J. Rosenberg
dynamicsoft

February 15, 2004

Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)

draft-ietf-sip-gruu-01

Status of this Memo

Copyright Notice

Abstract

Several applications of the Session Initiation Protocol (SIP) require a user agent (UA) to construct and distribute a URI which can be used by anyone on the Internet to route a call to that specific UA instance. A URI which routes to a specific UA instance is called a Globally Routable UA URI (GRUU). This document describes an extension to SIP for obtaining a GRUU from a server, and for communicating a GRUU to a peer within a dialog.

Table of Contents

1. Introduction

Several applications of the Session Initiation Protocol (SIP) [1] require a user agent (UA) to construct and distribute a URI which can be used by anyone on the Internet to route a call to that specific UA instance. An example of such an application is call transfer, based on the REFER method [4]. Another application is the usage of endpoint-hosted conferences within the conferencing framework [10]. We call these URIs Globally Routable UA URIs (GRUU). This specification provides a mechanism for obtaining and using GRUUs.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [2] and indicate requirement levels for compliant implementations.

3. Defining a GRUU

A GRUU is a SIP URI which has a specific set of characteristics:
Global: It can be used by any UAC connected to the Internet. In that regard, it is like an address-of-record (AOR) for a user. The address-of-record for a user, sip:joe@example.com, is meant to be used by anyone to call that user. The same is true for a GRUU.
Temporally Scoped: It may be temporally scoped. In that regard, its not like an AOR for a user. The general assumption is that an AOR for a user is valid so long as the user resides within that domain (of course, policies can be imposed to limit its validity, but that is not the default case). However, a GRUU has a limited lifetime by default. It can never be valid for longer than the duration of the registration of the UA to which it is bound. For example, if my PC registers to the SIP network, a GRUU for my PC is only valid as long as my PC is registered. If the PC unregisters, the GRUU is invalid; calls to it would result in a 404. If the PC comes back, the GRUU will be valid once more. Furthermore, it will frequently be the case that the GRUU has a lifetime shorter than the duration of the registration.
Instance Routing: It routes to a specific UA instance, and never forks. In that regard, it is unlike an address-of-record. When a call is made to a normal AOR which represents a user, routing logic is applied in proxies to deliver the call to one or more UAs. That logic can result in a different routing decision based on the time-of-day, or the identity of the caller. However, when a call is made to a GRUU, the routing logic is much more static. It has to cause the call to be delivered to a very specific UA instance. That UA instance has to be the same UA instance for any request sent to that GRUU. This does not mean that a GRUU

represents a fundamentally different type of URI; it only means that the logic a proxy applies to a GRUU is going to generally be simpler than that it applies to a normal AOR.

4. Use Cases

We have encountered several use cases for a GRUU.

4.1 REFER

Consider a blind transfer application [14]. User A is talking to user B. A wants to transfer the call to user C. So, it sends a REFER to user C. That REFER looks like, in part:

```
REFER sip:C@example.com SIP/2.0
From: sip:A@example.com;tag=99asd
To: sip:C@example.com
Refer-To: (URI that identifiers B's UA)
```

The Refer-To header needs to contain a URI that can be used by C to place a call to B. However, this call needs to route to the specific UA instace which B is using to talk to A. If it didn't, the transfer service would not execute. This URI is provided to A by B. Because B doesn't know who A will transfer the call to, the URI has to be usable by anyone. Therefore, it is a GRUU.

4.2 Conferencing

A similar need arises in conferencing [10]. In that framework, a conference is described by a URI which identifies the focus of the conference. The focus is a SIP UA at the center of a conference. Each conference participant has a dialog with the focus. One case described in the framework is where a user A has made a call to B. They then put B on hold, and call C. Now, A has two separate dialogs for two separate calls - one to B, and one to C. A would like to conference them. One model is that A morphs itself into a focus. It sends a re-INVITE on each existing dialog, and provides both B and C with an updated URI that now holds the conference URI. It also has a callee capabilities [6] parameter which indicates that this URI is a conference URI. A proceeds to mix the media streams from B and C. This is called an ad-hoc conference.

At this point, normal conferencing features can be applied. That means that B can send another user, D, the conference URI, perhaps in an email. D can send an INVITE to that URI, and join the conference. For this to work, the conference URI used by A in its re-INVITE has to be usable by anyone, and it has to route to the specific UA instance of A that is acting as the focus. If it didn't, basic

conferencing features would fail. Therefore, it is a GRUU.

4.3 Presence

In a SIP-based presence [15] system, the presence agent (PA) generates notifications about the state of a user. This state is represented with the Presence Information Document Format (PIDF) [13]. In a PIDF document, a user is represented by a series of tuples, each of which identifies the devices that the user has and provides information about them. Each tuple also has a contact URI, which is a SIP URI representing that device. A watcher can make a call to that URI, with the expectation that the call is routed to the device whose presence is represented in the tuple.

The URI in the presence document therefore has to route to the specific UA instance whose presence was reported. Furthermore, since the presence document could be used by anyone who subscribes to the user, the URI has to be usable by anyone. As a result, it is a GRUU.

It is interesting to note that the GRUU may need to be constructed by a presence agent, depending on how the presence document is computed by the server.

5. Overview of Operation

This section is tutorial in nature, and does not specify any normative behavior.

This extension allows a UA to obtain a GRUU, and to use a GRUU. These two mechanisms are separate, in that a UA can obtain a GRUU in any way it likes, and use the mechanisms in this specification to use them. Similarly, a UA can obtain a GRUU but never use it.

A UA can obtain a GRUU by generating a normal REGISTER request, as specified in RFC 3261 [1]. This request contains a Supported header field with the value "gruu", indicating to the registrar that the UA supports this extension. The UA includes a "sip.instance" media feature tag in the Contact header field of each Contact for which a GRUU is desired. This media feature tag contains a globally unique ID that identifies the UA instance. If the domain that the user is registering against also supports GRUU, the REGISTER responses will contain the "gruu" parameter in each Contact header field. This parameter contains a GRUU which the domain guarantees will route to that UA instace. That GRUU is guaranteed to remain valid for the duration of the registration. The GRUU is bound to the UA instace. Should the client change its Contact URI, but indicate that it represents the same instance ID, the server would provide the same GRUU. Furthermore, if the registration for the Contact expires, and

the UA registers the Contact at a later time with the same instance identifier, the server would provide the same GRUU.

Since the GRUU is a URI like any other, it can be handed out by a UA by placing it in any header field which can contain a URI. A UA will normally place the GRUU into the Contact header field of dialog creating requests and responses it generates. However, it is important for the UA receiving the message to know whether the Contact URI is a GRUU or not. To make this determination, the UA looks for the presence of the Supported header field in the request or response. If it is present with a value of "gruu", it means that the Contact URI is a GRUU.

When a UA uses a GRUU, it has the option of adding the "grid" URI parameter to the GRUU. This parameter is opaque to the proxy server handling the domain. However, when the server maps the GRUU to the corresponding Contact URI, the server will copy the grid parameter into the Contact URI. As a result, when the UA receives the request, the Request URI will contain the grid parameter it placed in the corresponding GRUU.

6. User Agent Behavior

User agent behavior is divided into two separate parts - REGISTER processing, and GRUU usage.

6.1 REGISTER Processing

When a UA wishes to obtain a GRUU within the domain of its AOR, when it generates a REGISTER request (initial or refresh), it MUST include the Supported header field in the request. The value of that header field MUST include "gruu" as one of the option tags. This alerts the registrar for the domain that the UA supports the GRUU mechanism.

Furthermore, for each Contact for which the UA desires to obtain a GRUU, the UA MUST include a "sip.instance" media feature tag as a UA characteristic [6]. As described in [6], this media feature tag will be encoded in the Contact header field as the "+sip.instance" Contact header field parameter. The value of this parameter, as described in Section 13.3, MUST be a globally unique identifier, and SHOULD remain the same across all registrations generated from that particular UA instance.

Besides the presence of the "gruu" option tag in the Supported header field and the "+sip.instance" Contact header field parameter, the REGISTER request is constructed identically to the case where this extension was not understood. Specifically, the Contact URI in the REGISTER request SHOULD NOT contain the gruu Contact header field

parameter. Any such parameters are ignored by the registrar, as the UA cannot propose a GRUU for usage with the Contact URI.

If a UA wishes to guarantee that the request is not processed unless the domain supports and uses this extension, it MAY include a Require header field in the request with a value that contains the "gruu" option tag.

If the response is a 2xx, each Contact header that contained the "+sip.instance" Contact header field parameter may also contain a "gruu" parameter. This parameter contains a SIP URI that represents a GRUU corresponding to that UA instance. Any requests sent to the GRUU URI will be routed by the domain to the Contact URI bound currently bound to that instance ID. The GRUU will not change in subsequent 2xx responses to REGISTER. Indeed, even if the UA lets the contact expire, when it re-registers it at any later time, the registrar will normally provide the same GRUU for the same address-of-record and the UA instance ID. However, this property cannot be guaranteed, and a UA MUST be prepared to receive a different GRUU in a subsequent registration.

6.2 Using the GRUU

A UA first obtains a GRUU using the procedures of Section 6.1, or by other means outside the scope of this specification.

A UA can use the GRUU in the same way it would use any other SIP URI. However, a UA compliant to this specification MUST use a GRUU when populating the Contact header field of dialog-creating requests and responses. This includes the INVITE request and its 2xx response, the SUBSCRIBE [3] request, its 2xx response, and the NOTIFY request, and the REFER [4] request and its 2xx response. Similarly, in those requests and responses where the GRUU is used in the Contact header field, the UA MUST include a Supported header field that contains the option tag "gruu". However, it is not necessary for a UA to know whether or not its peer in the dialog uses a GRUU before inserting one into the Contact header field.

When placing a GRUU into the Contact header field of a request or response, a UA MAY add the "grid" URI parameter to the GRUU. This parameter MAY take on any value permitted by the grammar for the parameter. Note that there are no limitations on the size of this parameter. When a UA sends a request to the GRUU, the proxy for the domain that owns the GRUU will translate the GRUU in the Request-URI, replacing it with the corresponding Contact URI. However, it will retain the "grid" parameter when this translation is performed. As a result, when the UA receives the request, the Request-URI will contain the "grid" created by the UA. This allows the UA to

effectively manufacture an infinite supply of GRUU, each of which differs by the value of the "grid" parameter. When a UA receives a request that was sent to the GRUU, it will be able to tell which GRUU was invoked by the "grid" parameter.

An implication of this behavior is that all mid-dialog requests will be routed through intermediate proxies. There will never be direct, UA to UA signaling. It is anticipated that this limitation will be addressed in future specifications.

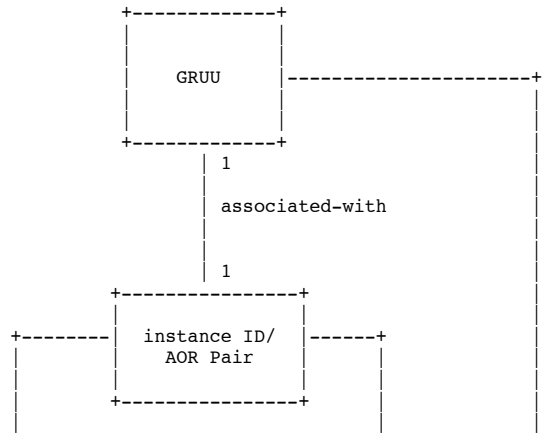
Once a UA knows that the Contact URI provided by its peer is a GRUU, it can use it in any application or SIP extension which requires a globally routable URI to operate. One such example is assisted call transfer.

7. Registrar Behavior

A registrar compliant to this specification is responsible for the creation and maintenance of GRUUs, and for providing those GRUU's to a UA in response to a REGISTER request.

7.1 Creation and Maintenance of GRUUs

A domain is responsible for creation and maintenance of a GRUU, along with its association to instance IDs, AORs and Contact URIs. These associations are modeled in the UML diagram in Figure 2.



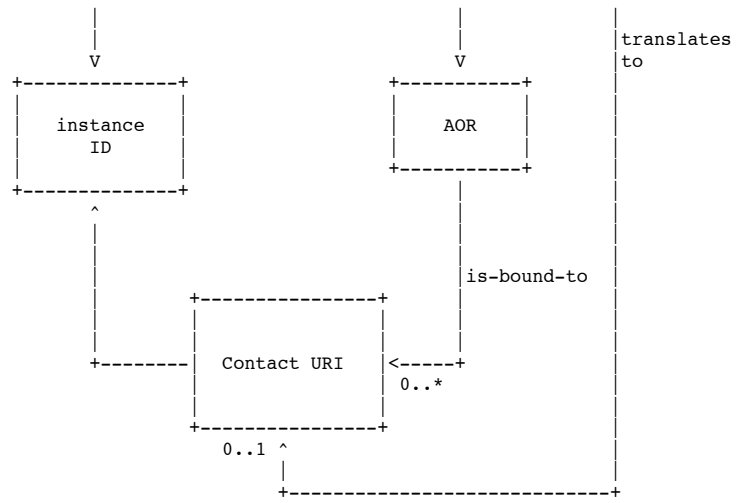


Figure 2

The combination of a UA instance ID and an AOR is referred to as an instance ID/AOR pair. There is a one-to-one mapping between such a pair and a GRUU; the GRUU is said to be associated with the pair, and the pair is associated with the GRUU. As a result, if two instance ID/AOR pairs are different, they each must be associated with a different GRUU. If two GRUUs are different, they each must be associated with a different instance ID/AOR pair. It is important to understand that this uniqueness is over the instance ID/AOR pair, not just the instance ID. For example, if a user registered the Contact sip:ua@pc.example.com;+sip.instance="1", representing a device with instance ID 1, to the AOR sip:user@example.com, and also registered the same Contact, representing the same instance ID - sip:ua@pc.example.com;+sip.instance="1" to a second AOR, say sip:boss@example.com, each of those UA instances would have a different GRUU, since they belong to different AORs.

A GRUU translates to zero or one Contact URIs. If the instance ID associated with the GRUU is the instance ID of a Contact URI currently bound to the AOR associated with that GRUU, then the GRUU translates to that Contact URI. If, however, the instance ID associated with the GRUU is not an instance ID of a Contact URI currently bound to the AOR associated with the GRUU (possibly because

there are no Contact URIs bound to the AOR), the GRUU maps to no Contact URI, and the GRUU is said to be invalid.

A registrar MAY create a GRUU for a particular instance ID/AOR pair at any time. Of course, if a UA requests a GRUU in a registration, and the registrar has not yet created one, it will need to do so in order to respond to the registration request. However, the registrar can create the GRUU in advance of any request from a UA.

This specification does not mandate a particular mechanism for construction of the GRUU. However, the GRUU MUST exhibit the following properties:

- The domain part of the URI is an IP address present on the public Internet, or, if it is a host name, exists in the global DNS and corresponds to an IP address present on the public Internet.
- When a request is sent to this URI, it routes to a proxy server in the same domain as that of the registrar.
- A proxy server in the domain can determine that the URI is a GRUU.
- When a proxy server in this domain receives a request sent to a URI that is a GRUU, that URI MUST be translated to the Contact URI currently bound to the AOR associated with that GRUU whose instance ID is the one associated with the GRUU.

In many cases, it will be desirable to construct the GRUU in such a way that it will not be possible, based on inspection of the URI, to determine the Contact URI that the GRUU translates to. It may also be desirable to construct it so that it will not be possible to determine the instance ID/AOR pair associated with the GRUU. Whether or not a GRUU should be constructed with this property is a local policy decision.

With these rules, it is possible, though not required, to construct a GRUU without requiring the maintenance of any additional state. To do that, the URI would be constructed in the following fashion:

```
user-part = "GRUU" + BASE64(E(K, (salt + instance ID + AOR)))
```

Where E(K,X) represents a suitable encryption function (such as AES with 128 bit keys) with key K applied to data block X, and the "+" operator implies concatenation. Salt represents a random string that prevents a client from obtaining pairs of known plaintext and ciphertext. A good choice would be at least 128 bits of randomness in the salt.

The benefit of this mechanism is that a server need not store additional information on mapping a GRUU to its corresponding Contact URI. The user part of the GRUU contains the instance ID and AOR. Assuming that the domain stores registrations in a database indexed by the AOR, the proxy processing the GRUU would look up the AOR,

extract the currently registered Contacts, and find the one matching the instance ID encoded in the request URI. The Contact URI whose instance ID is that instance ID is then used as the translated version of the URI. Encryption is needed to prevent attacks whereby the server is sent requests with faked GRUU, causing the server to direct requests to any named URI. Even with encryption, the proxy should validate the user part after decryption. In particular, the AOR should be one managed by the proxy in that domain. Should a UA send a request with a fake GRUU, the proxy would decrypt and then discard it because there would be no URI or an invalid URI inside.

Once an association from an instance ID/AOR to a GRUU is created, that mapping MUST remain in existence, and valid, as long as there exists any Contact bound to that AOR whose instance ID is that instance ID. If, through a de-registration or expiration, there is no longer any Contact bound to that AOR whose instance ID is that instance ID, the registrar MUST remove the mapping, and invalidate the GRUU. However, at any time in the future, should a UA register a Contact to that same AOR indicating that it represents that same instance ID, the registrar SHOULD provide the UA the same GRUU provided previously. Indeed, this requirement would ideally be a MUST if it was achieveable, but even with the stateless algorithm described above, key rotation or server failures may cause the GRUU associated with an instance ID/AOR pair to change. The value of associatig the GRUU with an instance ID/AOR pair, as opposed to a Contact URI/AOR pair, is that the association can transcend registrations. As a result, registrars SHOULD make every effort possible to maintain the association for as long as possible.

7.2 Providing GRUUs to User Agents

When a registrar compliant to this specification receives a REGISTER request, it checks for the presence of the Require header field in the request. If present, and if it contains the "gruu" option tag, the registrar MUST follow the procedures in the next paragraph for inclusion of the "gruu" parameter in a 2xx response to REGISTER. If not present, but a Supported header field was present with the "gruu" option tag, the registrar SHOULD follow the procedures in the next paragraph for inclusion of the "gruu" parameter in a 2xx response to REGISTER. If the Supported header field was not present, or it if was present but did not contain the value "gruu", the registrar SHOULD NOT follow the procedures of the next paragraph for inclusion of the "gruu" parameter in a 2xx response to REGISTER.

If the register request contained any "gruu" Contact header field parameters, these MUST be ignored by the registrar. A UA cannot suggest or otherwise provide a GRUU to the registrar.

A GRUU is provided to a UA by including it in the "gruu" Contact header field parameter for each Contact URI that contains a "+sip.instance" Contact header field parameter. The value of the gruu parameter is a quoted string containing the URI that is the GRUU for the associated instance ID/AOR pair. If the server does not currently have a GRUU associated with the instance ID/AOR, one is created according to the procedures of Section 7.1. Otherwise, if a GRUU already exists for that instance ID/AOR pair, the GRUU associated with that pair MUST be placed into the "gruu" Contact header field parameter of the REGISTER response.

Inclusion of a GRUU in the "gruu" Contact header field parameter of a REGISTER response is separate from the computation and storage of the GRUU. It is possible that the registrar has computed a GRUU for one UA, but a different UA that queries for the current set of registrations doesn't understand GRUU. In that case, the REGISTER response sent to that second UA would not contain the "gruu" Contact header field parameter, even though the UA has a GRUU for that Contact.

8. Proxy Behavior

When a proxy server receives a request, and the proxy owns the domain in the Request URI, and the proxy is supposed to access a Location Service in order to compute request targets (as specified in Section 16.5 of RFC 3261 [1]), the proxy MUST check if the Request URI is a GRUU created by that domain.

If the URI is a GRUU, the proxy MUST determine if there is still a Contact URI bound to AOR associated with the GRUU, whose instance ID is the instance ID associated with the GRUU. If that AOR no longer has any contacts bound to it, or if it does have contacts bound to it, but none of them have an instance ID equal to the instance ID associated with the GRUU, the proxy MUST generate a 404 (Not Found) response to the request.

Otherwise, the proxy MUST populate the target set with a single URI. This URI MUST be equal to the Contact URI that is translated from the GRUU. Furthermore, if the GRUU contained a "grid" URI parameter, the URI in the target set MUST also contain the same parameter with the same value.

A proxy MAY apply other processing to the request, such as execution of called party features. In particular, it is RECOMMENDED that non-routing called party features, such as call logging and screening, that are associated with the AOR are also applied to requests for all GRUUs associated with that AOR.

In many cases, a proxy will record-route an initial INVITE request, and the user agents will insert a GRUU into the Contact header field. When this happens, a mid-dialog request will arrive at the proxy with a Route header field that was inserted by the proxy, and a Request-URI that represents a GRUU. Proxies follow normal processing in this case; they will strip the Route header field, and then process the Request URI as described above.

The procedures of RFC 3261 are then followed to proxy the request. The request SHOULD NOT be redirected in this case. In many instances, a GRUU is used by a UA in order to assist in the traversal of NATs, and a redirection may prevent such a case from working.

9. Grammar

This specification defines two new Contact header field parameters, gruu and +sip.instance, and a new URI parameter, grid. The grammar for string-value is obtained from [6].

```
contact-params    =  c-p-q / c-p-expires / c-p-gruu / cp-instance
                      / contact-extension
c-p-gruu          =  "gruu" EQUAL DQUOTE SIP-URI DQUOTE
cp-instance       =  "+sip.instance" EQUAL LDQUOT string-value RDQUOT
uri-parameter     =  transport-param / user-param / method-param
                     / ttl-param / maddr-param / lr-param / grid-param
                     / other-param
grid-param        = "grid=" pvalue
```

10. Requirements

This specification was created in order to meet the following requirements:

REQ 1: When a UA invokes a GRUU, it MUST cause the request to be routed to the specific UA instance to which the GRUU refers.

REQ 2: It MUST be possible for a GRUU to be invoked from anywhere on the Internet, and still cause the request to be routed appropriately. That is, a GRUU MUST NOT be restricted to use within a specific addressing realm.

REQ 3: It MUST be possible for a GRUU to be constructed without requiring the network to store additional state.

REQ 4: It MUST be possible for a UA to obtain a multiplicity of GRUUs, each one of which routes to that UA instance. This is needed to support ad-hoc conferencing, for example, where a a UA instance needs a different URI for each conference it is hosting.

REQ 5: When a UA receives a request sent to a GRUU, it MUST be possible for the UA to know the GRUU which was used to invoke the request. This is necessary as a consequence of requirement 4.

REQ 6: It MUST be possible for a UA to add opaque content to a GRUU, which is not interpreted or altered by the network, and used only by the UA instance to whom the GRUU refers. This provides a basic cookie type of functionality, allowing a UA to build a GRUU with state embedded within it.

REQ 7: It MUST be possible for a proxy to execute services and features on behalf of a UA instace represented by a GRUU. As an example, if a user has call blocking features, a proxy may want to apply those call blocking features to calls made to the GRUU in addition to calls made to the user's AOR.

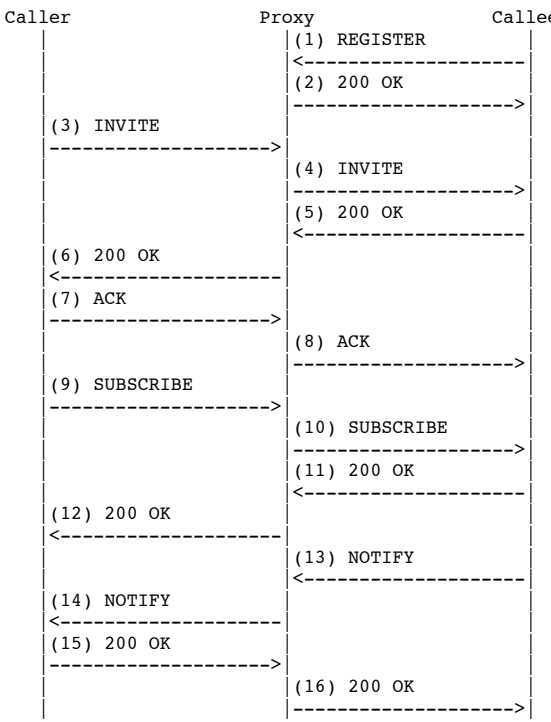
REQ 8: It MUST be possible for a UA in a dialog to inform its peer of its GRUU, and for the peer to know that the URI represents a GRUU. This is needed for the conferencing and dialog reuse applications of GRUUs, where the URIs are transferred within a dialog.

REQ 9: When transferring a GRUU per requirement 8, it MUST be possible for the UA receiving the GRUU to be assured of its integrity and authenticity.

REQ 10: It MUST be possible for a server, authoritative for a domain, to construct a GRUU which routes to a UA instace bound to an AOR in that domain. In other words, the proxy can construct a GRUU too. This is needed for the presence application.

11. Examples

The following call flow shows a basic registration and call setup, followed by a subscription directed to the GRUU.



The Callee supports the GRUU extension. As such, its REGISTER (1) looks like:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP client.example.com;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Callee <sip:callee@example.com>;tag=a73kszlfl
Supported: gruu
To: Callee <sip:callee@example.com>
Call-ID: 1j9FpLxk3uxtm8tn@client.example.com
CSeq: 1 REGISTER
Contact: <sip:callee@client.example.com>;+sip.instance="<hffua-ssdfff877>"
Content-Length: 0
```


The REGISTER response would look like:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP client.example.com;branch=z9hG4bKnashds7
From: Callee <sip:callee@example.com>;tag=a73kszlfl
To: Callee <sip:callee@example.com> ;tag=b88sn
Call-ID: 1j9FpLxk3uxtm8tn@client.example.com
CSeq: 1 REGISTER
Contact: <sip:callee@client.example.com>
  ;gruu="sip:hha9s8d=-999a@example.com"
  ;+sip.instance="<hffua-ssdfff877>"
Content-Length: 0
```

Note how the Contact header field in the REGISTER response contains the gruu parameter with the URI sip:hha9s8d=-999a@example.com. This represents a GRUU that translates to the Contact URI sip:callee@client.example.com.

The INVITE from the caller is a normal SIP INVITE. The 200 OK generated by the callee, however, now contains a GRUU in the Contact header field. The UA has also chosen to include a grid URI parameter into the GRUU.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bKnaa8
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK99a
From: Caller <sip:caller@example.com>;tag=n88ah
To: Callee <sip:callee@example.com> ;tag=a0z8
Call-ID: 1j9FpLxk3uxtma7@host.example.com
CSeq: 1 INVITE
Supported: gruu
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK
Contact: <sip:hha9s8d=-999a@example.com;grid=99a>
Content-Length: --
Content-Type: application/sdp

[SDP Not shown]
```

At some point later in the call, the caller decides to subscribe to the dialog event package [11] at that specific UA. To do that, it generates a SUBSCRIBE request (message 9), but directs it towards the GRUU contained in the Contact header field.

```
SUBSCRIBE sip:hha9s8d=-999a@example.com;grid=99a SIP/2.0
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:caller@example.com>;tag=kkaz-
To: Callee <sip:callee@example.com>
Call-ID: faif9a@host.example.com
CSeq: 2 SUBSCRIBE
Supported: gruu
Event: dialog
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK
Contact: <sip:bad998asd8asd0000a0@example.com>
Content-Length: 0
```

In this example, the caller itself supports the GRUU extension, and is using its own GRUU to populate the Contact header field of the SUBSCRIBE.

This request is routed to the proxy, which proceeds to perform a location lookup on the request URI. It is translated into the Contact URI of that GRUU, and then proxied there (message 10). Note how the grid parameter is maintained.

```
SUBSCRIBE sip:callee@client.example.com;grid=99a SIP/2.0
Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bK9555
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:caller@example.com>;tag=kkaz-
To: Callee <sip:callee@example.com>
Call-ID: faif9a@host.example.com
CSeq: 2 SUBSCRIBE
Supported: gruu
Event: dialog
Allow: INVITE, OPTIONS, CANCEL, BYE, ACK
Contact: <sip:bad998asd8asd0000a0@example.com>
Content-Length: 0
```

12. Security Considerations

Since GRUUs do not reveal information about the identity of the associated address-of-record or Contact URI, they provide routability without identity. However, GRUUs do not provide a complete or reliable solution for privacy. In particular, since the GRUU does not change during the lifetime of a registration, an attacker could correlate two calls as coming from the same source, which in and of itself reveals information about the caller. Furthermore, GRUUs do not address other aspects of privacy, such as the addresses used for media transport. For a discussion of how privacy services are provided in SIP, see RFC 3323 [9].

It is important for a UA to be assured of the integrity of a GRUU when it is given one in a REGISTER response. If the GRUU is tampered with by an attacker, the result could be denial of service to the UA. As a result, it is RECOMMENDED that a UA use the SIPS URI scheme when registering.

13. IANA Considerations

This specification defines a new Contact header field parameter, URI parameter and media feature tag.

13.1 Header Field Parameter

This specification defines a new header field parameter, as per the registry created by [7]. The required information is as follows:
Header field in which the parameter can appear: Contact
Name of the Parameter gruu
RFC Reference RFC XXXX [[NOTE TO IANA: Please replace XXXX with the RFC number of this specification.]]

13.2 URI Parameter

This specification defines a new SIP URI parameter, as per the registry created by [8].
Name of the Parameter grid
RFC Reference RFC XXXX [[NOTE TO IANA: Please replace XXXX with the RFC number of this specification.]]

13.3 Media Feature Tag

This section registers a new media feature tag, per the procedures defined in RFC 2506 [5]. The tag is placed into the sip tree, which is defined in [6].
Media feature tag name: sip.instance
ASN.1 Identifier: New assignment by IANA.
Summary of the media feature indicated by this tag: This feature tag contains a string that indicates a unique identifier associated with the UA instance registering the Contact. This identifier is globally unique, and remains bound to the UA instance for as long as is achievable. For UA instances that are implemented as hardware, such as an IP phone, the instance ID would ideally be burned into firmware when the device is manufactured. For software, the instance ID would generally be randomly created at installation time.
Values appropriate for use with this feature tag: String.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.
Examples of typical use: Routing a call to a specific device.
Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]
Security Considerations: This media feature tag can be used in ways which affect application behaviors. For example, the SIP caller preferences extension [12] allows for call routing decisions to be based on the values of these parameters. Therefore, if an attacker can modify the values of this tag, they may be able to affect the behavior of applications. As a result of this, applications which utilize this media feature tag SHOULD provide a means for ensuring its integrity. Similarly, this feature tag should only be trusted as valid when it comes from the user or user agent described by the tag. As a result, protocols for conveying this feature tag SHOULD provide a mechanism for guaranteeing authenticity.

14. Acknowledgements

The author would like to thank Rohan Mahy, Paul Kyzivat, Alan Johnston, and Cullen Jennings for their contributions to this work.

Normative References

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

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

[3] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[4] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

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

[6] Rosenberg, J., "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", draft-ietf-sip-callee-caps-03 (work in progress), January 2004.

[7] Camarillo, G., "The Internet Assigned Number Authority Header Field Parameter Registry for the Session Initiation Protocol", draft-ietf-sip-parameter-registry-01 (work in progress), November 2003.

[8] Camarillo, G., "The Internet Assigned Number Authority Universal Resource Identifier Parameter Registry for the Session Initiation Protocol", draft-ietf-sip-uri-parameter-reg-01 (work in progress), November 2003.

Informative References

[9] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[10] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-01 (work in progress), October 2003.

[11] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-03 (work in progress), October 2003.

[12] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", draft-ietf-sip-callerprefs-10 (work in progress), October 2003.

[13] Sugano, H. and S. Fujimoto, "Presence Information Data Format (PIDF)", draft-ietf-impp-cpim-pidf-08 (work in progress), May 2003.

[14] Sparks, R. and A. Johnston, "Session Initiation Protocol Call Control - Transfer", draft-ietf-sipping-cc-transfer-01 (work in progress), February 2003.

[15] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", draft-ietf-simple-presence-10 (work in progress), January 2003.

Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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

Internet Draft
Document: draft-ietf-sip-history-info-02.txt
Category: Standards Track

M. Barnes
Editor
Nortel Networks

Expires: August, 2004
February, 2004

An Extension to the Session Initiation Protocol for Request History Information

Status of this Memo

Copyright Notice

Abstract

This draft defines a standard mechanism for capturing the history information associated with a SIP request. This capability enables many enhanced services by providing the information as to how and why a call arrives at a specific application or user. This draft defines a new optional SIP header, History-Info, for capturing the history information in requests. A new option tag, Histinfo, to be included in the Supported header, is defined to allow UAs to indicate whether the History-Info should be returned in responses to a request which has captured the history information.

Table of Contents

Overview

Many services that SIP is anticipated to support require the ability to determine why and how the call arrived at a specific application. Examples of such services include (but are not limited to) sessions initiated to call centers via "click to talk" SIP URLs on a web page, "call history/logging" style services within intelligent "call management" software for SIP UAs and calls to voicemail servers and call centers. While SIP implicitly provides the redirect/retarget capabilities that enable calls to be routed to chosen applications, there is currently no standard mechanism within SIP for communicating the history of such a request. This "request history" information allows the receiving application to determine hints about how and why the call arrived at the application/user. This draft defines a new SIP header, History-Info, to provide a standard mechanism for capturing the request history information to enable a wide variety of services for networks and end users. The History-Info header provides a building block for development of new services.

Section 1 provides additional background motivation for the Request History capability. Section 2 identifies the requirements for a solution, with Section 3 providing an overall description of the solution.

Section 4 provides the details of the additions to the SIP protocol. An example use of the new header is included in Section 4.5, with

additional scenarios included in the Appendix. It is anticipated that these would be moved and progressed in a general Service examples draft such as [SIPSVCEX] or individual informational drafts describing these specific services, since the History-Info header is just one of the building blocks for implementing these services. Individual drafts would be particularly useful for documenting services for which there are multiple solutions, as it is not the intent, nor is it within the scope, of this draft to prescribe a complete solution for any of these applications.

Section 5 summarizes the application considerations identified in the previous sections. Section 6 summarizes the security solution as described in section 4.4.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In order to provide a cross reference of the solution description to the requirements without reiterating the entirety of the requirements inline, the requirements are referenced as [REQNAME-req] following the text or paragraph which explicitly satisfies the requirement.

1.Background: Why define a Generic "Request History" capability?

SIP implicitly provides redirect/retarget capabilities that enable calls to be routed to specific applications as defined in [RFC3261]. The term retarget will be used henceforth in this draft to refer to the process of a Proxy Server/UAC changing a URI in a request and thus changing the target of the request. This term is chosen to avoid associating this request history only with the specific SIP Redirect Server capability that provides for a response to be sent back to a UAC requesting that the UAC should retarget the original request to an alternate URI. The rules for determining request targets as described in section 16.5 of [RFC3261] are consistent with the use of the retarget term in this draft.

The motivation for the request history is that in the process of retargeting old routing information can be forever lost. This lost information may be important history that allows elements to which the call is retargeted to process the call in a locally defined, application specific manner. The proposal in this draft is to provide a mechanism for transporting the request history. It is not proposing any application specific behavior for a Proxy or UA upon receipt of the information. Indeed, such behavior should be a local decision for the recipient application.

Current network applications provide the ability for elements involved with the call to exchange additional information relating to how and why the call was routed to a particular destination. The following are examples of such applications:

1. Web "referral" applications, whereby an application residing within a web server determines that a visitor to a website has arrived at the site via an "associate" site which will receive some "referral" commission for generating this traffic,

2. Email forwarding whereby the forwarded-to user obtains a "history" of who sent the email to whom and at what time

3. Traditional telephony services such as Voicemail, call-center "automatic call distribution", and "follow-me" style services.

Several of the aforementioned applications currently define application specific mechanisms through which it is possible to obtain the necessary history information.

In addition, request history information could be used to enhance basic SIP functionality by providing the following:

4. Some diagnostic information for debugging SIP requests.

5. A stronger security solution for SIP. A side effect is that each proxy which captures the "request history" information in a secure manner provides an additional means (without requiring signed keys) for the original requestor to be assured that the request was properly retargeted.

2. "Request History" Requirements

The following list constitutes a set of requirements for a "Request History" capability.

1) CAPABILITY-req: The "Request History" capability provides a capability to inform proxies and UAs involved in processing a request about the history/progress of that request. While this is inherently provided when the retarget is in response to a SIP redirect, it is deemed useful for non-redirect retargeting scenarios, as well.

2) OPTIONALITY-req: The "Request History" information is optional.

2.1) In many cases, it is anticipated that whether the history is added to the Request would be a local policy decision enforced by the specific application, thus no specific protocol element is needed.

2.2) Due to the capability being "optional" from the SIP protocol perspective, the impact to an application of not having the "Request History" must be described. Applicability guidelines to be addressed by applications using this capability must be provided as part of the solution to these requirements.

3) GENERATION-req: "Request History" information is generated when the request is retargeted.

3.1) In some scenarios, it might be possible for more than one instance of retargeting to occur within the same Proxy. A proxy should also generate Request History information for the 'internal retargeting'.

3.2) An entity (UA or proxy) retargeting in response to a redirect or REFER should include any Request History information from the redirect/REFER in the new request.

4) ISSUER-req: "Request History" information can be generated by a UA or proxy. It can be passed in both requests and responses.

5) CONTENT-req: The "Request History" information for each occurrence of retargeting, shall include the following:

5.1) The new URI or address to which the request is in the process of being retargeted,

5.2) The URI or address from which the request was retargeted,

5.3) The reason for the Request-URI or address modification,

5.4) Chronological ordering of the Request History information.

6) REQUEST-VALIDITY-req: Request-History is applicable to requests not sent within an established dialog. (i.e. INVITE, REGISTER, MESSAGE, and OPTIONS).

7) BACKWARDS-req: Request-History information may be passed from the generating entity backwards towards the UAC. This is needed to enable services that inform the calling party about the dialog establishment attempts.

8) FORWARDS-req: Request-History information may also be included by the generating entity in the request, if it is forwarded onwards.

2.1 Security Requirements

The Request History information is being inserted by a network element retargeting a Request, resulting in a slightly different problem than the basic SIP header problem, thus requiring specific consideration. It is recognized that these security requirements can be generalized to a basic requirement of being able to secure information that is inserted by proxies.

The potential security problems include the following:
1) A rogue application could insert a bogus Request History entry either by adding an additional entry as a result of retargeting or entering invalid information.

2) A rogue application could re-arrange the Request History information to change the nature of the end application or to mislead the receiver of the information.

Thus, a security solution for "Request History" must meet the following requirements:

1) SEC-req-1: The entity receiving the Request History must be able to determine whether any of the previously added Request History content has been altered.

2) SEC-req-2: The ordering of the Request History information must be preserved at each instance of retargeting.

3) SEC-req-3: The entity receiving the information conveyed by the Request History must be able to authenticate the source of the information.

4) SEC-req-4: To ensure the confidentiality of the Request History information, only entities which process the request should have visibility to the information.

It should be noted that these security requirements apply to any entity making use of the Request History information, either by retargeting and capturing the information, or as an application making use of the information received in either a Request or Response.

2.2 Privacy Requirements

Since the Request URI that is captured could inadvertently reveal information about the originator, there are general privacy requirements that MUST be met:

1) PRIV-req-1: The entity retargeting the Request must ensure that it maintains the network-provided privacy (as described in [4]) associated with the Request as it is retargeted.

2) PRIV-req-2: The entity receiving the Request History must maintain the privacy associated with the information.

In addition, local policy at a proxy may identify privacy requirements associated with the Request URI being captured in the Request History information.

3) PRIV-req-3: Request History information subject to privacy requirements shall not be included in outgoing messages unless it is protected as described in [RFC3323].

3. Request History Information Description

The fundamental functionality provided by the request history information is the ability to inform proxies and UAs involved in processing a request about the history or progress of that request [CAPABILITY-req]. The solution is to capture the Request-URIs as a request is forwarded in a new header for SIP messages: History-Info [CONTENT-req]. This allows for the capturing of the history of a request that would be lost with the normal SIP processing involved in the subsequent forwarding of the request. This solution proposes no changes in the fundamental determination of request targets or in the request forwarding as defined in sections 16.5 and 16.6 of the SIP protocol specification [RFC3261].

The History-Info header can appear in any request not associated with an established dialog, which includes INVITE, REGISTER, MESSAGE, REFER and OPTIONS [REQUEST-VALIDITY-req] and any valid response to these requests.[ISSUER-req]

The History-Info header is added to a Request when a new request is created by a UAC or Proxy, or when the target of a request is changed. The term 'retarget' is introduced to refer to this changing of the target of a request and the subsequent forwarding of that request. It should be noted that retargeting only occurs when the Request-URI indicates a domain for which the processing entity is responsible. In terms of the SIP protocol, the processing associated with retargeting is described in sections 16.5, and 16.6 of [RFC3261]. As described in section 16.5 of [RFC3261], it is possible

for the target of a request to be changed by the same proxy multiple times (referred to as 'internal retargeting' in section 2), as the proxy MAY add targets to the target set after beginning Request Forwarding. Section 16.6 of [RFC3261] describes Request Forwarding. It is during this process of Request Forwarding, that the History Information is captured as an optional, additional header field. Thus, the addition of the History-Info header does not impact fundamental SIP Request Forwarding. An entity (UA or proxy) changing the target of a request in response to a redirect or REFER SHOULD also propagate any History-Info header from the initial Request in the new request [GENERATION-req, FORWARDS-req].

3.1 Optionality of History-Info

The History-Info header is optional in that neither UAs nor Proxies are required to support it. A new Supported header, Histinfo, is included in the Request to indicate whether the History-Info header is returned in Responses [BACKWARDS-req]. In addition to the Histinfo Supported header, local policy determines whether or not the header is added to any request, or for a specific Request-URI, being retargeted. It is possible that this could restrict the applicability of services which make use of the Request History Information to be limited to retargeting within domain(s) controlled by the same local policy, or between domain(s) which negotiate policies with other domains to ensure support of the given policy, or services for which "complete" History Information isn't required to provide the service. [OPTIONALITY-req] All applications making use of the History-info header MUST clearly define the impact of the information not being available and specify the processing of such a request.

3.2 Securing History-Info

This draft defines a new header for SIP. The draft does RECOMMEND the use of a secure transport mechanism such as TLS to ensure the overall confidentiality of the History-Info headers[SEC-req-4]. However, the problem is slightly different than the hop by hop security problem solved by TLS, as each hop is not required to add the History-Info header. Since the History-Info header is being inserted by an entity as it targets and forwards a Request, the resulting security requirements also introduce a slightly different problem than the basic SIP header or Identity [SIPATHID] problems, which are focused on securing the information in the initial request end to end. However, the requirements for the security solution are similar to the Via and Record-Route headers. For the History-Info header, the general requirement is to secure a header that is inserted by an intermediary and then subsequently referenced, by other intermediaries to build the next header entry, or by an end application using the information to provide a service. Thus, the general requirement takes the form of a middle to middle and middle

to end security solution, which is addressed in a separate document [SIPIISEC]. The use of the middle-to-end security solution discussed in [SIPIISEC] allows the integrity of the History-Info to be ascertained as it traverses the intermediaries. Thus, including the History-Info header in SIP Requests and securing in this manner adds an additional level of security end to end, assuring the initiator of a Request that it has indeed reached the intended recipient. Further discussion of the security mechanism for History-Info is provided in section 2.4.

3.3 Ensuring the Privacy of History-Info

Since the History-Info header can inadvertently reveal information about the requestor as described in [RFC3323], the Privacy header SHOULD be used to determine whether an intermediary can include the History-Info header in a Request that it receives and forwards [PRIV-req-2] or that it retargets [PRIV-req-1]. Thus, the History-Info header SHOULD not be included in Requests where the requestor has indicated a priv-value of Session or Header level privacy.

In addition, the History-Info header can reveal general routing information, which may be viewed by a specific intermediary or network, to be subject to privacy restrictions. Thus, local policy MAY also be used to determine whether to include the History-Info header at all, whether to capture a specific Request-URI in the header, or whether it be included only in the Request as it is retargeted within a specific domain. [PRIV-req-3]
[Issue-1: It has been proposed on the mailing list that there is a protocol requirement to support this functionality. It has been suggested that adding an additional field to the History-Info header (or extending the priv-values defined in RFC 3323) would facilitate the implementation of this functionality.]

It is recognized that satisfying the privacy requirements can impact the functionality of this solution by overriding the request to generate the information. As with the optionality and security requirements, applications making use of History-Info SHOULD address any impact this may have.

4 Request History Information Protocol Details

This section contains the details and usage of the proposed new SIP protocol elements. It also discusses the security aspects of the solution and provides some examples.

4.1 Protocol Structure of History-Info

History-Info is a header field as defined by [RFC3261]. It can appear in any request or response not associated with a dialog or which starts a dialog. For example, History-Info can appear in INVITE, REGISTER, MESSAGE, REFER and OPTIONS and any valid responses, plus NOTIFY requests which initiate a dialog .

The History-Info header carries the following information:

- Targeted-to-URI: the Request URI captured as the Request is forwarded.
- Index: A mandatory parameter for History-Info reflecting the chronological order of the information, indexed to also reflect the forking and nesting of requests. The format for this parameter is a string of digits, separated by dots to indicate the number of forward hops and retargets. This results in a tree representation of the history of the request, with the lowest level index reflecting a branch of the tree. By including the index and securing the header, the ordering of the History-info headers in the request is assured.[SEC-req-2]
- Reason: An optional parameter for History-info. The reason for the retargeting is captured by including the Reason Header [RFC3326] associated with the Request URI being retargeted. Thus, a reason is not included for a Request URI when it is first added in a History-info header, but rather is added when that particular Request-URI is retargeted. Note, that this does appear to complicate the security problem, however, retargeting only occurs when the Request-URI indicates a domain for which the processing entity is responsible, thus it would be the same processing entity that initially added the Request-URI to the header that would be updating it with the Reason.

The following summarizes the syntax of the History-Info header, based upon the standard SIP syntax [RFC3261]:

```
History-Info = "History-Info" HCOLON

                    hist-info *(COMMA hist-info)

hist-info = hi-targeted-to-uri *( SEMI hi-param )

hi-targeted-to-uri= name-addr

hi-param = hi-index / hi-extension

 hi-index = "index" EQUAL 1*DIGIT *(DOT 1*DIGIT)
```

hi-extension = generic-param

4.2 Protocol Examples

The following provides some examples of the History-Info header. Note that the backslash, CRLF, and spacing between the fields in the examples below are for readability purposes only.

```
History-Info:<sip:UserA@ims.nortelnetworks.com?Reason=SIP;\
  cause=302;text="Moved Temporarily">; index=1; foo=bar

History-Info: <sip:UserA@ims.nortelnetworks.com?Reason=SIP;\
   cause=302; text="Moved Temporarily">; index=1.1,
   <sip:UserB@nortelnetworks.com? Reason=SIP;cause=486;\
   text="Busy Here">;index=1.2,
   <sip:45432@vm.nortelnetworks.com> ; index=1.3
```

[Editor's note: need to insert row for Table 2].

4.3 Protocol usage

This section describes the processing specific to UAs and Proxies for the History-Info header and the Histinfo option tag. As discussed in section 1, the fundamental objective is to capture the target Request-URIs as a request is forwarded. This allows for the capturing of the history of a request that would be lost due to subsequent (re)targeting and forwarding. To accomplish this for the entire history of a request, either the UAC must capture the Request-URI in the initial request or a proxy must add History-Info headers for both the Request-URI in the initial request and the target Request-URI as the request is forwarded. The basic processing is for each entity forwarding a request to add a History-Info header for the target Request-URI, updating the index and adding the Reason as appropriate for any retargeted Request-URI.

[Editor's note: Once the Security solution is fully fleshed out, it may be reasonable to move this section 4.3 after section 4.4 and provide the detailed security related processing prior to this section, so that security aspects can be detailed in this section, as well.]

4.3.1 UAC Behavior

The UAC SHOULD include the Histinfo option tag in the Supported header in any request not associated with an established dialog for which the UAC would like the History-Info in the Response. In addition, the UAC SHOULD initiate the capturing of the History Information by adding a History-Info header using the Request-URI of the request as the hi-targeted-to-uri and initializing the index to 1 in the History-Info header

The processing of the History-Info header received in the Response is application specific and outside the scope of this draft. However, the validity of the information SHOULD be ensured prior to any application usage. [Editor's note: Further detail to be provided once the security solution is available.]

4.3.2 UAS Behavior

The processing of the History-Info header by a UAS in a Request depends upon local policy and specific applications at the UAS which might make use of the information. Prior to any application usage of the information, the validity SHOULD be ascertained. [Editor's note: Further detail to be provided once the security solution is available.]

If the Histinfo option tag is received in a request, the UAS should include any History-Info received in the request in the subsequent response.

4.3.3 Proxy Behavior

The inclusion of the History-Info header in a Request does not alter the fundamental processing of proxies for determining request targets as defined in section 16.5 of [RFC3261]. Whether a proxy adds the the History-Info header as it forwards a Request depends upon local policy, with the following being considerations in the definition of that policy:

- Whether the Request contains the Histinfo option tag in the Supported header.
- Whether the proxy supports the History-Info header.
- Whether any History-Info header added for a proxy/domain should go outside that domain. An example being the use of the History-Info header within the specific domain in which it is retargeted, however, policies (for privacy, user and network security, etc.) prohibit the exposure of that information outside that domain. An example of such an application is provided in Appendix C.
- Whether the History-Info header is added for a specific Request URI due to local privacy policy considerations.
- Within a given domain, whether there is a limit on the number of History-Info entries and the mechanism for applying the limit. [Issue-2: It has been highlighted that messages

carrying History-Info entries can become quite large in cases where there is a lot of retargeting. It seems that a reasonable recommendation could be provided for pruning the entries (albeit only entries added by that intermediary MAY be removed)].

An example policy would be a proxy that only adds the History-Info header if the Histinfo option tag is in the Supported header. Other proxies may have a policy that they always add the header, but never forward it outside a particular domain.

Each application making use of the History-Info header SHOULD address the impacts of the local policies on the specific application (e.g. what specification of local policy is optimally required for a specific application and any potential limitations imposed by local policy decisions).

Consistent with basic SIP processing of optional headers, proxies SHOULD maintain History-Info headers, received in messages being forwarded, independent of whether local policy supports History-Info.

The specific processing by proxies for adding the History-Info headers in Requests and Responses is described in detail in the following sections.

4.3.3.1 Adding the History-Info header to Requests

If the proxy supports the History-Info header, the proxy SHOULD add a History-Info header as it forwards a Request. Section 16.6 of [4] defines the steps to be followed as the proxy forwards a Request. Step 5 prescribes the addition of optional headers. Although, this would seem the appropriate step for adding the History-info header, the interaction with Step 6 "Postprocess routing information" and the impact of a strict route in the Route header could result in the Request-URI being changed, thus adding the History-info header between steps 8 (adding Via header) and 9 (adding Content-Length) is RECOMMENDED. Note, that in the case of loose routing, the Request-URI does not change during the forwarding of a Request, thus the capturing of History-Info for such a request would result in duplicate Request-URIs with different indices. The History-Info header SHOULD be added following any History-Info header received in the request being forwarded. Additionally, if a request is received that doesn't include a History-Info header, the proxy MAY add an additional History-Info header preceding the one being added for the current request being forwarded. The index for this entry is RECOMMENDED to start at 1.

For retargets that are the result of an explicit SIP response, the SIP Response Code that triggered the retargeting MUST be included in

the Reason header field of the Request URI that has been retargeted. For retargets as a result of timeouts or internal events, a Reason MAY be included in the Reason header field of the Request URI that has been retargeted.

In order to maintain ordering and accurately reflect the nesting and retargeting of the request, an index MUST be included along with the Targeted-to-URI being captured. Per the ABNF in section 4.1, the index consists of a dot delimited series of digits (e.g. 1.1.2), with each dot reflecting the number of hops or level of nesting of the request. Thus, the indexing results in a logical tree representation for the history of the Request. It is recommended that for each level of indexing, the index start at 1. For retargets within a proxy, the proxy MUST maintain the current level of nesting by incrementing the lowest/last digit of the index for each instance of retargeting, thus reflecting the number of retargets within the proxy.

The basic rules for adding the index are summarized as follows:

1. If the Request-URI in the original request indicates a resource for which this proxy is responsible, then the proxy reads the value from the History-Info header in the received request, if available, and adds another level of indexing by appending the DOT delimiter followed by an initial index for the new level of 1. For example, if the index in the last History-Info header field in the received request is 1.1, this proxy would initialize its index to 1.1.1. For each subsequent target that is forwarded by the same proxy, theindex is calculated by incrementing the last/lowest digit at the current level.

2. If the Request-URI indicates a resource that this proxy is not responsible for, then the lowest/last digit of the index is incremented (i.e. a new level is not created). For example, if the index in the History-Info header of the received request was 1.2, then the index in the History-Info header field added by this proxy would be 1.3.

If the request forwarding is done in parallel, the proxy MUST capture each of the Request-URIs to which the Request is forwarded in the manner previously described per rule 1 above. The index MUST be captured for each forked request per the rules above, with each new Request having a unique index. The proxy builds the subsequent requests and responses using the amalgamated information associated with each of those requests and including the header entries in the order indicated by the indexing. Section 4.5 provides an example of a parallel request scenario, highlighting this indexing mechanism.

4.3.3.2 Processing History-Info in Responses

A proxy that receives a Request with the Histinfo option tag in the Supported header, and depending upon a local policy supporting the capture of History-Info, SHOULD return captured History-Info in subsequent, provisional and final responses to the Request.

It should be noted that local policy considerations, for network and intermediary privacy, MAY restrict the sending of the History-Info headers added by the intermediary in subsequent responses. Thus, in such cases, the proxy MAY remove from these responses the History-Info headers which it inserted in the original forwarded request.

4.3.4 Redirect Server Behavior

A redirect server SHOULD NOT add any new History-Info, as that would be done by the entity receiving the 3xx response. However, a redirect server MAY include History-Info in responses by adding any History-Info headers received in a request to a subsequent response.

4.4 Security for History-Info

As discussed in Section 1, the security requirements are partially met by recommending the use of TLS (a basic SIP requirement per [RFC3261]) for hop by hop security. In addition, the use of the middle-to-end security solution discussed in [SIPIISEC] allows the integrity of the History-Info to be ascertained as it traverses the intermediaries.
For the History-Info header, the general requirement is to secure a header that is inserted by an intermediary and then subsequently referenced, by other intermediaries to build the next header entry or by an end application using the information to provide a service. In terms of exactly what is being secured, it is primarily the captured Request-URIs that are the security concern, since they can reflect some aspect of a user's identity and service routing. However, the indices are also important in that they can be used to determine if specific Request-URIs have been removed from the header. Thus, the primary objective of the security solution is to ensure that the entire History-Info header is protected from being accessed or manipulated by non-authorized entities, with the fundamental assumption that retargeting entities are implicitly authorized.

The security associated with the Request History Information is optional and depends upon local policy and the impact on specific applications of having the information compromised. Since, the Request History Information itself is also optional and it has been recommended that applications document the impact of the information not being available, it is also suggested that the impact of not supporting the security recommendations also be documented by the application to ensure that the impacts have been sufficiently addressed by the application.

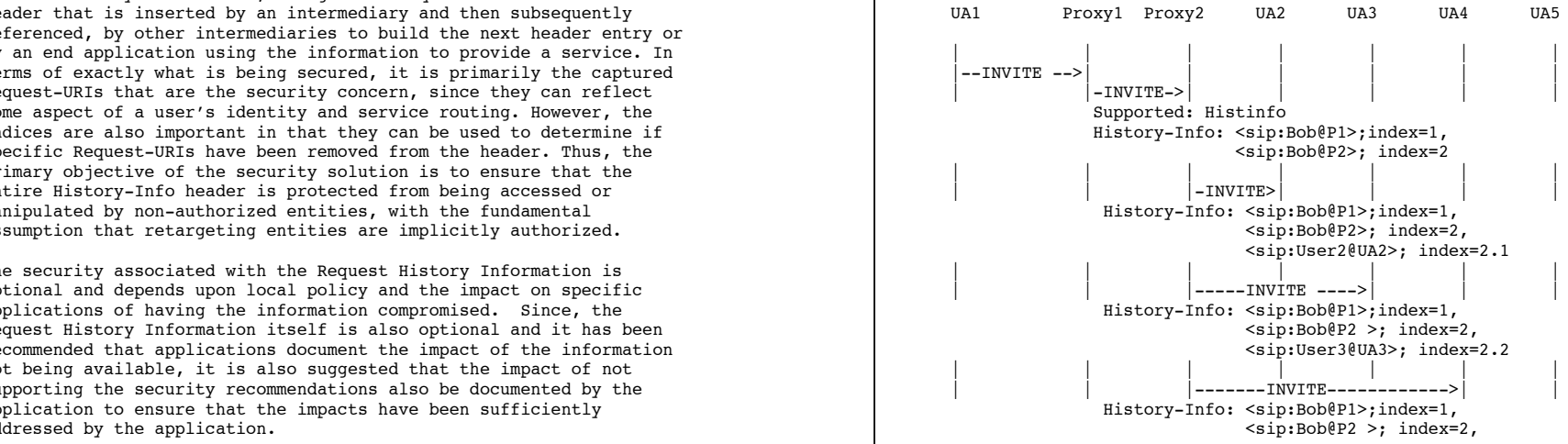
4.4.1 Security examples

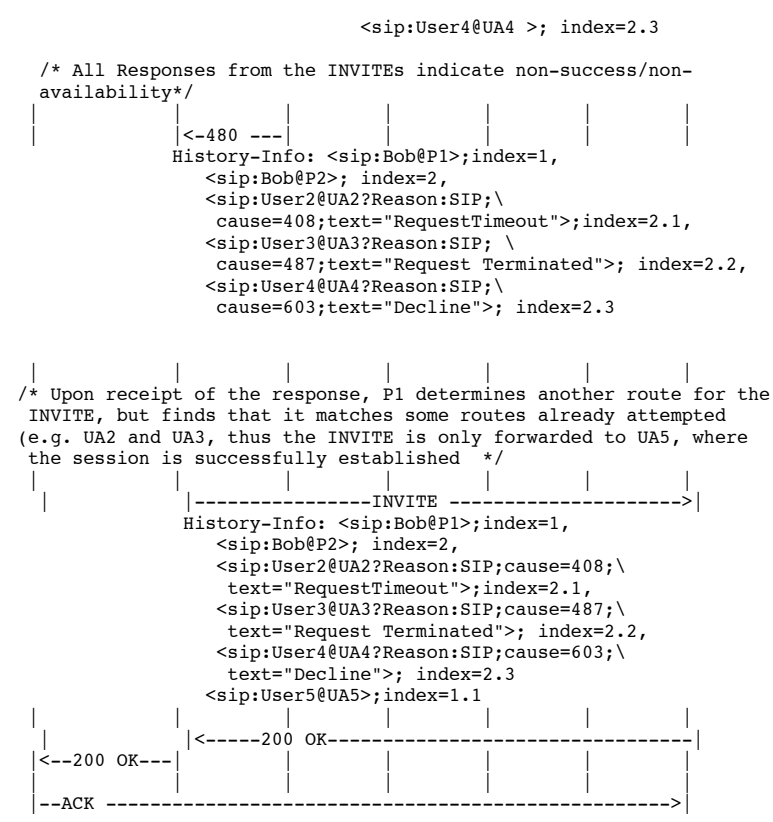
[Editor's Note: Need to add some protocol details for protecting History-Info once [SIPIISEC] is further along].

4.5 Example Applications using History-Info

This scenario highlights an example where the History-Info in the response is primarily of use in not retrying routes that have already been tried by another proxy. Note, that this is just an example and that there may be valid reasons why a Proxy would want to retry the routes and thus, this would likely be a local proxy or even user specific policy.

UA 1 sends a call to "Bob" to proxy 1. Proxy 1 forwards the request to Proxy 2. Proxy 2 sends the requests in parallel and tries several places (UA2, UA3 and UA4) before sending a response to Proxy 1 that all the places are busy. Proxy 1, without the History-Info, would try several of the same places (UA3 and UA4) based upon registered contacts for "Bob", before completing at UA5. However, with the History-Info, Proxy 1 determines that UA3 and UA4 have already received the invite, thus the INVITE goes directly to UA5.





5. Application Considerations

As seen by the example scenarios in the appendix, History-Info provides a very flexible building block that can be used by intermediaries and UAs for a variety of services. As such, any services making use of History-Info must be designed with the following considerations:

1) History-Info is optional, thus a service should define default behavior for requests and responses not containing History-Info headers.
2) History-Info may be impacted by privacy considerations. Applications requiring History-Info need to be aware that if Header or Session level privacy is requested by a UA (or imposed by an intermediary) that History-Info may not be available in a request or response. This would be addressed by an application in the same manner as the previous consideration by ensuring there is reasonable default behavior should the information not be available.
3) History-Info may be impacted by local policy. Each application making use of the History-Info header SHOULD address the impacts of the local policies on the specific application (e.g. what specification of local policy is optimally required for a specific application and any potential limitations imposed by local policy decisions). Note, that this is related to the optionality and privacy considerations identified in 1 and 2 above, but goes beyond that. For example, due to the optionality and privacy considerations, an entity may receive only partial History-Info entries; will this suffice? Note, that this would be a limitation for debugging purposes, but might be perfectly satisfactory for some models whereby only the information from a specific intermediary is required.
4) The security associated with the Request History Information is optional. Whether there is security applied to the entries depends upon local policy. The impact of lack of having the information compromised depends upon the nature of the specific application (e.g. is the information something that appears on a display or is it processed by automata which could have negative impacts on the subsequent processing of a request?). It is suggested that the impact of an intermediary not supporting the security recommendations should be evaluated by the application to ensure that the impacts have been sufficiently addressed by the application. For the display example, a visual indicator could be applied highlighting that the information has not been, or could not be, validated.

6. Security Considerations

This draft provides a proposal in sections 3.2 and 4.4 for addressing the Security requirements identified in section 2.1 by proposing the use of TLS between entities, and by securing the History-Info headers added by proxies as described in [SIPIISEC].

7. IANA Considerations

(Note to RFC Editor: Please fill in all occurrences of XXXX in this section with the RFC number of this specification).

This document defines a new SIP header field name: History-Info and a new option tag: Histinfo.

The following changes should be made to http:///www.iana.org/assignments/sip-parameters

The following row should be added to the header field section:

Header Name	Compact Form	Reference
History-Info	none	[RFCXXXX]

The following should be added to the Options Tags section:

Name	Description	Reference
Histinfo	When used with the Supported header, this option tag indicates support for the History Information to be captured for requests and returned in subsequent responses. This tag is not used in a Proxy-Require or Require header field since support of History-Info is optional.	[RFCXXXX]

Open Issues

The following summarizes the current open issues in this document:

- Issue-1: Privacy indication for specific History-Info entries. It has been proposed on the mailing list that there is a requirement beyond the basic Header or Session privacy provided by RFC 3323 for History-Info entries in terms of supporting local policy based privacy requirements. It has been suggested that adding an additional field to the History-Info header (or extending the priv-values defined in RFC 3323) would facilitate the implementation of this functionality. Adding such information to the HI entries would impact the protocol structure in section 4.1 and processing in 4.3.3 (and 4.3.3.1 and 4.3.3.2)

- Issue-2: Bounding the History-Info entries and a mechanism for applying the limit. It has been highlighted by developers that messages carrying History-Info entries can become quite large in cases where there is a lot of retargeting. It seems that a reasonable recommendation could be provided for pruning the entries (albeit only entries added by that intermediary should be removed). For example:

  - Keeping only the first and last entries
  - Keeping only the last leaf of each of the branches.
  - Restricting the breadth and depth of the History-Info tree.

  Such bounding would require normative processing guidelines in section 4.3.3 and introduce an additional application consideration in section 5.

Changes since last version

Changes from the û01 to the û02 version:

- Merged the SIPPING WG requirements draft into this document. Note that this increments the section references in the remainder of the document by 2 (and by 3 for Security and IANA considerations due to new section added). Also, removed redirect server from ISSUER-req since the solution identified this as not being required (or desirable).
- Added an explicit privacy requirement (PRIV-req-3) for the proxy's role in recognizing and maintaining privacy associated with a Request-URI being captured in History-Info due to local policy. (Note, that the text was already there, it just wasn't highlighted as an explicit requirement).
- Clarified the use of CRLF and spacing in the example headers in section 4.2.
- Removed the compact form for the header since unknown headers with multiple entries would not be recognized (i.e. this may cause parsing problems).
- Added a summary of Application Considerations to address concerns about the optional usage of History-Info.
- Converted the references from numbers to labels to avoid the continual problem of renumbering.
- Minor editorial changes (per NITS highlighted by Rohan and Eric and some minor rewording for clarity).

Changes from the û00 to the û01 version:

- Attempted to be more explicit about the fundamental processing associated with the header. Removed definitions of new terms, only referencing the terms from the requirements in the context of the fundamental SIP processing implied by the terms.
- Attempted to clarify the Index and the related processing.
- Added more detail addressing the privacy requirements.
- Added a bit more detail on security. The security solution remains in a separate document and this document will need updating once that is completed.

- Updated the examples (in section 2.5 and appendix) and clarified the definition and the maintenance of the Index in sections 2.1 and 2.3.3.1.
- Clarified the Reason description in section 2.1. There had been an error in the description of the processing that was a remnant of the change to include only a single URI for each History-Info header.
- Miscellaneous editorial changes (i.e. HistInfo -> Histinfo, etc.)

Changes from individual draft-barnes-sipping-history-info-02 to the û 00 WG version:

- Updated references and added reference to Security solution draft.
- Removed appendix D which included background on analysis of solution options.
- Cleaned up the document format per rfc2223bis.
- Strengthened the inclusion of the INDEX as a MUST (per discussion at IETF-56).
- Added text around the capturing of the Reason (SHOULD be captured for SIP responses and MAY be captured for other things such as timeouts).
- Clarified the response processing 2.3.3.2 to include provisional responses and the sending of a 183 to convey History-Info.
- Added section 2.3.4 to address Redirect Server behavior.

Normative References

[RFC3261] J. Rosenberg et al, "SIP: Session initiation protocol," RFC 3261, June, 2002.

[RFC3326] H. Schulzrinne, D. Oran, G. Camarillo, "The Reason Header Field for the Session Initiation Protocol", RFC 3326, December, 2002.

[RFC3323] J. Peterson, "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November, 2002.

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

[RFC2234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[SIPIISEC] M. Barnes, "A Mechanism to Secure SIP Headers Inserted by Intermediaries", draft-barnes-sipping-inserted-info-01.txt, October, 2003.

Informational References

[SIPSVCEX] A. Johnson, "SIP Service Examples", draft-ietf-sipping-service-examples-05.txt, November, 2002.

[SIPATHID] J. Peterson, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-01.txt, February, 2003.

Acknowledgements

The editor would like to acknowledge the constructive feedback provided by Robert Sparks, Paul Kyzivat, Scott Orton, John Elwell, Nir Chen, Francois Audet, Palash Jain, Brian Stucker, Norma Ng, Anthony Brown, Jayshree Bharatia, Jonathan Rosenberg and Eric Burger.

The editor would like to acknowledge the significant input from Rohan Mahy on some of the normative aspects of the ABNF, particularly around the need for and format of the index and around the enhanced SIP security aspects enabled by this draft.

Contributors' Addresses

Cullen, Mark and Jon contributed to the development of the initial requirements.

Cullen and Mark provided substantial input in the form of email discussion in the development of the initial version of the individual solution document.

Cullen Jennings
Cisco Systems
170 West Tasman Dr
MS: SJC-21/3

Tel: +1 408 527 9132
Email: fluffy@cisco.com

Jon Peterson
NeuStar, Inc.
1800 Sutter Street, Suite 570
Concord, CA 94520
USA

Phone: +1 925-363-8720
EMail: Jon.Peterson@NeuStar.biz

Mark Watson
Nortel Networks (UK)
Maidenhead Office Park (Bray House)
Westacott Way
Maidenhead,
Berkshire
England

Tel: +44 (0)1628-434456
Email: mwatson@nortelnetworks.com

Author's Address

Mary Barnes
Nortel Networks
2380 Performance Drive
Richardson, TX USA

Phone: 1-972-684-5432
Email: mary.barnes@nortelnetworks.com

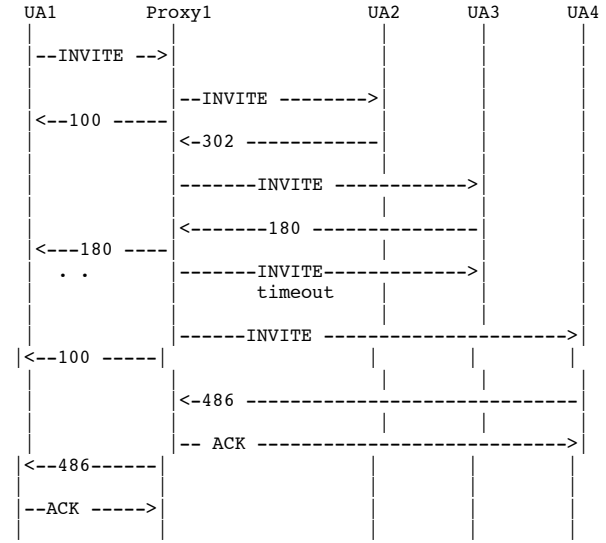
Appendix A Forking Scenarios

A.1 Sequentially forking (History-Info in Response)

This scenario highlights an example where the History-Info in the response is useful to an application or user that originated the request.

UA 1 sends a call to "Bob" via proxy 1. Proxy 1 sequentially tries several places (UA2, UA3 and UA4) unsuccessfully before sending a response to UA1.

This scenario is provided to show that by providing the History-Info to UA1, the end user or an application at UA1 could make a decision on how best to attempt finding "Bob". Without this mechanism UA1 might well attempt UA3 (and thus UA4) and then re-attempt UA4 on a third manual attempt at reaching "Bob". With this mechanism, either the end user or application could know that "Bob" is busy on his home phone and is physically not in the office. If there were an alternative address for "Bob" known to this end user or application, that hasn't been attempted, then either the application or the end user could attempt that. The intent here is to highlight an example of the flexibility of this mechanism that enables applications well beyond SIP as it is certainly well beyond the scope of this draft to prescribe detailed applications.

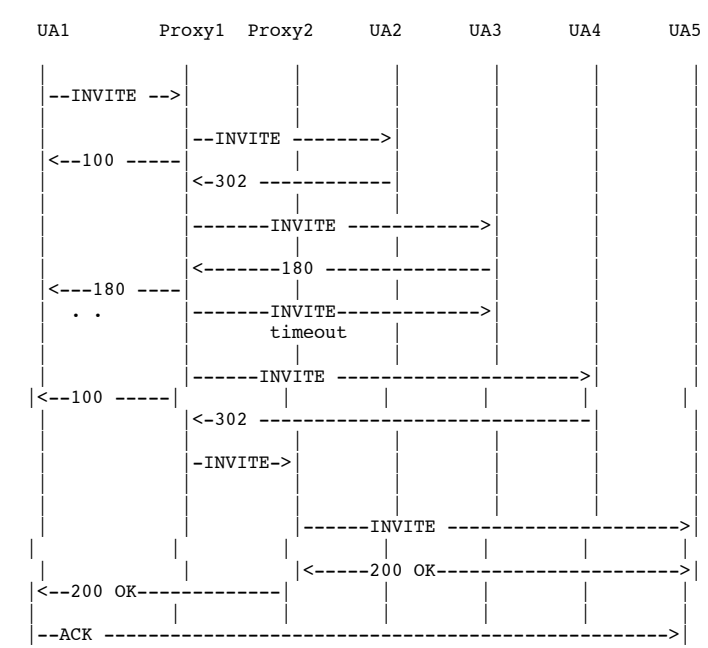


[Editor's Note: Need to detail the message flow.]

A.2 Sequential Forking (with Success)

This scenario highlights an example where the History-Info in the request is primarily of use in not retrying routes that have already been tried by another proxy. Note, that this is just an example and that there may be valid reasons why a Proxy would want to retry the routes and thus, this would like be a local proxy or even user specific policy.

UA 1 sends a call to "Bob" to proxy 1. Proxy 1 sequentially tries several places (UA2, UA3 and UA4) before retargeting the call to Proxy 2. Proxy 2, without the History-Info, would try several of the same places (UA3 and UA4)based upon registered contacts for "Bob", before completing at UA5. However, with the History-Info, Proxy 2 determines that UA3 and UA4 have already received the invite, thus the INVITE goes directly to UA5.



[Editor's Note: Need to add the details of the messages here.]

Appendix B Voicemail

This scenario highlights an example where the History-Info in the request is primarily of use by an edge service (e.g. Voicemail Server). It should be noted that this isn't intended to be a complete specification for this specific edge service as it is quite likely that additional information is need by the edge service. History-Info is just one building block that this service makes use of.

UA 1 called UA A which had been forwarded to UA B which forwarded to a UA VM (voicemail server). Based upon the retargeted URIs and Reasons (and other information) in the INVITE, the VM server makes a policy decision about what mailbox to use, which greeting to play etc.

Message Details

INVITE F1 UA1->Proxy

```
INVITE sip:UserA@nortelnetworks.com SIP/2.0
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Contact: BigGuy <sip:User1@here.com>
Content-Type: application/sdp
Content-Length: <appropriate value>

  v=0
  o=UserA 2890844526 2890844526 IN IP4 client.here.com
  s=Session SDP
  c=IN IP4 100.101.102.103
  t=0 0
  m=audio 49170 RTP/AVP 0
  a=rtpmap:0 PCMU/8000

  /*Client for UA1 prepares to receive data on port 49170
```

```
from the network. */

INVITE F2 Proxy->UA-A

INVITE sip:UserA@ims.nortelnetworks.com SIP/2.0
Via: SIP/2.0/UDPims.nortelnetworks.com:5060;branch=1
  Via: SIP/2.0/UDP here.com:5060
Record-Route: <sip:UserA@nortelnetworks.com>
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>
Call-Id: 12345600@here.com
CSeq: 1 INVITE
History-Info: <sip:UserA@ims.nortelnetworks.com>; index=1
Contact: BigGuy <sip:User1@here.com>
Content-Type: application/sdp
Content-Length: <appropriate value>

v=0
o=UserA 2890844526 2890844526 IN IP4 client.here.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

100 Trying F3 Proxy->UA1

SIP/2.0 100 Trying
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Content-Length: 0


302 Moved Temporarily F4  UserA->Proxy
SIP/2.0 302 Moved Temporarily
Via: SIP/2.0/UDP ims.nortelnetworks.com:5060;branch=1
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>;tag=3
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Contact: <sip:UserB@nortelnetworks.com>
Content-Length: 0
```



```
INVITE F5 Proxy-> UA-B

INVITE sip:UserB@nortelnetworks.com SIP/2.0
Via: SIP/2.0/UDP ims.nortelnetworks.com:5060;branch=2
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>
Call-Id: 12345600@here.com
History-Info: <sip:UserA@ims.nortelnetworks.com?Reason=SIP;\
cause=302; text="Moved Temporarily">; index=1,
<sip:UserB@nortelnetworks.com>;index=2
CSeq: 1 INVITE
Contact: BigGuy <sip:User1@here.com>
Content-Type: application/sdp
Content-Length: <appropriate value>

v=0
o=User1 2890844526 2890844526 IN IP4 client.here.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

180 Ringing F6  UA-B ->Proxy

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP there.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>;tag=5
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Content-Length: 0

180 Ringing F7  Proxy-> UA1

SIP/2.0 180 Ringing
SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Content-Length: 0

/* User B is not available. INVITE is sent multiple
times until it times out. */

  /* The proxy forwards the INVITE to UA-VM after adding the
additional History Information entry. */
```

```
INVITE F8  Proxy-> UA-VM

INVITE sip:VM@nortelnetworks.com SIP/2.0
Via: SIP/2.0/UDP ims.nortelnetworks.com:5060;branch=3
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
   To: LittleGuy <sip:UserA@nortelnetworks.com>
Call-Id: 12345600@here.com
History-Info:<sip:UserA@ims.nortelnetworks.com?Reason=SIP;\
cause=302; text="Moved Temporarily">;index=1,
<sip:UserB@nortelnetworks.com?Reason=SIP;cause=480;\
text="Temporarily Unavailable" >;index=2,
<sip:VM@nortelnetworks.com>;index=3
CSeq: 1 INVITE
Contact: BigGuy <sip:User1@here.com>
Content-Type: application/sdp
Content-Length: <appropriate value>

v=0
o=User1 2890844526 2890844526 IN IP4 client.here.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000


200 OK F9

SIP/2.0 200 OK UA-VM->Proxy

Via: SIP/2.0/UDP ims.nortelnetworks.com:5060;branch=3
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>;tag=3
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Contact: TheVoiceMail <sip:VM@nortelnetworks.com>
Content-Type: application/sdp
Content-Length: <appropriate value>

v=0
o=UserA 2890844527 2890844527 IN IP4 vm.nortelnetworks.com
s=Session SDP
c=IN IP4 110.111.112.114
t=0 0
m=audio 3456 RTP/AVP 0
```



```
a=rtpmap:0 PCMU/8000


200 OK F10  Proxy->UA1

SIP/2.0 200 OK
Via: SIP/2.0/UDP ims.nortelnetworks.com:5060;branch=3
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy <sip:UserA@nortelnetworks.com>;tag=3
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Contact: TheVoiceMail <sip:VM@nortelnetworks.com>
Content-Type: application/sdp
Content-Length: <appropriate value>

v=0
o=UserA 2890844527 2890844527 IN IP4 vm.nortelnetworks.com
s=Session SDP
c=IN IP4 110.111.112.114
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000

ACK F11 UA1-> UA-VM

ACK sip:VM@nortelnetworks.com SIP/2.0
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:User1@here.com>
To: LittleGuy<sip:UserA@nortelnetworks.com>;tag=3
Call-Id: 12345600@here.com
CSeq: 1 ACK
Content-Length: 0

/* RTP streams are established between UA1 and
UA-VM. UA-VM starts announcement for UA1 */
```

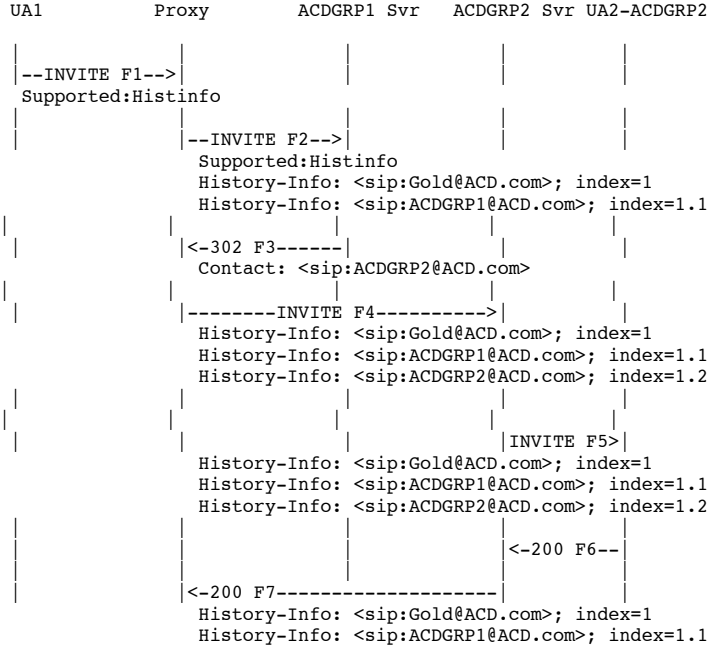
Appendix C Automatic Call Distribution Example

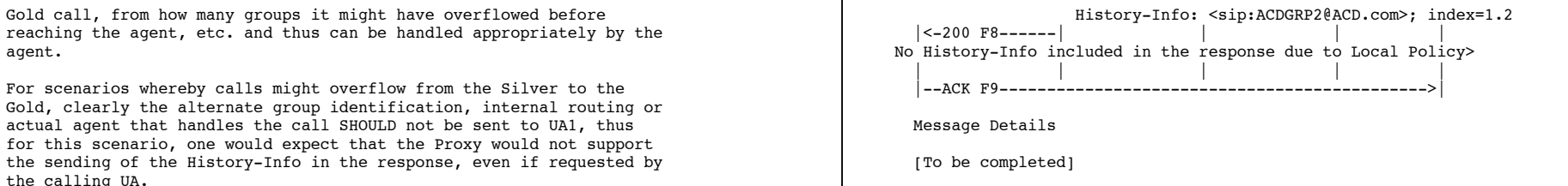
This scenario highlights an example of an Automatic Call Distribution service, where the agents are divided into groups based upon the type of customers they handle. In this example, the Gold customers are given higher priority than Silver customers, so a Gold call would get serviced even if all the agents servicing the Gold group (ACDGRP1) were busy, by retargeting the request to the Silver Group. Upon receipt of the call at the agent assigned to handle the incoming call, based upon the History-Info header in the message, the application at the agent can provide an indication that this is a

Gold call, from how many groups it might have overflowed before reaching the agent, etc. and thus can be handled appropriately by the agent.

For scenarios whereby calls might overflow from the Silver to the Gold, clearly the alternate group identification, internal routing or actual agent that handles the call SHOULD not be sent to UA1, thus for this scenario, one would expect that the Proxy would not support the sending of the History-Info in the response, even if requested by the calling UA.

As with the other examples, this is not prescriptive of how one would do this type of service but an example of a subset of processing that might be associated with such a service. In addition, this example is not addressing any aspects of Agent availability, which might also be done via a SIP interface.





Full Copyright Statement

SIP WG C. Jennings
Internet-Draft Cisco Systems
Expires: August 14, 2004 February 14, 2004

SIP Conventions for Voicemail URIs
draft-jennings-sip-voicemail-uri-01

Status of this Memo

Copyright Notice

Abstract

SIP systems are often used to initiate connections to voicemail or unified messaging systems. This document describes a convention for forming SIP Request URIs that request particular services from unified messaging systems.

This work is being discussed on the sip@ietf.org mailing list.

Table of Contents

1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

2. Introduction

Unified messaging systems (UM) have developed out of traditional voice mail systems. They can be used for storing and interacting with voice, video, faxes, email and instant messaging. Users often use SIP to initiate communications with them. When a SIP call is routed to a UM, there is a requirement for the UM to be able to figure out several bits of information from the call so that it can deliver the desired services. The UM needs to know what mailbox should be used for the context of this call and possible reasons about what type of service is desired. This includes knowing the type of media (voice or IM for example). Many voice mail systems provide different greetings depending whether the reason the call was sent to voicemail was that the user was busy or because the user did not answer. All of this information can be delivered in existing SIP signaling from the call control that retargets the call to the UM, but there are no standardized conventions for describing how the desired mailbox and service requested are expressed. It would be possible for every vendor to make this configurable so that any site can get it to work; however, this is not a very realistic view of achieving interoperability among call control, gateways, and unified messaging systems from different vendors. These requirements and more are described in the History Requirements [9]. This document describes a convention for describing this mailbox and service information in the SIP URI so that vendors and operators can build interoperable systems. It meets some but not all of the requirements in [9].

The work in the History Info [10] draft can be used in similar systems. It is more comprehensive and covers a much wider set of requirements. A key difference from this system is that history allows the UM to look at the history of the call and decided on what the best treatment is for the call. This work requires the call control system to know something about the history of the call and specifically ask the UM to invoke a particular service.

If there were no need to interoperate with TDM based voicemail systems or allow TDM systems to use VoIP unified messaging systems, this problem would be a little easier. The problem that is introduced in the VoIP to TDM case is as follows. The SIP system needs to tell a PSTN GW both the subscriber's mailbox identifier (which typically looks like a phone number) and the address of the voicemail system in the TDM network (again a phone number).

One topic that causes some confusion in the requirements for this has to do with the fact that the related PSTN mechanism can carry two addresses. These correspond to the original target of the call and the most recent target to which it has been redirected. In general, the original target is used to find the voice mail box. The target that most recently redirected is not as useful for voicemail but is very useful for billing. It is often used to bill the most recent portion of the call leg. This work addresses only the requirements for UM system, and billing is completely out of scope. The History draft is much more extensive and covers more cases that might be useful for billing, but this work does not.

The question has been asked why the To header cannot be used to understand which mailbox to use. One of the problems with this is that the call control proxies cannot modify the To header, and the UAC often set it incorrectly because they do not have information about the subscribers in the domain they are trying to call. This happens because the routing of the call often translates the URI multiple times before it results in an identifier for the desired user that is valid in the namespace that the UM system understands.

Another set of requirements that this mechanism can deal with is the call coverage naming issues. The problem is when Bill calls the 800 number that sends him to the helpdesk, the proxy may first fork the call to Alice (who works at the help desk), and then if Alice does not answer in a few seconds fork the call on to Bob (who also works at the helpdesk). Both Alice and Bob would like to be informed that the call was to the help desk before they answer the call. If neither answers, the call may get sent to the help desk's voice mailbox, not Bob's or Alice's.

3. Mechanism (UAS and Proxy)

The mechanism works by encoding the information for the desired service in the SIP URI that is sent to the UM system. Two chunks of information are encoded, the first being the target mailbox to use and the second being the SIP error code that caused this retargeting and indicates the desired service. The target mailbox can be put in the user part of the URI and is also put in a target URI parameter while the reason is put in the Reason header. For example, if the proxy wished to use Alice's mailbox because her phone was busy, the URI sent to the UM system could be something like:

sip:alice@um.example.com;target=alice

and include a Reason header like:

Reason: SIP ;cause=486

3.1 Target

The target parameter indicates the mailbox to use. In many cases the user portion of the SIP URI could be set to the same value but it does not have to be. For example in the case of a voice mail system on the PSTN, the user portion will contain the phone number of the voice mail system while the target will contain the phone number of the subscriber's mailbox.

3.2 Reason Header Usage

The Reason header, defined in RFC-3326 [7], is used to indicates the target="RFC2119"service that the UAS receiving the message should perform. It corresponds to the SIP Status-Code that results in the desired service being requested. A mapping between some common services and reason codes are:

Service	Reason Parameter
Busy	486
No answer	408
Unconditional	302
Deflect	487
No Contacts/Failure of UA	410

This drafts extends the Reason headers to be allowed in a SIP request outside of a Dialog.

3.3 Retrieving Messages

The UM system MAY use the fact that the From header is the same as the URI target as a hint that the user wishes to retrieve messages.

4. Interaction with Netann

This approach is designed to interact well with the netann mechanism. A netann parameter[8] can be used to indicate exactly which initial prompt to play.

5. Interaction with History

The History mechanism[10] provides considerably more information that is useful for a UM system. This work does not stop a UM system from taking advantage of the History information if it is present and using that to handle the call.

6. Limitations of Voicemail URI

This system requires the proxy that is requesting the service to understand what are valid targets on the UM system. For practical purposes this means that the approach is unlikely to work in many cases where the proxy is not configured with information about the UM system or if the proxy is not in the same administrative domain.

This system requires the call control proxy to know what it wants the UM to do instead of giving the UM system the information about the call that allows the UM system to decide what to do. For example, if a call to the help desk got forwarded first to Alice, then to Bob, then finally to the helpdesk UM system, the UM system may want to leave a copy of the message in the primary help desk mail box and also leave a copy in Alice's mailbox since she was the primary person at the helpdesk. In addition the UM system might want to page Alice, Bob and their supervisor to let them know that no one is staffing the help desk. This system does not provide enough information to the UM system about what happened to the call to meet the needs of a scenario such as the one above.

This system only works when the service the call control wants applied is fairly simple. For example it does not allow the proxy to express information like "Do not offer to connect to the target's colleague because that address was already tried".

Some systems have expressed requirements for the UAC to understand when the call is re-targeted and get updated information about where it was targeted to as the call proceeds. This work does not address this requirement - History does, as does the option of just sending a 1xx class message with a Reason header[7].

The mechanism in this document does not address any billing issues associated with forwarded calls. This is a separate problem.

These limitations discussed in this section are addressed by the History[10] work.

7. Examples

7.1 Proxy Forwards No Answer to Voicemail

In this example, Alice calls Bob. Bob's proxy runs a timer and determines that Bob has not answered his phone, and the proxy forwards the call to Bob's voicemail. Alice's phone is at 192.168.0.1 while Bob's phone is at 192.168.0.2. The important things to note is the URI and Reason header in message F4.

F1: INVITE 192.168.0.1 -> proxy.example.com

```
INVITE sip:15555551002@example.com;user=phone SIP/2.0
Via: SIP/2.0/TCP 192.168.0.1:5060;branch=z9hG4bK-74bf9
From: Alice <sip:5551001@example.com>;tag=9fxced76sl
To: sip:15555551002@example.com;user=phone
Call-ID: c3x842276298220188511
CSeq: 1 INVITE
Max-Forwards: 70
Contact: <sip:x123456x@192.168.0.1;transport=tcp>
Content-Type: application/sdp
Content-Length: *Body length goes here*

* SDP goes here*
```

F2: INVITE proxy.example.com -> 192.168.0.2

```
INVITE sip:line1@192.168.0.2 SIP/2.0
Via: SIP/2.0/TCP 192.168.1.4:5060;branch=z9hG4bK-ik80k7g-1
Via: SIP/2.0/TCP 192.168.0.1:5060;branch=z9hG4bK-74bf9
From: Alice <sip:5551001@example.com>;tag=9fxced76sl
To: sip:15555551002@example.com;user=phone
Call-ID: c3x842276298220188511
CSeq: 1 INVITE
Max-Forwards: 70
Contact: <sip:x123456x@192.168.0.1;transport=tcp>
Content-Type: application/sdp
Content-Length: *Body length goes here*

* SDP goes here*
```

F3: 486 192.168.0.2 -> proxy.example.com

```
SIP/2.0 486 Busy Here
Via: SIP/2.0/TCP 192.168.1.4:5060;branch=z9hG4bK-ik80k7g-1
Via: SIP/2.0/TCP 192.168.0.1:5060;branch=z9hG4bK-74bf9
From: Alice <sip:5551001@example.com>;tag=9fxced76sl
To: sip:15555551002@example.com;user=phone;tag=09xde23d80
Call-ID: c3x842276298220188511
CSeq: 1 INVITE
Contact: <sip:x654321x@192.168.0.2;transport=tcp>
Content-Length: 0
```

F4: INVITE proxy.example.com -> um.example.com


```
INVITE sip:bob@um.example.com;target=bob SIP/2.0
Reason SIP ;cause=486
Via: SIP/2.0/TCP 192.168.1.4:5060;branch=z9hG4bK-ik80k7g-2
Via: SIP/2.0/TCP 192.168.0.1:5060;branch=z9hG4bK-74bf9
From: Alice <sip:5551001@example.com>;tag=9fxced76sl
To: sip:15555551002@example.com;user=phone
Call-ID: c3x842276298220188511
CSeq: 1 INVITE
Max-Forwards: 70
Contact: <sip:x123456x@192.168.0.1;transport=tcp>
Content-Type: application/sdp
Content-Length: *Body length goes here*

* SDP goes here*
```

7.2 Zero Configuration UM System

In this example, the UM system has no configuration information specific to any user. The proxy is configured to pass a URI that provides the prompt to play and an email address in the user portion of the URI to send the recorded message to.

The call flow is the same as in the previous example except that the URI in F4 changes to specify the user part as Bob's email address, and the netann URI play parameter specifies where the greeting to play can be fetched from.

F4: INVITE proxy.example.com -> um.example.com

```
INVITE
   sip:bob@um.example.com;target=mailto:bob@example.com;
   play=http://www.example.com/bob/busy.way
   SIP/2.0
Reason: SIP ;cause=486
Via: SIP/2.0/TCP 192.168.1.4:5060;branch=z9hG4bK-ik80k7g-2
Via: SIP/2.0/TCP 192.168.0.1:5060;branch=z9hG4bK-74bf9
From: Alice <sip:5551001@example.com>;tag=9fxced76sl
To: sip:15555551002@example.com;user=phone
Call-ID: c3x842276298220188511
CSeq: 1 INVITE
Max-Forwards: 70
Contact: <sip:x123456x@192.168.0.1;transport=tcp>
Content-Type: application/sdp
Content-Length: *Body length goes here*

* SDP goes here*
```

In addition, if the proxy wished to indicate a VXML script that the UM should execute, it could add a parameter to the URI in the above message that looked like:

voicexml=http://www.example.com/bob/busy.vxml

7.3 TDM Voice Mail Connected via a Gateway

In this example, the voicemail system has a TDM interconnect to a gateway to the VoIP system. Bob's mailbox is +1 555 555-1002 while the address of the voicemail system on the TDM network is +1 555 555-2000.

The call flow is the same as in the previous example except for the URI in F4.

F4: INVITE proxy.example.com -> gw.example.com

```
INVITE sip:+1-555-555-2000@um.example.com;user=phone;\
       target=tel:+1-555-555-1002
       SIP/2.0
Reason: SIP ;cause=486
Via: SIP/2.0/TCP 192.168.1.4:5060;branch=z9hG4bK-ik80k7g-2
Via: SIP/2.0/TCP 192.168.0.1:5060;branch=z9hG4bK-74bf9
From: Alice <sip:5551001@example.com>;tag=9fxced76sl
To: sip:15555551002@example.com;user=phone
Call-ID: c3x842276298220188511
CSeq: 1 INVITE
Max-Forwards: 70
Contact: <sip:x123456x@192.168.0.1;transport=tcp>
Content-Type: application/sdp
Content-Length: *Body length goes here*

* SDP goes here*
```

7.4 Call Coverage

In this example a user on the PSTN calls a 800 number. The GW sends this to the proxy which recognizes that the helpdesk is the target. Alice and Bob are staffing the help desk and are tried sequentially but neither answers, so the call is forwarded to the helpdesk's voice mail.

The key item in this flow is that the invite to Alice and Bob looks like


```
INVITE sip:bob@um.example.com;target=helpdesk SIP/2.0
Reason: SIP ;cause=302
```

8. Syntax

This document updates the BNF in Section 25 of RFC 3261 [3] to add the target-param to the uri-parameter as shown below.

```
uri-parameter     =  transport-param / user-param /
                     method-param / ttl-param / maddr-param /
                     lr-param / target-param / other-param

target-param      =  "target=" pvalue
```

9. PSTN Mapping

The mapping to PSTN protocol is important both for gateways that connect the IP network to existing TDM equipment, such as PBX's and voicemail systems, and for gateways that connect the IP network to the PSTN network. Both ISDN and ISUP have signaling for this information that can be treated as roughly equivalent for the purposes here.

The user portion of the URI SHOULD be used as the address of the voicemail system on the PSTN, while the target SHOULD be mapped to the original redirecting party on the PSTN side.

If the gateway and Proxy are in the same Trust Domain (defined in RFC 3325 [5]) and the Spec(T) includes compliance with this document and the Spec(T) asserts that the Proxy will do screening (whatever that means), then the gateway MAY claim it is screened; otherwise it SHOULD NOT assert that the diversion information is screened.

This draft says nothing about what to put into the redirecting numbers, as that has billing implications outside the scope of this work. The requirements here will work fine if the redirecting number is not set on the PSTN side. It is not recommended that the GW map the target information into the redirecting party information, but doing so is not in violation of this document.

The following SHOULD be used as the mapping between reason parameters and ISUP/ISDN redirect reason codes:

ISUP or ISDN	PSTN Reason	SIP Reason Parameter
0000	Unknown	300
0001	Call forwarding busy or called DTE busy	486
0010	Call forwarding no reply	408
1111	Call forwarding unconditional or systematic call redirection	302
1010	Call deflection or call forwarding by the called DTE	487
1001	Call forwarding DTE out of order	410

The redirection counters SHOULD be set to one unless additional information is available.

10. IANA Considerations

This document adds a new value to the IANA registration in the sub-registry at http://www.iana.org/assignments/sip-parameters as defined in [6].

Parameter Name	Reference
target	RFC XXXX

Note to RFC Editor - replace XXXX with the RFC number of this document.

11. Security Considerations

This draft inherently discusses transactions involving at least 3 parties. This makes the privacy issues somewhat more complex.

The new URI parameters defined in this draft are generally sent from a Proxy or call control system to a unified messaging (UM) system or gateway to the PSTN, and then to a voicemail system. This tells the UM what service the proxy wishes to have performed. Just as any message sent from the proxy to the UM needs to be integrity protected, these need to be integrity protected. This stops attackers from doing things like causing a voicemail meant for the CEO of the company to go to an attacker's mailbox. RFC 3261 provides TLS and IPSEC mechanisms suitable for protecting against this.

The signaling from the Proxy to the UM will reveal who is calling whom and possibly some information about the presence of a user based

on whether a call got sent to voicemail instead of being answered. This information can be protected by encrypting the SIP traffic between the Proxy and UM. Again, RFC 3261 contains mechanisms for accomplishing this using TLS and IPSEC.

The S/MIME based mechanisms in RFC 3261 will generally not be applicable for protecting this information because they are meant for end to end issues and this is primarily a middle to end scenario. Ongoing work on middle to end [11] may allow S/MIME based schemes to be used for protecting this information. These schemes would allow the information to be hidden and integrity protected if there was another administrative domain between the Proxy and UM. The current scheme is based on hop by hop security and requires all hops between the Proxy and UM to be trusted, which is the case in many deployment scenarios.

11.1 Integrity Protection of Forwarding in SIP

Forwarding of a call in SIP brings up a very strange trust issue. Consider the normal case of when A calls B, and then the call gets forwarded by a network element in the domain of B to C, and then C answers the call. A called B but ended up talking to C. This may be hard to separate from a man in the middle attack.

There are two possible solutions for this. One is that B sends back information to A saying don't call me, call C and signs it as B. The problem with this is that it reveals the fact that B has forwarded to C and often B does not want to do this. For example, B may be a work phone that has been forwarded to a mobile or home phone. The user does not want to reveal their mobile or home phone number but, even more importantly, does not want to reveal that they are not in the office but are instead working from home.

The other possible solution for this is that A needs to trust B only to forward to a trusted identity. This requires a hop by hop transitive trust such that each hop will only send to a trusted next hop and each hop will only do things that the user at that hop desired. This solution is enforced in SIP using the SIPS URI and TLS based hop by hop security. It protects from an off axis attack but if one of the hops is not trustworthy, the call may be subverted to an attacker.

Any redirection of a call to an attacker's mailbox is a very serious issue. It is trivial for the attacker to make the mailbox seem very much like the real mailbox and forward the message to the real mailbox so that the fact that the messages have been intercepted or even tampered with is not detected.

11.2 Privacy Related Issues on the Second Call Leg

When A calls B and gets redirected to C, occasionally people say there is a requirement for the call leg from B to C to be anonymous. This is not the PSTN: there is no call leg from B to C; instead there is a VoIP session between A and C. If A had put a To header containing B in the initial invite message, unless something special is done about it, C will see that To header. If the person who answers phone C says "I think you dialed the wrong number, who were you trying to reach?" A will probably specify B.

If A does not want C to see that the call was to B, A needs a special relationship with the Proxy that does the forwarding so that it will not reveal that information, and the call should go through an anonymizer service that provides session or user level privacy (as described in RFC 3323 [4]) service before going to C. It's not hard to figure out how to meet this requirement, but it is difficult to figure out why anyone would want this service.

If B wants to make sure that C does not see that the call was to B, it is easier but a bit weird. The usual argument is Bill wants to forward his phone to Monica but does not want Monica to find out his phone number. It is a little weird that Monica would want to accept all Bill's calls without knowing how to call Bill to complain. The only person Monica will be able to complain to is Hillary who tried to call Bill. Several popular web portals will send SMS alert message about things like stock prices and weather to mobile phone users today. Some of these contain no information about the account on the web portal that imitated them, making it nearly impossible for the mobile phone owner to stop them. This anonymous message forwarding has turned out to be a really bad idea even where no malice was intended. Clearly some people are fairly dubious about the need for this, but never mind: let's look at how it is solved.

In the general case, the proxy needs to route the call through an Anonymization Service and everything will be cleaned up. Any Anonymization service that performs the "Privacy: Header" Service in RFC 3323 [5] MUST remove the reason and target URI parameters from the URI. RFC 3325 already makes it pretty clear you would need to clean up this sort of information.

There is a specialized case of some interest where the mechanism in this document is being used in conjunction with RFC 3325 and the UM and the Proxy are both in the trust domain. It this limited case, the problem that B does not want reveal their address to C can be solved by ensuring that the target parameter URI should never be in a message that is forwarded outside the trust domain. If it is passed to a PSTN device in the trust domain, the appropriate privacy flag

needs to be set in the ISUP or ISDN signaling.

12. Changes from 00 Version

The reason information was moved from being a tag in the URI to using the Reason header.

13. Acknowledgements

Mary Barnes, Dean Willis, and Steve Levy have spent significant time with me helping me understand the requirements and pros and cons of various approaches. I would like to thank them very much for this, and since this is an acknowledgements section I would also like to acknowledge Rohan Mahy's help with the various documents on this subject and his encouragement to work on a solution that brings some consensus to this topic.

Normative References

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

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

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

[4] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[5] Jennings, C., Peterson, J. and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002.

[6] Camarillo, G., "The Internet Assigned Number Authority Universal Resource Identifier Parameter Registry for the Session Initiation Protocol", draft-ietf-sip-uri-parameter-reg-01 (work in progress), November 2003.

[7] Schulzrinne, H., Oran, D. and G. Camarillo, "The Reason Header Field for the Session Initiation Protocol (SIP)", RFC 3326, December 2002.

Informative References

[8] Burger, E., "Basic Network Media Services with SIP",

draft-burger-sipping-netann-07 (work in progress), September 2003.

[9] Barnes, M., "SIP Generic Request History Capability Requirements", draft-ietf-sipping-req-history-04 (work in progress), June 2003.

[10] Barnes, M., "An Extension to the Session Initiation Protocol for Request History Information", draft-ietf-sip-history-info-01 (work in progress), October 2003.

[11] Ono, K. and S. Tachimoto, "End-to-middle security in the Session Initiation Protocol(SIP)", draft-ono-sipping-end2middle-security-00 (work in progress), June 2003.

Author's Address

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

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

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

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

SIPPING WG
Internet-Draft
Expires: August 7, 2004

C. Jennings
Cisco Systems, Inc.
February 7, 2004

Instance Identifiers for SIP User Agents
draft-jennings-sipping-instance-id-00.txt

Status of this Memo

Copyright Notice

Abstract

There are places in building SIP [2] based communications systems where it is useful to have a stable identifier for particular user agents that are used for user communications. This draft defines a convention for names that can be used to satisfy these needs.

Table of Contents

1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [3].

2. Introduction and Use Cases

There are a few cases in which it is convenient to be able to identify instances of a user agent. Some examples are described. They all require the name to be stable across reboots of the device.

In the config framework[4], a user agent sends a subscribe to fetch its configuration. It needs to get the same configuration each time.

A particular user, Alice, has several user agents that all register as Alice. A registrar wishes to report which user agent are currently registered to a network management system. For this reporting to make sense, each of Alice's user agents must have a stable name.

A system that is using the dialog package to monitor a particular user agent would like to be able to assign an alias like "My Office Phone" for display purposes to that particular user agent.

When several presence user agents are providing presence data, it must be possible to correlate a particular set of data with the particular device that provided it.

In all these cases, the user agent could be a software program running on a computer with more than one user.

3. Requirements

The identifier needs to be unique.

Identifiers are needed for user agents that are in dedicated pieces of hardware such as IP phones.

Identifiers are needed for software user agents running on multi-user computers.

In some of the cases with IP phones, it is desirable for this same identifier to be recorded as a bar code on the outside of the box that the IP phone comes in.

4. Solution

User agents that follow the convention of this document MUST put a unique identifier in a new tag, called "instance", of the Contact header when sending a SIP request. They MAY omit this for a particular sequence of SIP messages if the user has requested it be removed for privacy reasons.

The unique identifier has no real semantic information other than uniqueness. In cases in which the user agent runs on a single computer and this is the only user agent on that computer, the MAC address of the primary network card is the preferred identifier. In cases in which it is impossible to use the MAC address, then when the user agent is first run, it should generate a random 64 bit number and use this as the identifier. It MUST store this number in some non volatile storage that is stable over reboots and power outages. The user agent SHOULD use the same instance identifier tag even if it is registering different AOR or contacts.

If the identifier is a MAC address, it MUST be formatted as the letters "MAC-" followed by a 12 digit hexadecimal representation of the MAC address. The address can not include ":", whitespace, or other formatting. If the identifier is a random number, it MUST be formatted as the letters "RANID-" followed by a 16 digit hexadecimal representation of the number. Note that the identifiers are case sensitive and all alpha characters are upper case.

The MAC and RANDID identify the namespace for the unique identifier. In the future this unique identifer namespace may be extended with other namespaces that use unique identifiers from things like USB, Bluetooth, or Firewire.

These same identifiers may be used in the user portion of request URIs when that is appropriate. A SUBSCRIBE for configuration information is a good example.

5. Discussion

The contact header in a SIP request identifies an address that can be used to reach the device that is sending the request. This address may change each time the device running the user agent gets a new IP address, but it is very reasonable for the display name to give a unique identifier for what this instance of the user agent wishes to be known by. Right now SIP does not give any recommendation on what to place in the field. This document suggests a naming convention for this.

MAC addresses are usually put on the outside of the box for IP phones

in a form that humans can read and also by a barcode scanner.

6. BNF

The following ABNF follows the rules in RFC-2234 [1] and updates the BNF in RFC 3261.

```
contact-params = c-p-q / c-p-expires / c-p-instance
                 / contact-extentions
c-p-intance = "instance" EQUAL uniq-ident
UHEX  =  DIGIT / %x41-46 ;uppercase A-F
MAC  =  %x4d.41.43 ; MAC in caps
RANDID  = %x52.41.4e.44.49.44 ; RANDID in caps
uniq-ident = ( mac-ident / rand-ident )
mac-ident = MAC "-" 12UHEX
rand-ident = RANDID "-" 16UHEX
```

7. Example

The following are some valid Contact headers:

```
Contact: <sip:alice@host22.example.com>;instance=MAC-123456789ABC
Contact: <sip:alice@host22.example.com>;instance=
         RANDID-0123456789ABCDEF
```

8. Security Consideration

The unique identifer reveals further privacy related information to other people that see the SIP signalling. Currently user agents put an IP address or DNS name in the contact header, so the amount of extra information this reveals is very minimal. The MAC address may reveal the manufacturer of the user agent.

9. Open Issues

Would this be better in an "Instance-ID" header?

Would this be better in the User-Agent header? Some systems are doing already doing this.

Is 64 bits the right size for the random identifier?

Is requiring upper case appropriate?

10. Acknowledgments

Many thank for the useful comments and improvements from Louis Pratt, Steve Levy, Rohan Mahy, and Randy Baird as well as the list discussion from Jonathan Rosenberg.

Normative References

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

[2] 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.

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

Informative References

[4] Petrie, D., "A Framework for SIP User Agent Configuration", draft-ietf-sipping-config-framework-00 (work in progress), March 2003.

Author's Address

Cullen Jennings
Cisco Systems, Inc.
170 West Tasman Dr.
MS: SJC-21/2
San Jose, CA 95134
USA

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

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

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

SIP WG H. Khartabil
Internet-Draft Nokia
Expires: August 4, 2004 February 4, 2004

Analysis: HTTP Authentication for SIP draft-khartabil-sip-auth-analysis-00

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) provides a stateless, challenge-based mechanism for authentication that is based on authentication in HTTP. RFC 3261 fails to indicate the behaviour of SIP intermediaries and User Agents in certain scenarios. This documents presents such scenarios, analysis the currently available text, and where possible, offers a solution.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] provides a stateless, challenge-based mechanism for authentication that is based on authentication in HTTP. RFC 3261 fails to indicate the behaviour of SIP intermediaries and User Agents in certain scenarios. This documents presents such scenarios, analysis the currently available text, and where possible, offers a solution.

One of the main issues presented is proxies with the same realm appearing in a chain.

2. Use of 401 and 407 Responses, www-authenticate and proxy-authenticate headers

Section 22.1 of RFC 3261 says "In SIP, a UAS uses the 401 (Unauthorized) response to challenge the identity of a UAC. Additionally, registrars and redirect servers MAY make use of 401 (Unauthorized) responses for authentication, but proxies MUST NOT, and instead MAY use the 407 (Proxy Authentication Required) response".

The use of 401 and 407 must be more normative. Text should read: A UAS that require authentication MUST use 401 in a response and MUST challenge with a www-authenticate header. Registrars and redirect servers MUST use 401 and www-authenticate header. Proxies MUST use 407 and proxy-authenticate header.

3. Rendering to user

Section 22.1 of RFC 3261 says: "When a UAC receives a challenge, it SHOULD render to the user the contents of the "realm" parameter in the challenge (which appears in either a WWW-Authenticate header field or Proxy-Authenticate header field) if the UAC device does not already know of a credential for the realm in question".

This is talking about a terminal being pre-configured with credentials. Some implementation may choose to render to the user the challenge if a previously sent response to a challenge failed due to incorrect credentials, regardless if the credentials were pre configured or not. The text following the quote above gives the impression that this is not possible.

"A service provider that pre-configures UAs with credentials for its realm should be aware that users will not have the opportunity to present their own credentials for this realm when challenged at a pre-configured device".

4. Rejected or not

Section 22.1 of RFC 3261 says: "A UAC MUST NOT re-attempt requests with the credentials that have just been rejected (though the request may be retried if the nonce was stale)".

The problem with this is that how do you know if the request has been rejected or it is just the next hop proxy challenging with the same realm?

5. Use of To-header vs. Remote URI (Request-URI)

Section 22.2 of RFC 3261 says: "The UAC may require input from the originating user before proceeding. Once authentication credentials have been supplied (either directly by the user, or discovered in an internal key-ring), UAs SHOULD cache the credentials for a given value of the To header field and "realm" and attempt to re-use these values on the next request for that destination".

It is more appropriate to cache request-URI, and not rely on what is in the To-header.

6. Caching of User-to-User Authentication Credentials

Section 22.2 of RFC 3261 says: "UAs MAY cache credentials in any way they would like".

This means within dialogs as well as across dialogs. This is talking about www-authenticate challenges.

7. Caching outbound proxy credentials

Section 22.3 of RFC 3261 says: "if a UA is configured with the realm of its local outbound proxy, when one exists, then the UA MAY cache credentials for that realm across dialogs".

It cannot be assumed that the domain name of a configured outbound proxy is its realm. Therefore a terminal configured with only the outbound proxy URI cannot and MUST NOT cache credentials or any proxy challenge across dialogs.

This introduces the problem of multiple proxies in a chain challenging with the same realm. How does a UAC know that the challenge was from the outbound proxy and not from a downstream proxy with the same realm as the outbound proxy? Can we mandate that the outbound proxy must have a unique realm?

8. User Cancelling when challenged

Section 22.3 of RFC 3261 says: "If a request is forked (as described in Section 16.7), various proxy servers and/or UAs may wish to challenge the UAC ... .If the UAC does not provide credentials for each challenge, the proxy servers that issued the challenges will not forward requests to the UA".

It is probably worth noting here that if the user enters his credentials for at least one of the challenges and not the rest (cancels the rest), the UAC must re-issue a new SIP request with a response to the challenges that have not been cancelled. If all challenges have not been responded to by the user or terminal, then the UAC does not re-issue the SIP request and simply return a 403 to the user.

9. The realm Issue

Section 22.3 of RFC 3261 says: "When multiple proxies are used in a chain, a Proxy-Authorization header field value MUST NOT be consumed by any proxy whose realm does not match the "realm" parameter specified in that value".

Section 22.3 of RFC 3261 also says: "It is possible for multiple challenges associated with the same realm to appear in the same 401 (Unauthorized) or 407 (Proxy Authentication Required). This can occur, for example, when multiple proxies within the same administrative domain, which use a common realm, are reached by a forking request."

The problem here is with forking: A proxy will fork the second request with the credentials. How does a proxy that the request has been forked to (or a UAS) know, by just examining the realm, that this is a digest response to a challenge it issued? The solution here needs to specify that a proxy examines all digest response matching the realm as well as the nonce.

Another question: Is it possible that multiple proxies with the same realm are placed in a chain? I.e. the first proxy challenges, user provide correct authentication. That proxy authenticates and forwards the request to a down stream proxy. That down stream proxy has the same realm as first proxy. There are a few issues here:

- A proxy MUST consume a proxy-authorization header intended for it. A proxy knows that by examining the realm, as quoted above. But, if proxies with the same realm are placed in a chain, a Proxy needs to examine the nonce and make sure it created it before it consumes the header.

- If a proxy MAY (in comparison to MUST) consume the header, how does the second proxy know that the credentials in the request are not for it? It needs to examine the nonce and find out it is not from it. Is that ok?

Consuming the header does not solve the forking problem. Therefore examining the nonce is the only solution.

Another issue is at the UAC side. If 2 proxies are in a chain and share a realm, how does the UAC know, when the second proxy challenges, that it is not the first proxy re-challenging because the credentials provided to it were wrong? There are a couple of things that can be done here:

- mandate that the proxy places stale=false when it is re-challenging due to wrong credentials. This means stale=false is different that stale not present at all. The UAC replaces the provided proxy-authorization header with a new one.

- mandate that the proxy does not change the nonce when it is re-challenging due to wrong credentials. The UAC replaces the provided proxy-authorization header with a new one.

- Note: a UAS does not need to do the above since the UAC knows that if a re-challenge occurs and stale is not true, then new credentials need to be provided. This works also for a proxy forking to multiple UASs with the same realm.

10. Authentication-Info

RFC 3261 allows the usage of Authentication-Info header. The BNF in RFC 3261 allows multiple authentication-info headers where RFC 2617 allows only one. Is it only the terminating UAS that is allowed to insert this header. If so, why allow multiples to be present. If not, how does the UAC know which proxy added this header? It cannot know since there is no parameter indicating so, not even nonce.

11. Acknowledgements

The author would like to thank Pekka Pessi for his feedback.

References

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

Author's Address

Hisham Khartabil
Nokia
P.O. Box 321
Helsinki
Finland

Phone: +358 7180 76161
EMail: hisham.khartabil@nokia.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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

SIMPLE WG H. Khartabil
Internet-Draft A. Niemi
Expires: August 5, 2004 Nokia
February 5, 2004

Conveying a Session Policy Uniform Resource Identifier (URI) in the Session Initiation Protocol (SIP)

draft-khartabil-sip-policy-uri-call-info-purpose-00

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) defines the Call-Info header. This header field provides additional information about the originator or recipient of the request. Information in the Call-Info is generally a URI, and the exact purpose of the URI is described by the "purpose" parameter. This document introduces a new purpose parameter value of "policy-uri" that can be used by a request originator or recipient to convey a URI where a certain policy pertaining to a session can be accessed. For example, "policy-uri" can be used by a conference server to indicate the URI of the conference policy to a UA participating in that conference.

This document also sets up an IANA registry for the "purpose" parameter in the Call-info header of the Session Initiation Protocol.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] defines the Call-Info header. The purpose of the Call-Info header field is to provide additional information about the caller or callee, depending on whether the header is used in a request or a response. This information consists of a URI, and the purpose of the URI is described by the "purpose" parameter. Some of the already defined purposes include:

- o icon; for providing an image suitable for an iconic representation of the caller or callee
- o info; for providing ageneral description of the caller or callee, e.g., in the form of a web page
- o card; for providing contact information, e.g., in the form of a business card

In addition to these purpose values, additional purpose values can be registered with IANA.

This document introduces a new purpose parameter value of "policy-uri" that is used by a caller or a callee to convey a URI where a certain policy pertaining to the session can be accessed. The main use case for the "policy-uri" involves a a conference server indicating the URI of the conference policy [2] to a participant UA. This UA can then use the Call-Info URI to see and modify the conference policy.

It was also discovered that there was no IANA registry set up for the "purpose" parameter in the Call-info header of the Session Initiation Protocol [1]. This document sets up the IANA registry.

2. policy-uri Description

In a conferencing scenario, it was identified that some participants may not want to subscribe to the conference state event package [3], but would still like to learn the conference policy URI [2]. In this case, it is suggested that the Call-info header carries such information.

In order to do that, a new Call-info purpose needs to be created that indicates that the call-info header carries a policy URI pointing to an address where the policy can be fetched. This policy URI is not restricted to conference policy URI, but any policy related to a session.

3. IANA Registration

This specification establishes the call-info "purpose" parameter "call-info-purpose" sub-registry under http://www.iana.org/assignments/sip-parameters and initiates its population with the values listed in Section 20.9 of [1]. Additional call-info-purpose are registered by RFC publication.

This document also registers one new "call-info-purpose" value: policy-uri.

"policy-uri" is described in Section 2.

Short descriptions suitable for the IANA registry is: a URI that informs the recipient of the location of a policy related to the session being initiated.

4. Acknowledgements

The authors would like to thank Jonathan Rosengerg for pointing out that the IANA registry is missing and for suggesting that in can be included in this document.

References

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

[2] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-01.txt, October 2003.

[3] Rosenberg, J., Shulzrinne, H. and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State", draft-ietf-sipping-conference-package-02, October 2003.

Authors' Addresses

Hisham Khartabil
Nokia
P.O. Box 321
Helsinki
Finland

Phone: +358 7180 76161
EMail: hisham.khartabil@nokia.com

Aki Niemi
Nokia
P.O. Box 321
Helsinki
Finland

Phone: +358 50 389 1644
EMail: aki.niemi@nokia.com

Intellectual Property Statement

Full Copyright Statement

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgment

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

Network Working Group R. Sparks
Internet-Draft dynamicsoft
Expires: August 6, 2004 February 6, 2004

Actions addressing identified issues with the Session Initiation Protocol's non-INVITE Transaction
draft-sparks-sip-nit-actions-00

Status of this Memo

Copyright Notice

Abstract

This draft describes modifications to the Session Initiation Protocol (SIP) to address problems that have been identified with the SIP non-INVITE transaction. These modifications reduce the probability of messages losing the race condition inherent in the non-INVITE transaction and reduce useless network traffic. They also improve the robustness of SIP networks when elements stop responding. These changes update behavior defined in RFCs 3261 and 3263.

Table of Contents

1. Introduction

There are a number of unpleasant edge conditions created by the SIP non-INVITE transaction model's fixed duration. The negative aspects of some of these are exacerbated by the effect provisional responses have on the non-INVITE transaction state machines. These problems are documented in [3]. In summary:

A non-INVITE transaction must complete immediately or risk losing a race

Losing the race will cause the requester to stop sending traffic to the responder (the responder will be temporarily blacklisted)

Provisional responses can delay recovery from lost final responses

The 408 response is useless for the non-INVITE transaction

As non-INVITE transactions through N proxies time-out, there can be an O(N^2) storm of the useless 408 responses

This draft specifies updates to RFC 3261 [1] and RFC 3263 [2] to improve the behavior of SIP elements when these edge conditions arise.

2. Improving the situation when responses are only delayed

There are two goals to achieve when we constrain the problem to those cases where all elements are ultimately responsive and networks ultimately deliver messages:

- Reduce the probability of losing the race, preferably to the point that it is negligible
- Reduce or eliminate useless messaging

2.1 Action 1: Make the best use of provisional responses

- Disallow non-100 provisionals to non-INVITE requests
- Disallow 100 Trying to non-INVITE requests before Timer E reaches T2 (for UDP hops)
- Allow 100 Trying after Timer E reaches T2 (for UDP hops)
- Allow 100 Trying for hops over reliable transports

Since non-INVITE transactions must complete rapidly ([3]), any information beyond "I'm here" (which can be provided by a 100 Trying) can be just as usefully delayed to the final response. Sending non-100 provisionals wastes bandwidth.

As shown in [3], sending any provisional response inside a NIT before Timer E reaches T2 damages recovery from failure of an unreliable transport.

Without a provisional, a late final response is the same as no response at all and will likely result in blacklisting the late responding element ([3]), If an element is delaying its final response at all, sending a 100 Trying after Timer E reaches T2 prevents this blacklisting without damaging recovery from unreliable transport failure.

Blacklisting on a late response occurs even over reliable transports. Thus, if an element processing a request received over a reliable transport is delaying its final response at all, sending a 100 Trying well in advance of the timeout will prevent blacklisting. Sending a 100 Trying immediately will not harm the transaction as it would over UDP, but a policy of always sending such a message results in unneccessary traffic. A policy of sending a 100 Trying after the period of time in which Timer E reaches T2 had this been a UDP hop is one reasonable compromise.

2.2 Action 2: Remove the useless late-response storm

- Disallow 408 to non-INVITE requests
- Absorb stray non-INVITE responses at proxies

A 408 to non-INVITE will always arrive too late to be useful ([3]), The client already has full knowledge of the timeout. The only information this message would convey is whether or not the server believed the transaction timed out. However, with the current design of the NIT, a client can't do anything with this knowledge. Thus the 408 simply wasting network resources and contributes to the response bombardment illustrated in [3].

Late non-INVITE responses by definition arrive after the client transaction's Timer F has fired and the client transaction has entered the Terminated state. Thus, these responses cannot be distinguished from strays. Changing the protocol behavior to prohibit forwarding non-INVITE stray responses stops the late response storm. It also improves the proxy's defenses against malicious users counting on the RFC 3261 requirement to forward such strays.

3. Improving the situation when an element is not going to respond

When we expand the scope of the problem to also deal with element or network failure, we have more goals to achieve:

- Identifying when an element is non-responsive
- Minimizing or eliminating falsely identifying responsive elements as non-responsive
- Avoiding non-responsive elements with future requests

Action 1 dramatically improves an elements ability to distinguish between failure and delayed response from the next downstream element. Ssome response, either provisional or final, will almost certainly be received before the transaction times out. So, an element can more safely assume that no response at all indicates the peer is not available and follow the existing requirements in [1] and [2] (as amended by this memo) for that case.

As [3] discusses, behavior once an element is identified as non-responsive is currently underspecified. [2] speaks only non-normatively about caching the addresses of servers that have successfully been communicated with for an unspecified period of time.

3.1 Action 3: Strengthen specification of caching success and failures in RFC 3263

- Make the caching recommendation normative for servers successfully reached
- Add failures due to non-responsiveness to that cache

This cache will also be used to remember servers that have issued a 503 with or without a Retry-After.

4. Normative Updates to RFC 3261

4.1 Action 1

A SIP element MUST NOT send any provisional response with a Status-Code other than 100 to a non-INVITE request.

A SIP element MUST NOT respond to a request with a Status-Code of 100 over any unreliable transport, such as UDP, before the amount of time it takes a client transaction's Timer E to be reset to T2.

A SIP element MAY respond to a request with a Status-Code of 100 over an unreliable transport after the amount of time it takes a client transaction's Timer E to be reset to T2.

A SIP element MAY respond to a request with a Status-Code of 100 over a reliable transport at any time.

4.2 Action 2

A transaction-stateful SIP element MUST NOT send a response with Status-Code of 408 to a non-INVITE request. As a consequence, an element that can not respond before the transaction expires will not send a final response at all.

A transaction-stateful SIP proxy MUST NOT send any response to a non-INVITE request unless it has a matching server transaction that is not in the Terminated state. As a consequence, this proxy will not forward any "late" non-INVITE response.

5. Normative Updates to RFC 3263

5.1 Action 3

(Note that RFC 3263 uses "client" for "any SIP element wishing to send a request".)

Once a client identifies an available server for a domain name using the algorithms defined in RFC 3263, it SHOULD cache the identity of that server in an available-cache. This identity MUST be periodically removed from the cache, and its time-to-live in that cache SHOULD be short. If the server with that identity becomes unavailable, the identity MUST be immediately removed from the cache and SHOULD be placed in an unavailable-cache. The next attempt to reach that domain name MUST invoke the algorithms in RFC 3263.

If any attempt to contact a server based on the output of the algorithms of RFC 3263 yeilds that the server is unavailable (the request times out or the server returns a 503 Status-Code), the identity of that server SHOULD be placed in an unavailable-cache. This identity MUST be periodically removed from that cache, and its time-to-live in that cache SHOULD be short. If information about the period of unavailability is present (such as in a Retry-After header field in a 503 response), the time-to-live in this cache SHOULD reflect that information.

If the algorithms of RFC 3263 yeild a server identity that is in an unavailable-cache, that identity MUST be discarded and the algorithm MUST be continued to search for another candidate.

OPEN ISSUE: Can we strengthen placing identities in an unavailable-cache to MUST? RFC 3263 failover for non-INVITE will not work without it.

OPEN ISSUE: Is it possible to recommend a time more specific than "short" in these requirements?

References

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

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

[3] Sparks, R., "Problems identified associated with the Session Initiation Protocol's non-INVITE Transaction", draft-sparks-sip-nit-problems (work in progress), February 2004.

Author's Address

Robert J. Sparks
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, TX 75024

EMail: rsparks@dynamicsoft.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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

Network Working Group R. Sparks
Internet-Draft dynamicsoft
Expires: August 6, 2004 February 6, 2004

Future work addressing issues with the Session Initiation Protocol's non-INVITE Transaction

draft-sparks-sip-nit-future-00

Status of this Memo

Copyright Notice

Abstract

This draft explores several possible remedies for a set of issues that have been identified with the Session Initiation Protocol's non-INVITE transaction. These proposed solutions are in rough form and have not yet accumulated working group consensus. They range from minor extensions to protocol behavior to fundamentally changing the non-INVITE transaction model.

Table of Contents

1. Introduction

There are a number of unpleasant edge conditions created by the SIP non-INVITE transaction model's fixed duration. The negative aspects of some of these are exacerbated by the effect provisional responses have on the non-INVITE transaction state machines. These problems are documented in [3]. In summary:

A non-INVITE transaction must complete immediately or risk losing a race

Losing the race will cause the requester to stop sending traffic to the responder (the responder will be temporarily blacklisted)

Provisional responses can delay recovery from lost final responses

The 408 response is useless for the non-INVITE transaction

As non-INVITE transactions through N proxies time-out, there can be an O(N^2) storm of the useless 408 responses

Consensus is forming around serveral solutions to many of these problems. Those solutions are documented in [4]. This draft explores a set of solutions to address the remaining problems. The solutions are broken into two alternatives. Alternative A focuses on incremental repair to the existing non-INVITE transaction model. Alternative B proposes changing the model.

2. Alternative A: Improving the situation with a fixed NIT duration

2.1 Improving the situation when responses are only delayed

2.1.1 Improve a UAS's knowledge of how much time it has to respond

Consider the race lost in [3]. The UAS could win this race if it responded soon enough for its 200 to reach the UAC before the UAC timed out. Unfortunately, there is no way, given the current specifications, for the UAC to know how much time it really has left. It might make a rough guess at the propagation time due to network transmission by counting Via header field values and assuming each hop took at most T1, but it has no idea at all what the propagation delay through each of the proxies was.

The UAS's situation could be dramatically improved if the next upstream element explicitly indicated how much time was left. Each element would assume a network delay for any message of T1, and estimate the sum of its own internal propagation delay for both the request and the final response, resulting in the messaging shown in

Figure 1 (which for compactness assumes T1=500ms at each hop). Assume the internal delay introduced by P1, P2, and P3 is 1.5s, 3s, and 0.5s respectively. P1 advertises a timeleft of 32 - 1.5 - 2*T1 = 29.5. P2 advertises a timeleft of 29.5 - 3 - 2*T1 = 25.5. P3 advertises 25.5 - 0.5 - 2*T1 = 24

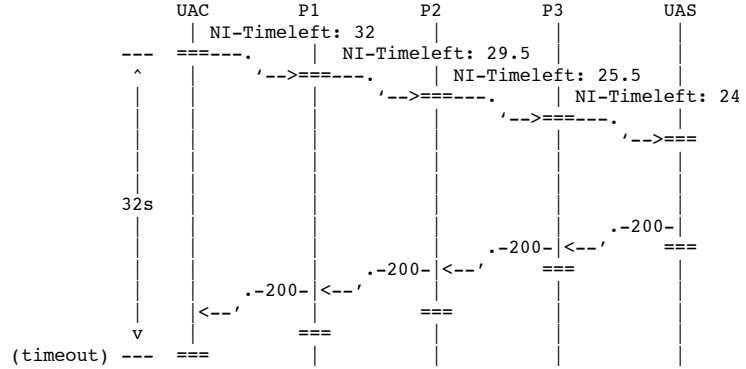


Figure 1: Explicitly indicating timeleft

Note that each element determines how much time was and will be lost to network propagation delay over the first upstream hop in incorporates that into its calculation. The UAS will need to do this as well, so in our example above, it knows that it only has 23 seconds to respond.

The estimate of timeleft can be improved if an element has better knowledge of the real network propagation delay. The element can measure its internal propagation delay for the request, but will have to estimate the propagation delay for the response.

To improve behavior in the presence of existing elements that will not supply a timeleft indication, an element that receives a non-INVITE request without the indication could behave as if it had received value of

```
    64*T1 - (2*T1 + IPD)*(n_Via-1)
```

where

IPD = estimate of internal processing delay of a

request and a response (strawman: 1s)
n_Via = number of Via header field values in the request

2.1.2 When an application needs more time

Application designers are faced with significant challenges when the semantics of processing a request require more time (human intervention for example) than the non-INVITE transaction allows. SIP Events ([2]) deals with this by spreading the semantics of processing a new subscription request across two or more non-INVITE requests - a SUBSCRIBE and subsequent NOTIFYs. For example, if a server receives a request for a subscription that cannot be granted or refused until a human provides input, the SUBSCRIBE request will be accepted with a 202 Accepted. A subsequent NOTIFY will convey whether or not the subscription has been allowed or denied.

An alternate approach is to allow a server to tell a client "I can't do this right now, but try again in a little while".

2.1.2.1 Specify try again later behavior

When a server discovers it needs more time than the current non-INVITE transaction will allow to finish the work needed to process the request, it could return a 302 response with:

- A contact pointing to itself with NO expiration time so that this value cannot be cached.
- A Retry-After header indicating when the client should try the request again

A client receiving this response SHOULD retry the request at the indicated time. A server MUST NOT apply the results of the request until the client successfully retries the request. (This limits the set of problems this tool can be used with to those whose side effects can be undone.) A client can effectively CANCEL a request by not coming back.

There are several issues that would need to be resolved if this approach is pursued:

- [1] forbids emitting a 302 with a contact equal to the Request-URI, so the "contact point to self" above would have to change each time (with respect to URI equality) such that the request still arrived at the same agent (requiring a GRUU).
- Emitting and handling 300-class responses for requests inside a dialog is not well-specified in [1]. It is unlikely that existing implementations would exhibit interoperable behavior if they encountered them.
- Proxies would need to know to not recurse on this kind of 302 response. This might require an explicitly signaled extension, or indicate that a 4xx or 5xx class response is more appropriate.

2.1.3 Improving the situation when an element is not going to respond

The mechnanism described in Section 2.1.1 gives a proxy the information it needs to respond in time to avoid the proxy doom problem described in [3].

3. Alternative B: Allowing NITs to pend

The root causes of the problems described in [3] is the fixed-length non-INVITE transaction and the extra mechanics for providing reliability over unreliable transports.

3.1 Allow the non-INVITE transaction to pend indefinitely

We could change the definition of the non-INVITE transaction to allow it to pend indefinitely by removing Timer F. By doing so,

- the race condition goes away
- the 408 response would become meaningful once again
- the late response blacklisting problem disappears
- the 408 bombardment problem disappears
- the proxy doom problem is eliminated

Clients would use CANCEL to pending non-INVITEs to stimulate a final response when they are through waiting, similar to INVITE. Proxies will be spared the doom described in [3] since they can force branches to complete with CANCEL before sending a final response.

Responsibility for reliability over UDP would remain with the requester. This means that provisional responses will still not squelch request retransmission. A long pending non-INVITE request would be retransmitted once 4 seconds (for the default value of T2) once timer E reaches T2, but only over UDP. This might be mitigated by replacing T2 with another, larger, configurable value for use with the non-INVITE transaction.

The primary disadvantage of this approach is that it raises the expense for handling non-INVITE transactions at proxies to the same level as INVITE transactions. Proxies will have to maintain state for NITs longer than they currently do. Proxies will need a way to end the transaction. We can give them this by duplicating INVITE behavior: create a timer analogous to Timer C. When it fires, send CANCELs down any outstanding branches and once they complete, send a 408 (assuming no branch returned a better final response) to the requester.

This change is backwards-safe, if not completely backwards compatible:

- Existing client, proposed server: The client's experience is unchanged. It will still abandon the transaction after Timer F fires. The failure scenarios are exactly those we currently have. The server will need to protect itself against never receiving a CANCEL (with an analog to Timer C).

- Proposed client, existing server: The behavior here is an improvement over the existing client-server behavior. The 408 emitted by an existing server would become meaningful to the proposed client. New methods that take advantage of the pending property will be rejected by the existing server with a 501. Existing servers might not be expecting CANCEL to non-INVITEs, but are not compliant to the existing specification if such a CANCEL induces incorrect behavior. We would need to add a constraint, similar to that already on the INVITE transaction, binding clients that receive no response within a short time to abandon the transaction instead of pending indefinitely to account for server failure.

4. Acknowledgments

This document attempts to capture many conversations about non-INVITE issues. Significant contributers include Ben Campbell, Gonzalo Camarillo, Steve Donovan, Rohan Mahy, Dan Petrie, Adam Roach, Jonathan Rosenberg, and Dean Willis.

References

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

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] Sparks, R., "Problems identified associated with the Session Initiation Protocol's non-INVITE Transaction", draft-sparks-sip-nit-problems (work in progress), February 2004.

[4] Sparks, R., "Actions addressing identified issues with the Session Initiation Protocol's non-INVITE transaction", draft-sparks-sip-nit-actions (work in progress), February 2004.

Author's Address

Robert J. Sparks
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, TX 75024

EMail: rsparks@dynamicsoft.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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

Network Working Group R. Sparks
Internet-Draft dynamicsoft
Expires: August 6, 2004 February 6, 2004

Problems identified associated with the Session Initiation Protocol's non-INVITE Transaction

draft-sparks-sip-nit-problems-00

Status of this Memo

Copyright Notice

Abstract

This draft describes several problems that have been identified with the Session Initiation Protocol's non-INVITE transaction.

Table of Contents

1. Problems under the current specifications

There are a number of unpleasant edge conditions created by the SIP non-INVITE transaction model's fixed duration. The negative aspects of some of these are exacerbated by the effect provisional responses have on the non-INVITE transaction state machines as currently defined.

1.1 NITs must complete immediately or risk losing a race

The non-INVITE transaction defined in RFC 3261 [1] is designed to have a fixed and finite duration (dependent on T1). A consequence of this design is that participants must strive to complete the transaction as quickly as possible. Consider the race condition shown in Figure 1.

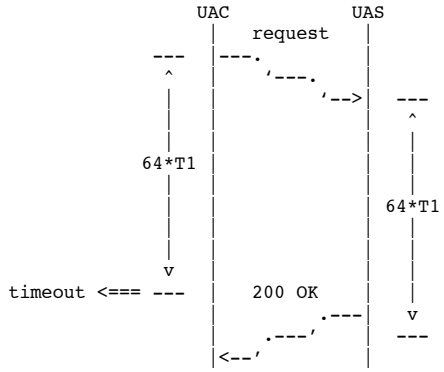


Figure 1: NI Race Condition

The UAS in this figure believes it has responded to the request in time, and that the request succeeded. The UAC, on the other hand, believes the request has timed-out, hence failed. No longer having a matching client transaction, the UAC core will ignore what it believes to be a spurious response. As far as the UAC is concerned, it received no response at all to its request. The ultimate result is the UAS and UAC have conflicting views of the outcome of the transaction.

Therefore, a UAS cannot wait until the last possible moment to send a final response within a NIT. It must, instead, send its response so that it will arrive at the UAC before that UAC times out. Unfortunately, the UAS has no way to accurately measure the propagation time of the request or predict the propagation time of the response. The uncertainty it faces is compounded by each proxy that participates in the transaction. Thus, the UAS's only choice is to send its final response as soon as it possibly can and hope for the best.

This result constrains the set of problems that can be solved with a single NIT. Any delay introduced during processing of a request increases the probability of losing the race. If the timing characteristics of that processing are not predictable and controllable, a single NIT is an inappropriate model for handling the request. One viable alternative is to accept the request with a 202 and send the ultimate results in a new request in the reciprocal direction.

In specialized networks, a UAS might have some reliable knowledge of inter-hop latency and could use that knowledge to determine if it has time to delay its final response in order to perform some processing such as a database lookup while mitigating its risk of losing the race in Figure 1. Establishing this knowledge across arbitrary networks (perhaps using resource reservation techniques and deterministic transports) is not currently feasible.

1.2 Provisional responses can delay recovery from lost final responses

The non-INVITE client transaction state machine provides reliability for NITs over unreliable transports (UDP) through retransmission of the request message. Timer E is set to T1 when a request is initially transmitted. As long as the machine remains in the Trying state, each time Timer E fires, it will be reset to twice its previous value (capping at T2) and the request is retransmitted.

If the non-INVITE client transaction state machine sees a provisional response, it transitions to the Proceeding state, where retransmission continues, but the algorithm for resetting Timer E is simply to use T2 instead of doubling at each firing. (Note that Timer E is not altered during the transition to Proceeding).

Making the transition to the Proceeding state before Timer E is reset to T2 can cause recovery from a lost final response to take extra time. Figure 2 shows recovery from a lost final response with and without a provisional message during this window. Recovery occurs within 2*T1 in the case without the provisional. With the provisional, recovery is delayed until T2, which by default is 8*T1.

In practical terms, a provisional response to a NIT in currently deployed networks can delay transaction completion by up to 3.5 seconds.

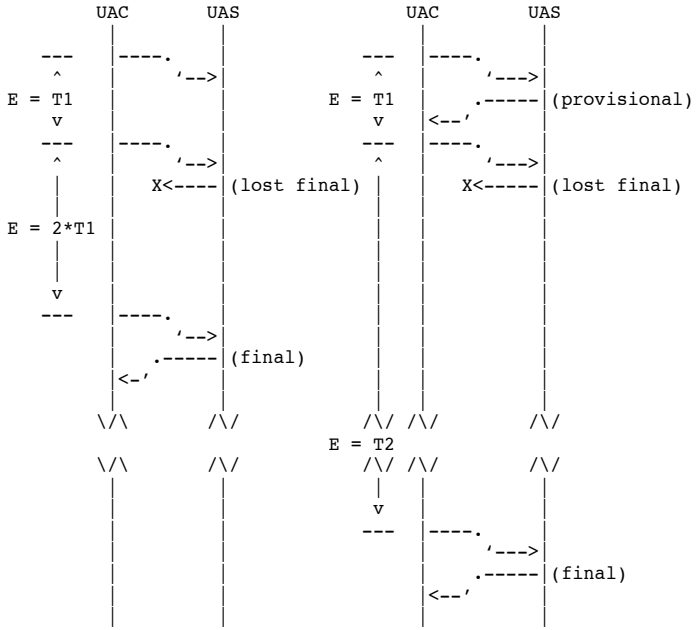


Figure 2: Provisionals can harm recovery

No additional delay is introduced if the first provisional response is received after Timer E has reached its maximum reset interval of T2.

1.3 Delayed responses will temporarily blacklist an element

A SIP element's use of SRV is specified in RFC 3263 [2]. That specification discusses how SIP assures high availability by having upstream elements detect failure of downstream elements. It proceeds to define several types of failure detection and instructions for

failover. Two of the behaviors it describes are important to this document:

- Within a transaction, transport failure is detected either through an explicit report from the transport layer or through timeout. Note specifically that timeout will indicates transport failure regardless of the transport in use. When transport failure is detected, the request is retried at the next element from the sorted results of the SRV query.

- Between transactions, locations reporting temporary failure (through 503/Retry-After for example) are not used until their requested black-out period expires.

The specification notes the benefit of caching locations that are successfully contacted, but does not discuss how such a cache is maintained. It is unclear whether an element should stop using (temporarily blacklist) a location returned in the SRV query that results in a transport error. If it does, when should such a location be removed from the blacklist?

Without such a blacklist (or equivalent mechanism), the intended availability mechanism fails miserably. Consider traffic between two domains. Proxy pA in domain A needs to forward a sequence of non-INVITE requests to domain B. Through DNS SRV, pA discovers pB1 and pB2, and the ordering rules of [2] and [3] indicate it should use pB1 first. The first request to pB1 times out. Since pA is a proxy and a NIT has a fixed duration, pA has no opportunity to retry the request at pB2. If pA does not remember pB1's failure, the second request (and all subsequent non-INVITE requests until pB1 recovers) are doomed to the same failure. Caching would allow the subsequent requests to be tried at pB2.

Since miserable failure is not acceptable in deployed networks, we should anticipate that elements will, in fact, cache timeout failures between transactions. Then the race in Figure 1 becomes important. If an element fails to respond "soon enough", it has effectively not responded at all, and will be blacklisted at its peer for some period of time.

(Note that even with caching, the first request timeout results in a timeout failure all the way back to the original submitter. The failover mechanisms in [2] work well to increase the resiliency of a given INVITE transaction, but do nothing for a given non-INVITE transaction.)

1.4 408 for non-INVITE is not useful

Consider the race condition in Figure 1 when the final response is 408 instead of 200. Under the current specification, the race is guaranteed to be lost. Most existing endpoints will emit a 408 for a non-INVITE request 64*T1 after receiving the request if they haven't emitted an earlier final response. Such a 408 is guaranteed to arrive at the next upstream element too late to be useful. In fact, in the presence of proxies, these messages are even harmful. When the 408 arrives, each proxy will have already terminated its associated client transaction due to timeout. So, each proxy must forward the 408 upstream statelessly. This, in turn, is guaranteed to arrive too late. As Figure 3 shows, this can ultimately result in bombarding the original requester with spurious 408s. (Note that the proxy's client transaction state machine never enters the Completed state, so Timer K does not enter into play).

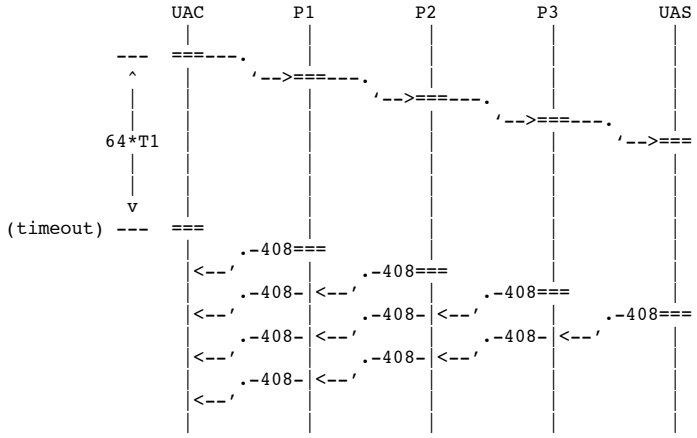


Figure 3: late 408s to non-INVITEs

This response bombardment is not limited to the 408 response, though it only exists when participating client transaction state machines are timing out. Figure 4 generalizes Figure 1 to include multiple hops. Note that even though the UAS responds "in time" to P3, the response is too late for P2, P1 and the UAC.

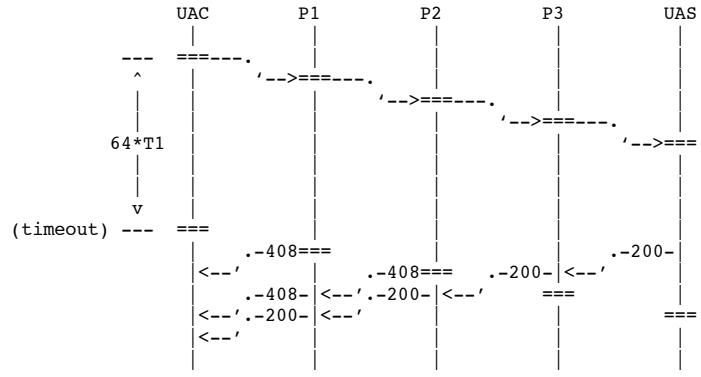


Figure 4: Additional timeout related error

1.5 Non-INVITE timeouts doom forking proxies

A single branch with a delayed or missing final response will dominate the processing at proxy that receives no 2xx responses to a forked non-INVITE request. Since this proxy is required to allow all of its client transactions to terminate before choosing a "best response". This forces the proxy's server transaction to lose the race in Figure 1. Any response it ultimately forwards (a 401 for example) will arrive at the upstream elements too late to be used. Thus, if no element among the branches would return a 2xx response, failure of a single element (or its transport) dooms the proxy to failure.

1.6 Mismatched timer values make winning the race harder

There are many failure scenarios due to misconfiguration or misbehavior that the SIP specification does not discuss. One is placing two elements with different configured values for T1 and T2 on the same network. Review of Figure 1 illustrates that the race failure is only made more likely in this misconfigured state (it may appear that shortening T1 at the element behaving as a UAS improves this particular situation, but remember that these elements may trade roles on the next request). Since the protocol provides no mechanism for discovering/negotiating a peer's timer values, exceptional care must be taken when deploying systems with non-defaults to ensure they will _never_ directly communicate with elements with default values.

2. Acknowledgments

This document captures many conversations about non-INVITE issues. Significant contributers include Ben Campbell, Gonzalo Camarillo, Steve Donovan, Rohan Mahy, Dan Petrie, Adam Roach, Jonathan Rosenberg, and Dean Willis.

References

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

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

[3] Gulbrandsen, A., Vixie, P. and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

Author's Address

Robert J. Sparks
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, TX 75024

EMail: rsparks@dynamicsoft.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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

Session Initiation Protocol (SIP)
Internet Draft
Document: draft-stucker-sip-guid-00.txt
Expires: July 2004

B. Stucker
Nortel Networks

February 2004

Client Globally Unique ID for SIP

Status of this Memo

Abstract

A number of applications using the Session Initiation Protocol (SIP) protocol require or can be enhanced by being able to uniquely identify a particular user agent (UA) instance in the network. This document describes an extension to SIP that allows clients to generate globally unique identifiers (GUID) for use within SIP based applications, providing an example of their use.

Table of Contents

1. Introduction

Within SIP, there arise situations where it is necessary to ensure that an action is applied to a particular user agent (UA) instance, but the existing mechanisms within SIP are not always reliable. For example, although registrations identify a routable address and port of a particular UA, in an environment that uses dynamically assigned IP addresses (NATs, VPNs, short-lease DHCP networks) there is no ready way of always tying registrations together across time for a particular UA instance. In these environments, the usual IP/port combination that defines a particular routing location of a UA is unreliable over time as an identifier of that UA.

As a result, an identifier that UAs can use as a "finger-printing" mechanism to identify themselves is useful. Whereas the Globally Routable UA URIs (GRUU) draft [4] seeks to address a server-generated identifier for the UA, this draft seeks to define a client-generated approach to a similar problem.

The mechanism defined in this document allows a particular UA instance to construct a globally unique identifier which can be used by SIP services to process requests that require, or are enhanced by, the ability to identify a particular UA instance in the network over a long period of time.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [ii].

3. Creating a GUID

This section covers the details of creating a GUID on the client UA.

3.1 Characteristics of a GUID

The idea of a globally unique ID is hardly a new concept. Designers and developers of all sorts of applications in the physical world and the Internet have required the ability to uniquely identify a particular entity from a larger set. This is especially true when every other property of the entity is subject to substantial changes over time that would render it difficult or impossible to uniquely identify over time.

For example, governments frequently assign us a number (or other identifying string) when we are born because they have a need to identify us as taxpayers throughout our lives. There are several other examples of unique IDs, such as vehicle identification numbers and serial numbers on items we buy from large manufacturers.

A common characteristic of these identification numbers is that they have two basic properties:

- They are unique to the entity they are associated with.
- A central authority coordinates the assignment of IDs to ensure that no two entities are given the same identifier.

Note, that there is no requirement that there be any sort of registry that knows which entity has what identifier. This would be needed if the identifier were to be used for non-repudiation purposes, but that is not always a goal that needs to be fulfilled depending on the application.

Sometimes entities need to be able to be identified uniquely, but to have a central authority assign an identifier would be difficult or impossible. In these situations, it is still possible for the entity to assign itself a unique identifier. This can be achieved by using a mechanism that ensures that the odds of any two entities having the same identifier are statistically insignificant.

An example of this mechanism would be human fingerprints. Fingerprints can be used as a globally unique identifier of who you

are, and the odds of two people having the same fingerprints are statistically insignificant (even twins have a different set). No central authority coordinates the assignment of who gets what fingerprints, and yet they can be used to uniquely identify a particular person. If they are registered with a central authority, they can be used for purposes of non-repudiation. In either case, they are very useful, as other characteristics of people may change wildly over time.

3.2 Construction of a SIP GUID

Constructing an identifier that describes a UA is trivialquite straightforward. SIP TAGs are frequently generated to identify a particular UA session within SIP. Ensuring that the identifier is unique within a small, controlled set of UAs is more difficult, but still manageable by simply assigning them directly to the UA upon creation (like assigning a static IP address to a machine on a LAN). However, making that identifier unique across very large sets could be very difficult by simple assignment through sheer logistics (think about your experiences trying to get a driver's license).

Because a straightforward assignment of a GUID is problematic at best (and impossible at worst) this approach is ruled out in favor of using a standard mechanism: use time and space to your advantage. All SIP GUIDs MUST be generated based off the time that they were generated, and the "space" in which they were generated.

Obviously, generating a SIP GUID that is composed of a three-digit number would not satisfy most reasonable definitions of "unique" within a SIP network. Therefore, SIP GUIDs MUST be at least 128-bits in length.

3.2.1 Time Component

Time can be used to create uniqueness because each instant in time only occurs once. This can be used to constrain the set of all UAs that wish to create a GUID at that instant from the set of all UAs that will ever exist (ie. all of the UAs that wish to create a GUID on February 6th, 2004 at 10:45pm as opposed to all UAs that will ever exist from now to eternity). This means that a component of a GUID should be based on the current local time. It is not necessary that every UA generating a GUID need to have synchronized clocks with every other UA. This is because we're not interested in being able to tell the exact moment a GUID is created. It's used simply as a component of the GUID in order to constrain the larger set.

Many computers and development platforms vary in the scale at which time can be measured. Because we are using time to constrain the set of all UAs that may ever wish to generate a GUID, it is important

that the smallest available unit be used by the UA generating the GUID. Additionally, a large random number from a cryptographically-strong random number generator can be appended to the current time to create a pseudo-timestamp with very fine resolution.

Here's an example:

- A computer's clock can be resolved down to 1 millisecond.
- The computer's random number generator can produce a random integer up to (2^31)-1.
- From this a "pseudo-clock" can then be constructed that resolves time down to the order of a pico-second (10^-12 seconds, or trillionths of a second).

Friday, February 6th, 2004 at 21:30:54 CST can be expressed as 1076124654957 milliseconds since January 1, 1970, 00:00:00 GMT.

A possible random number generated by a cryptographically-strong random number generator: 190182543.

Taken together, it is possible to create a "pseudo-time" of 1076124654957190182543 pico-seconds since January 1, 1970 00:00:00 GMT.

This is a very powerful notion, and if further resolution is required, successive random numbers can be appended to further resolve the "pseudo-clock" to fantastically small instants of time. It is critical, however, that an actual clock source be used as the most-significant digits of the "pseudo-clock".

In the example given, even if 1 billion SIP UAs decided to generate a new GUID at the same time, it is still a 1 in a trillion chance that they come up with the same "pseudo-clock" time.

SIP GUIDs MUST use a "psuedo-clock" that resolves to a minimum of 10^-12 seconds.

3.2.2 Space Component

The other component to a well-formed globally unique identifier that is not assigned by a central authority is to use space (or an approximation of it) as a component. It can obviously be the same time in multiple places, but no two UAs can ultimately occupy the same position in "space".

Because we are dealing with the electronic world, the notion of space is used somewhat conceptually; depending on the application, what

constitutes "space" may vary. The MAC address of the device that the UA instance runs upon would be a good way to denote its position in space, where space is given as the network. However, there are security implications involved with handing out a MAC address at the application level. For one, it can be used to discover the manufacturer of the device, which may help an attacker determine a method of attack.

Therefore, MAC addresses SHOULD NOT be used as an identifier of space for the purposes of a SIP GUID.

Additionally, there may be multiple UA instances executing on the same CPU. For this reason, it is RECOMMENDED that the space component of a SIP GUID be a location in memory that is uniquely held by that UA instance, as well as the IP address of the UA. Taken together with the time component, this still provides a high level of uniqueness within the network. It is extremely unlikely that two UA instances would be stored in the same location in memory, on two computers with the exact same IP address, at the exact same "pseudo-clock" time.

SIP GUIDs MUST contain a space component that provides no fewer than 64 bits of uniqueness.

3.3 Comparing SIP GUIDs

When comparing two SIP GUIDs, their values SHOULD be considered a unique identifier for the UA instance associated with the party that sent the SIP request, including any aliases of the user or entity identified by the sending party.

3.4 The GUID Header

The GUID header identifies a UA GUID. This header denotes the GUID for that UA instance. The GUID header MUST NOT appear in a SIP response, and if present MUST be ignored by the recipient. The GUID header MAY appear in any SIP request type. It is RECOMMENDED that user agents include their GUID in any REGISTER request sent.

3.4.1 Syntax

This document adds the following entry to Table 2 of [1]. Additions to this table are also provided for extension methods defined at the time of publication of this document. This is provided as a courtesy to the reader and is not normative in any way. MESSAGE, SUBSCRIBE and NOTIFY, REFER, INFO, UPDATE, PRACK, and PUBLISH are defined respectively in [6], [7], [5], [8], [9], [10], and [11].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	MSG
GUID	R		o	o	o	o	o	o	o

Header field	where	SUB	NOT	REF	INF	UPD	PRA	PUB
GUID	R	o	o	o	o	o	o	o

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC-2234 [3].

```
GUID        = "GUID" HCOLON token
```

A SIP request MUST contain no more than one GUID.

Examples:

```
GUID: f7ca930e2412f1bf016eb4940441672d3c26b17

GUID: 10761246549571901825 43+47bfc83e+10.33.15.8
```

3.5 Proxy Behavior

Proxies MUST NOT modify the contents of the GUID header during processing. It MAY be stripped according to the privacy policies of the system should header privacy have been requested by the UA sending the request in accordance with RFC-3323.

4. Security Considerations

The extension defined in this document may impact the security of a particular SIP application. Depending on the use of the GUID in a given application, considerations may need to be made to use a secure transport mechanism such as TLS for sending SIP requests containing a GUID.

5. IANA Considerations

5.1 Registration of the "GUID" SIP header

Name of Header: GUID

Short form: none

Normative description: section 3.4 of this document.

6. References

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

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

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

[4] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-00 (work in progress), January 2004.

[5] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[6] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C. and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[7] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[8] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[9] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

[10] Rosenberg, J. and Schulzrinne, H., "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 2362, June 2002.

[11] Niemi, A., "SIMPLE Presence Publication Mechanism", draft-ietf-sip-publish-02 (work in progress), January 2004.

7. Acknowledgements

Thanks to Jennifer Beckman, Mary Barnes, Alberto Villarica, Trip Ingle, and William Janning for comments and helping to create the foundation for this document. Additionally, thanks is given to Jonathan Rosenberg for introduction of the GRUU draft which describes the server-side generation of unique identifiers within SIP.

Author's Addresses

Brian Stucker
Nortel Networks
2380 Performance Drive
Richardson, Texas
Email: bstucker@nortelnetworks.com