Network Working Group
Internet-Draft
Updates: 3261,3327 (if approved)
Expires: September 6, 2006

C. Jennings, Ed.
Cisco Systems
R. Mahy, Ed.
Plantronics
March 5, 2006

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

draft-ietf-sip-outbound-02

Status of this Memo

Copyright Notice

Abstract

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. Even when a proxy server can open a TCP connection to a User Agent,

most User Agents lack a certificate suitable to act as a TLS (Transport Layer Security) server. 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. Keep Alive Technique 10
4. User Agent Mechanisms 10
4.1. Instance ID Creation 10
4.2. Initial Registrations 12
4.2.1. Registration by Other Instances 13
4.3. Sending Requests 13
4.3.1. Selecting the First Hop 13
4.3.2. Forming Flows 13
4.4. Detecting Flow Failure 14
4.4.1. Keep Alive with STUN 14
4.4.2. Keep Alive with Double CRLF 15
4.5. Flow Recovery 15
5. Edge Proxy Mechanisms 16
5.1. Processing Register Requests 16
5.2. Generating Flow Tokens 16
5.3. Forwarding Requests 17
6. Registrar and Location Server Mechanisms 17
6.1. Processing Register Requests 18
6.2. Forwarding Requests 19
7. Mechanisms for All Servers (Proxys, Registars, UAS) 19
7.1. STUN Processing 19
7.2. Double CRLF Processing 20
8. Example Message Flow 20
9. Grammar 23
10. IANA Considerations 24
10.1. Contact Header Field 24
10.2. SIP/SIPS URI Paramters 24
10.3. SIP Option Tag 24
10.4. Media Feature Tag 25
11. Security Considerations 26

12. Open Issues 26
13. Requirements 27
14. Changes 27
14.1. Changes from 01 Version 27
14.2. Changes from 00 Version 27
15. Acknowledgments 27
16. References 28
16.1. Normative References 28
16.2. Informative References 29
Authors' Addresses 30
Intellectual Property and Copyright Statements 31

1. Introduction

There are many environments for SIP [5] deployments in which the User Agent (UA) can form a connection to a Registrar or Proxy but in which the connections in the reverse direction to the UA are not possible. This can happen for several reasons. Connection 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 there may be a NAT, which are 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 definitely do not have a long-term, stable DNS name that is appropriate for binding to a certificate. It is impractical for them to have a certificate that can be used as a client-side TLS certificate for SIP. However, these systems can still form TLS 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 mechanism also uses a keep alive mechanism over each flow so that the UA can 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 [4].

2.1. Definitions

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

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 configured 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.

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.

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 use the STUN (Simple Traversal of UDP through NATs) protocol as the keep alive mechanism to keep their flow to the proxy or registrar alive.

3.2. Single Registrar and UA

In this example, 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 is used to allow the registrar to detect and avoid using 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 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:00000000-0000-0000-0000-000A95A0E128>"
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 and reg-id 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 proxy uses the most recently created registration first. This allows a UA that has rebooted to replace its previous registration for each flow with minimal impact on overall system load.

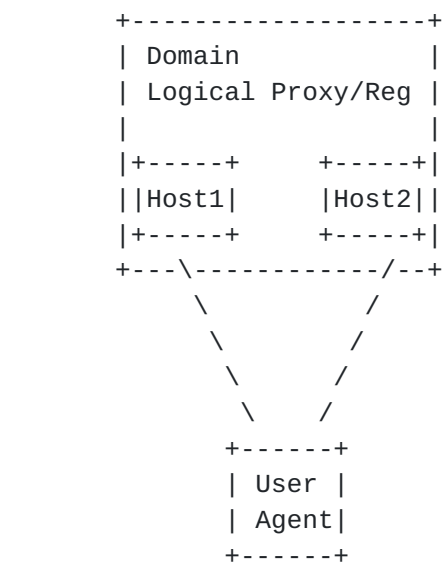
When Alice sends a request to Bob, his proxy selects the target set. The proxy forwards the request to elements in the target set based on the proxy's policy. The proxy looks at the target set and uses the instance-id to understand that two targets both end up routing to the same UA. When the proxy goes to forward a request to a given target, it looks and finds the flows that received the registration. The proxy then forwards the request on that flow instead of 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 registration bindings for this AOR that has the same instance-id as the selected UA. In general, if two registrations have the same reg-id and instance-id, the proxy will favor the most recently registered flow. This is so that if a UA reboots, the proxy will prefer to use the most recent flow that goes to this UA instead of trying one of the old flows which would presumably fail.

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 this example system, the logical proxy/registrar for the domain is running on two hosts that share the appropriate state and can both provide registrar and proxy functionality for the domain. The UA will form connections to two of the physical hosts that can perform the 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 backup 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 backup hosts. Designs having only one set were also considered, but in this

case there would have to be some way to ensure that the two connection did not accidentally resolve to the same host. Various approaches for this are possible but all probably require extensions to the SIP protocol so they were not included in this specification. This approach can work with the disadvantage that slightly more configuration of DNS is required.



The UA is configured with a primary and backup registration URI. 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;sip-stun" and "sip: backup.example.com;sip-stun". The "sip-stun" tag indicates that a SIP server supports STUN and SIP muxed 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 things like SRV can be used to do load balancing across a proxy farm.

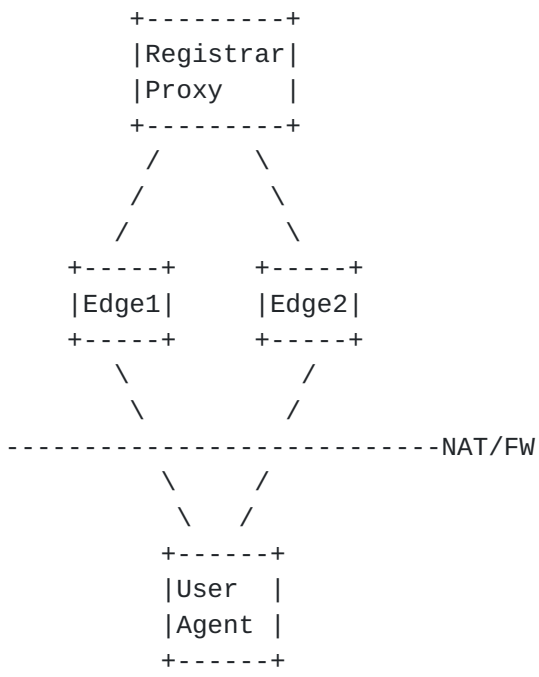
The domain also needs to ensure that a request for the UA sent to host1 or host2 is then sent across the appropriate flow to the UA. The domain might choose to use the Path header (as described in the next section) approach to store this internal routing information on host1 or host2.

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, it can delay some significant 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.

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 [12] 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 can be done by putting the value 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 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 [12].

3.5. Keep Alive Technique

A keep alive mechanism needs to detect failure of a connection and changes to the NAT public mapping, as well as keeping any NAT bindings refreshed. This specification uses STUN [7] over the same flow as the SIP traffic to perform the keep alive. 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, requests are sent over the same flow that is being used for the SIP traffic. The proxy or registrar acts as a STUN 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. It may also be possible to detect a changes flow with 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. The TCP KEEP_ALIVE mechanism was not used because most operating systems do not allow the time to be set on a per connection basis. Linux, Solaris, OS X, and Windows all allow KEEP_ALIVEs to be turned on or off on a single socket using the SO_KEEPALIVE socket options but can not change the duration of the timer for an individual socket. The length of the timer typically defaults to 7200 seconds. The length of the timer can be changed to a smaller value by setting a kernel parameter but that affects all TCP connections on the host and thus is not appropriate to use.

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.

4. User Agent Mechanisms

4.1. Instance ID Creation

Each UA MUST have an Instance Identifer 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 persitant across power cylces of the device.

A UA SHOULD use a UUID URN [9]. 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 [9] 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 [15]. 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.

The UA SHOULD include a "sip.instance" media feature tag as a UA characteristic [10] in requests and responses. As described in [10], 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 [3]. 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 anoymous request or some other privacy concern requires that the UA not reveal its identity.

> RFC 3840 [10] 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 [10] and in the caller preferences specification [2]. 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 [3]. Lexical equality may 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 lexigraphically different to previous registrations.

4.2. Initial Registrations

UAs are configured with one or more SIP URIs representing the default outbound-proxy-set. The specification assumes the set is determined via configuration but future specifications may define other 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 (primary and backup) and SHOULD support sets with up to four URIs. For each outbound proxy URI in the set, the UA MUST 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 [20].

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

These ordinary registration requests MUST also add a distinct reg-id parameter to the Contact header field. Each one of these registrations will form a new flow from the UA to the proxy. The reg-id sequence does not have to be sequential but MUST be exactly the same reg-id sequence each time the device power cycles or reboots so that the reg-id values will collide with the previously used reg-id values. This is so the proxy can realize that the older registrations are probably not useful.

The UAC MUST indicate that it supports the Path header [12] 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.

Note that the UA needs to honor 503 responses to registrations as described in RFC 3261 and RFC 3263 [6]. In particular, implementors should note that when receiving a 503 response with a Retry-After header field, the UA should wait the indicated amount of time and retry the registration. A Retry-After header field value of 0 is valid and indicates the UA should 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.

4.2.1. Registration by Other Instances

A User Agent MUST NOT include an instance-id or reg-id 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 deregister all Contacts (a single Contact header field value with the value of "*").

4.3. Sending Requests

As described in Section 4.1, all requests need to include the instance-id media feature tag unless privacy concerns require otherwise.

4.3.1. Selecting the First Hop

When an 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 [21] (Service Route) and [20].

4.3.2. Forming Flows

The UA performs normal DNS resolution on the next hop URI (as described in RFC 3263 [6]) to find a protocol, IP address, and port. For non TLS protocols, 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, the existing flow is only used if, in addition to matching the IP address, port, and protocol, the host production in the next hop URI MUST match 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.

4.4. Detecting Flow Failure

The UA needs to detect when a specific flow fails. If a flow has failed, the UA follows the procedures in Section 4.2 to form a new flow to replace the failed one. The UA proactively tries to detect failure by periodically sending keep alive messages using one of the techniques described in this section.

The time between keep alive requests when using UDP based transports SHOULD be a random number between 24 and 29 seconds while for TCP based transports it SHOULD be a random number between 95 and 120 seconds. These times MAY be configurable.

- 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 the keep alive requests unsynchronized to evenly spread the load on the servers. For TCP, the 120 seconds was chosen based on the idea that for a good user experience, failures should 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 may 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.

4.4.1. Keep Alive with STUN

User Agents that form flows MUST check if the configured URI they are connecting to has the "sip-stun" URI parameter (defined in Section 10). If the parameter is present, the UA needs to periodically perform keep alive checks by sending a STUN [7] Binding Requests over the flow.

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

4.4.2. Keep Alive with Double CRLF

User Agents that form flows MUST check if the configured URI they are connecting to has the "crlf-ping" URI parameter (defined in Section 10). If the parameter is present, the UA needs to send keep alive requests by sending a CRLF over the flow.

If the UA does not receive any data back over the flow within 7 seconds of sending the CRLF, then it MUST consider the lack of response to be a flow failure.

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 4.3.2; 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 time to wait is computed in the following way. If all of the flows to every URI in the 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 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( 1800, (base-time * (2 ^ consecutive-failures)))
```

These three times MAY be configurable in the UA. The three times are the max-time with a default of 1800 seconds, the base-time-all-fail with a default of 30 seconds, and the base-time-not-failed with a default of 60 seconds. For example if the base time was 30 seconds, and there had been three failures, then the wait time would be 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 forever with a random time between 900 and 1800 seconds between the attempts.

5. Edge Proxy Mechanisms

5.1. Processing Register Requests

When an Edge Proxy receives a registration request with a sip.instance media feature tag in the Contact header field, it MUST form a flow identifier token that is unique to this network flow. The Edge Proxy MUST insert this token into a URI referring to this proxy and place this URI into a Path header field as described in RFC 3327 [12]. The token MAY be placed in the userpart of the URI.

5.2. Generating Flow Tokens

A trivial but impractical way to satisfy the flow token requirement 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. Two stateless examples are 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 can not be modified by attackers.

Algorithm 1: The proxy generates a flow token for connection-oriented transports by concatenating the file descriptor (or equivalent) with the NTP time the connection was created, and base64 encoding the result. This results in an approximately 16 octet identifier. The proxy generates a flow token for UDP by concatenating the file descriptor and the remote IP address and port, then base64 encoding the result. This algorithm MUST NOT be used unless all messages between the Edge proxy and Registrar use a SIPS protected transport. If the SIPS level of integrity protection is not available, an attacker can hijack another user's calls.

Algorithm 2: When the proxy boots it selects a 20 byte 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 [16]. The concatenation of the HMAC and S are base64 encoded, as defined in [18], and used as the flow identifier. When using IPv4 addresses, this will result in a

32 octet identifier.

5.3. Forwarding Requests

When the Edge Proxy receives a request, it applies normal routing procedures with the following addition. If the top-most Route header refers to the Edge Proxy and contains a valid flow identifier token created by this proxy, the proxy MUST forward the request over the flow that received the REGISTER request that caused the flow identifier token to be created. For connection-oriented transports, if the flow no longer exists the proxy SHOULD send a 410 response to the request.

The advantage to a stateless approach to managing the flow information is that there is no state on the edge proxy that requires clean up or that has to be synchronized with the registrar.

Proxies which used one of the two algorithms described in this document to form a flow token follow the procedures below to determine the correct flow.

Algorithm 1: The proxy base64 decodes the user part of the Route header. For TCP, if a connection specified by the file descriptor is present and the creation time of the file descriptor matches the creation time encoded in the Route header, the proxy forwards the request over that connection. For UDP, the proxy forwards the request from the encoded file descriptor to the source IP address and port.

Algorithm 2: 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 it matches. If the HMAC is not correct, the proxy SHOULD send a 403 response. If the HMAC was 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 410 response to the request.

Note that techniques to ensure that mid-dialog requests are routed over an existing flow are out of scope and therefore 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.

6. Registrar and Location Server Mechanisms

6.1. Processing Register Requests

This specification updates the definition of a binding in RFC 3261 [5] Section 10 and RFC 3327 [12] Section 5.3.

When no instance-id 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 an instance-id is present in a Contact header field value in a REGISTER request, the corresponding binding is between an AOR and the combination of instance-id and reg-id. 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 (the Contact URI is not part of the "key" for the binding). The registrar MUST be prepared to receive, simultaneously for the same AOR, some registrations that use instance-id and reg-id and some that do not.

Registrars which implement this specification, MUST support the Path header mechanism [12].

In addition to the normal information stored in the binding record, some additional information MUST be stored for any registration that contains a reg-id header parameter in the Contact header field value. 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 file descriptor. 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 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 [12] requires the registrar to store this information as well. If the registrar receives a re-registration, it MUST update the information that uniquely identifies the network flow over which the request arrived and SHOULD update the time the binding was last updated.

The Registrar MUST include the 'outbound' option-tag in a Supported header field value in its responses to REGISTER requests. The Registrar MAY be configured with local policy to reject any registrations that do not include the instance-id and reg-id to eliminate the amplification attack described in [19]. 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.

6.2. 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 410 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 two bindings have the same instance-id and reg-id, the proxy SHOULD prefer the contact that was most recently updated.

The proxy uses normal forwarding rules looking at the Route 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. Additionally, when the proxy forwards a request to a binding that contains a reg-id, the proxy MUST send the request over the same network flow that was saved with the binding. This means that for TCP, the request MUST be sent on the same TCP socket that received the REGISTER request. For UDP, the request MUST be 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 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 with flows that use that file descriptor.

7. Mechanisms for All Servers (Proxys, Registars, UAS)

A SIP device that receives SIP messages directly from a UA needs to behave as specified in this section. Such devices would generally include a Registrar and an Edge Proxy, as they both receive register requests directly from a UA.

7.1. STUN Processing

This document defines a new STUN usage for inband connectivity

checks. The only STUN messages required by this usage are Binding Requests, Binding Responses, and Error Responses. The UAC sends Binding Requests over the same UDP flow, TCP connection, or TLS channel used for sending SIP messages, once a SIP registration has been successfully processed on that flow. 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. After a successful STUN response is received over TCP or TLS over TCP, the underlying TCP connection is left in the active state.

If the server 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. Specifically it MUST be capable of receiving and responding to STUN Binding Requests.

> It is easy to distinguish STUN and SIP packets 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 device that supports STUN as described in this section, the URI parameter "sip-stun", as defined in Section 10 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. The sip-stun tag typically would be present in the URI in the Route header field value of a REGISTER request and not be in the Request URI.

7.2. Double CRLF Processing

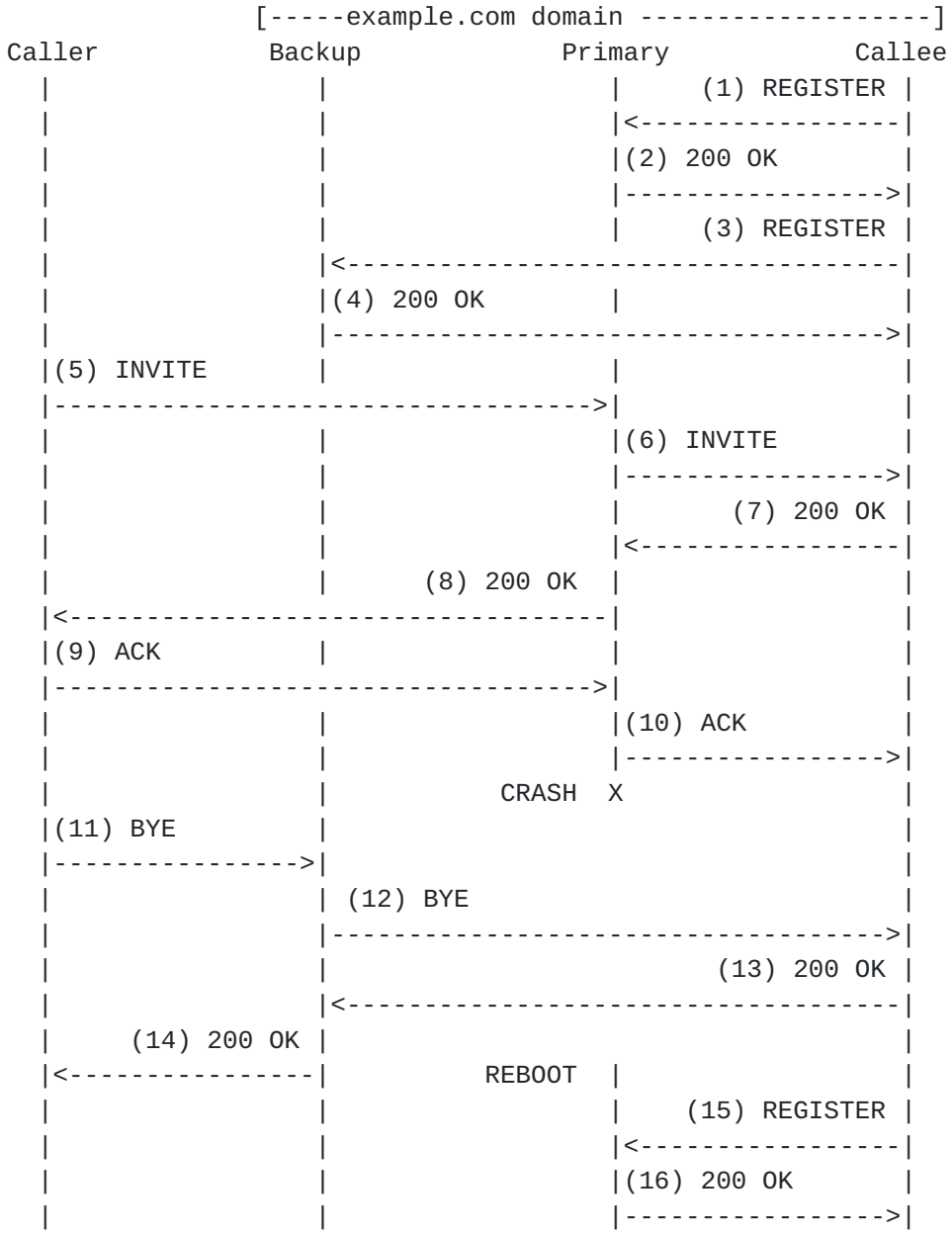
If the SIP server is acting as the TCP client and initiated the TCP connection (meaning that this host did the active open), then the SIP server MUST NOT perform any of the processing in this section. The following only applies when the SIP server is acting as the TCP server (meaning that this host did the passive open).

When the server receives a CRLF before the start line of a message on a flow, it MUST send some data back on that same flow within 3 seconds. If no message is actively being sent, it SHOULD send back a CRLF after waiting at least 1 second. The reason for waiting at least 1 second is that if the other end has an incorrect implementation and incorrectly echoes the CRLF, this will stop the flow from going into a live-lock state.

8. Example Message Flow

The following call flow shows a basic registration and an incoming

call. Part way through the call, the flow to the Primary proxy is lost. The BYE message for the call is rerouted to the callee via the Backup proxy. When connectivity to the primary proxy is established, the Callee registers again to replace the lost flow as shown in message 15.



This call flow assumes that the Callee has been configured with a proxy set that consists of "sip:primary.example.com;lr;sip-stun" and "sip:backup.example.com;lr;sip-stun". The Callee REGISTER in message (1) looks like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 10.0.1.1;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com>
Call-ID: 1j9FpLxk3uxtm8tn@10.0.1.1
CSeq: 1 REGISTER
Supported: path
Route: <sip:primary.example.com;lr;sip-stun>
Contact: <sip:callee@10.0.1.1>
  ;+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 10.0.1.1;branch=z9hG4bKnashds7
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com> ;tag=b88sn
Call-ID: 1j9FpLxk3uxtm8tn@10.0.1.1
CSeq: 1 REGISTER
Supported: outbound
Contact: <sip:callee@10.0.1.1>
  ;+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 backup instead of the primary. They look like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 10.0.1.1;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com>
Call-ID: 1j9FpLxk3uxtm8tn-2@10.0.1.1
CSeq: 1 REGISTER
Supported: path
Route: <sip:backup.example.com;lr;sip-stun>
Contact: <sip:callee@10.0.1.1>
  ;+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 10.0.1.1;branch=z9hG4bKnashds7
  From: Callee <sip:callee@example.com>;tag=a73kszlfl
  To: Callee <sip:callee@example.com> ;tag=b88sn
  Call-ID: 1j9FpLxk3uxtm8tn-2@10.0.1.1
  Supported: outbound
  CSeq: 1 REGISTER
  Contact: <sip:callee@10.0.1.1>
    ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
    ;reg-id=1
    ;expires=3600
  Contact: <sip:callee@10.0.1.1>
    ;+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 6 contains the following Record-Route header field:

```
Record-Route: <sip:example.com;lr>
```

Message 11 seems seams strange in that it goes to the backup instead of the primary. The Caller actually sends the message to the domain of the callee to a host (primary or backup) that is currently available. How the domain does this is an implementation detail up to the domain and not part of this specification.

The registrations in message 15 and 16 are the same as message 1 and 2 other than the Call-ID has changed.

9. Grammar

This specification defines new Contact header field parameters, reg-id and +sip.instance. The grammar includes the definitions from RFC 3261 [5] and includes the definition of uric from RFC 2396 [11]. The ABNF[8] is:

```
contact-params = c-p-q / c-p-expires / c-p-flow / c-p-instance
                 / contact-extension

c-p-flow       = "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 2396
```

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

10. IANA Considerations

10.1. 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 [13] . The required information is:

Header Field	Parameter Name	Predefined Values	Reference
Contact	reg-id	Yes	[RFC AAAA]

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

10.2. SIP/SIPS URI Paramters

This specification arguments the "SIP/SIPS URI Parameters" sub-registry as per the registry created by [14] . The required information is:

Parameter Name	Predefined Values	Reference
sip-stun	No	[RFC AAAA]
crlf-ping	No	[RFC AAAA]

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

10.3. 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 to the registering User Agent the Registrars support for this extension.

10.4. Media Feature Tag

This section registers a new media feature tag, per the procedures defined in RFC 2506 [1]. The tag is placed into the sip tree, which is defined in RFC 3840 [10].

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 [23] 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 may 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.

11. 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.

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 proxy 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.

12. Open Issues

Do we want to include the Double CRLF keep alive option?

Are thre any deployments that could use Algorithm 1 and if not can we remove it?

We should change syntax from "sip-stun" to "keep-alive=sip-stun".

13. 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 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.

14. Changes

Note to RFC Editor: Please remove this whole section.

14.1. Changes from 01 Version

Moved definition of instance-id from GRUU[17] draft to this draft.

Added tentative text about Double CRLF Keep Alive

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

14.2. Changes from 00 Version

Moved TCP keep alive 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.

15. 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, and Lyndsay Campbell.

16. References

16.1. Normative References

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

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

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

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

[5] 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.

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

[7] Rosenberg, J., "Simple Traversal of UDP Through Network Address Translators (NAT) (STUN)", draft-ietf-behave-rfc3489bis-02 (work in progress), July 2005.

[8] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

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

[10] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating

User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[11] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

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

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

[14] 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.

16.2. Informative References

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

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

[17] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-04 (work in progress), July 2005.

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

[19] Lawrence, S., Hawrylyshen, A., and R. Sparks, "Problems with Max-Forwards Processing (and Potential Solutions)", October 2005.

[20] Rosenberg, J., "Clarifying Construction of the Route Header Field in the Session Initiation Protocol (SIP)", draft-rosenberg-sip-route-construct-00 (work in progress), July 2005.

[21] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration", RFC 3608, October 2003.

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

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.