the Contact header field, or not. If the Edge Proxy determines that this is the case, it inserts its URI in a Path header field value as described in RFC 3327 [5]. If the Edge Proxy is the first SIP node after the UAC, it either MUST store a "flow token"--containing information about the flow from the previous hop--in its Path URI, or reject the request. The flow token MUST be an identifier that is unique to this network flow. The flow token MAY be placed in the userpart of the URI. In addition, the first node MUST include an 'ob' URI parameter in its Path header field value. If the Edge Proxy is not the first SIP node after the UAC it MUST NOT place an 'ob' URI parameter in a Path header field value. The Edge Proxy can determine if it is the first hop by examining the Via header field.

5.2. Generating Flow Tokens

A trivial but impractical way to satisfy the flow token requirement in Section 5.1 involves storing a mapping between an incrementing counter and the connection information; however this would require the Edge Proxy to keep an impractical amount of state. It is unclear when this state could be removed and the approach would have problems if the proxy crashed and lost the value of the counter. A stateless example is provided below. A proxy can use any algorithm it wants as long as the flow token is unique to a flow, the flow can be recovered from the token, and the token cannot be modified by attackers.

Example Algorithm: When the proxy boots it selects a 20-octet crypto random key called K that only the Edge Proxy knows. A byte array, called S, is formed that contains the following information about the flow the request was received on: an enumeration indicating the protocol, the local IP address and port, the remote IP address and port. The HMAC of S is computed using the key K and the HMAC-SHA1-80 algorithm, as defined in [11]. The concatenation of the HMAC and S are base64 encoded, as defined in [12], and used as the

flow identifier. When using IPv4 addresses, this will result in a 32-octet identifier.

5.3. Forwarding Requests

When an Edge Proxy receives a request, it applies normal routing procedures with the following addition. If the Edge Proxy receives a request where the edge proxy is the host in the topmost Route header field value, and the Route header field value contains a flow token, the proxy compares the flow in the flow token with the source of the request to determine if this is an "incoming" or "outgoing" request. If the flow in the flow token in the topmost Route header field value matches the source of the request, the request in an "outgoing" request. For an "outgoing" request, the edge proxy just removes the Route header and continues processing the request. Otherwise, this is an "incoming" request. For an incoming request, the proxy removes the Route header field value and forwards the request over the 'logical flow' identified by the flow token, that is known to deliver data to the specific target UA instance. For connection-oriented transports, if the flow no longer exists the proxy SHOULD send a 430 (Flow Failed) response to the request.

Proxies which used the example algorithm described in this document to form a flow token follow the procedures below to determine the correct flow.

Example Algorithm: To decode the flow token, take the flow identifier in the user portion of the URI and base64 decode it, then verify the HMAC is correct by recomputing the HMAC and checking that it matches. If the HMAC is not correct, the proxy SHOULD send a 403 (Forbidden) response. If the HMAC is correct then the proxy SHOULD forward the request on the flow that was specified by the information in the flow identifier. If this flow no longer exists, the proxy SHOULD send a 430 (Flow Failed) response to the request.

Note that this specification needs mid-dialog requests to be routed over the same flows as those stored in the Path vector from the initial registration, but techniques to ensure that mid-dialog requests are routed over an existing flow are not part of this specification. However, an approach such as having the Edge Proxy Record-Route with a flow token is one way to ensure that mid-dialog requests are routed over the correct flow. The Edge Proxy can use the presence of the "ob" parameter in the UAC's Contact URI to determine if it should add a flow token.

5.4. Edge Proxy Keepalive Handling

All edge proxies compliant with this specification MUST implement support for the STUN NAT Keepalive usage on its SIP UDP ports as described in Section 8.

When a server receives a double CRLF sequence on a connection oriented transport such as TCP or SCTP, it MUST immediately respond with a single CRLF over the same connection.

6. Registrar Mechanisms: Processing REGISTER Requests

This specification updates the definition of a binding in RFC 3261 [1] Section 10 and RFC 3327 [5] Section 5.3.

When no +sip.instance media feature parameter is present in a Contact header field value in a REGISTER request, the corresponding binding is still between an AOR and the URI from that Contact header field value. When a +sip.instance media feature parameter is present in a Contact header field value in a REGISTER request, the corresponding binding is between an AOR and the combination of the instance-id (from the +sip.instance media feature parameter) and the value of reg-id parameter if it is present. For a binding with an instance-id, the registrar still stores the Contact header field value URI with the binding, but does not consider the Contact URI for comparison purposes. A Contact header field value with an instance-id but no reg-id is valid, but one with a reg-id but no instance-id is not. If the registrar processes a Contact header field value with a reg-id but no instance-id, it simply ignores the reg-id parameter. The registrar MUST be prepared to receive, simultaneously for the same AOR, some registrations that use instance-id and reg-id and some registrations that do not.

Registrars which implement this specification MUST support the Path header mechanism [5].

In addition to the normal information stored in the binding record, some additional information needs to be stored for any registration that contains an instance-id and a reg-id header parameter in the Contact header field value. First the registrar examines the first Path header field value, if any. If the Path header field exists and the first URI does not have an 'ob' URI parameter, the registrar MUST ignore the reg-id parameter and continue processing the request as if it did not support this specification. Likewise if the REGISTER request visited an edge proxy, but no Path header field values are present, the registrar MUST ignore the reg-id parameter. Specifically, if it ignores the 'reg-id' parameter the registrar MUST

use RFC 3261 Contact binding rules, and MUST NOT include the 'outbound' option-tag in its Supported header field. The registrar can determine if it is the first hop by examining the Via header field.

If the UAC has a direct flow with the registrar, the registrar MUST store enough information to uniquely identify the network flow over which the request arrived. For common operating systems with TCP, this would typically just be the handle to the file descriptor where the handle would become invalid if the TCP session was closed. For common operating systems with UDP this would typically be the file descriptor for the local socket that received the request, the local interface, and the IP address and port number of the remote side that sent the request. The registrar MAY store this information by adding itself to the Path header field with an appropriate flow token.

The registrar MUST also store all the Contact header field information including the reg-id and instance-id parameters and SHOULD also store the time at which the binding was last updated. If a Path header field is present, RFC 3327 [5] requires the registrar to store this information as well. If the registrar receives a re-registration, it MUST update any information that uniquely identifies the network flow over which the request arrived if that information has changed, and SHOULD update the time the binding was last updated.

The Registrar MUST include the 'outbound' option-tag (defined in Section (Section 12.1)) in a Supported header field value in its responses to REGISTER requests for which it has performed outbound processing, and MUST NOT include this option-tag if it did not perform outbound processing. Furthermore, the Registrar MUST NOR include this option-tag in its response if the Registrar skipped outbound processing by ignoring the reg-id parameter as described in this specification. Note that the requirements in this section applies to both REGISTER requests received from an Edge Proxy as well as requests received directly from the UAC. The Registrar MAY be configured with local policy to reject any registrations that do not include the instance-id and reg-id, or with Path header field values that do not contain the 'ob' parameter.

To be compliant with this specification, registrars which can receive SIP requests directly from a UAC without intervening edge proxies MUST implement the STUN NAT Keepalive usage on its SIP UDP ports as described in Section 8 and when it receives a double-CRLF sequence on a connection oriented transport such as TCP or SCTP, it MUST immediately respond with a single CRLF over the same connection.

7. Authoritative Proxy Mechanisms: Forwarding Requests

When a proxy uses the location service to look up a registration binding and then proxies a request to a particular contact, it selects a contact to use normally, with a few additional rules:

- The proxy MUST NOT populate the target set with more than one contact with the same AOR and instance-id at a time.
- If a request for a particular AOR and instance-id fails with a 430 (Flow Failed) response, the proxy SHOULD replace the failed branch with another target (if one is available) with the same AOR and instance-id, but a different reg-id.
- If the proxy receives a final response from a branch other than a 408 (Request Timeout) or a 430 (Flow Failed) response, the proxy MUST NOT forward the same request to another target representing the same AOR and instance-id. The targeted instance has already provided its response.

The proxy uses normal forwarding rules looking at the next-hop target of the message and the value of any stored Path header field vector in the registration binding to decide how to forward the request and populate the Route header in the request. If the proxy stored information about the flow over which it received the REGISTER for the binding, then the proxy MUST send the request over the same 'logical flow' saved with the binding that is known to deliver data to the specific target UA instance.

Typically this means that for TCP, the request is sent on the same TCP socket that received the REGISTER request. For UDP, the request is sent from the same local IP address and port over which the registration was received, to the same IP address and port from which the REGISTER was received.

If a proxy or registrar receives information from the network that indicates that no future messages will be delivered on a specific flow, then the proxy MUST invalidate all the bindings in the target set that use that flow (regardless of AOR). Examples of this are a TCP socket closing or receiving a destination unreachable ICMP error on a UDP flow. Similarly, if a proxy closes a file descriptor, it MUST invalidate all the bindings in the target set with flows that use that file descriptor.

8. STUN Keepalive Processing

This section describes changes to the SIP transport layer that allow SIP and the STUN [3] NAT Keepalive usage to be mixed over the same flow. The STUN messages are used to verify that connectivity is

still available over a UDP flow, and to provide periodic keepalives. Note that these STUN keepalives are always sent to the next SIP hop. STUN messages are not delivered end-to-end.

The only STUN messages required by this usage are Binding Requests, Binding Responses, and Binding Error Responses. The UAC sends Binding Requests over the same UDP flow that is used for sending SIP messages. These Binding Requests do not require any STUN attributes. The UAS responds to a valid Binding Request with a Binding Response which MUST include the XOR-MAPPED-ADDRESS attribute.

If a server compliant to this section receives SIP requests on a given interface and port, it MUST also provide a limited version of a STUN server on the same interface and port as described in Section 12.3 of [3].

> It is easy to distinguish STUN and SIP packets sent over UDP, because the first octet of a STUN packet has a value of 0 or 1 while the first octet of a SIP message is never a 0 or 1.

When a URI is created that refers to a SIP node that supports STUN as described in this section, the 'keep-stun' URI parameter, as defined in Section 12 MUST be added to the URI. This allows a UA to inspect the URI to decide if it should attempt to send STUN requests to this location. For example, an edge proxy could insert this parameter into its Path URI so that the registering UA can discover the edge proxy supports STUN keepalives.

Because sending and receiving binary STUN data on the same ports used for SIP is a significant and non-backwards compatible change to RFC 3261, this section requires a number of checks before sending STUN messages to a SIP node. If a SIP node sends STUN requests (for example due to incorrect configuration) despite these warnings, the node could be blacklisted for UDP traffic.

A SIP node MUST NOT send STUN requests over a flow unless it has an explicit indication that the target next hop SIP server claims to support STUN. For example, automatic or manual configuration of an outbound-proxy-set which contains the 'keep-stun' parameter, or receiving the parameter in the Path header of the edge proxy, is considered sufficient explicit indication. Note that UACs MUST NOT use an ambiguous configuration option such as "Work through NATs?" or "Do Keepalives?" to imply next hop STUN support.

> Typically, a SIP node first sends a SIP request and waits to receive a final response (other than a 408 response) over a flow to a new target destination, before sending any STUN messages. When scheduled for the next NAT refresh, the SIP node sends a STUN

request to the target.

Once a flow is established, failure of a STUN request (including its retransmissions) is considered a failure of the underlying flow. For SIP over UDP flows, if the XOR-MAPPED-ADDRESS returned over the flow changes, this indicates that the underlying connectivity has changed, and is considered a flow failure.

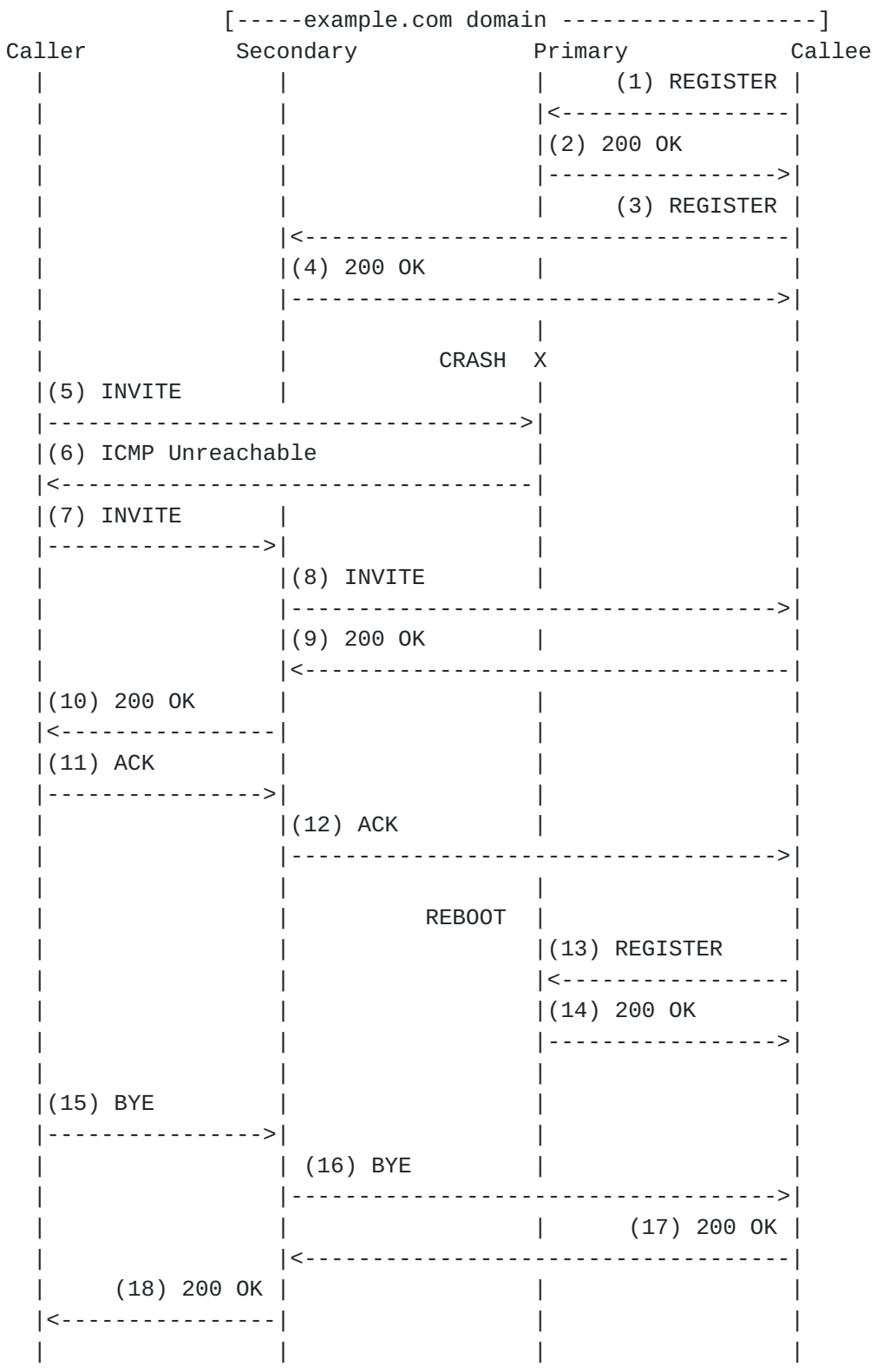
8.1. Use with Sigcomp

When STUN is used together with SigComp [24] compressed SIP messages over the same flow, how the STUN messages are sent depends on the transport protocol. For UDP flows, the STUN messages are simply sent uncompressed, "outside" of SigComp. This is supported by multiplexing STUN messages with SigComp messages by checking the two topmost bits of the message. These bits are always one for SigComp, or zero for STUN.

All SigComp messages contain a prefix (the five most-significant bits of the first byte are set to one) that does not occur in UTF-8 [13] encoded text messages, so for applications which use this encoding (or ASCII encoding) it is possible to multiplex uncompressed application messages and SigComp messages on the same UDP port.
The most significant two bits of every STUN message are both zeroes. This, combined with the magic cookie, aids in differentiating STUN packets from other protocols when STUN is multiplexed with other protocols on the same port.

9. Example Message Flow

The following call flow shows a basic registration and an incoming call. At some point, the flow to the Primary proxy is lost. An incoming INVITE tries to reach the Callee through the Primary flow, but receives an ICMP Unreachable message. The Caller retries using the Secondary Edge Proxy, which uses a separate flow. Later, after the Primary reboots, The Callee discovers the flow failure and reestablishes a new flow to the Primary.



This call flow assumes that the Callee has been configured with a proxy set that consists of "sip:pri.example.com;lr;keep-stun" and "sip:sec.example.com;lr;keep-stun". The Callee REGISTER in message

(1) looks like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=7F94778B653B
To: Callee <sip:callee@example.com>
Call-ID: 16CB75F21C70
CSeq: 1 REGISTER
Supported: path
Route: <sip:pri.example.com;lr;keep-stun>
Contact: <sip:callee@192.0.2.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;reg-id=1
Content-Length: 0
```

In the message, note that the Route is set and the Contact header field value contains the instance-id and reg-id. The response to the REGISTER in message (2) would look like:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKnashds7
From: Callee <sip:callee@example.com>;tag=7F94778B653B
To: Callee <sip:callee@example.com>;tag=6AF99445E44A
Call-ID: 16CB75F21C70
CSeq: 1 REGISTER
Supported: outbound
Contact: <sip:callee@192.0.2.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;reg-id=1
  ;expires=3600
Content-Length: 0
```

The second registration in message 3 and 4 are similar other than the Call-ID has changed, the reg-id is 2, and the route is set to the secondary instead of the primary. They look like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKnqr9bym
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=755285EABDE2
To: Callee <sip:callee@example.com>
Call-ID: E05133BD26DD
CSeq: 1 REGISTER
Supported: path
Route: <sip:sec.example.com;lr;keep-stun>
Contact: <sip:callee@192.0.2.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;reg-id=2
Content-Length: 0


SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKnqr9bym
From: Callee <sip:callee@example.com>;tag=755285EABDE2
To: Callee <sip:callee@example.com>;tag=49A9AD0B3F6A
Call-ID: E05133BD26DD
Supported: outbound
CSeq: 1 REGISTER
Contact: <sip:callee@192.0.2.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;reg-id=1
  ;expires=3600
Contact: <sip:callee@192.0.2.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;reg-id=2
  ;expires=3600
Content-Length: 0
```

The messages in the call flow are very normal. The only interesting thing to note is that the INVITE in message 8 contains a Record-Route header for the Secondary proxy, with its flow token.

```
Record-Route:
 <sip:PQPbqQE+Ynf+tzRPD27lU6uxkjQ8LLUG@sec.example.com;lr>
```

The registrations in message 13 and 14 are the same as message 1 and 2 other than the Call-ID and tags have changed. Because these messages will contain the same instance-id and reg-id as those in 1 and 2, this flow will partially supersede that for messages 1 and 2 and will be tried first by Primary.

10. Grammar

This specification defines new Contact header field parameters, reg-id and +sip.instance. The grammar includes the definitions from RFC 3261 [1] and includes the definition of uric from RFC 3986 [14].

> Note: The "=/" syntax used in this ABNF indicates an extension of the production on the left hand side.

The ABNF[15] is:

```
contact-params =/ c-p-reg / c-p-instance

c-p-reg        = "reg-id" EQUAL 1*DIGIT ; 1 to 2**31

c-p-instance   =  "+sip.instance" EQUAL
                  LDQUOT "<" instance-val ">" RDQUOT

instance-val   = *uric ; defined in RFC 3986
```

The value of the reg-id MUST NOT be 0 and MUST be less than 2**31.

11. Definition of 430 Flow Failed response code

This specification defines a new SIP response code '430 Flow Failed'. This response code is used by an Edge Proxy to indicate to the Authoritative Proxy that a specific flow to a UA instance has failed. Other flows to the same instance could still succeed. The Authoritative Proxy SHOULD attempt to forward to another target (flow) with the same instance-id and AOR.

12. IANA Considerations

12.1. Contact Header Field

This specification defines a new Contact header field parameter called reg-id in the "Header Field Parameters and Parameter Values" sub-registry as per the registry created by [16]. The required information is:

Header Field	Parameter Name	Predefined Values	Reference
Contact	reg-id	No	[RFC AAAA]

[NOTE TO RFC Editor: Please replace AAAA with

the RFC number of this specification.]

12.2. SIP/SIPS URI Parameters

This specification augments the "SIP/SIPS URI Parameters" sub-registry as per the registry created by [17]. The required information is:

Parameter Name	Predefined Values	Reference
keep-crlf	No	[RFC AAAA]
keep-stun	No	[RFC AAAA]
timed-keepalive	No	[RFC AAAA]
ob	No	[RFC AAAA]

[NOTE TO RFC Editor: Please replace AAAA with the RFC number of this specification.]

12.3. SIP Option Tag

This specification registers a new SIP option tag, as per the guidelines in Section 27.1 of RFC 3261.

Name: outbound
Description: This option-tag is used to identify Registrars which support extensions for Client Initiated Connections. A Registrar places this option-tag in a Supported header to communicate the Registrar's support for this extension to the registering User Agent.

12.4. Response Code

This section registers a new SIP Response Code, as per the guidelines in Section 27.4 of RFC 3261.

Code: 430
Default Reason Phrase: Flow Failed
Reference: This document

12.5. Media Feature Tag

This section registers a new media feature tag, per the procedures defined in RFC 2506 [18]. The tag is placed into the sip tree, which is defined in RFC 3840 [7].

Media feature tag name: sip.instance

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag contains a string containing a URN that indicates a unique identifier associated with the UA instance registering the Contact.

Values appropriate for use with this feature tag: String.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Routing a call to a specific device.

Related standards or documents: RFC XXXX

[[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

Security Considerations: This media feature tag can be used in ways which affect application behaviors. For example, the SIP caller preferences extension [9] allows for call routing decisions to be based on the values of these parameters. Therefore, if an attacker can modify the values of this tag, they might be able to affect the behavior of applications. As a result, applications which utilize this media feature tag SHOULD provide a means for ensuring its integrity. Similarly, this feature tag should only be trusted as valid when it comes from the user or user agent described by the tag. As a result, protocols for conveying this feature tag SHOULD provide a mechanism for guaranteeing authenticity.

13. Security Considerations

One of the key security concerns in this work is making sure that an attacker cannot hijack the sessions of a valid user and cause all calls destined to that user to be sent to the attacker. Note that the intent is not to prevent existing active attacks on SIP UDP and TCP traffic, but to insure that no new attacks are added by introducing the outbound mechanism.

The simple case is when there are no edge proxies. In this case, the only time an entry can be added to the routing for a given AOR is when the registration succeeds. SIP already protects against attackers being able to successfully register, and this scheme relies on that security. Some implementers have considered the idea of just saving the instance-id without relating it to the AOR with which it registered. This idea will not work because an attacker's UA can impersonate a valid user's instance-id and hijack that user's calls.

The more complex case involves one or more edge proxies. When a UA sends a REGISTER request through an Edge Proxy on to the registrar, the Edge Proxy inserts a Path header field value. If the registration is successfully authenticated, the registrar stores the value of the Path header field. Later when the registrar forwards a request destined for the UA, it copies the stored value of the Path header field into the Route header field of the request and forwards the request to the Edge Proxy.

The only time an Edge Proxy will route over a particular flow is when it has received a Route header that has the flow identifier information that it has created. An incoming request would have gotten this information from the registrar. The registrar will only save this information for a given AOR if the registration for the AOR has been successful; and the registration will only be successful if the UA can correctly authenticate. Even if an attacker has spoofed some bad information in the Path header sent to the registrar, the attacker will not be able to get the registrar to accept this information for an AOR that does not belong to the attacker. The registrar will not hand out this bad information to others, and others will not be misled into contacting the attacker.

14. Operational Notes on Transports

This entire section is non-normative.

RFC 3261 requires proxies, registrars, and User Agents to implement both TCP and UDP but deployments can chose which transport protocols they want to use. Deployments need to be careful in choosing what transports to use. Many SIP features and extensions, such as large presence notification bodies, result in SIP requests that can be too large to be reasonably transported over UDP. RFC 3261 states that when a request is too large for UDP, the device sending the request attempts to switch over to TCP. No known deployments currently use this feature but it is important to note that when using outbound, this will only work if the UA has formed both UDP and TCP outbound flows. This specification allows the UA to do so but in most cases it will probably make more sense for the UA to form a TCP outbound connection only, rather than forming both UDP and TCP flows. One of the key reasons that many deployments choose not to use TCP has to do with the difficulty of building proxies that can maintain a very large number of active TCP connections. Many deployments today use SIP in such a way that the messages are small enough that they work over UDP but they can not take advantage of all the functionality SIP offers. Deployments that use only UDP outbound connections are going to fail with sufficiently large SIP messages.

15. Requirements

This specification was developed to meet the following requirements:

1. Must be able to detect that a UA supports these mechanisms.
2. Support UAs behind NATs.
3. Support TLS to a UA without a stable DNS name or IP address.
4. Detect failure of a connection and be able to correct for this.
5. Support many UAs simultaneously rebooting.
6. Support a NAT rebooting or resetting.
7. Minimize initial startup load on a proxy.
8. Support architectures with edge proxies.

16. Changes

Note to RFC Editor: Please remove this whole section.

16.1. Changes from 08 Version

UAs now include the 'ob' parameter in their Contact header for non-REGISTER requests, as a hint to the Edge Proxy (so the EP can Record-Route with a flow-token for example).

Switched to CRLF for keepalives of connection-oriented transports after brutal consensus at IETF 68.

Added timed-keepalive parameter and removed the unnecessary keep-tcp param, per consensus at IETF68.

Removed example "Algorithm 1" which only worked over SIPS, per consensus at IETF68.

Deleted text about probing and validating with options, per consensus at IETF68.

Deleted provision for waiting 120 secs before declaring flow stable, per consensus at IETF68.

fixed example UUIDs

16.2. Changes from 07 Version

Add language to show the working group what adding CRLF keepalives would look like.

Changed syntax of keep-alive=stun to keep-stun so that it was easier to support multiple tags in the same URI.

16.3. Changes from 06 Version

Added the section on operational selection of transports.

Fixed various editorial typos.

Put back in requirement flow token needs to be unique to flow as it had accidentally been dropped in earlier version. This did not change any of the flow token algorithms.

Reordered some of the text on STUN keepalive validation to make it clearer to implementors. Did not change the actual algorithm or requirements. Added note to explain how if the proxy changes, the revalidation will happen.

16.4. Changes from 05 Version

Mention the relevance of the 'rport' parameter.

Change registrar verification so that only first-hop proxy and the registrar need to support outbound. Other intermediaries in between do not any more.

Relaxed flow-token language slightly. Instead of flow-token saving specific UDP address/port tuples over which the request arrived, make language fuzzy to save token which points to a 'logical flow' that is known to deliver data to that specific UA instance.

Added comment that keep-stun could be added to Path.

Added comment that battery concerns could motivate longer TCP keepalive intervals than the defaults.

Scrubbed document for avoidable lowercase may, should, and must.

Added text about how Edge Proxies could determine they are the first hop.

16.5. Changes from 04 Version

Moved STUN to a separate section. Reference this section from within the relevant sections in the rest of the document.

Add language clarifying that UA MUST NOT send STUN without an explicit indication the server supports STUN.

Add language describing that UA MUST stop sending STUN if it appears the server does not support it.

Defined a 'sip-stun' option tag. UAs can optionally probe servers for it with OPTIONS. Clarified that UAs SHOULD NOT put this in a Proxy-Require. Explain that the first-hop MUST support this option-tag.

Clarify that SIP/STUN in TLS is on the "inside". STUN used with Sigcomp-compressed SIP is "outside" the compression layer for UDP, but wrapped inside the well-known shim header for TCP-based transports.

Clarify how to decide what a consecutive registration timer is. Flow must be up for some time (default 120 seconds) otherwise previous registration is not considered successful.

Change UAC MUST-->SHOULD register a flow for each member of outbound-proxy-set.

Reworded registrar and proxy in some places (introduce the term "Authoritative Proxy").

Loosened restrictions on always storing a complete Path vector back to the registrar/authoritative proxy if a previous hop in the path vector is reachable.

Added comment about re-registration typically happening over same flow as original registration.

Changed 410 Gone to new response code 430 Flow Failed. Was going to change this to 480 Temporarily Unavailable. Unfortunately this would mean that the authoritative proxy deletes all flows of phones who use 480 for Do Not Disturb. Oops!

Restored sanity by restoring text which explains that registrations with the same reg-id replace the old registration.

Added text about the 'ob' parameter which is used in Path header field URIs to make sure that the previous proxy that added a Path understood outbound processing. The registrar doesn't include Supported: outbound unless it could actually do outbound processing (ex: any Path headers have to have the 'ob' parameter).

Added some text describing what a registration means when there is an instance-id, but no reg-id.

16.6. Changes from 03 Version

Added non-normative text motivating STUN vs. SIP PING, OPTIONS, and Double CRLF. Added discussion about why TCP Keepalives are not

always available.

Explained more clearly that outbound-proxy-set can be "configured" using any current or future, manual or automatic configuration/ discovery mechanism.

Added a sentence which prevents an Edge Proxy from forwarding back over the flow over which the request is received if the request happens to contain a flow token for that flow. This was an oversight.

Updated example message flow to show a fail-over example using a new dialog-creating request instead of a mid-dialog request. The old scenario was leftover from before the outbound / gruu reorganization.

Fixed tags, Call-IDs, and branch parameters in the example messages.

Made the ABNF use the "=/" production extension mechanism recommended by Bill Fenner.

Added a table in an appendix expanding the default flow recovery timers.

Incorporated numerous clarifications and rewordings for better comprehension.

Fixed many typos and spelling steaks.

16.7. Changes from 02 Version

Removed Double CRLF Keepalive

Changed ;sip-stun syntax to ;keepalive=stun

Fixed incorrect text about TCP keepalives.

16.8. Changes from 01 Version

Moved definition of instance-id from GRUU[25] draft to this draft.

Added tentative text about Double CRLF Keepalive

Removed pin-route stuff

Changed the name of "flow-id" to "reg-id"

Reorganized document flow

Described the use of STUN as a proper STUN usage

Added 'outbound' option-tag to detect if registrar supports outbound

16.9. Changes from 00 Version

Moved TCP keepalive to be STUN.

Allowed SUBSCRIBE to create flow mappings. Added pin-route option tags to support this.

Added text about updating dialog state on each usage after a connection failure.

17. Acknowledgments

Jonathan Rosenberg provided many comments and useful text. Dave Oran came up with the idea of using the most recent registration first in the proxy. Alan Hawrylyshen co-authored the draft that formed the initial text of this specification. Additionally, many of the concepts here originated at a connection reuse meeting at IETF 60 that included the authors, Jon Peterson, Jonathan Rosenberg, Alan Hawrylyshen, and Paul Kyzivat. The TCP design team consisting of Chris Boulton, Scott Lawrence, Rajnish Jain, Vijay K. Gurbani, and Ganesh Jayadevan provided input and text. Nils Ohlmeier provided many fixes and initial implementation experience. In addition, thanks to the following folks for useful comments: Francois Audet, Flemming Andreasen, Mike Hammer, Dan Wing, Srivatsa Srinivasan, Dale Worely, Juha Heinanen, Eric Rescorla, Lyndsay Campbell, Christer Holmberg, Kevin Johns, and Erkki Koivusalo.

Appendix A. Default Flow Registration Backoff Times

The base-time used for the flow re-registration backoff times described in Section 4.5 are configurable. If the base-time-all-fail value is set to the default of 30 seconds and the base-time-not-failed value is set to the default of 90 seconds, the following table shows the resulting delay values.

# of reg failures	all flows unusable	>1 non-failed flow
0	0 secs	0 secs
1	30-60 secs	90-180 secs
2	1-2 mins	3-6 mins
3	2-4 mins	6-12 mins
4	4-8 mins	12-24 mins
5	8-16 mins	15-30 mins
6 or more	15-30 mins	15-30 mins

18. References

18.1. Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Rosenberg, J., "Simple Traversal Underneath Network Address Translators (NAT) (STUN)", draft-ietf-behave-rfc3489bis-05 (work in progress), October 2006.

[4] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[5] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[6] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[7] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[8] Moats, R., "URN Syntax", RFC 2141, May 1997.

[9] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[10] Rosenberg, J. and H. Schulzrinne, "An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing", RFC 3581, August 2003.

[11] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[12] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003.

[13] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[14] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Sy ntax", STD 66, RFC 3986, January 2005.

[15] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[16] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", BCP 98, RFC 3968, December 2004.

[17] Camarillo, G., "The Internet Assigned Number Authority (IANA) Uniform Resource Identifier (URI) Parameter Registry for the Session Initiation Protocol (SIP)", BCP 99, RFC 3969, December 2004.

[18] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, March 1999.

18.2. Informative References

[19] Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-09 (work in progress), October 2006.

[20] Hakala, J., "Using National Bibliography Numbers as Uniform Resource Names", RFC 3188, October 2001.

[21] Rosenberg, J., "Construction of the Route Header Field in the Session Initiation Protocol (SIP)", draft-rosenberg-sip-route-construct-02 (work in progress), October 2006.

[22] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During

Registration", RFC 3608, October 2003.

[23] Boulton, C., "Best Current Practices for NAT Traversal for SIP", draft-ietf-sipping-nat-scenarios-05 (work in progress), June 2006.

[24] Price, R., Bormann, C., Christoffersson, J., Hannu, H., Liu, Z., and J. Rosenberg, "Signaling Compression (SigComp)", RFC 3320, January 2003.

[25] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-11 (work in progress), October 2006.

Authors' Addresses

Cullen Jennings (editor)
Cisco Systems
170 West Tasman Drive
Mailstop SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 902-3341
Email: fluffy@cisco.com

Rohan Mahy (editor)
Plantronics
345 Encincal St
Santa Cruz, CA 95060
USA

Email: rohan@ekabal.com

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).