Network Working Group
Request for Comments: 5626
Updates: 3261, 3327
Category: Standards Track

C. Jennings, Ed.
Cisco Systems
R. Mahy, Ed.
Unaffiliated
F. Audet, Ed.
Skype Labs
October 2009

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

Abstract

The Session Initiation Protocol (SIP) allows proxy servers to initiate TCP connections or to send asynchronous UDP datagrams to User Agents in order to deliver requests. However, in a large number of real deployments, many practical considerations, such as the existence of firewalls and Network Address Translators (NATs) or the use of TLS with server-provided certificates, prevent servers from connecting to User Agents in this way. This specification defines behaviors for User Agents, registrars, and proxy servers that allow requests to be delivered on existing connections established by the User Agent. It also defines keep-alive behaviors needed to keep NAT bindings open and specifies the usage of multiple connections from the User Agent to its registrar.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

There are many environments for SIP [RFC3261] deployments in which the User Agent (UA) can form a connection to a registrar or proxy but in which connections in the reverse direction to the UA are not possible. This can happen for several reasons, but the most likely is a NAT or a firewall in between the SIP UA and the proxy. Many such devices will only allow outgoing connections. This specification allows a SIP User Agent behind such a firewall or NAT to receive inbound traffic associated with registrations or dialogs that it initiates.

Most IP phones and personal computers get their network configurations dynamically via a protocol such as the Dynamic Host Configuration Protocol (DHCP) [RFC2131]. These systems typically do not have a useful name in the Domain Name System (DNS) [RFC1035], and they almost never have a long-term, stable DNS name that is appropriate for use in the subjectAltName of a certificate, as required by [RFC3261]. However, these systems can still act as a Transport Layer Security (TLS) [RFC5246] client and form outbound connections to a proxy or registrar that authenticates with a server certificate. The server can authenticate the UA using a shared secret in a digest challenge (as defined in Section 22 of RFC 3261) over that TLS connection. This specification allows a SIP User Agent who has to initiate the TLS connection to receive inbound traffic associated with registrations or dialogs that it initiates.

The key idea of this specification is that when a UA sends a REGISTER request or a dialog-forming 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 in another transport protocol -- to forward any incoming requests that need to go to this UA in the context of the registration or dialog.

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 register over multiple flows at the same time. This specification also defines two keep-alive schemes. The keep-alive mechanism is used to keep NAT bindings fresh, and to allow the UA to detect when a flow has failed.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2.1. Definitions

Authoritative Proxy: A proxy that handles non-REGISTER requests for a specific Address-of-Record (AOR), performs the logical Location Server lookup described in [RFC3261], and forwards those requests to specific Contact URIs. (In [RFC3261], the role that is authoritative for REGISTER requests for a specific AOR is a Registration Server.)

Edge Proxy: An edge proxy is any proxy that is located topologically between the registering User Agent and the Authoritative Proxy. The "first" edge proxy refers to the first edge proxy encountered when a UA sends a request.

Flow: A Flow is a transport-layer association between two hosts that is represented by the network address and port number of both ends and by the transport 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.

Flow Token: An identifier that uniquely identifies a flow which can be included in a SIP URI (Uniform Resource Identifier [RFC3986]).

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 for a different flow, each concurrent 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 which appears as a "+sip.instance" Contact header field parameter. This is a Uniform Resource Name (URN) that uniquely identifies this specific UA instance.

"ob" Parameter: The "ob" parameter is a SIP URI parameter that has a different meaning depending on context. In a Path header field value, it is used by the first edge proxy to indicate that a flow token was added to the URI. In a Contact or Route header field value, it indicates that the UA would like other requests in the same dialog to be routed over the same flow.

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

3. Overview

The mechanisms defined in this document are useful in several scenarios 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.

This entire section is non-normative.

3.1. Summary of Mechanism

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 flows 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 correspond to the same UA. The registrar can use the reg-id label to recognize whether a UA is creating a new flow or refreshing or replacing an old one, possibly after a reboot or a network failure.

When a proxy goes to route a message to a UA for which it has a binding, it can use any one of the flows on which a successful registration has been completed. A failure to deliver a request 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.

When sending a dialog-forming request, a UA can also ask its first edge proxy to route subsequent requests in that dialog over the same flow. This is necessary whether the UA has registered or not.

UAs use a simple periodic message as a keep-alive mechanism to keep their flow to the proxy or registrar alive. For connection-oriented transports such as TCP this is based on carriage-return and line-feed sequences (CRLF), while for transports that are not connection oriented, this is accomplished by using a SIP-specific usage profile of STUN (Session Traversal Utilities for NAT) [RFC5389].

3.2. Single Registrar and UA

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

```
+-----------+
| Registrar |
| Proxy     |
+-----+-----+
      |
      |
 +----+--+
 | User  |
 | Agent |
 +-------+
```

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

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

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/TCP 192.0.2.2;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, outbound
Contact: <sip:line1@192.0.2.2;transport=tcp>; reg-id=1;
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-000A95A0E128>"
Content-Length: 0
```

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

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

When Alice sends a request to Bob, his authoritative proxy selects the target set. The proxy forwards the request to elements in the target set based on the proxy’s policy. The proxy looks at the target set and uses the instance-id to understand if two targets both end up routing to the same UA. When the proxy goes to forward a request to a given target, it looks and finds the flows over which it received the registration. The proxy then forwards the request over an existing flow, instead of resolving the Contact URI using the procedures in [RFC3263] and trying to form a new flow to that contact.

As described in the next section, if the proxy has multiple flows that all go to this UA, the proxy can choose any one of the registration bindings for this AOR that has the same instance-id as the selected UA.

3.3. Multiple Connections from a User Agent

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

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

```
       +-------------------+
       | Domain            |
       | Logical Proxy/Reg |
       |                   |
       |+-----+     +-----+|
       ||Host1|     |Host2||
       |+-----+     +-----+|
       +---\------------/--+
            \          /
             \        /
              \      /
               \    /
              +------+
              | User |
              | Agent|
              +------+
```

The UA is configured with multiple outbound proxy registration URIs. These URIs are configured into the UA through whatever the normal mechanism is to configure the proxy address and AOR in the UA. If the AOR is alice@example.com, the outbound-proxy-set might look something like "sip:primary.example.com" and "sip: secondary.example.com". Note that each URI in the outbound-proxy-set could resolve to several different physical hosts. The administrative domain that created these URIs should ensure that the two URIs resolve to separate hosts. These URIs are handled according to normal SIP processing rules, so mechanisms like DNS SRV [RFC2782] can be used to do load-balancing across a proxy farm. The approach in this document does not prevent future extensions, such as the SIP UA configuration framework [CONFIG-FMWK], from adding other ways for a User Agent to discover its outbound-proxy-set.

The domain also needs to ensure that a request for the UA sent to Host1 or Host2 is then sent across the appropriate flow to the UA. The domain might choose to use the Path header approach (as described in the next section) to store this internal routing information on Host1 or Host2.

When a single server fails, all the UAs that have a flow through it will detect a flow failure and try to reconnect. This can cause large loads on the server. When large numbers of hosts reconnect nearly simultaneously, this is referred to as the avalanche restart problem, and is further discussed in Section 4.5. The multiple flows to many servers help reduce the load caused by the avalanche restart. If a UA has multiple flows, and one of the servers fails, the UA delays a recommended amount of 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.

Scalability is achieved by using DNS SRV [RFC2782] to load-balance the primary connection across a set of machines that can service the primary connection, and also using DNS SRV to load-balance across a separate set of machines that can service the secondary connection. The deployment here requires that DNS is configured with one entry that resolves to all the primary hosts and another entry that resolves to all the secondary hosts. While this introduces additional DNS configuration, the approach works and requires no additional SIP extensions to [RFC3263].

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

3.4. Edge Proxies

Some SIP deployments use edge proxies such that the UA sends the REGISTER to an edge proxy that then forwards the REGISTER to the registrar. There could be a NAT or firewall between the UA and the edge proxy.

```
                +---------+
                |Registrar|
                |Proxy    |
                +---------+
                 /      \
                /        \
               /          \
            +-----+     +-----+
            |Edge1|     |Edge2|
            +-----+     +-----+
               \           /
                \         /
        ----------------------------NAT/FW
                  \     /
                   \   /
                  +------+
                  |User  |
                  |Agent |
                  +------+
```

The edge proxy includes a Path header [RFC3327] so that when the proxy/registrar later forwards a request to this UA, the request is routed through the edge proxy.

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

The term edge proxy is often used to refer to deployments where the edge proxy is in the same administrative domain as the registrar. However, in this specification we use the term to refer to any proxy between the UA and the registrar. For example, the edge proxy may be inside an enterprise that requires its use, and the registrar could be from a service provider with no relationship to the enterprise. Regardless of whether they are in the same administrative domain, this specification requires that registrars and edge proxies support the Path header mechanism in [RFC3327].

3.5. Keep-Alive Technique

This document describes two keep-alive mechanisms: a CRLF keep-alive and a STUN keep-alive. Each of these mechanisms uses a client-to-server "ping" keep-alive and a corresponding server-to-client "pong" message. This ping-pong sequence allows the client, and optionally the server, to tell if its flow is still active and useful for SIP traffic. The server responds to pings by sending pongs. If the client does not receive a pong in response to its ping (allowing for retransmission for STUN as described in Section 4.4.2), it declares the flow dead and opens a new flow in its place.

This document also suggests timer values for these client keep-alive mechanisms. These timer values were chosen to keep most NAT and firewall bindings open, to detect unresponsive servers within 2 minutes, and to mitigate against the avalanche restart problem. However, the client may choose different timer values to suit its needs, for example to optimize battery life. In some environments, the server can also keep track of the time since a ping was received over a flow to guess the likelihood that the flow is still useful for delivering SIP messages.

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 4.5 to provide congestion relief when a large number of agents simultaneously reboot.

A keep-alive mechanism needs to keep NAT bindings refreshed; for connections, it also needs to detect failure of a connection; and for connectionless transports, it needs to detect flow failures including changes to the NAT public mapping. For connection-oriented transports such as TCP [RFC0793] and SCTP [RFC4960], this specification describes a keep-alive approach based on sending CRLFs. For connectionless transport, such as UDP [RFC0768], this specification describes using STUN [RFC5389] over the same flow as the SIP traffic to perform the keep-alive.

UAs and Proxies are also free to use native transport keep-alives; however, the application may not be able to set these timers on a per-connection basis, and the server certainly cannot make any assumption about what values are used. Use of native transport keep-alives is outside the scope of this document.

3.5.1. CRLF Keep-Alive Technique

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

> Note: Sending a CRLF over a connection-oriented transport is backwards compatible (because of requirements in Section 7.5 of [RFC3261]), but only implementations which support this specification will respond to a "ping" with a "pong".

3.5.2. STUN Keep-Alive Technique

This approach can only be used for connection-less transports, such as UDP.

For connection-less transports, a flow definition could change because a NAT device in the network path reboots and the resulting public IP address or port mapping for the UA changes. To detect this, STUN requests are sent over the same flow that is being used for the SIP traffic. The proxy or registrar acts as a limited Session Traversal Utilities for NAT (STUN) [RFC5389] server on the SIP signaling port.

> Note: The STUN mechanism is very robust and allows the detection of a changed IP address and port. Many other options were considered, but the SIP Working Group selected the STUN-based approach. Approaches using SIP requests were abandoned because many believed that good performance and full backwards compatibility using this method were mutually exclusive.

4. User Agent Procedures

4.1. Instance ID Creation

Each UA MUST have an Instance Identifier Uniform Resource Name (URN) [RFC2141] that uniquely identifies the device. Usage of a URN provides a persistent and unique name for the UA instance. It also provides an easy way to guarantee uniqueness within the AOR. This URN MUST be persistent across power cycles of the device. The instance ID MUST NOT change as the device moves from one network to another.

A UA SHOULD create a Universally Unique Identifier (UUID) URN [RFC4122] as its instance-id. The UUID URN allows for non-centralized computation of a URN based on time, unique names (such as a MAC address), or a random number generator.

> Note: A device like a "soft phone", when first installed, can generate a UUID [RFC4122] 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 on this device.

If a URN scheme other than UUID is used, the UA MUST only use URNs for which an RFC (from the IETF stream) defines how the specific URN needs to be constructed and used in the "+sip.instance" Contact header field parameter for outbound behavior.

To convey its instance-id in both requests and responses, the UA includes a "sip.instance" media feature tag as a UA characteristic [RFC3840]. This media feature tag is encoded in the Contact header field as the "+sip.instance" Contact header field parameter. One case where a UA could prefer to omit the "sip.instance" media feature tag is when it is making an anonymous request or some other privacy concern requires that the UA not reveal its identity.

> Note: [RFC3840] 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 the callee capabilities [RFC3840] and the caller preferences [RFC3841] specifications. When the instance ID is used in this specification, it is "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 [RFC2141]. Lexical equality could result in two URNs being considered unequal when they are actually equal. In this specific usage of URNs, the only element that provides the URN is the SIP UA instance identified by that URN. As a result, the UA instance has to 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 to (yet lexicographically different from) previous registrations.

4.2. Registrations

4.2.1. Initial Registrations

At configuration time, UAs obtain one or more SIP URIs representing the default outbound-proxy-set. This specification assumes the set is determined via any of a number of configuration mechanisms, and future specifications can define additional mechanisms such as using DNS to discover this set. How the UA is configured is outside the scope of this specification. However, a UA MUST support sets with at least two outbound proxy URIs and SHOULD support sets with up to four URIs.

For each outbound proxy URI in the set, the User Agent Client (UAC) SHOULD send a REGISTER request using this URI as the default outbound proxy. (Alternatively, the UA could limit the number of flows formed to conserve battery power, for example). If the set has more than one URI, the UAC MUST send a REGISTER request to at least two of the default outbound proxies from the set. UAs that support this specification MUST include the outbound option tag in a Supported header field in a REGISTER request. Each of these REGISTER requests will use a unique Call-ID. 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.

REGISTER requests, other than those described in Section 4.2.3, MUST include an instance-id media feature tag as specified in Section 4.1.

A UAC conforming to this specification MUST include in the Contact header field, a "reg-id" parameter that is distinct from other "reg-id" parameters used in other registrations that use the same "+sip.instance" Contact header field parameter and AOR. Each one of these registrations will form a new flow from the UA to the proxy. The sequence of reg-id values does not have to be sequential but MUST be exactly the same sequence of reg-id values each time the UA instance power cycles or reboots, so that the reg-id values will collide with the previously used reg-id values. This is so the registrar can replace the older registrations.

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

The UAC MUST support the Path header [RFC3327] mechanism, and indicate its support 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 examines successful registration responses for the presence of an outbound option-tag in a Require header field value. Presence of this option-tag indicates that the registrar is compliant with this specification, and that any edge proxies which needed to participate are also compliant. If the registrar did not support outbound, the UA has potentially registered an un-routable contact. It is the responsibility of the UA to remove any inappropriate Contacts.

If outbound registration succeeded, as indicated by the presence of the outbound option-tag in the Require header field of a successful registration response, the UA begins sending keep-alives as described in Section 4.4.

> Note: The UA needs to honor 503 (Service Unavailable) responses to registrations as described in [RFC3261] and [RFC3263]. In particular, implementors should note that when receiving a 503 (Service Unavailable) response with a Retry-After header field, the UA is expected to wait the indicated amount of time and retry the registration. A Retry-After header field value of 0 is valid and indicates the UA is expected to retry the REGISTER request immediately. Implementations need to ensure that when retrying the REGISTER request, 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.

If the registering UA receives a 439 (First Hop Lacks Outbound Support) response to a REGISTER request, it MAY re-attempt registration without using the outbound mechanism (subject to local policy at the client). If the client has one or more alternate outbound proxies available, it MAY re-attempt registration through such outbound proxies. See Section 11.6 for more information on the 439 response code.

4.2.2. Subsequent REGISTER Requests

Registrations for refreshing a binding and for removing a binding use the same instance-id and reg-id values as the corresponding initial registration where the binding was added. Registrations that merely refresh an existing binding are sent over the same flow as the original registration where the binding was added.

If a re-registration is rejected with a recoverable error response, for example by a 503 (Service Unavailable) containing a Retry-After header, the UAC SHOULD NOT tear down the corresponding flow if the flow uses a connection-oriented transport such as TCP. As long as "pongs" are received in response to "pings", the flow SHOULD be kept active until a non-recoverable error response is received. This prevents unnecessary closing and opening of connections.

4.2.3. Third-Party Registrations

In an initial registration or re-registration, a UA MUST NOT include a "reg-id" header field parameter in the Contact header field 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.)

A UAC also MUST NOT include an instance-id feature tag or "reg-id" Contact header field parameter in a request to un-register all Contacts (a single Contact header field value with the value of "*").

4.3. Sending Non-REGISTER Requests

When a UAC 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. UAs that support this specification SHOULD include the outbound option tag in a Supported header field in a request that is not a REGISTER request.

The UAC performs normal DNS resolution on the next hop URI (as described in [RFC3263]) to find a protocol, IP address, and port. For protocols that don't use TLS, if the UAC has an existing flow to this IP address, and port with the correct protocol, then the UAC MUST use the existing connection. For TLS protocols, there MUST also be a match between the host production in the next hop and one of the URIs contained in the subjectAltName in the peer certificate. If the UAC 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.

Typically, a UAC using the procedures of this document and sending a dialog-forming request will want all subsequent requests in the dialog to arrive over the same flow. If the UAC is using a Globally Routable UA URI (GRUU) [RFC5627] that was instantiated using a Contact header field value that included an "ob" parameter, the UAC sends the request over the flow used for registration, and subsequent requests will arrive over that same flow. If the UAC is not using such a GRUU, then the UAC adds an "ob" parameter to its Contact header field value. This will cause all subsequent requests in the dialog to arrive over the flow instantiated by the dialog-forming request. This case is typical when the request is sent prior to registration, such as in the initial subscription dialog for the configuration framework [CONFIG-FMWK].

> Note: If the UAC wants a UDP flow to work through NATs or firewalls, it still needs to put the ’rport’ parameter [RFC3581] in its Via header field value, and send from the port it is prepared to receive on. More general information about NAT traversal in SIP is described in [NAT-SCEN].

4.4. Keep-Alives and Detecting Flow Failure

Keep-alives are used for refreshing NAT/firewall bindings and detecting flow failure. Flows can fail for many reasons including the rebooting of NATs and the crashing of edge proxies.

As described in Section 4.2, a UA that registers will begin sending keep-alives after an appropriate registration response. A UA that does not register (for example, a PSTN gateway behind a firewall) can also send keep-alives under certain circumstances.

Under specific circumstances, a UAC might be allowed to send STUN keep-alives even if the procedures in Section 4.2 were not completed, provided that there is an explicit indication that the target first-hop SIP node supports STUN keep-alives. For example, this applies to a non-registering UA or to a case where the UA registration succeeded, but the response did not include the outbound option-tag in the Require header field.

> Note: A UA can "always" send a double CRLF (a "ping") over connection-oriented transports as this is already allowed by Section 7.5 of [RFC3261]. However a UA that did not register using outbound registration cannot expect a CRLF in response (a "pong") unless the UA has an explicit indication that CRLF keep-alives are supported as described in this section. Likewise, a UA that did not successfully register with outbound procedures needs explicit indication that the target first-hop SIP node supports STUN keep-alives before it can send any STUN messages.

A configuration option indicating keep-alive support for a specific target is considered an explicit indication. If these conditions are satisfied, the UA sends its keep-alives according to the same guidelines as those used when UAs register; these guidelines are described below.

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

When a successful registration response contains the Flow-Timer header field, the value of this header field is the number of seconds the server is prepared to wait without seeing keep-alives before it could consider the corresponding flow dead. Note that the server would wait for an amount of time larger than the Flow-Timer in order to have a grace period to account for transport delay. The UA MUST send keep-alives at least as often as this number of seconds. If the UA uses the server-recommended keep-alive frequency it SHOULD send its keep-alives so that the interval between each keep-alive is randomly distributed between 80% and 100% of the server-provided time. For example, if the server suggests 120 seconds, the UA would send each keep-alive with a different frequency between 95 and 120 seconds.

If no Flow-Timer header field was present in a register response for this flow, the UA can send keep-alives at its discretion. The sections below provide RECOMMENDED default values for these keep-alives.

The client needs to perform normal [RFC3263] SIP DNS resolution on the URI from the outbound-proxy-set to pick a transport. Once a transport is selected, the UA selects the keep-alive approach that is recommended for that transport.

Section 4.4.1 describes a keep-alive mechanism for connection-oriented transports such as TCP or SCTP. Section 4.4.2 describes a keep-alive mechanism for connection-less transports such as UDP. Support for other transports such as DCCP [RFC4340] is for further study.

4.4.1. Keep-Alive with CRLF

This approach MUST only be used with connection oriented transports such as TCP or SCTP; it MUST NOT be used with connection-less transports such as UDP.

A User Agent that forms flows checks if the configured URI to which the UA is connecting resolves to a connection-oriented transport (e.g., TCP and TLS over TCP).

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

```
CRLF = CR LF
double-CRLF = CR LF CR LF
CR = %x0D
LF = %x0A
```

The "ping" and "pong" need to be sent between SIP messages and cannot be sent in the middle of a SIP message. If sending over TLS, the CRLFs are sent inside the TLS protected channel. If sending over a SigComp [RFC3320] compressed data stream, the CRLF keep-alives are sent inside the compressed stream. The double CRLF is considered a single SigComp message. The specific mechanism for representing these characters is an implementation-specific matter to be handled by the SigComp compressor at the sending end.

If a pong is not received within 10 seconds after sending a ping (or immediately after processing any incoming message being received when that 10 seconds expires), then the client MUST treat the flow as failed. Clients MUST support this CRLF keep-alive.

> Note: This value of 10-second timeout was selected to be long enough that it allows plenty of time for a server to send a response even if the server is temporarily busy with an administrative activity. At the same time, it was selected to be small enough that a UA registered to two redundant servers with unremarkable hardware uptime could still easily provide very high levels of overall reliability. Although some Internet protocols are designed for round-trip times over 10 seconds, SIP for real-time communications is not really usable in these type of environments as users often abandon calls before waiting much more than a few seconds.

When a Flow-Timer header field is not provided in the most recent success registration response, the proper selection of keep-alive frequency is primarily a trade-off between battery usage and availability. The UA MUST select a random number between a fixed or configurable upper bound and a lower bound, where the lower bound is 20% less then the upper bound. The fixed upper bound or the default configurable upper bound SHOULD be 120 seconds (95 seconds for the lower bound) where battery power is not a concern and 840 seconds (672 seconds for the lower bound) where battery power is a concern. The random number will be different for each keep-alive "ping".

> Note on selection of time values: the 120-second upper bound was chosen based on the idea that for a good user experience, failures normally will be detected in this amount of time and a new connection will be set up. The 14-minute upper bound for battery-powered devices was selected based on NATs with TCP timeouts as low as 15 minutes. Operators that wish to change the relationship between load on servers and the expected time that a user might not receive inbound communications will probably adjust this time. The 95-second lower bound was chosen so that the jitter introduced will result in a relatively even load on the servers after 30 minutes.

4.4.2. Keep-Alive with STUN

This approach MUST only be used with connection-less transports, such as UDP; it MUST NOT be used for connection-oriented transports such as TCP and SCTP.

A User Agent that forms flows checks if the configured URI to which the UA is connecting resolves to use the UDP transport. The UA can periodically perform keep-alive checks by sending STUN [RFC5389] Binding Requests over the flow as described in Section 8. Clients MUST support STUN-based keep-alives.

When a Flow-Timer header field is not included in a successful registration response, the time between each keep-alive request SHOULD be a random number between 24 and 29 seconds.

> Note on selection of time values: the upper bound of 29 seconds was selected, as many NATs have UDP timeouts as low as 30 seconds. The 24-second lower bound was selected so that after 10 minutes the jitter introduced by different timers will make the keep-alive requests unsynchronized to evenly spread the load on the servers. Note that the short NAT timeouts with UDP have a negative impact on battery life.

If a STUN Binding Error Response is received, or if no Binding Response is received after 7 retransmissions (16 times the STUN "RTO" timer -- where RTO is an estimate of round-trip time), the UA considers the flow failed. If the XOR-MAPPED-ADDRESS in the STUN Binding Response changes, the UA MUST treat this event as a failure on the flow.

4.5. Flow Recovery

When a flow used for registration (through 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 value as the registration it replaces. This is done in much the same way as forming a brand new flow as described in Section 4.2; however, if there is a failure in forming this flow, the UA needs to wait a certain amount of time before retrying to form a flow to this particular next hop.

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

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

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

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

- max-time with a default of 1800 seconds
- base-time (if all failed) with a default of 30 seconds
- base-time (if all have not failed) with a default of 90 seconds

For example, if the base-time is 30 seconds, and there were three failures, then the upper-bound wait time is min(1800, 30*(2^3)) or 240 seconds. The actual amount of time the UA waits before retrying registration (the retry delay time) is computed by selecting a uniform random time between 50 and 100% of the upper-bound wait time. The UA MUST wait for at least the value of the retry delay time before trying another registration to form a new flow for that URI (a 503 response to an earlier failed registration attempt with a Retry-After header field value may cause the UA to wait longer).

To be explicitly clear on the boundary conditions: when the UA boots, it immediately tries to register. If this fails and no registration on other flows succeed, the first retry happens somewhere between 30 and 60 seconds after the failure of the first registration request. If the number of consecutive-failures is large enough that the maximum of 1800 seconds is reached, the UA will keep trying indefinitely with a random time of 15 to 30 minutes between each attempt.

5. Edge Proxy Procedures

5.1. Processing Register Requests

When an edge proxy receives a registration request with a "reg-id" header field parameter in the Contact header field, it needs to determine if it (the edge proxy) will have to be visited for any subsequent requests sent to the User Agent identified in the Contact header field, or not. If the edge proxy is the first hop, as indicated by the Via header field, it MUST insert its URI in a Path header field value as described in [RFC3327]. If it is not the first hop, it might still decide to add itself to the Path header based on local policy. In addition, if the edge proxy is the first SIP node after the UAC, the edge proxy either MUST store a "flow token" (containing information about the flow from the previous hop) in its Path URI or reject the request. The flow token MUST be an identifier that is unique to this network flow. The flow token MAY be placed in the userpart of the URI. In addition, the first node MUST include an "ob" URI parameter in its Path header field value. If the edge proxy is not the first SIP node after the UAC it MUST NOT place an "ob" URI parameter in a Path header field value. The edge proxy can determine if it is the first hop by examining the Via header field.

5.2. Generating Flow Tokens

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

> Example Algorithm: When the proxy boots, it selects a 20-octet crypto random key called K that only the edge proxy knows. A byte array, called S, is formed that contains the following information about the flow the request was received on: an enumeration indicating the protocol, the local IP address and port, the remote IP address and port. The HMAC of S is computed using the key K and the HMAC-SHA1-80 algorithm, as defined in [RFC2104]. The concatenation of the HMAC and S are base64 encoded, as defined in [RFC4648], and used as the flow identifier. When using IPv4 addresses, this will result in a 32-octet identifier.

5.3. Forwarding Non-REGISTER Requests

When an edge proxy receives a request, it applies normal routing procedures with the following additions. If the edge proxy receives a request where the edge proxy is the host in the topmost Route header field value, and the Route header field value contains a flow token, the proxy follows the procedures of this section. Otherwise the edge proxy skips the procedures in this section, removes itself from the Route header field, and continues processing the request.

The proxy decodes the flow token and compares the flow in the flow token with the source of the request to determine if this is an "incoming" or "outgoing" request.

If the flow in the flow token identified by the topmost Route header field value matches the source IP address and port of the request, the request is an "outgoing" request; otherwise, it is an "incoming" request.

5.3.1. Processing Incoming Requests

If the Route header value contains an "ob" URI parameter, the Route header was probably copied from the Path header in a registration. If the Route header value contains an "ob" URI parameter, and the request is a new dialog-forming request, the proxy needs to adjust the route set to ensure that subsequent requests in the dialog can be delivered over a valid flow to the UA instance identified by the flow token.

> Note: A simple approach to satisfy this requirement is for the proxy to add a Record-Route header field value that contains the flow-token, by copying the URI in the Route header minus the "ob" parameter.

Next, whether the Route header field contained an "ob" URI parameter or not, the proxy removes the Route header field value and forwards the request over the ’logical flow’ identified by the flow token, that is known to deliver data to the specific target UA instance. If the flow token has been tampered with, the proxy SHOULD send a 403 (Forbidden) response. If the flow no longer exists, the proxy SHOULD send a 430 (Flow Failed) response to the request.

Proxies that used the example algorithm described in Section 5.2 to form a flow token follow the procedures below to determine the correct flow. To decode the flow token, take the flow identifier in the user portion of the URI and base64 decode it, then verify the HMAC is correct by recomputing the HMAC and checking that it matches. If the HMAC is not correct, the request has been tampered with.

5.3.2. Processing Outgoing Requests

For mid-dialog requests to work with outbound UAs, the requests need to be forwarded over some valid flow to the appropriate UA instance. If the edge proxy receives an outgoing dialog-forming request, the edge proxy can use the presence of the "ob" URI parameter in the UAC’s Contact URI (or topmost Route header field) to determine if the edge proxy needs to assist in mid-dialog request routing.

> Implementation note: Specific procedures at the edge proxy to ensure that mid-dialog requests are routed over an existing flow are not part of this specification. However, an approach such as having the edge proxy add a Record-Route header with a flow token is one way to ensure that mid-dialog requests are routed over the correct flow.

5.4. Edge Proxy Keep-Alive Handling

All edge proxies compliant with this specification MUST implement support for STUN NAT keep-alives on their SIP UDP ports as described in Section 8.

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

The last proxy to forward a successful registration response to a UA MAY include a Flow-Timer header field if the response contains the outbound option-tag in a Require header field value in the response. The reason a proxy would send a Flow-Timer is if it wishes to detect flow failures proactively and take appropriate action (e.g., log alarms, provide alternative treatment if incoming requests for the UA are received, etc.). The server MUST wait for an amount of time larger than the Flow-Timer in order to have a grace period to account for transport delay.

6. Registrar Procedures

This specification updates the definition of a binding in [RFC3261], Section 10 and [RFC3327], Section 5.3.

Registrars that implement this specification MUST support the Path header mechanism [RFC3327].

When receiving a REGISTER request, the registrar MUST check from its Via header field if the registrar is the first hop or not. If the registrar is not the first hop, it MUST examine the Path header of the request. If the Path header field is missing or it exists but the first URI does not have an "ob" URI parameter, then outbound processing MUST NOT be applied to the registration. In this case, the following processing applies: if the REGISTER request contains the reg-id and the outbound option tag in a Supported header field, then the registrar MUST respond to the REGISTER request with a 439 (First Hop Lacks Outbound Support) response; otherwise, the registrar MUST ignore the "reg-id" parameter of the Contact header. See Section 11.6 for more information on the 439 response code.

A Contact header field value with an instance-id media feature tag but no "reg-id" header field parameter is valid (this combination will result in the creation of a GRUU, as described in the GRUU specification [RFC5627]), but one with a reg-id but no instance-id is not valid. If the registrar processes a Contact header field value with a reg-id but no instance-id, it simply ignores the reg-id parameter.

A registration containing a "reg-id" header field parameter and a non-zero expiration is used to register a single UA instance over a single flow, and can also de-register any Contact header fields with zero expiration. Therefore, if the Contact header field contains more than one header field value with a non-zero expiration and any of these header field values contain a "reg-id" Contact header field parameter, the entire registration SHOULD be rejected with a 400 (Bad Request) response. The justification for recommending rejection versus making it mandatory is that the receiver is allowed by [RFC3261] to squelch (not respond to) excessively malformed or malicious messages.

If the Contact header did not contain a "reg-id" Contact header field parameter or if that parameter was ignored (as described above), the registrar MUST NOT include the outbound option-tag in the Require header field of its response.

The registrar MUST be prepared to receive, simultaneously for the same AOR, some registrations that use instance-id and reg-id and some registrations that do not. The registrar MAY be configured with local policy to reject any registrations that do not include the instance-id and reg-id, or with Path header field values that do not contain the "ob" URI parameter. If the Contact header field does not contain a "+sip.instance" Contact header field parameter, the registrar processes the request using the Contact binding rules in [RFC3261].

When a "+sip.instance" Contact header field parameter and a "reg-id" Contact header field parameter are present in a Contact header field of a REGISTER request (after the Contact header validation as described above), the corresponding binding is between an AOR and the combination of the instance-id (from the "+sip.instance" Contact header parameter) and the value of "reg-id" Contact header field parameter parameter. The registrar MUST store in the binding the Contact URI, all the Contact header field parameters, and any Path header field values. (Even though the Contact URI is not used for binding comparisons, it is still needed by the authoritative proxy to form the target set.) Provided that the UAC had included an outbound option-tag (defined in Section 11.4) in a Supported header field value in the REGISTER request, the registrar MUST include the outbound option-tag in a Require header field value in its response to that REGISTER request.

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

If the registrar receives a re-registration for a specific combination of AOR, and instance-id and reg-id values, the registrar MUST update any information that uniquely identifies the network flow over which the request arrived if that information has changed, and SHOULD update the time the binding was last updated.

To be compliant with this specification, registrars that can receive SIP requests directly from a UAC without intervening edge proxies MUST implement the same keep-alive mechanisms as edge proxies (Section 5.4). Registrars with a direct flow with a UA MAY include a Flow-Timer header in a 2xx class registration response that includes the outbound option-tag in the Require header.

7. Authoritative Proxy Procedures: Forwarding Requests

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

- The proxy MUST NOT populate the target set with more than one contact with the same AOR and instance-id at a time.

- If a request for a particular AOR and instance-id fails with a 430 (Flow Failed) response, the proxy SHOULD replace the failed branch with another target (if one is available) with the same AOR and instance-id, but a different reg-id.

- If the proxy receives a final response from a branch other than a 408 (Request Timeout) or a 430 (Flow Failed) response, the proxy MUST NOT forward the same request to another target representing the same AOR and instance-id. The targeted instance has already provided its response.

The proxy uses the next-hop target of the message and the value of any stored Path header field vector in the registration binding to decide how to forward and populate the Route header in the request. If the proxy is co-located with the registrar and stored information about the flow to the UA that created the binding, then the proxy MUST send the request over the same 'logical flow' saved with the binding, since that flow is known to deliver data to the specific target UA instance's network flow that was saved with the binding.

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

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

8. STUN Keep-Alive Processing

This section describes changes to the SIP transport layer that allow SIP and STUN [RFC5389] Binding Requests to be mixed over the same flow. This constitutes a new STUN usage. The STUN messages are used to verify that connectivity is still available over a UDP flow, and to provide periodic keep-alives. These STUN keep-alives are always sent to the next SIP hop. STUN messages are not delivered end-to-end.

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

If a server compliant to this section receives SIP requests on a given interface and UDP port, it MUST also provide a limited version of a STUN server on the same interface and UDP port.

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

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

A SIP node MUST NOT send STUN requests over a flow unless it has an explicit indication that the target next-hop SIP server claims to support this specification. UACs MUST NOT use an ambiguous configuration option such as "Work through NATs?" or "Do keep-alives?" to imply next-hop STUN support. A UAC MAY use the presence of an "ob" URI parameter in the Path header in a registration response as an indication that its first edge proxy supports the keep-alives defined in this document.

> Note: Typically, a SIP node first sends a SIP request and waits to receive a 2xx class response over a flow to a new target destination, before sending any STUN messages. When scheduled for the next NAT refresh, the SIP node sends a STUN request to the target.

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

The SIP keep-alive STUN usage requires no backwards compatibility with [RFC3489].

8.1. Use with SigComp

When STUN is used together with SigComp [RFC3320] compressed SIP messages over the same flow, the STUN messages are simply sent uncompressed, "outside" of SigComp. This is supported by multiplexing STUN messages with SigComp messages by checking the two topmost bits of the message. These bits are always one for SigComp, or zero for STUN.

> Note: All SigComp messages contain a prefix (the five most significant bits of the first byte are set to one) that does not occur in UTF-8 [RFC3629] encoded text messages, so for

applications that use this encoding (or ASCII encoding) it is possible to multiplex uncompressed application messages and SigComp messages on the same UDP port. The most significant two bits of every STUN Binding method are both zeroes. This, combined with the magic cookie, aids in differentiating STUN packets from other protocols when STUN is multiplexed with other protocols on the same port.

9. Example Message Flow

Below is an example message flow illustrating most of the concepts discussed in this specification. In many cases, Via, Content-Length, and Max-Forwards headers are omitted for brevity and readability.

In these examples, "EP1" and "EP2" are outbound proxies, and "Proxy" is the authoritativeProxy.

The section is subdivided into independent calls flows; however, they are structured in sequential order of a hypothetical sequence of call flows.

9.1. Subscription to Configuration Package

If the outbound proxy set is already configured on Bob’s UA, then this subsection can be skipped. Otherwise, if the outbound proxy set is learned through the configuration package, Bob’s UA sends a SUBSCRIBE request for the UA profile configuration package [CONFIG-FMWK]. This request is a poll (Expires is zero). After receiving the NOTIFY request, Bob’s UA fetches the external configuration using HTTPS (not shown) and obtains a configuration file that contains the outbound-proxy-set "sip:ep1.example.com;lr" and "sip:ep2.example.com;lr".

```
     [----example.com domain-------------------------]
     Bob         EP1   EP2     Proxy             Config
      |           |     |        |                  |
    1)|SUBSCRIBE->|     |        |                  |
    2)|           |---SUBSCRIBE Event: ua-profile ->|
    3)|           |<--200 OK -----------------------|
    4)|<--200 OK--|     |        |                  |
    5)|           |<--NOTIFY------------------------|
    6)|<--NOTIFY--|     |        |                  |
    7)|---200 OK->|     |        |                  |
    8)|           |---200 OK ---------------------->|
      |           |     |        |                  |
```

In this example, the DNS server happens to be configured so that sip: example.com resolves to EP1 and EP2.

Example Message #1:

```
SUBSCRIBE sip:00000000-0000-1000-8000-AABBCCDDEEFF@example.com
  SIP/2.0
Via: SIP/2.0/TCP 192.0.2.2;branch=z9hG4bKnlsdkdj2
Max-Forwards: 70
From: <anonymous@example.com>;tag=23324
To: <sip:00000000-0000-1000-8000-AABBCCDDEEFF@example.com>
Call-ID: nSz1TWN54x7My0GvpEBj
CSeq: 1 SUBSCRIBE
Event: ua-profile ;profile-type=device
 ;vendor="example.com";model="uPhone";version="1.1"
Expires: 0
Supported: path, outbound
Accept: message/external-body, application/x-uPhone-config
Contact: <sip:192.0.2.2;transport=tcp;ob>
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-AABBCCDDEEFF>"
Content-Length: 0
```

In message #2, EP1 adds the following Record-Route header:

```
Record-Route:
 <sip:GopIKSsn0oGLPXRdV9BAXpT3coNuiGKV@ep1.example.com;lr>
```

In message #5, the configuration server sends a NOTIFY with an external URL for Bob to fetch his configuration. The NOTIFY has a Subscription-State header that ends the subscription.

Message #5

```
NOTIFY sip:192.0.2.2;transport=tcp;ob SIP/2.0
Via: SIP/2.0/TCP 192.0.2.5;branch=z9hG4bKn81dd2
Max-Forwards: 70
To: <anonymous@example.com>;tag=23324
From: <sip:00000000-0000-1000-8000-AABBCCDDEEFF@example.com>;tag=0983
Call-ID: nSz1TWN54x7My0GvpEBj
CSeq: 1 NOTIFY
Route: <sip:GopIKSsn0oGLPXRdV9BAXpT3coNuiGKV@ep1.example.com;lr>
Subscription-State: terminated;reason=timeout
Event: ua-profile
Content-Type: message/external-body; access-type="URL"
 ;expiration="Thu, 01 Jan 2009 09:00:00 UTC"
 ;URL="http://example.com/uPhone.cfg"
 ;size=9999;hash=10AB568E91245681AC1B
Content-Length: 0
```

EP1 receives this NOTIFY request, strips off the Route header, extracts the flow-token, calculates the correct flow, and forwards the request (message #6) over that flow to Bob.

Bob’s UA fetches the configuration file and learns the outbound proxy set.

9.2. Registration

Now that Bob’s UA is configured with the outbound-proxy-set whether through configuration or using the configuration framework procedures of the previous section, Bob’s UA sends REGISTER requests through each edge proxy in the set. Once the registrations succeed, Bob’s UA begins sending CRLF keep-alives about every 2 minutes.

```
     Bob         EP1   EP2     Proxy     Alice
      |           |     |        |         |
    9)|-REGISTER->|     |        |         |
   10)|           |---REGISTER-->|         |
   11)|           |<----200 OK---|         |
   12)|<-200 OK---|     |        |         |
   13)|----REGISTER---->|        |         |
   14)|           |     |--REG-->|         |
   15)|           |     |<-200---|         |
   16)|<----200 OK------|        |         |
      |           |     |        |         |
      |  about 120 seconds later...        |
      |           |     |        |         |
   17)|--2CRLF--->|     |        |         |
   18)|<--CRLF----|     |        |         |
   19)|------2CRLF----->|        |         |
   20)|<------CRLF------|        |         |
      |           |     |        |         |
```

In message #9, Bob’s UA sends its first registration through the first edge proxy in the outbound-proxy-set by including a loose route. The UA includes an instance-id and reg-id in its Contact header field value. Note the option-tags in the Supported header.

Message #9

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/TCP 192.0.2.2;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Bob <sip:bob@example.com>;tag=7F94778B653B
To: Bob <sip:bob@example.com>
Call-ID: 16CB75F21C70
CSeq: 1 REGISTER
Supported: path, outbound
Route: <sip:ep1.example.com;lr>
Contact: <sip:bob@192.0.2.2;transport=tcp>;reg-id=1
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-AABBCCDDEEFF>"
Content-Length: 0
```

Message #10 is similar. EP1 removes the Route header field value, decrements Max-Forwards, and adds its Via header field value. Since EP1 is the first edge proxy, it adds a Path header with a flow token and includes the "ob" parameter.

```
Path: <sip:VskztcQ/S8p4WPbOnHbuyh5iJvJIW3ib@ep1.example.com;lr;ob>
```

Since the response to the REGISTER (message #11) contains the outbound option-tag in the Require header field, Bob's UA will know that the registrar used outbound binding rules. The response also contains the currently active Contacts, and the Path for the current registration.

Message #11

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP 192.0.2.15;branch=z9hG4bKnuiqisi
Via: SIP/2.0/TCP 192.0.2.2;branch=z9hG4bKnashds7
From: Bob <sip:bob@example.com>;tag=7F94778B653B
To: Bob <sip:bob@example.com>;tag=6AF99445E44A
Call-ID: 16CB75F21C70
CSeq: 1 REGISTER
Supported: path, outbound
Require: outbound
Contact: <sip:bob@192.0.2.2;transport=tcp>;reg-id=1;expires=3600
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-AABBCCDDEEFF>"
Path: <sip:VskztcQ/S8p4WPbOnHbuyh5iJvJIW3ib@ep1.example.com;lr;ob>
Content-Length: 0
```

The second registration through EP2 (message #13) is similar except that the Call-ID has changed, the reg-id is 2, and the Route header goes through EP2.

Message #13

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/TCP 192.0.2.2;branch=z9hG4bKnqr9bym
Max-Forwards: 70
From: Bob <sip:bob@example.com>;tag=755285EABDE2
To: Bob <sip:bob@example.com>
Call-ID: E05133BD26DD
CSeq: 1 REGISTER
Supported: path, outbound
Route: <sip:ep2.example.com;lr>
Contact: <sip:bob@192.0.2.2;transport=tcp>;reg-id=2
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-AABBCCDDEEFF>"
Content-Length: 0
```

Likewise in message #14, EP2 adds a Path header with flow token and "ob" parameter.

```
Path: <sip:wazHDLdIMtUg6r0I/oRZ15zx3zHE1w1Z@ep2.example.com;lr;ob>
```

Message #16 tells Bob’s UA that outbound registration was successful, and shows both Contacts. Note that only the Path corresponding to the current registration is returned.

Message #16

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP 192.0.2.2;branch=z9hG4bKnqr9bym
From: Bob <sip:bob@example.com>;tag=755285EABDE2
To: Bob <sip:bob@example.com>;tag=49A9AD0B3F6A
Call-ID: E05133BD26DD
Supported: path, outbound
Require: outbound
CSeq: 1 REGISTER
Contact: <sip:bob@192.0.2.2;transport=tcp>;reg-id=1;expires=3600
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-AABBCCDDEEFF>"
Contact: <sip:bob@192.0.2.2;transport=tcp>;reg-id=2;expires=3600
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-AABBCCDDEEFF>"
Path: <sip:wazHDLdIMtUg6r0I/oRZ15zx3zHE1w1Z@ep2.example.com;lr;ob>
Content-Length: 0
```

9.3. Incoming Call and Proxy Crash

In this example, after registration, EP1 crashes and reboots. Before Bob’s UA notices that its flow to EP1 is no longer responding, Alice calls Bob. Bob’s authoritative proxy first tries the flow to EP1, but EP1 no longer has a flow to Bob, so it responds with a 430 (Flow Failed) response. The proxy removes the stale registration and tries the next binding for the same instance.

```
     Bob         EP1   EP2     Proxy     Alice
      |           |     |        |         |
      |    CRASH  X     |        |         |
      |        Reboot   |        |         |
      |           |     |        |         |
   21)|           |     |        |<-INVITE-|
   22)|           |<---INVITE----|         |
   23)|           |----430------>|         |
   24)|           |     |<-INVITE|         |
   25)|<---INVITE-------|        |         |
   26)|----200 OK------>|        |         |
   27)|           |     |200 OK->|         |
   28)|           |     |        |-200 OK->|
   29)|           |     |<----------ACK----|
   30)|<---ACK----------|        |         |
      |           |     |        |         |
   31)|           |     |<----------BYE----|
   32)|<---BYE----------|        |         |
   33)|----200 OK------>|        |         |
   34)|           |     |--------200 OK--->|
      |           |     |        |         |
```

Message #21

```
INVITE sip:bob@example.com SIP/2.0
To: Bob <sip:bob@example.com>
From: Alice <sip:alice@a.example>;tag=02935
Call-ID: klmvCxVWGp6MxJp2T2mb
CSeq: 1 INVITE
```

Bob’s proxy rewrites the Request-URI to the Contact URI used in Bob’s registration, and places the path for one of the registrations towards Bob’s UA instance into a Route header field. This Route goes through EP1.

Message #22

```
INVITE sip:bob@192.0.2.2;transport=tcp SIP/2.0
To: Bob <sip:bob@example.com>
From: Alice <sip:alice@a.example>;tag=02935
Call-ID: klmvCxVWGp6MxJp2T2mb
CSeq: 1 INVITE
Route: <sip:VskztcQ/S8p4WPbOnHbuyh5iJvJIW3ib@ep1.example.com;lr;ob>
```

Since EP1 just rebooted, it does not have the flow described in the flow token. It returns a 430 (Flow Failed) response.

Message #23

```
SIP/2.0 430 Flow Failed
To: Bob <sip:bob@example.com>
From: Alice <sip:alice@a.example>;tag=02935
Call-ID: klmvCxVWGp6MxJp2T2mb
CSeq: 1 INVITE
```

The proxy deletes the binding for this path and tries to forward the INVITE again, this time with the path through EP2.

Message #24

```
INVITE sip:bob@192.0.2.2;transport=tcp SIP/2.0
To: Bob <sip:bob@example.com>
From: Alice <sip:alice@a.example>;tag=02935
Call-ID: klmvCxVWGp6MxJp2T2mb
CSeq: 1 INVITE
Route: <sip:wazHDLdIMtUg6r0I/oRZ15zx3zHE1w1Z@ep2.example.com;lr;ob>
```

In message #25, EP2 needs to add a Record-Route header field value, so that any subsequent in-dialog messages from Alice’s UA arrive at Bob’s UA. EP2 can determine it needs to Record-Route since the request is a dialog-forming request and the Route header contained a flow token and an "ob" parameter. This Record-Route information is passed back to Alice’s UA in the responses (messages #26, 27, and 28).

Message #25

```
INVITE sip:bob@192.0.2.2;transport=tcp SIP/2.0
To: Bob <sip:bob@example.com>
From: Alice <sip:alice@a.example>;tag=02935
Call-ID: klmvCxVWGp6MxJp2T2mb
CSeq: 1 INVITE
Record-Route:
  <sip:wazHDLdIMtUg6r0I/oRZ15zx3zHE1w1Z@ep2.example.com;lr>
```

Message #26

```
SIP/2.0 200 OK
To: Bob <sip:bob@example.com>;tag=skduk2
From: Alice <sip:alice@a.example>;tag=02935
Call-ID: klmvCxVWGp6MxJp2T2mb
CSeq: 1 INVITE
Record-Route:
  <sip:wazHDLdIMtUg6r0I/oRZ15zx3zHE1w1Z@ep2.example.com;lr>
```

At this point, both UAs have the correct route-set for the dialog. Any subsequent requests in this dialog will route correctly. For example, the ACK request in message #29 is sent from Alice’s UA directly to EP2. The BYE request in message #31 uses the same route-set.

Message #29

```
ACK sip:bob@192.0.2.2;transport=tcp SIP/2.0
To: Bob <sip:bob@example.com>;tag=skduk2
From: Alice <sip:alice@a.example>;tag=02935
Call-ID: klmvCxVWGp6MxJp2T2mb
CSeq: 1 ACK
Route: <sip:wazHDLdIMtUg6r0I/oRZ15zx3zHE1w1Z@ep2.example.com;lr>
```

Message #31

```
BYE sip:bob@192.0.2.2;transport=tcp SIP/2.0
To: Bob <sip:bob@example.com>;tag=skduk2
From: Alice <sip:alice@a.example>;tag=02935
Call-ID: klmvCxVWGp6MxJp2T2mb
CSeq: 2 BYE
Route: <sip:wazHDLdIMtUg6r0I/oRZ15zx3zHE1w1Z@ep2.example.com;lr>
```

9.4. Re-Registration

Somewhat later, Bob’s UA sends keep-alives to both its edge proxies, but it discovers that the flow with EP1 failed. Bob’s UA re-registers through EP1 using the same reg-id and Call-ID it previously used.

```
     Bob         EP1   EP2     Proxy     Alice
      |           |     |        |         |
   35)|------2CRLF----->|        |         |
   36)|<------CRLF------|        |         |
   37)|--2CRLF->X |     |        |         |
      |           |     |        |         |
   38)|-REGISTER->|     |        |         |
   39)|           |---REGISTER-->|         |
   40)|           |<----200 OK---|         |
   41)|<-200 OK---|     |        |         |
      |           |     |        |         |
```

Message #38

```
REGISTER sip:example.com SIP/2.0
From: Bob <sip:bob@example.com>;tag=7F94778B653B
To: Bob <sip:bob@example.com>
Call-ID: 16CB75F21C70
CSeq: 2 REGISTER
Supported: path, outbound
Route: <sip:ep1.example.com;lr>
Contact: <sip:bob@192.0.2.2;transport=tcp>;reg-id=1
 ;+sip.instance="<urn:uuid:00000000-0000-1000-8000-AABBCCDDEEFF>"
```

In message #39, EP1 inserts a Path header with a new flow token:

```
Path: <sip:3yJEbr1GYZK9cPYk5Snocez6DzO7w+AX@ep1.example.com;lr;ob>
```

9.5. Outgoing Call

Finally, Bob makes an outgoing call to Alice. Bob’s UA includes an "ob" parameter in its Contact URI in message #42. EP1 adds a Record-Route with a flow-token in message #43. The route-set is returned to Bob in the response (messages #45, 46, and 47), and either Bob or Alice can send in-dialog requests.

```
     Bob         EP1   EP2     Proxy     Alice
      |           |     |        |         |
   42)|--INVITE-->|     |        |         |
   43)|           |---INVITE---->|         |
   44)|           |     |        |-INVITE->|
   45)|           |     |        |<--200---|
   46)|           |<----200 OK---|         |
   47)|<-200 OK---|     |        |         |
   48)|--ACK----->|     |        |         |
   49)|           |-----ACK--------------->|
      |           |     |        |         |
   50)|-- BYE---->|     |        |         |
   51)|           |-----------BYE--------->|
   52)|           |<----------200 OK-------|
   53)|<--200 OK--|     |        |         |
      |           |     |        |         |
```

Message #42

```
INVITE sip:alice@a.example SIP/2.0
From: Bob <sip:bob@example.com>;tag=ldw22z
To: Alice <sip:alice@a.example>
Call-ID: 95KGsk2V/Eis9LcpBYy3
CSeq: 1 INVITE
Route: <sip:ep1.example.com;lr>
Contact: <sip:bob@192.0.2.2;transport=tcp;ob>
```

In message #43, EP1 adds the following Record-Route header.

```
Record-Route:
  <sip:3yJEbr1GYZK9cPYk5Snocez6DzO7w+AX@ep1.example.com;lr>
```

When EP1 receives the BYE (message #50) from Bob’s UA, it can tell that the request is an "outgoing" request (since the source of the request matches the flow in the flow token) and simply deletes its Route header field value and forwards the request on to Alice’s UA.

Message #50

```
BYE sip:alice@a.example SIP/2.0
From: Bob <sip:bob@example.com>;tag=ldw22z
To: Alice <sip:alice@a.example>;tag=plqus8
Call-ID: 95KGsk2V/Eis9LcpBYy3
CSeq: 2 BYE
Route: <sip:3yJEbr1GYZK9cPYk5Snocez6DzO7w+AX@ep1.example.com;lr>
Contact: <sip:bob@192.0.2.2;transport=tcp;ob>
```

10. Grammar

This specification defines a new header field "Flow-Timer", and new Contact header field parameters, "reg-id" and "+sip.instance". The grammar includes the definitions from [RFC3261]. Flow-Timer is an extension-header from the message-header in the [RFC3261] ABNF.

The ABNF [RFC5234] is:

```
 Flow-Timer     = "Flow-Timer" HCOLON 1*DIGIT

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

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

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

 instance-val   = 1*uric ; defined in RFC 3261
```

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

11. IANA Considerations

11.1. Flow-Timer Header Field

This specification defines a new SIP header field "Flow-Timer" whose syntax is defined in Section 10.

Header Name	compact	Reference
Flow-Timer		[RFC5626]

11.2. "reg-id" Contact Header Field Parameter

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 [RFC3968]. The syntax is defined in Section 10. The required information is:

Header Field	Parameter Name	Predefined Values	Reference
Contact	reg-id	No	[RFC5626]

11.3. SIP/SIPS URI Parameters

This specification augments the "SIP/SIPS URI Parameters" sub-registry as per the registry created by [RFC3969]. The required information is:

Parameter Name	Predefined Values	Reference
ob	No	[RFC5626]

11.4. SIP Option Tag

This specification registers a new SIP option tag, as per the guidelines in Section 27.1 of [RFC3261].

Name: outbound

Description: This option-tag is used to identify UAs and registrars that support extensions for Client-Initiated Connections. A UA places this option in a Supported header to communicate its support for this extension. A registrar places this option-tag in a Require header to indicate to the registering User Agent that the registrar used registrations using the binding rules defined in this extension.

11.5. 430 (Flow Failed) Response Code

This document registers a new SIP response code (430 Flow Failed), as per the guidelines in Section 27.4 of [RFC3261]. This response code is used by an edge proxy to indicate to the Authoritative Proxy that a specific flow to a UA instance has failed. Other flows to the same instance could still succeed. The Authoritative Proxy SHOULD attempt to forward to another target (flow) with the same instance-id and AOR. Endpoints should never receive a 430 response. If an endpoint receives a 430 response, it should treat it as a 400 (Bad Request) per normal procedures, as in Section 8.1.3.2 of [RFC3261]. This response code is defined by the following information, which has been added to the method and response-code sub-registry under the SIP Parameters registry.

Response Code	Reference
Request Failure 4xx	
430 Flow Failed	[RFC5626]

11.6. 439 (First Hop Lacks Outbound Support) Response Code

This document registers a new SIP response code (439 First Hop Lacks Outbound Support), as per the guidelines in Section 27.4 of [RFC3261]. This response code is used by a registrar to indicate that it supports the ’outbound’ feature described in this specification, but that the first outbound proxy that the user is attempting to register through does not. Note that this response code is only appropriate in the case that the registering User Agent advertises support for outbound processing by including the outbound option tag in a Supported header field. Proxies MUST NOT send a 439 response to any requests that do not contain a "reg-id" parameter and an outbound option tag in a Supported header field. This response code is defined by the following information, which has been added to the method and response-code sub-registry under the SIP Parameters registry.

Response Code	Reference
Request Failure 4xx	
439 First Hop Lacks Outbound Support	[RFC&rfc.number;]

11.7. Media Feature Tag

This section registers a new media feature tag, per the procedures defined in [RFC2506]. The tag is placed into the sip tree, which is defined in [RFC3840].

Media feature tag name: sip.instance

ASN.1 Identifier: 23

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 (equality relationship).

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 5626

Security Considerations: This media feature tag can be used in ways which affect application behaviors. For example, the SIP caller preferences extension [RFC3841] allows for call routing decisions to be based on the values of these parameters. Therefore, if an attacker can modify the values of this tag, they might be able to affect the behavior of applications. As a result, applications that 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.

12. Security Considerations

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

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

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

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

The Security Considerations discussed in [RFC3261] and [RFC3327] are also relevant to this document. For the security considerations of generating flow tokens, please also see Section 5.2. A discussion of preventing the avalanche restart problem is in Section 4.5.

This document does not change the mandatory-to-implement security mechanisms in SIP. User Agents are already required to implement Digest authentication while support of TLS is recommended; proxy servers are already required to implement Digest and TLS.

13. Operational Notes on Transports

This entire section is non-normative.

[RFC3261] requires proxies, registrars, and User Agents to implement both TCP and UDP but deployments can chose which transport protocols they want to use. Deployments need to be careful in choosing what transports to use. Many SIP features and extensions, such as large presence notification bodies, result in SIP requests that can be too large to be reasonably transported over UDP. [RFC3261] states that when a request is too large for UDP, the device sending the request attempts to switch over to TCP. It is important to note that when using outbound, this will only work if the UA has formed both UDP and TCP outbound flows. This specification allows the UA to do so, but in most cases it will probably make more sense for the UA to form a TCP outbound connection only, rather than forming both UDP and TCP flows. One of the key reasons that many deployments choose not to use TCP has to do with the difficulty of building proxies that can maintain a very large number of active TCP connections. Many deployments today use SIP in such a way that the messages are small enough that they work over UDP but they can not take advantage of all the functionality SIP offers. Deployments that use only UDP outbound connections are going to fail with sufficiently large SIP messages.

14. Requirements

This specification was developed to meet the following requirements:

1. Must be able to detect that a UA supports these mechanisms.

2. Support UAs behind NATs.

3. Support TLS to a UA without a stable DNS name or IP address.

4. Detect failure of a connection and be able to correct for this.

5. Support many UAs simultaneously rebooting.

6. Support a NAT rebooting or resetting.

7. Minimize initial startup load on a proxy.

8. Support architectures with edge proxies.

15. Acknowledgments

Francois Audet acted as document shepherd for this document, tracking hundreds of comments and incorporating many grammatical fixes as well as prodding the editors to "get on with it". Jonathan Rosenberg, Erkki Koivusalo, and Byron Campen 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 document that formed the initial text of this specification. Additionally, many of the concepts here originated at a connection reuse meeting at IETF 60 that included the authors, Jon Peterson, Jonathan Rosenberg, Alan Hawrylyshen, and Paul Kyzivat. The TCP design team consisting of Chris Boulton, Scott Lawrence, Rajnish Jain, Vijay K. Gurbani, and Ganesh Jayadevan provided input and text. Nils Ohlmeier provided many fixes and initial implementation experience. In addition, thanks to the following folks for useful comments: Francois Audet, Flemming Andreasen, Mike Hammer, Dan Wing, Srivatsa Srinivasan, Dale Worely, Juha Heinanen, Eric Rescorla, Lyndsay Campbell, Christer Holmberg, Kevin Johns, Jeroen van Bemmel, Derek MacDonald, Dean Willis, and Robert Sparks.

16. References

16.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

[RFC2506] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, March 1999.

[RFC3261] 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.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC3327] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[RFC3581] Rosenberg, J. and H. Schulzrinne, "An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing", RFC 3581, August 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[RFC3841] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[RFC3968] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", BCP 98, RFC 3968, December 2004.

[RFC3969] 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.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

16.2. Informative References

[CONFIG-FMWK]	Petrie, D. and S. Channabasappa, Ed., "A Framework for Session Initiation Protocol User Agent Profile Delivery", Work in Progress, February 2008.
[NAT-SCEN]	Boulton, C., Rosenberg, J., Camarillo, G., and F. Audet, "Best Current Practices for NAT Traversal for Client-Server SIP", Work in Progress, September 2008.
[RFC0768]	Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.
[RFC0793]	Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
[RFC1035]	Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
[RFC2104]	Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.
[RFC2131]	Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.
[RFC2782]	Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
[RFC3320]	Price, R., Bormann, C., Christoffersson, J., Hannu, H., Liu, Z., and J. Rosenberg, "Signaling Compression (SigComp)", RFC 3320, January 2003.
[RFC3489]	Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, March 2003.
[RFC3986]	Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
[RFC4340]	Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

[RFC4648]   Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC4960]   Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5246]   Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5627]   Rosenberg, J., "Obtaining and Using Globally Routable User Agent URIs (GRUUs) in the Session Initiation Protocol (SIP)", RFC 5627, October 2009.

Appendix A. Default Flow Registration Backoff Times

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

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

Appendix B. ABNF

This appendix contains the ABNF defined earlier in this document.

```
CRLF = CR LF
double-CRLF = CR LF CR LF
CR = %x0D
LF = %x0A

Flow-Timer     = "Flow-Timer" HCOLON 1*DIGIT

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

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

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

instance-val   = 1*uric ; defined in RFC 3261
```

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)
Unaffiliated

EMail: rohan@ekabal.com

Francois Audet (editor)
Skype Labs

EMail: francois.audet@skypelabs.com