Discovering Outbound Proxies and Providing High Availability with Client Initiated Connections in the Session Initiation Protocol (SIP)
draft-rosenberg-sip-outbound-discovery-mid-dialog-00

Abstract

In many deployment configurations, Session Initiation Protocol (SIP) clients are capable of initiating connection requests towards their SIP server, but the SIP server cannot open connections towards the client. Specifications have been developed which allow for a client-initiated connection to be used for incoming requests towards the client. This outbound connection involves the use of a SIP proxy, called an outbound proxy, that the client connects to. However, the specification does not provide a means to discover the outbound...
proxy, nor does it support high availability for failures of the outbound proxy mid-dialog. This specification fills those gaps. The resulting mechanism additionally provides solutions for inter-proxy connection reuse and usage of certificates with SIP.

Table of Contents

1. Introduction .................................................. 3
2. Requirements .................................................. 5
3. Overview of Operation ........................................ 7
4. Client Behavior ............................................... 12
   4.1. Choosing a Flow for Sending a Request ................. 12
   4.2. Associating URI with Flows .............................. 13
5. Server Behavior ............................................... 14
6. Connection Management ........................................ 15
7. User Agent Behavior .......................................... 15
8. Outbound Proxy Behavior ...................................... 16
   8.1. Receipt of Initial REGISTER ............................. 16
   8.2. Receipt of REGISTER Refresh ............................ 17
   8.3. Record-Routing Dialog Forming Requests ............... 17
   8.4. Additional Flow Selection Rule ......................... 17
   8.5. Mid-Dialog Request Handling ......................... 18
9. Home Proxy Behavior ........................................... 18
10. Impacts to SIP Outbound ..................................... 18
11. Backwards Compatibility Considerations .................. 19
12. Interactions with Connection Reuse ......................... 20
13. Security Considerations .................................... 20
14. IANA Considerations ......................................... 21
15. Example ....................................................... 22
16. Acknowledgements ........................................... 28
17. References ................................................... 28
   17.1. Normative References ................................. 28
   17.2. Informative References ............................... 29
18. Author's Address ............................................. 30
19. Intellectual Property and Copyright Statements ........ 31
1. Introduction

The Session Initiation Protocol (SIP), RFC 3261 [1] allows a client, called a user agent (UA) to initiate and receive communications sessions. In order to receive an incoming SIP request, which may arrive at any time, a SIP proxy will need to send a SIP request towards a UA. In RFC 3261, this was done by having the proxy resolve the domain name of the Request-URI [2] (if it had a hostname), and then sending a request towards the resulting IP address, or opening a TCP connection towards that address. In environments where there are no NATs or firewalls between the UA and the proxy, this approach works well. However, most modern deployments have NAT and firewall devices between the UA and their proxy.

To resolve this, a specification, generally called sip-outbound [3] was developed. This specification has the UA create a "flow" towards its outbound proxy at the time of registration. A flow is equivalent to a TCP connection in the case of TCP, and in the case of UDP, the flow represents the 5-tuple used to send the UDP packet. Once the flow is created, it is held open by the UA and the outbound proxy. When an incoming request arrives from the network towards the UA, it is routed to the outbound proxy. The outbound proxy then routes the request over the existing flow created by the client. SIP-outbound also provides a keepalive mechanism, utilizing the Simple Traversal Underneath NAT (STUN) [5], generated by the client, which keeps any bindings allocated by the NAT fresh. The keepalive also allows the client to detect failures of its outbound proxy, and initiate a new connection when this happens. SIP-outbound also allows a client to open multiple flows to different outbound proxies, which provides high availability. In case of proxy failure, a new incoming request can get routed over the second connection, and the UA can eventually re-establish a second connection.

However, sip-outbound does not address two issues that are key to deployment. Firstly, it does not specify how a client discovers its outbound proxy. In many deployments, a domain will separate the outbound proxy from the home proxy; the latter being the proxy which provides registration processing and location service functions. A UA will initially only have its address-of-record. A DNS lookup of its AOR using RFC 3263 [2] will necessarily return the IP address of the home proxy, since that is the proxy that needs to be contacted by clients that direct calls towards the UA. However, a UA needs to direct its initial registration towards its outbound proxy.

The second issue is that of mid-dialog failover. SIP-outbound explicitly states that it does not support mid-dialog failover. Specifically, if a UA has flows to a multiplicity of outbound proxies, and in the middle of a dialog, the proxy through which
requests within that dialog are flowing should fail, any mid-dialog requests will not be delivered. Supporting mid-dialog failover requires three properties which currently are not present in SIP-outbound, as illustrated in Figure 1. There are three outbound proxies, and the UA has established flows through numbers 2 and 3. Proxy 2 fails. For a mid-dialog request to be delivered, the following must happen:

1. When a mid-dialog request arrives at the home proxy (any home proxy), it has to know to deliver the request to another outbound proxy, and specifically, the outbound proxy to which the UA has another flow (proxy 3)
2. When the request arrives at the backup outbound proxy, it has to know to deliver it over an existing flow towards the UA.

3. When the UA originates a request, it needs to know to deliver it through the backup proxy.

Currently, sip-outbound will not result in these three behaviors. When the original dialog-forming request passes through outbound proxy 2, it will record-route with a URI which routes back to itself. Thus, mid-dialog requests would not be delivered to the backup proxy. Should proxy two instead record-route with a cluster URI (one that identifies all three outbound proxies), there is no way for the home proxy to know that outbound proxy 1 could not be used for delivering message 1.

2. Requirements

We believe that there are several key requirements to the solution:

REQ 1: The solution should ensure that, if a proxy inserted a URI into a Record-Route header field, that it should continue to see that URI in the Route header fields of mid-dialog requests, even if the mid-dialog request is sent to a backup proxy. Many implementations place state and other critical information into the user part of the Record-Route header field, and use that information in all mid-dialog requests for proper processing of the call. Consequently, this property has to be retained for existing implementations to continue to rely on this feature of SIP.

REQ 2: The solution should not require that edge proxies have to replicate any kind of dynamic state between them. However, it is acceptable that the solution be limited to failover cases where the proxies in a cluster which back each other up are all from the same vendor, and thus can share common proprietary algorithms, such as those for construction of URIs.

REQ 3: The solution shall work in cases where the outbound proxy is in the same domain as the home proxy, and ones where it is not in the same domain.

REQ 4: The solution shall not require that there are no further proxies between the outbound proxy and the home proxy. For example, in 3gpp IMS, the outbound proxy would be the P-CSCF, and the home proxy the S-CSCF. There can be an additional proxy, the I-CSCF, between the two.
REQ 5: The solution shall not require any non-standard uses to DNS, nor shall it require any unduly complication DNS configurations to be supported. In particular, it shall not introduce any additional DNS configuration requirements beyond those already specified in RFC 3263. Coordination of DNS and SIP network configurations is a common source of operational difficulty, and this solution should not exacerbate it.

REQ 6: The solution shall work even for clients with no DNS, which have only a static mapping between their AOR and a set of IP addresses for the home proxy it maps to. Unfortunately, some deployments support very thin clients which do not rely on DNS at all.

REQ 7: The solution shall support failures of the outbound proxy, failures of the NAT between the client and its outbound proxy, and transient errors and network problems which cause loss or disconnection of TCP connections between the client and its outbound proxy.
3. Overview of Operation

We explain operation of the system using the network model of Figure 2. This figure shows a UA, X, with an AOR of sip:X@example.com. There are three outbound proxies, and two home proxies. The home proxies share registration state through some database mechanism, not shown.
The mechanism is best explained by example. Though the algorithm itself is simple and relatively easy to specify, the rationale of why it works is less clear and requires a call flow to best understand.

The mechanism assumes that the UA starts with knowledge of its AOR (sip:x@example.com) and credentials for authenticating itself to that AOR. The mechanism also assumes that the UA is aware of zero or one domain names for an outbound proxy farm. In cases where the SIP server DHCP option [7] is used, this will provide the client with a domain name with which it can construct a SIP URI of the form sip: domain-name. In such a case, this will give the client a single outbound proxy URI. In this example, DHCP would return outbound.example.com, which points to the cluster of three outbound proxies. If DHCP is not used, proprietary link-layer technologies, such as those used in IMS, can provide the client with an IP address or port for its outbound proxy. If no such mechanism is used, the client starts with zero outbound proxy URIs.

If the UA has no outbound proxy learned from DHCP, it sends its REGISTER request to the home proxy. The home proxy redirects the request with a 305, pointing towards the outbound proxy farm.

The REGISTER is then processed by a proxy in the outbound proxy farm (say, proxy 1), and a home proxy (say proxy 1). Prior to forwarding the 200 OK response to the initial REGISTER request, outbound proxy 1 will add an Alternative-Proxies header field, which contains a URI for each additional proxy that A would like to client to use as an outbound proxy. The UA will add those URIs to its outbound proxy set, and proceed to REGISTER through them. This provides a dynamic discovery mechanism, whereby a client can start with zero outbound proxies and learn a set of N outbound proxies, selected by the provider at the time of registration. This dynamic selection gives great flexibility into the set used for any particular UA. Furthermore, since the selection is done by the proxies themselves, high availability is facilitated.

To achieve high availability, new rules for determining where to send a request are defined. The idea is that each flow is associated with URI learned through SIP transactions passing over that flow. When a client (whether its a UA or a proxy) goes to send a request, it chooses the flow that is associated with the URI which is a ‘longest match’ to the URI that the request is to be sent to. This causes a form of ‘stickiness’, whereby once a request for a particular URI has been sent through a flow, it tends to stay on that flow. To take advantage of this for HA, the outbound proxies insert a URI into the Path header field of the REGISTER (which is in turn echoed in the Service-Route) which points to the entire cluster (outbound.example.com) but contains a user part that is distinct for
the user performing the registration and identifies the proxies that have been selected for that user. This stickiness applies to both the flow between the UA and outbound proxies, and between the outbound proxies and the home proxy which processed the each registration. This is shown in Figure 3. The Path URI used for the registration from UA X (which we denote as "X-Path") is associated by the UA with the flows to outbound proxies 2 and 3. The same URI is linked to the flow from home proxy 1 to outbound proxy 2 and home proxy 2 to outbound proxy 3 (we assume each home proxy processed one of the two registrations).
When the UA initiates a call, the topmost Route will be the Service Route (which is identical to the Path URI), causing the request to 'stick' to one of the two existing flows. Similarly, when an incoming request arrives at home proxy 1, the resulting Path will produce a Route who’s URI is an exact match for the flow to outbound proxy 2, causing the request to go to the ‘right’ place. Had the initial two registrations both been handled by home proxy 2, and an incoming request was delivered to home proxy 1, home proxy 1 would not have the Path URI associated with a flow to the right outbound proxies. Thus, it will choose one of the three proxies arbitrarily (based on RFC 3263 [2]. If it should pick the wrong one (outbound proxy 2), the remedy is simple. The user part of the Path URI contains information about which of the outbound proxies were used for UA X. Outbound proxy 1 would see that it was not one of the proxies, and use a 305 [6] to redirect home proxy 1 towards home proxy 2 or 3. This request would then be delivered, and also cause home proxy 1 to now have a cached ‘stickiness’ from the Path URI to the right outbound proxy. This would avoid the need for future redirections as long as the home proxy retains this association.

As the call setup request passes through the outbound proxy, it record-routes with the same URI that was used in the Path, but with an additional dialog-specific URI parameter. This parameter can be used to contain additional dialog state (if needed). This causes both the home proxy and the UA to associate this URI to the flow through the proxy used for the dialog. This is shown pictorially in Figure 4. In this example, the call was setup through outbound proxy 2 and home proxy 1. The outbound proxy record-routed with URI "2-RR", which is identical to "X-Path" but with the addition of an additional parameter. This URI is therefore the most specific match to "2-RR" but is also a less-specific match to "X-Path". This is the key to mid-dialog HA.
Consider once more the failure of outbound proxy 2. The remote UA in the dialog sends a mid-dialog request, such as a BYE. This arrives at home proxy 2. The next Route header is "2-RR". Home proxy 1 finds this to be the most specific match to the flow to outbound proxy 2. However, this proxy has crashed (a fact that is discovered by home proxy 1 through any number of means). "2-RR" is a less specific match for X-Path, and thus the request goes on that flow next, arriving correctly at the backup proxy, outbound proxy 3.

Outbound proxy 3 has access to any state or state references that
outbound proxy 2 placed into the Record-Route URI. It extracts information from the user part of the record-route, which identifies that this is a URI for user X, and thus can be handled by sending the request over the flow that the proxy has towards UA X.

Similar processing happens for requests initiated by UA X. A mid-dialog request will have a topmost Route header field with a value of "2-RR" and thus be the most specific match for outbound proxy 2, but fall back to outbound proxy 3 should number 2 be unavailable.

When outbound proxy 2 recovers it will immediately begin receiving mid-dialog traffic for the dialogs. If it likes, it can redirect such requests towards outbound proxy 3, using a URI which will ‘stick’ future mid-dialog requests to outbound proxy 3 instead.

The association of URI to flows has to be maintained by proxies and the UA. This requires them to track dialog state in order to be most effective. However, if this state is lost, it is gracefully recovered through subsequent signaling, at the cost of additional redirects that cause it to be re-built. For this reason, this state does not need to be replicated between components in a cluster.

4. Client Behavior

The rules defined here apply to any client, whether it’s a user agent client or proxy. Two sets of rules are defined. One is for associating URIs with flows, and the other for choosing a flow for sending a request.

All clients compliant with this specification MUST support the 305 redirect mechanism and the revised Service-Route semantics described in [6]. This mechanism will cause a UA, if it receives a 305, to use the URI in the Contact of the 305 in a Route header of the redirect, rather than populating the Request-URI.

4.1. Choosing a Flow for Sending a Request

When a client sends a request, that request will always be directed towards a SIP URI. This would be the topmost Route header field in the request if Route was present, else the Request-URI. Call this the target URI. The client will need to decide whether to send a request for a particular target URI on an existing flow or a new flow, and if on an existing flow, which one. This is done by comparing the target URI to the URIs associated with each existing flow. The essential idea is to pick a flow that is the ‘most specific’ match for a URI. A match is called "most specific" when the two URI are strongly equivalent [4], which means that they have
the same set of URI parameters with the same values. A match is called "partly specific" when the two URI are equal based on RFC 3261 rules for equality. This requires the user part and domains to match, but allows one URI to contain URI parameters while the other does not. Finally, a match is called "least specific" when the only thing that matches is the domain part of the URI. When a client wants to send a request to a target URI, it MUST choose a flow which is a most specific match. If there are no most specific matches, it MUST choose one that is partly specific. If none are partly specific, it MUST perform an RFC 3263 [2] resolution of the hostname. If the resulting IP address and port match that of an existing connection which is a least specific match for the URI, that connection MUST be used. Otherwise, the agent MUST initiate a new connection. The usage of RFC 3263 here allows for load balancing to still occur. If, in any of the previous steps, more than one flow is a match, the UA chooses one randomly.

4.2. Associating URI with Flows

Once a request is sent on a flow, it is possible that the request will result in further URI being associated with the flow. Whether this happens depends on the method and the response code.

If the request was a REGISTER request that generated a successful response (2xx), the client searches for the Service-Route header field in the response. If the client is a UA, it takes the topmost value in the Service-Route header field. If the client is a proxy, the proxy takes the URI it inserted into the Path header field of the request, assuming it had inserted one. It searches the values in the Service-Route header field for that URI. When it finds it, it selects the next URI in the Service-Route header field. The client MUST associate that URI with the flow on which the request was sent. The client MAY unassociate the URI with the flow at the time of registration expiry. Note that this requires clients compliant to this mechanism to track registration lifetimes in order to expire these associations.

If the request was a dialog-forming request, such as INVITE or SUBSCRIBE, which generated a successful response, the client searches for the Record-Route header field in the response. If the client is a UA, it takes the topmost value in the Record-Route header field. If the client is a proxy, the proxy takes the URI it inserted into the Record-Route header field of the request, assuming it had inserted one. It searches the values in the Record-Route header field for that URI. When it finds it, it selects the next URI in the Record-Route header field. The client MUST associate that URI with the flow on which the request was sent. The client MAY unassociate the URI with the flow at the time of termination of the dialog. Note
that this requires clients compliant to this mechanism to track dialog states in order to expire these associations.

If the result of a request is 305, the client MUST follow the procedures in [6], which will cause the client to recurse and send the request to the Contacts. If, prior to the redirection, the target URI was associated with the flow on which the request was sent, the client SHOULD unassociate the URI with the flow upon receipt of a 305. This provides a way for a server to unstick itself from a particular URI, and is useful for redirecting traffic after recovery from a failure. If, after recursion, the result of the request is a success, the rules described above are used to associate URI with the flow on which the recursed request was sent.

If the response to a request is a 5xx, the client SHOULD retry the request by re-running the selection algorithm described here, but acting as if the flow that was just described (and its associated URI) do not exist. This will have the effect of causing failover across flows. A successful response to a retried request, based on the results above, may cause the association of URI with other flows. Note that a 5xx response will not unstick the URI to the flow on which the first request was sent.

5. Server Behavior

When a server, which is either a UAS or proxy server, receives a request, it performs the processing in this section.

If the server is on the receiving side of a TLS connection open, and the client offered a TLS certificate, the server MUST create a SIP URI of the form sip:domain, where the domain is identical to the domain in the subjectAltName of the client-provided certificate. This URI MUST be associated with the connection that was just established. This processing is specified to TLS. [[OPEN ISSUE: In principle, this mechanism could be applied to UDP or other transports as long as there was some way to securely bind the sender with an identity. However the only defined way in SIP is TLS]].

If the server receives a REGISTER request, the server examines the topmost Path header field value in the request. If the Path header field is present, and the domain in the topmost URI matches the domain of the URIs associated with the flow, the URI is added to the list of URI associated with the flow if the server generates a successful response. If, when the request was received, no URI were associated with the flow, the topmost Path URI MUST NOT be associated with the flow.
If the server receives a dialog-forming request, the server examines the topmost Record-Route header field value in the request. If there is a Record-Route header field present, and the domain in the topmost URI matches the domain of the URIs associated with the flow, the URI is added to the list of URI associated with the flow if the server generates a successful response. If, when the request was received, no URI were associated with the flow, the topmost Record-Route URI MUST NOT be associated with the flow.

6. Connection Management

The procedures in this section apply to all clients and servers.

Once a connection has been established and becomes associated with URI, the connection SHOULD be retained as long as any URI remain associated with the connection. Even if there are no URI associated with a connection, it is RECOMMENDED that connections be retained in order to avoid the delays associating with re-establishing them at a later time. Should a connection close as a result of error, or be closed by a peer, the agent SHOULD retain the list of associated URI and also the IP address, port and protocol that the connection had been established to. If a request is to be sent which matches one of those URI, the agent SHOULD attempt to re-open the connection to the IP address and port and protocol. If the IP address is owned by another host now, this won’t work. May need to add some text on checking resulting TLS certs against the domain names). If, after 10 seconds, the connection could not be re-established, the agent SHOULD discard the list of associated URI. Caching the list of associated URI helps proxies and user agents be robust against temporary disconnections due to transient conditions. It takes time to rebuild the set of associated URI, and this state should not be lightly discarded.

7. User Agent Behavior

A UA compliant to this specification MUST include a Supported header field in its REGISTER requests with the "alt-proxies" option tag.

A UA MUST, upon receipt of a REGISTER response containing an Alternative-Proxies header field, process that header field. For each URI in the header field value, it MUST add that URI as a new additional outbound proxy. Furthermore, when generating a REGISTER request for that outbound proxy, it MUST create a new flow towards the URI, regardless of whether an existing flow matches the URI. Often these URI will use the maddr parameter in order to cause the UA to use a specific host as an outbound proxy. Note that, in the case
of TLS, the server certificate is matched against the domain of the URI, rather than any value in the maddr parameter.  [[OPEN ISSUE: still mildly uncomfortable on this; need to consider whether there is a real issue here or not.  I can’t think of a problem per se.]].

8. Outbound Proxy Behavior

Beyond the general rules for clients and servers defined in Section 4 and Section 5, an outbound proxy follows the rules defined here.

8.1. Receipt of Initial REGISTER

An outbound proxy identifies an initial REGISTER by the topmost Route header field in the REGISTER request. If this URI has no user part, and the hostname matches the hostname for the server cluster for that proxy, the request is an initial REGISTER request.

An proxy MUST add a Path header field to the request. The host portion of the URI MUST match the hostname for the server cluster for that proxy, which will in turn be equal to the hostname in the Route header field of the received REGISTER request. The user part is encoded as a matter of local policy. However, it MUST have the following properties. It MUST encode the identity of the other proxies that have been selected as the outbound proxy set for this user. It MUST encode the instance ID and reg ID of the request in such a way that any other proxy in the cluster can understand it (this is assuming the usage of SIP outbound, where there will always be just one contact). Furthermore, it MUST contain the authenticated identity of the sender of the request [[TODO: need to describe procedures when this is not present]]. This identity, in the case of SIP digest, is obtained from the username field of the Authorization or Proxy-Authorization header field. Note that the outbound proxy doesn’t actually have to perform the authentication. If it does not, but rather it is done by a downstream proxy trusted by the outbound proxy, that username is used. If an outbound proxy does not authenticate, and has no trust relationship with the downstream proxy performing the authentication, the mechanism in this specification cannot be applied.

OPEN ISSUE: this seems moderately brittle. In theory at least its hard for an outbound proxy to know what downstream element performed the authentication exchange. However in nearly all real deployments, including ones where the proxies are in different domains, there is knowledge as a consequence of the deployment topology. Is this acceptable?

When the 200 OK to the REGISTER is received, the outbound proxy
selects which other proxies it desires the client to initiate an outbound connection to. This MUST be the same set that was used during the encoding of the Path header field. For each additional outbound proxy that is to be used, the proxy MUST add a value to the Alternative-Proxies header field. This value MUST be identical to the URI inserted into the Path header field, with the exception of the removal of the keepalive=stun parameter and the addition of the maddr parameter, which identifies the specific proxy or farm of proxies from which the additional outbound proxy is to be selected.

The proxy MUST associate the username, instance ID and reg-ID with the flow that the request arrived on. It MUST maintain this association for the duration of the registration. It MUST retain the chosen Path URI for the duration of the registration and associate it with the username/realm and instance ID.

TODO: add text on handling case of a second flow on the same proxy.

8.2. Receipt of REGISTER Refresh

A proxy can distinguish an initial REGISTER from a request by the topmost URI. If the topmost URI matches a Path URI it, or another proxy in the cluster, had previously associated with a registration, it is a refresh. The outbound proxy MUST insert the same URI into the Path header field of the REGISTER request. It MUST NOT insert the Alternative-Proxies header field into the response.

8.3. Record-Routing Dialog Forming Requests

A proxy can detect a request as out of dialog based on the topmost Route URI. If it equals the Path URI of an existing registration, and the request method is not REGISTER, and the To tag is absent, it is a non-dialog request.

If the request is a dialog forming request, an outbound proxy MUST record-route. The URI used for record-routing MUST be "partly equal" to the Path URI (which will be present in the topmost URI). The proxy MAY add additional URI parameters which would cause the record-routed URI to not be strongly equal to the Path URI.

8.4. Additional Flow Selection Rule

Outbound proxies follow the general client and server processing rules defined in Section 4 and Section 5, with one very important addition. If, when trying to send a request towards a UA, the procedures in Section 4 instruct the outbound proxy to open a new flow, the outbound proxy performs one additional check to see if it can reuse an existing flow. If the Route header field in the request
that was received had a structure indicates that it contains a
username, instance ID and reg-ID, and it was not received over a flow
from a client (in other words, it came from the home proxy or another
proxy before it) the outbound proxy checks to see if it has any flows
associated with that username and instance ID, with any reg-ID. If
it finds a match, the request is sent on that flow.

If there is no match, the proxy decodes the user part of the Route
URI in the request it received. This user part encodes the
identities of the outbound proxies assigned to this user. The proxy
MUST generate a 305 redirect, and include a single Contact header
field value for each outbound proxy. Each value MUST be equal to the
Path URI (based on RFC 3261 equivalence rules), but MUST contain an
maddr parameter that points the request to the specific outbound
proxy that was selected.

8.5. Mid-Dialouge Request Handling

There are no additional behaviors required for mid-dialog request
processing, beyond those specified in Section 4 and Section 5 and the
additional flow selection rule defined above. These rules, when
combined together, produce the desired high availability effect.

9. Home Proxy Behavior

Home proxies follow the procedures defined for all clients and
servers as described above.

Additionally, if a home proxy receives a REGISTER request, and this
request didn’t arrive from a proxy that is a valid edge proxy, the
home proxy MUST redirect the request, and include a SIP URI in the
Contact header field which identifies the outbound proxy farm for its
own domain. This redirect MUST follow the procedures of [6].

10. Impacts to SIP Outbound

At this time, SIP outbound is still under specification. Though it
is not an objective of SIP outbound to specify the mid-dialog HA
mechanisms described here, it is preferable that SIP outbound be
specified in such a way that a UA compliant only to SIP-outbound
would still get high availability, once the network is upgraded with
the mechanisms defined here.

The following changes are suggested to SIP outbound to accomplish
this:
Firstly, the client algorithm described above needs to be moved into sip-outbound and trimmed to apply only to UA. This provides the essential HA mechanism for the client.

The additional behavior defined here for user agents is the discovery mechanism afforded by the Alternative-Proxies header field and the modified Service-Route semantics in [6]. Without these mechanisms, the client can either be configured with its outbound proxies or it can learn local ones via DHCP. If configured ones are used, as long as they are configured as URI and present in the topmost Route header of all requests that are not part of a dialog (in which case the record-route set is used), HA will work from the client. In the case of proxies learned by DHCP, only a hostname is provided, and just a single one. The recommendation here is that sip-outbound recommend several DNS queries to obtain two IP addresses, and then to use sip: <hostname>;maddr=<IP-addr> as the URI for each outbound proxy. This will also allow for HA to work without further client change. However, lack of Service-Route support will require the outbound proxies to perform state replication to compensate.

Subsequent support of the remainder of this specification (including the modified Service-Route mechanism) would eliminate the need for state replication in the outbound proxies, provide more dynamic HA mechanisms (such that each agent can have a different proxy assigned to it), and allow for different numbers of outbound proxies on a UA-by-UA basis.

11. Backwards Compatibility Considerations

This specification defines two fairly separable mechanisms, and each has its own backwards compatibility consideration. The first is the Alternative-Proxies header field, and the second is the ‘stickiness’ behavior defined for clients and servers.

Backwards compatibility for the Alternative-Proxies header field is readily accomplished with an option tag. A UA includes the “alt-proxies” option tag in the Supported header field of its registrations, indicating support for this header. An outbound proxy inserting an Alternative-Proxies header field into a REGISTER response would add a Require header field with that option tag, informing the UA that the mechanism must be used.

The connection stickiness is more problematic. A proxy will want to know whether its neighbors support the mechanism prior to construction of a URI for a Record-Route or Path header field. For peers which support the mechanism, the URIs would have the structure described here, which has the important property that the host part
identifies the entire cluster and not a specific proxy. However, this will not work as well with peers which don’t support the stickiness, since requests targeted to that URI would go anywhere in the cluster (based on RFC 3263 processing alone). There are several ways to handle this.

Firstly, a proxy MAY prod a specific peer with an OPTIONS request to determine support for this mechanism. A proxy compliant to the client and server stickiness rules described above will indicate such with a "sticky" option tag in the Supported header field of an OPTIONS response.

Alternatively, a proxy can simply assume all peers support the mechanism. If they do not, the worse case is that requests get routed to the wrong proxy in the cluster. As long as all proxies in a cluster support the mechanism here, the mis-targeted proxy can either redirect the request to the right proxy (assuming the upstream element supports the new 305 semantics; backwards compatibility for THAT mechanism is discussed in [6]). If the upstream element doesn’t support the new 305 semantics, the mistargeted element can literally proxy the request to the correct place. Consequently, lack of backwards compatibility still allows the system to function - it just adds additional costs of an extra proxy hop to get requests to the correct place. Finally, in some deployments it is simply known what the topology is, and support for this specification can be a configured property of an element that a proxy communicates with.

12. Interactions with Connection Reuse

The mechanism defined in this specification has an important and unintended side effect. When two proxies are communicating with mutual TLS, it results in connection reuse of that TLS connection. This connection reuse mechanism works regardless of the presence of NAT between a pair of proxies. It also results in load balancing, though with less control than is normally afforded by the priority and preference fields in DNS.

Consequently, it is an open question as to whether this specification would serve as an alternative mechanism to the one defined in [8]. It is worth pointing out that the connection reuse described here is, to a large degree, a proxy-to-proxy generalization of the SIP outbound specification.

13. Security Considerations

This specification provides an alternative way for proxies to
construct the URI that they use in Path and Record-Route header fields, compared to the algorithms in [3]. One of the central security issues there is construction of the flow token (the user part of the Path header field) such that another UA cannot steal calls targeted to another UA. The algorithm described here deals with this problem by encoding the authenticated identity of the client into the flow token. Consequently, as long as a client authenticates itself, it cannot cause the proxy to generate a flow token that would be identical to the one constructed for a different UA. This avoids the attack, but comes at the cost of requiring the proxy to perform and/or track the authentication procedures in a downstream proxy.

Another potential attack is to generate requests with Record-Route or Path header fields which are intended to ‘stick’ subsequent requests to connections that they would never normally go to based on RFC 3263 procedures. For example, a malicious proxy in bad.example.com might try to fool a proxy into thinking requests for good.example.com should go to it. It would do this by including a Record-Route in an INVITE which contains good.example.com. This attack is mitigated by MUST strength processing for servers, which says that they cannot ‘stick’ a URI to a connection unless there was already a URI stuck to the connection with the same domain. The first URI gets stuck to a connection either because an element learned the target of the connection through DNS (which was then verified through TLS), or because the incoming connection was received from an element that presented a domain certificate that created a URI that got stuck to the connection. In either case, this implies that the domain of all URI that get stuck to a connection are always equal to the identity presented in a certificate from the entity on the other side of that connection. This provides a robust mechanism that avoids the attack in question, even in cases where the DNS has been subverted.

Indeed, the ‘stickiness’ algorithm described here addresses some of the questions and issues raised in the domain certificates specification [9]. In particular, because proxies always "path" and "record-route" with URI whose domain parts match their TLS certificates, there are no complicates with record-routing. No additional certificates need to be obtained for individual elements. All that is required is the same thing that is required for web servers - a common certificate for all proxies in a cluster.

14. IANA Considerations

TODO.
15. Example

This section builds on the basic example used in the introduction, and fills in the blanks on how each URI is constructed and how each request is routed to the next hop.

Consider once again the deployment topology of Figure 2. Let us assume that UA X does not learn a SIP server via DHCP, and thus starts with nothing but its AOR and credentials. Let us further assume that the domain would like the UA to use two flows. Finally, the example assumes TLS on all links. The mechanism works in absence of TLS as well, but works better with (fewer messages). The flow considers only a single home proxy, home proxy 1.

```
    X     OB Proxy 2     OB Proxy 3    Home Proxy 1   Correspondent
(1) TLS Connect
(2) REGISTER
(3) 305
(4) TLS Connect
(5) REGISTER
(6) TLS Connect
(7) REGISTER
(8) 401
(9) 401
(10) REGISTER
(11) REGISTER
(12) 200 OK
(13) 200 OK
(14) TLS Connect
(15) REGISTER
(16) TLS Connect
```

Rosenberg                Expires April 16, 2007                [Page 22]
Figure 5

The basic call flow is shown in Figure 5.

UA X starts with only its AOR. It generates a REGISTER request with no Route header fields and a Request-URI of sip:example.com. This causes the client to perform a DNS query (not shown) and find the address of one of the home proxies (only one is shown). It opens a TLS connection towards that proxy (message 1), and then sends the REGISTER (message 2). The home proxy, seeing that the request came from neither outbound proxy, redirects it with a 305 (message 3). It includes a Contact header field in the redirect whose value is sip: outbound.example.com.

The client still has no outbound proxies configured per se. It looks up outbound.example.com in the DNS and finds three A records, one for each of the three outbound proxies. It selects the one for proxy 2, and then proceeds to open a TLS connection towards it (message 4). It then generates a REGISTER request whose Request-URI is sip: example.com but that contains a Route header field of sip: outbound.example.com (message 5). The server certificate provided by the proxy will cause the SIP URI sip:outbound.example.com to be associated with this TLS connection. The REGISTER arrives at proxy 2. Proxy 2 recognizes it as an initial REGISTER due to its lack of a user part in the Route header field. Consequently, it notes the instance ID and reg-ID but finds no username since the request doesn’t (yet) contain an Authorization header field. It selects a backup proxy (proxy 3). It constructs a Path URI and adds it to the REGISTER request. The Path URI has a user part that encodes the identity of proxy 3, and the instance ID and reg ID. It then opens a TLS connection to the home proxy (discovered by a DNS lookup of the Request-URI sip:example.com) (message 6), and sends the request there (message 7). Proxy 2 now associates the SIP URI sip:example.com to the connection to the example.com proxy, as a consequence of the certificate provided by the home proxy. In addition, proxy 2 utilizes mutual TLS authentication. When the home proxy receives the TLS connect in message 6, the home proxy notes the subjectAltName of the client certificate, which will be outbound.example.com, and associates the SIP URI sip:outbound.example.com with the TLS connection just created by proxy 2.

The home proxy will reject the request with a 401 in order to challenge for credentials (message 8). This is forwarded to the UA (message 9). Since the request was rejected, no URI get stuck to any
flows by any element. The UA applies its credentials. It generates a
new REGISTER message, whose Route header field is sip:example.com.
This bounces off of the home proxy and is redirected once more (not
shown). It recurses on the 305 and now generates a REGISTER (still
with credentials) towards sip:outbound.example.com. The UA notes
that it has a TLS connection with an associated URI with the same
value, so that existing TLS connection to proxy 2 is reused. The
REGISTER message is sent there (message 10).

Now, proxy 2 sees an initial REGISTER request once more. This time,
the request contains a username (x) in the Authorization header
field, along with an instance ID (urn:instanceID) in the Contact
header field and a reg-id (1) as well. It constructs a Path URI of the
form

```
sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun
```

and inserts it into the request. This URI encodes the address of the
two proxies being used for this user (proxies 2 and 3), along with
the instance ID and reg-ID. It associates the instance ID, username
and reg-ID with the TLS connection. Since it is forwarding the
request to example.com, it finds a matching TLS connection and sends
the request there (message 11). The home proxy processes the
registration as per sip-outbound. It then copies the Path URI into a
Service-Route header field and places that into the REGISTER response
(message 12). It adds itself to the Service-Route by adding the URI
sip:example.com as the second Service-Route.

Since the REGISTER is successful, the home proxy examines the topmost
Path from the REGISTER request. Since its domain,
outbound.example.com, matches the domain of the URI already
associated with the TLS connection, the home proxy adds that URI to
the list of URI bound to that connection. Consequently, its
connection table looks like:

```
TLS connection to proxy 2:
sip:outbound.example.com
sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun
```

The 200 OK is received at proxy 2. Following the procedures in
Section 4, it finds its own Path URI in the Service-Route, and looks
at the next URI, which is sip:example.com. This URI is already
associated with the TLS connection to the home proxy, so nothing more
is done. Proxy 2 adds an Alternative-Proxies header field to the
REGISTER response, with a value of sip:proxy2+proxy3+x+instanceID+1@
outbound.example.com;maddr=192.0.2.1. Proxy A then forwards the 200
OK to the UA (message 13).

UA X now does several things. Firstly, since the response contains a
Service-Route, it updates its outbound proxy to the URI in the Service-Route (sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun). This is now the first outbound proxy. Note how it has a keepalive=stun parameter inserted by the proxy, which validates that it supports the keepalive mechanism.

Next, the UA associates the connection towards A with this URI, so that its connection table looks like:

TLS connection to 2:
- sip:outbound.example.com
- sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun

Finally, it extracts the URI from the Alternative-Proxies header field, which points to proxy 3. It runs through an identical registration procedure with proxy 3. This procedure will leave the UA with two outbound proxies, both of which have exactly the same URI. Its connection table will look like:

TLS connection to 2:
- sip:outbound.example.com
- sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun

TCP connection to 3:
- sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;maddr=192.0.2.1
- sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun

The connection table at the home proxy will look like:

TLS connection to 2:
- sip:outbound.example.com
- sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun

TLS connection to 3:
- sip:outbound.example.com
- sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun

Next, the UA decides to make a call. It generates a SIP INVITE (message 24). This INVITE contains the full service route in the Route header field. The topmost Route is sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun. The Request-URI identifies the called party. The client matches that URI against the URI associated with TLS connections and finds a match for both the TLS connection to 2 and 3. Thus, either connection can be used. The client arbitrarily chooses the connection to proxy 2,
and sends the request.

This request is received at proxy 2, which record-routes. It does so by using a URI of the form

```
sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;
keepalive=stun;aor-qual=abc
```

The aor-qual URI parameter contains some state that is specific to the SIP dialog. Proxy 2 strips its URI from the topmost Route, finding sip:example.com as the next Route. It has a TLS connection to the home proxy, and so sends the request there (message 25). The home proxy itself record-routes (with any URI it chooses) and forwards the request to the terminating domain (message 27) after opening a TLS connection there (message 26). A 200 OK is received (message 28), which is forwarded to proxy 2 (message 29). This next step is important. Following the procedures of Section 5, the home proxy adds the topmost Record-Route value that was present in the request to the URI list associated with the TLS connection to proxy 2. Consequently, its connection table now looks like (lines folded for readability):

```
TLS connection to 2:
    sip:outbound.example.com
    sip:proxy2+proxy3+x+instanceID+1@outbound.example.com
        ;keepalive=stun
    sip:proxy2+proxy3+x+instanceID+1@outbound.example.com
        ;keepalive=stun;aor-qual=abc

TLS connection to 3:
    sip:outbound.example.com
    sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun
```

The 200 OK is received by proxy 2. It adds the home proxy’s record-route URI to the list of URI associated with its TLS connection to the home proxy (assuming the URI hostname is example.com). The response is then forwarded to the UA (message 30). The ACKs are then sent (messages 31 to 33).

Now something interesting happens. Proxy 2 fails. Its TLS connection to the home proxy and the UA are severed. Let us assume that neither the UA or the home proxy have yet detected this. The called party sends a BYE at this very moment to terminate the call (message 34), which arrives at the home proxy. The next Route header field will have the value

```
sip:proxy2+proxy3+x+instanceID+1@outbound.example.com;keepalive=stun
    ;aor-qual=abc
```

The most specific match is for the connection to proxy 2. The request is sent there (message 35). This will quickly timeout due to TCP connection failures. This causes the home proxy to try the next best TLS connection, to proxy 3. It sends the
request thre (message 36).

This BYE request arrives at proxy 3. There are no further Route headers. Thus, proxy 3 attempts to route the request using the request URI, which will be the private IP address of the UA and thus match no existing TLS connection. Before attempting this anyway, the proxy examines the Route header field in the request it received. It recognizes the structure, and extracts the username and instance ID. These do in fact match an existing TLS connection to UA X, and so the request is sent over that connection and delivered to the UA (message 37).

16. Acknowledgements

Thanks to Cullen Jennings for reviewing and commenting on earlier versions of this document. His ideas led to substantial improvements in the mechanism. Paul Kyzivat provided helpful feedback.

17. References

17.1. Normative References


17.2. Informative References


Author’s Address

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net
Intellectual Property Statement

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

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

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

Disclaimer of Validity

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

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

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