SIP WG
Internet-Draft
Expires: April 26, 2006

C. Jennings, Ed.
Cisco Systems
R. Mahy, Ed.
SIP Edge LLC
October 23, 2005

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

draft-ietf-sip-outbound-01

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 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 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 for high availability systems.

Table of Contents

1. Introduction

There are many environments for SIP 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 a firewall or NAT.

Most IP phones and personal computers get their network configurations dynamically via a protocol such as DHCP. These systems typically do not have a useful name in 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 such that the UA authenticates the server certificate, and the server authenticates 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 connection, be it UDP, TCP, or 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 the server. Since the server can't connect to the UA, the UA has to make sure that a connection is always active. This requires the UA to detect when a connection fails. Since, such detection takes time and leaves a window of opportunity for missed incoming requests, this mechanism allows the UA to use multiple connections, referred to as "flows", to the proxy or registrar and using a keep alive mechanism on 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 [2].

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 connection between two hosts that is represented by the network address of both ends and the protocol. For TCP and UDP this would include the IP addresses and ports of both ends and the protocol (TCP or UDP). With TCP, a flow would often have a one to one correspondence with a single file descriptor in the operating system.

flow-id: This refers to the value of a new header field parameter value for the contact header. When a UA register multiple times, each registration gets a unique flow-id value. This does not refer to flow.

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 as defined in [1]. This is a URN that uniquely identifies the UA.

3. Overview

Several scenarios in which this technique is useful are discussed below, including the simple collocated 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 (found in the GRUU[1]) that stays the same for this UA even if the UA reboots or is power cycled. Each UA can register multiple times for the same AOR to achieve high reliability. Each registration includes the instance-id for the UA and a flow-id label that is different for each connection.

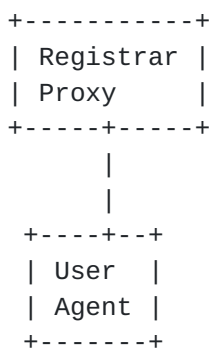
UAs use STUN as the keep alive mechanism to keep their flow to the proxy or registrar alive. A UA can create more than one flow using multiple registrations for the same AOR. The instance-id parameter is used by the proxy to identify which UA a flow is associated with. The flow-id is used by the proxy and registrar to tell the difference between a UA re-registering and one that is registering over an additional flow. The proxies keep track of the flows used for successful registrations.

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 looking at the instance-id. Proxies can

tell that a flow replaces a previously abandoned flow by looking at the flow-id.

3.2. Single Registrar and UA

In this example there a single server is acting as both a registrar and proxy.



User Agents forming only a single connection continue to register normally but include the instance-id as described in the GRUU [1] specification and can also add a flow-id parameter to the Contact header field value. The flow-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/UDP 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
Contact: <sip:line1@192.168.0.2>; flow-id=1;
  ;+sip.instance="<urn:uuid:00000000-0000-0000-0000-000A95A0E128>"
Content-Length: 0
```

Note: Implementors often ask why the value of the sip.instance is inside angle brackets. This is a requirement of RFC 3840 [7] which defines media feature tags in SIP. Feature tags which are strings are compared by case sensitive string comparison. To differentiate these tags from tokens (which are not case sensitive), case sensitive parameters such as the sip.instance

media feature tag are placed inside angle brackets.

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 (as defined in [1]) and flow-id (as defined in Section 9) along with the rest of the Contact header field. If the instance-id and flow-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.

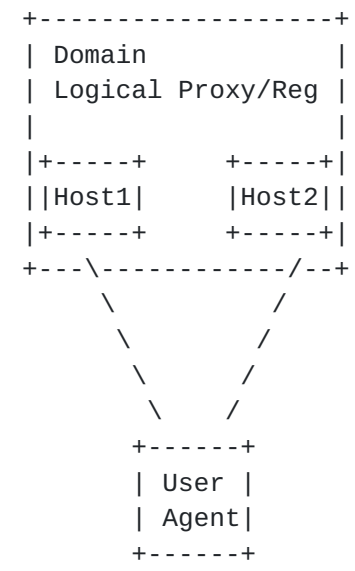
Later when Alice sends a request to Bob, his proxy selects the target set. The proxy forwards the request to elements in the target set based on the proxy's policy. The proxy looks at the the target set and uses the instance-id to understand that two targets both end up routing to the same UA. When the 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 down the same flow that did the registration for this AOR. If the proxy had multiple flows that all went to this UA, it would choose any one of registration bindings that it had for this AOR and that had the same instance-id as the selected UA. In general, if two registrations have the same flow-id and instance-id, the proxy would 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 will presumably fail.

3.3. Multiple Connections from a User Agent

There are various ways to deploy SIP to build a reliable and scaleable system. This section discusses one such design that is possible with the mechanisms in this draft. 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 connections to the domain. Scaleability 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 be configured with an entry that resolves to all the primary hosts and another 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.



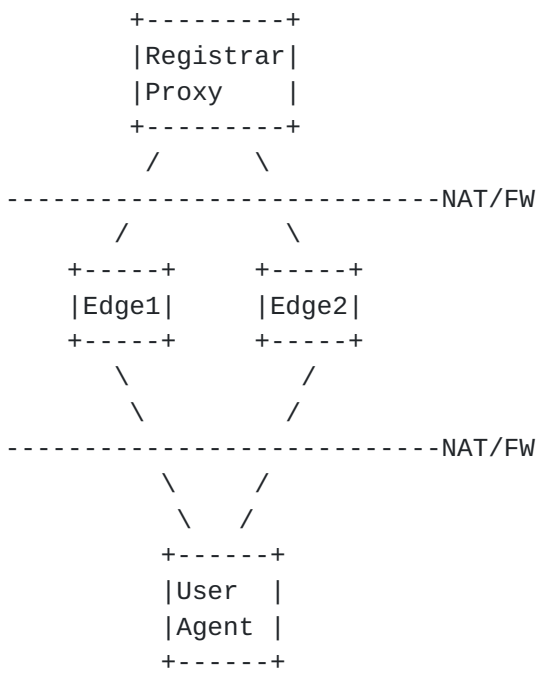
The UA is configured with a primary and backup registration URI. These URIs are configured into the UA through whatever the normal mechanism is to configure the proxy or registrar for the UA. They might look something like "sip:primary.example.com;sip-stun" and "sip:backup.example.com;sip-stun" if the domain was example.com. The "sip-stun" tag indicates that they support STUN as described later in this specification. Note that each of them could resolve to several different hosts. The administrative domain that created these URIs MUST ensure that the two URIs resolve to separate hosts. These URIs have normal SIP processing so things like SRV can be used to do load balancing across a proxy farm.

The User Agent would get a GRUU from the domain to use at its contact. The GRUU would refer to the domain, not host1 or host2. Regardless of which host received a request to GRUU, the domain would need to ensure that the request got sent to host1 or host2 and 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 form this internal routing to host1 or host2.

When a single server fails, all the UAs that have a registration with it will detect this and try to reconnect. This can cause large loads on the server and is referred to as the avalanche restart problem further discussed in Section 4.3. 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 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 [10] 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 for FW between the UA and the Edge Proxy and there could also be one between the Edge Proxy and the Registrar. This second case typically happens when the Edge Proxy is in an enterprise the Registrar is located at a service provider.



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. This is 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 that was put in the Path header to the associated flow.

The term Edge Proxy is often used to refer to deployments where the 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 [10].

3.5. Keep Alive Technique

A keep alive mechanism needs to detect both failure of a connection and changes to the NAT public mapping as well as keeping any NAT bindings refreshed. This specification uses STUN [5] 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 connection 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 do this with OPTIONS messages and rport; although this approach has the advantage of being backwards compatible, it also increases the load on the proxy or registrar server. The TCP KEEP_ALIVE mechanism is 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.

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

4. User Agent Mechanisms

The UA behavior is divided up into sections. The first describes what a client must do when forming a new connection, the second when detecting failure of a connection, and the third on failure recovery.

4.1. Forming Flows

When a User Agent initiates a dialog, it MUST provide a Contact URI which has GRUU properties if it is in possession of an appropriate GRUU. If it can not provide a GRUU, it needs to follow the procedure specified in Section 4.1.1.

UAs are configured with one or more SIP URIs representing the default outbound proxies with which to register. 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 discussed in [15] but typically results in sending the REGISTER with the Route header field containing a loose route to the outbound proxy URI. The UA MUST include the instance-id as described in [1]. The UA MUST also add a distinct flow-id parameter to the Contact header field. The UA SHOULD use a flow-id value of 1 for the first URI in the set, and a flow-id value of 2 for the second, and so on. Each one of these registrations will form a new flow from the UA to the proxy. The flow-id sequence does not have to be exactly 1,2,3 but it does have to be exactly the same flow-id sequence each time the device power cycles or reboots so that the flow-id values will collide with the previously used flow-id values and the proxy can realize that the older registrations are probably not useful.

If the 200 response to a REGISTER contains a Service Route header field value as defined in RFC 3608 [16], then whichever proxy sends the 200 response last will affect where all future requests from this UA are directed.

Note that the UA needs to honor 503 responses to registrations as described in RFC 3261 and RFC 3263 [4]. In particular, implementors should note that when receiving a 503 with a Retry-After, 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 redo the DNS resolution process such that if multiple hosts are reachable from the URI, there is a chance that the UA will select an alternate host from the one it chose the previous time the URI was resolved.

Note on Instance-ID Selection: The instance-id needs to be a URN but there are many ways one can be generated. A particularly simple way for both "hard" phones and "soft" phones is to use a UUID as defined in [6]. A device like a soft-phone, when first installed, should 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 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.

4.1.1. Request without GRUU

If the UA does not have a GRUU, it MUST send the request with a Contact header field containing a +sip.instance media feature parameter, and it MUST include the "pin-route" option-tag in both a Proxy-Require and a Require header field value. A User Agent compliant with this specification MUST NOT initiate a dialog with an INVITE without a GRUU in the Contact header field. (At the time of this writing this is allowed only for dialogs initiated with the SUBSCRIBE method.)

This mechanism without a GRUU is not reliable if any of the proxies on the path fail so it SHOULD not be used for long lived subscriptions. Once a UA acquires an appropriate GRUU, it should terminate these subscriptions and re-subscribe using the normal GRUU based approach.

4.2. Detecting Flow Failure

The UA needs to detect if a given flow has failed, and if it has failed, follow the procedures in Section 4.1 to form a new flow to replace the failed one.

User Agents that form flows MUST check if the configured URI they are connecting to has the "sip-stun" tag (defined in Section 10) and, if the tag is present, then the UA needs to periodically perform STUN [5] requests over the flow. The time between STUN requests when using UDP SHOULD be a random number between 24 and 29 seconds while for other transport protocols it SHOULD be a random number between 95 and 120 seconds. The 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 would be sent before 30 seconds based on information that some NATs had UDP

timeouts as low as 30 seconds. The 24 second lower bound was selected so that after 10 minutes the jitter this introduce would have unsyncronized the STUN requests from different devices 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 would be detected in this 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 widely. The 95 seconds lower bound was chosen so that the jitter introduced would result in a relatively even load on the servers after 30 minutes.

If the mapped address in the STUN response changes, the UA must treat this as a failure on the flow. Any time a SIP message is sent and the proxy does not respond, this is also considered a failure, the flow is discarded and the procedures in Section 4.3 are followed to form a new flow.

4.3. Flow Failure Recovery

When a flow to a particular URI in the proxy set fails, the UA needs to form a new flow to replace it. The new flow MUST have the same flow-id as the flow it is replacing. This is done in much the same way as the flows are described as being formed in Section 4.1; 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 URI in the proxy set. 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 the base time multiplied by two to power of the number of consecutive registration failures to that URI up to a maximum of 1800 seconds.

```
wait-time = min( 1800, (base-time * (2 ^ consecutive-failures)))
```

These three times SHOULD 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 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 had succeeded, the first retry would happen 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 being reached, then the UA keep trying forever with a random time between 900 and 1800 seconds between the attempts.

SIP dialogs can be used for one or more "usages". For example, a session created with INVITE (a session "usage") and a subscription (a subscription "usage") can share a dialog. On failure of a flow, a User Agent might wish to resynchronizing the state of any active usages on any dialogs using the flow. For example, the User Agent could send a new subscription for each subscription usage and an INVITE with replaces for each session usage. Note that when a flow was obtained via a REGISTER request, the flow might be used by many dialogs and dialog usages. A flow obtained via another request (e.g. a SUBSCRIBE request) only has usages from a single dialog. The only reason to do this is that a message may have been lost while the flow was being reestablished. The GRUU will ensure that any future messages are still delivered to the UA even if it does not re-subscribe, re-INVITE, or otherwise refresh the usage. Deployments need to carefully consider the implications of these sorts of operations. This approach only helps in a very narrow corner case and it will cause a huge load on the system if a single proxy crashes. In some deployments, this will cause more harm than good.

4.4. Registration by Other Instances

A User Agent MUST NOT include an instance-id or flow-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.)

5. Registrar Mechanisms

5.1. Processing Register Requests

Registrars which implement this specification, MUST support the Path header mechanism[10] and processes REGISTER requests as described in Section 10 of RFC 3261 with the following change. Any time the registrar checks if a new contact matches an existing contact in the location database, it MUST also check and see if both the instance-id and flow-id match. If they do not both match, then they are not the same contact. Additionally, if the both the instance-id and flow-id are present and do match, then it is considered a match regardless of if the value of the contact header field value matches. The

registrar MUST be prepared to receive some registrations that use instance-id and flow-id and some that do not, simultaneously for the same AOR.

In addition to the normal information stored in the binding record, some additional information MUST be stored for any registration that contains a flow-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 flow-id and instance-id and SHOULD also store the time at which the binding was last updated. If a Path header field is present RFC 3327 [10] requires this to be stored and the registrar MUST store the Path header field value with the binding record. Any time a messages is forwarded over the flow that created this binding, this stored Path header field value will be used to route the message. 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 MAY be configured with local policy to reject any registrations that do not include the instance-id and flow-id to eliminate the amplification attack described in [14].

5.2. 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 with the same AOR and instance-id, but a different flow-id.
- If two bindings have the same instance-id and flow-id, it SHOULD prefer the contact that was most recently updated.

Note that if the request URI is a GRUU, the proxy will only select contacts with the AOR and instance-id associated with the GRUU. The rules above still apply to a GRUU. This allows a request routed to a

GRUU to first try one of the flows to a UA, then if that fails, try another flow to the same UA instance.

The proxy uses normal forwarding rules looking at the Route of the message and any values of the of the stored Path header field value 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 flow-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 an indication from the network that indicates that no future messages on this flow will work, then it MUST remove 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 remove all the bindings that use that flow.

6. Edge Proxy Mechanisms

6.1. Processing Register Requests

When an Edge Proxy receives a registration request it MUST form a flow identifier token that is unique to this network flow and use this token as the user part of the URI that this proxy inserts into the Path header. Edge proxies MUST use a Path header. A trivial way to satisfy this requirement 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.

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 [8]. The concatenation of the HMAC and S are base64 encoded, as defined in [9], and used as the flow identifier. With IPv4 address, this will result in a 32 octet identifier.

Algorithm 1 MUST NOT be used unless the REGISTER request is over a SIPS protected transport. If the SIPS level of integrity protection is not available, an attacker can hijack another user's calls.

6.2. Forwarding Requests

When the Edge Proxy receives a request it applies normal routing procedures with the addition that it is routed to a URI with a flow identifier token that this proxy created, then 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 that has to be synchronized with the registrar.

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 verity 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.

Edge Proxies MUST Record-Route so that mid-dialog requests still are routed over the correct flow.

7. Mechanisms for All Servers

7.1. STUN Processing

TODO: This section needs to be brought into sync with the STUN draft and check there are not issues for SIP and STUN on TCP or UDP connections.

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.

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 requests with the exception that it does not need to support STUN requests with the changed port or changed address flag set. This allows the STUN server to run with only one port and IP address.

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 would typically show up in the URI in the Route header field value of a REGISTER request and would not be in the request URI.

7.2. Pin-Route Processing

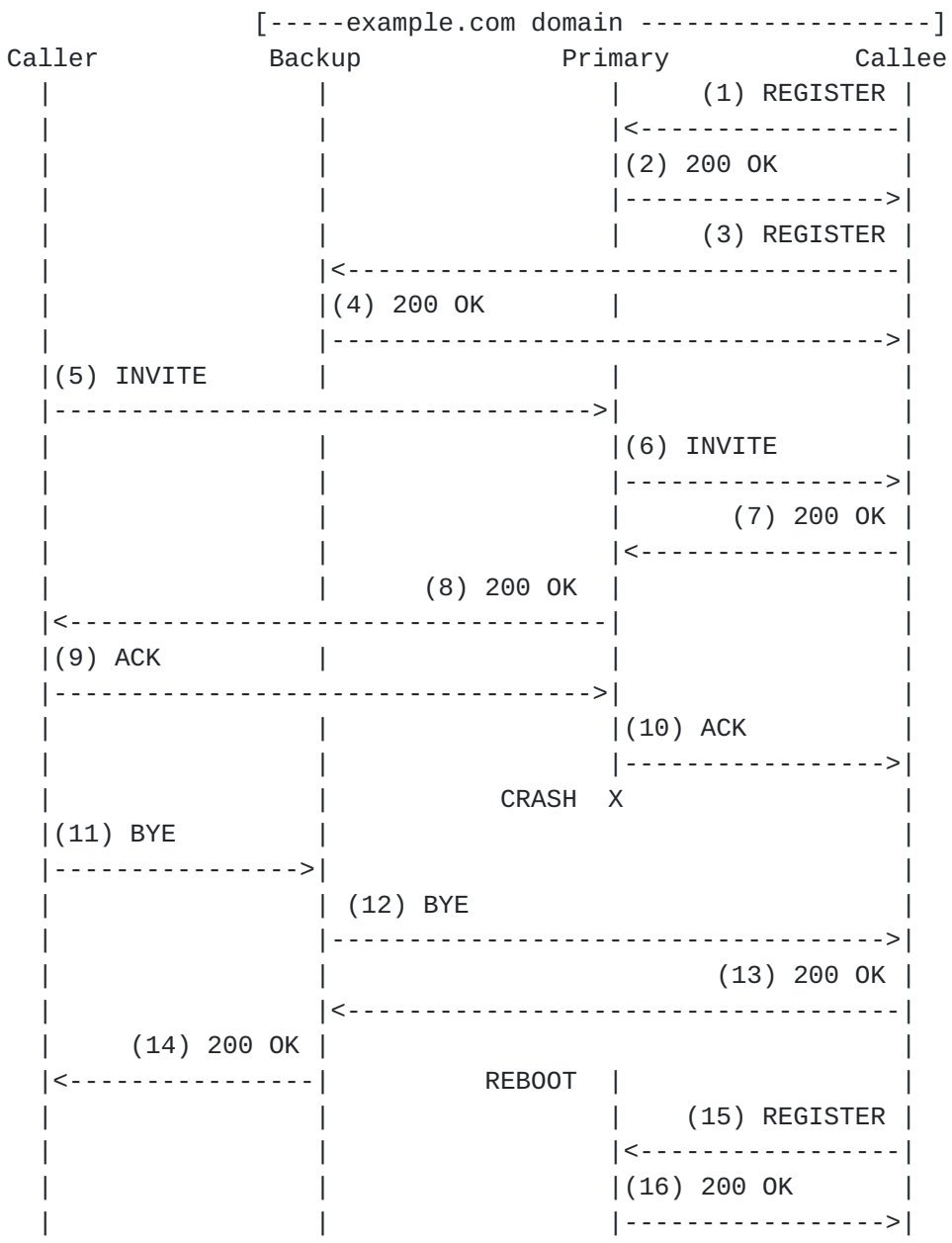
A sip device receives a request with the "pin-route" options tag set in the Proxy-Require header field or the Require header field needs to follow the procedures in this section.

A UAS that receives a request with the "pin-route" option tag in the Require header MUST either reject the request if pin-route is not supported, or if pin-route is supported by this UAS, the UAS MUST ensure that any message send in the dialog formed by this request is sent on the same flow as the initial request. This specification does not mandate that all UAs support this option but certain UAs, such as the NOTIFIER in the configuration framework, will want to support this so they can form subscriptions with devices that do not have a GRUU.

A proxy that receives a request with the "pin-route" option tag in the Proxy-Require header MUST add a record-route header field value that resolves to this proxy and it MUST ensure that any future requests or responses in this dialog are forwarded on the same flow as the original request. The suggested way to do this is to form a flow identifier token in the same way that an Edge Proxy would form this for the Path header and insert this flow identifier token in the user portion of the URI used in the record route header field value.

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
Route: <sip:primary.example.com;lr;sip-stun>
Contact: <sip:callee@10.0.1.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;flow-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 flow-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
Contact: <sip:callee@10.0.1.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;flow-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 flow-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
Route: <sip:backup.example.com;lr;sip-stun>
Contact: <sip:callee@10.0.1.1>
  ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
  ;flow-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
   CSeq: 1 REGISTER
   Contact: <sip:callee@10.0.1.1>
     ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
     ;flow-id=1
     ;expires=3600
   Contact: <sip:callee@10.0.1.1>
     ;+sip.instance="<urn:uuid:0C67446E-F1A1-11D9-94D3-000A95A0E128>"
     ;flow-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 will have a:

```
   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 based on the GRUU that the callee provided in their Contact header field value when the dialog was formed and the domain selected a host (primary or backup) that was currently available. How the domain does this is an implementation detail up to the domain.

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 a new Contact header field parameter, flow-id. The grammar for DIGIT and EQUAL is obtained from RFC 3261 [3].

```
   contact-params = c-p-q / c-p-expires / c-p-flow / contact-extension
   c-p-flow       = "flow-id" EQUAL 1*DIGIT
```

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

10. IANA Considerations

This specification defines a new Contact header field parameter called flow-id in the "Header Field Parameters and Parameter Values" sub-registry as per the registry created by [11] at http://www.iana.org/assignments/sip-parameters. The required information is:

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

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

This specification defines a new value in the "SIP/SIPS URI Parameters" sub-registry as per the registry created by [12] at http://www.iana.org/assignments/sip-parameters. The required information is:

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

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

TODO: Add IANA section for "pin-route" option tag.

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 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

Service Route: The current interaction of this draft and draft-rosenberg-sip-route-construct [15] does not work. Currently the Service Route specification, RCFC 3608, suggests that the service route is appended to the outbound proxy set. That will work with this specification. However the [15] draft is suggesting to change the behavior so that the Service Route replaces the outbound proxy. This is basically so that SIP can be used to make configuration changes to the UA. The problem is that this specification requires two or more URIs for the outbound configuration (so that reliability is possible) and the Service Route would only be able to provide a single URI. If it is desirable to use Service Route this way, it probably needs to be modified in many ways including allowing it to return different Service Routes to different devices registering for the same AOR.

Record Routing Edge Proxies: If an Edge Proxy record routes with a name that resolves explicitly to it and then crashes, all future requests in that dialog will fail. If an Edge Proxy record routes with a name that resolves to many edge proxies or does not record route at all, then requests that do not have GRUU as a contact will

not work. A suggested resolution to this is to require GRUU for long lived dialogs and have the Edge proxies use path headers and not record route.

SUBSCRIBEs without a GRUU. Earlier version of draft assumed that a REGISTER was always the first message. However the configuration framework[13] needs to perform a SUBSCRIBE to get the configuration that will allow the UA to register. This specification needs to deal with situations where there is a SUBSCRIBE but no REGISTER. The current resolution is to record route for these special cases and mitigate the reliability implications of this by not allowing these dialogs to be long lived.

The terminology of flow, flow-id, connection is confusing. Do we want to change it?

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.
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. Support proxy farms with multiple hosts for scaling and reliability purposes.
8. Minimize initial startup load on a proxy.
9. Support proxies that provide geographic redundancy.
10. Support architectures with edge proxies.
11. Must be able to receive notifications over the same flow used to send a subscription, even before any registrations have been established. This ensures compatibility with the SIP configuration framework [13].

14. 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] 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.

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

[3] 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.

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

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

[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.

16.2. Informative References

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

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

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

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

[12] 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.

[13] Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-07 (work in progress), July 2005.

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

[15] 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.

[16] 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)
SIP Edge LLC
5617 Scotts Valley Drive, Suite 200
Scotts Valley, CA 95066
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.