Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 16, 2012

I. Baz Castillo
J. Luis Millan
XtraTelecom S.A.
V. Pascual
Acme Packet

September 13, 2011

WebSocket Transport for Session Initiation Protocol (SIP)
draft-ibc-rtcweb-sip-websocket-00

Abstract

This document specifies a WebSocket subprotocol for a new transport in SIP (Session Initiation Protocol). The WebSocket protocol enables two-way realtime communication between clients (typically web-based applications) and servers. The main goal of this specification is to integrate the SIP protocol within web applications.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 16, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

Integrating the SIP protocol [RFC3261] within modern web-based applications has been a hard task historically due to the specification complexity and inherent limitations in web browsers and HTTP protocol [RFC2616]. The arrival of WebSocket [I-D.ietf-hybi-thewebsocketprotocol] and [RTC-Web] (Real Time Collaboration on the World Wide Web) provides a two-way communication technology for web-based applications along with multimedia capabilities for audio and video sessions in web browsers, making feasible the requeriments of the SIP protocol.

This specification defines a new WebSocket subprotocol for transporting SIP messages between a WebSocket client and server, a new transport for the SIP protocol and procedures for SIP proxies when behaving as a bridge between WebSocket and other SIP transports. No changes have been made to the SIP protocol [RFC3261].

2. 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 [RFC2119].

3. Scope

The WebSocket protocol is mostly suitable for web-based applications running in a web browser. Other applications running out of web browsers do not have the constraints of web applications since typically they can directly access to the transport layer.

In the same manner, the WebSocket protocol adds a network overhead since it works as an intermediary layer between the transport and application layers. There is no benefit on using SIP over WebSocket transport between two SIP nodes when none of them runs within a web browser. Even more, the WebSocket protocol is not symmetric since just a WebSocket client can open a connection to a WebSocket server (a WebSocket client does not listen for incoming connections).

Given these arguments, this specification is mostly focused on integrating the SIP protocol within web-based applications or any client application using the WebSocket protocol. Other aspects such as DNS NAPTR/SRV resolution for SIP over WebSocket transport are not covered by this specification since they are mainly useless given the WebSocket protocol nature.

This document just covers SIP as a signalling protocol, leaving multimedia capabilities integration for a separate document once [RTC-Web] (Real Time Collaboration on the World Wide Web) becomes a standard.

4. SIP WebSocket Transport

WebSocket [I-D.ietf-hybi-thewebsocketprotocol] is a reliable protocol and therefore the WebSocket subprotocol for a SIP transport defined by this document is also a reliable transport. Thus, client and server transactions using WebSocket transport MUST follow the procedures and timer values for a reliable transport as defined in [RFC3261].

4.1. Via Transport Parameter

Via header fields carry the transport protocol identifier. This document defines the value "WS" to be used for requests over plain WebSocket protocol and "WSS" for requests over secure WebSocket protocol (in which the WebSocket session is established on top of TLS [RFC5246] over TCP transport).

The updated augmented BNF (Backus-Naur Form) [RFC5234] for this parameter is the following (the original BNF for this parameter can be found in [RFC3261]):

```
transport         =  "UDP" / "TCP" / "TLS" / "WS" / "WSS"
                     / other-transport
```

The following are examples of Via header fields using "WS" and "WSS":

```
Via: SIP/2.0/WS 1.2.3.4:28456
Via: SIP/2.0/WSS [2001:0:63ba:74c:1806:7ea2:9aab:f892]:32802
```

4.2. SIP URI Transport Parameter

This document defines the value "ws" as the transport parameter value for a SIP URI [RFC3986] to be contacted using WebSocket protocol. Whether to select a plain or secure WebSocket connection depends on the SIP URI schema ("sip" schema means plain WebSocket connection while "sips" schema requires secure WebSocket connection).

The updated augmented BNF (Backus-Naur Form) [RFC5234] for this parameter is the following (the original BNF for this parameter can be found in [RFC3261]):

```
transport-param   =  "transport="
                     ( "udp" / "tcp" / "sctp" / "tls" / "ws"
                     / other-transport )
```

The following are examples of SIP URI’s containing a "ws" transport parameter:

```
sip:alice@1.2.3.4:28456;transport=ws
sips:bob@[2001:0:63ba:74c:1806:7ea2:9aab:f892]:32802;transport=ws
```

4.3. Sending Responses

The SIP server transport uses the value of the top Via header field in order to determine where to send a response. If the "sent-protocol" is "WS" or "WSS" the response MUST be sent using the existing WebSocket connection to the source of the original request, if that connection is still open. This requires the server transport to maintain an association between server transactions and transport connections. If that connection is no longer open, the server MUST NOT attempt to open a WebSocket connection to the Via "sent-by"/"received"/"rport".

> This is due the nature of the WebSocket protocol in which just the WebSocket client can establish a connection with the WebSocket server. A WebSocket client does not listen for incoming connections.

5. The WebSocket SIP Subprotocol

The term WebSocket subprotocol refers to the application-level protocol layered over a WebSocket connection. This document specifies the WebSocket SIP subprotocol for carrying SIP requests and responses through a WebSocket connection.

WebSocket [I-D.ietf-hybi-thewebsocketprotocol] defines message units as application data exchange for communication endpoints, becoming a message boundary protocol. These messages can contain UTF-8 text or binary data. The WebSocket SIP subprotocol specified in this document mandates messages of type UTF-8 text.

The WebSocket client and WebSocket server send SIP messages to each other. Each SIP message MUST be carried within a single WebSocket message and MUST be a complete SIP message, so a Content-Length header field is not mandatory. Sending more than one SIP message within a single WebSocket message is not allowed, neither sending an incomplete SIP message.

> This makes parsing of SIP messages easier on client side (typically web-based applications with an strict and simple API for receiving WebSocket messages). There is no need to establish boundaries (typically using Content-Length headers) between different messages. Same advantage is present in other message-based SIP transports as UDP or SCTP [RFC4168].

6. WebSocket Client Usage

As stated in [I-D.ietf-hybi-thewebsocketprotocol], a WebSocket URI [RFC3986] is given to the WebSocket client (typically within a web-based application) who resolves the URI destination and establishes a WebSocket connection with the corresponding server (by performing the handshake and negotiation procedures described in [I-D.ietf-hybi-thewebsocketprotocol]).

The client application is supposed to be provided with SIP account configuration values (as an AoR, outbound proxy and so on). Such values are used by the client application when generating SIP messages.

After establishing the WebSocket connection, the client SHOULD discover the source IP and port from which the server has received the TCP connection. Such IP and port are required for constructing the client's SIP local URI (to be used in Contact header during SIP registration and SIP dialogs).

> The mechanism used by the client application in order to discover its source IP and port is currently out of the scope of this specification, although it might be defined in future revisions of this document.

The client's SIP local URI MUST be constructed as follows:

- If the WebSocket connection is secure (given WebSocket URI has "wss" schema) the URI MUST have "sips" schema, "sip" otherwise.
- The URI username is up to the application.
- The URI hostport is determined by the local IP and port previously retrieved.
- A "transport" parameter with value "ws" MUST be added to the URI.

This SIP local URI MUST be used by the client as a registration binding (Contact URI in a REGISTER) and as a local target for SIP dialogs (Contact URI in a request or response) since this URI is the only adddress in which the client can be contacted, and just through the WebSocket server.

Any new request sent by the client MUST contain the discovered local IP and port in the Via "sent-by" field. Via "sent-transport" field MUST be set to "WSS" if the WebSocket connection is secure, to "WS" otherwise.

Due to the nature of the WebSocket protocol, the client sends all the SIP requests to the WebSocket server it is connected to, so the WebSocket server behaves as a de facto outbound SIP proxy.

In case the client application decides to close the WebSocket connection (for example when performing "logout" in a web application) it is recommended to remove the existing SIP registration binding (if present) by specifying an expiration interval of "0" for that contact address in a REGISTER request as described in section 10.2.2 of [RFC3261].

6.1. WebSocket Disconnection

In some circumstances the WebSocket connection could be terminated by the WebSocket server (for example when the server is restarted). If the client application wants to become reachable again it SHOULD reconnect to the WebSocket server and perform the SIP local URI discovery process again followed by a new SIP registration.

The client MAY also remove the previous registration binding in the registrar server, as such address is no longer reachable.

When the WebSocket server is also the SIP registrar server, it MAY remove the SIP registration bindings associated to a WebSocket connection after such connection has been closed. Such a decision is out of the scope of this specification and depends on the SIP network topology.

7. WebSocket Server Usage

Here we assume that a SIP proxy or UAS (User Agent Server) is also acting as a WebSocket server implementing the WebSocket subprotocol described in this document. The server receives WebSocket connection attempts from clients. How the server authorizes or not those connections is out of the scope of this specification. Once the WebSocket subprotocol defined in this document has been negotiated, both client and server can send SIP messages to each other.

The server can only contact a SIP URI with the parameter "transport=ws" in case the destination address belongs to an existing WebSocket connection established from a WebSocket client. If not, a local transport error MUST be generated (which involves a 500 or 503 SIP response code).

> Such a case could happen when an existing SIP registration binding points to an already closed WebSocket connection which was not removed.

7.1. SIP Proxy Considerations

A SIP proxy implementing WebSocket transport can intercommunicate clients using SIP over WebSocket with other SIP clients or nodes using any other transport.

When the proxy bridges between WebSocket transport and any other SIP transport (including WebSocket transport) it MUST perform Loose Routing as specified in [RFC3261]. Otherwise in-dialog requests would fail since WebSocket clients cannot contact destinations other than their WebSocket server, and non-WebSocket SIP nodes cannot establish a connection to WebSocket clients. It is also recommended that the proxy follows recommendations in [RFC5658] and uses double Record-Route technique in these cases.

In the same way, if the SIP proxy implementing the WebSocket server behaves as an outbound proxy for REGISTER requests, it MUST add a Path header as described in [RFC3327]. Otherwise the WebSocket client would never receive incoming requests from the SIP registrar server after the lookup procedures in the SIP location service.

8. WebSocket Connection Keep Alive

It is recommended that the WebSocket client or server keeps the WebSocket connection open by sending periodic Ping frames as described in [I-D.ietf-hybi-thewebsocketprotocol] section 5.5.2. The mechanisms of decision for a WebSocket endpoint to maintain, or not, the connection over time is out of scope of this document.

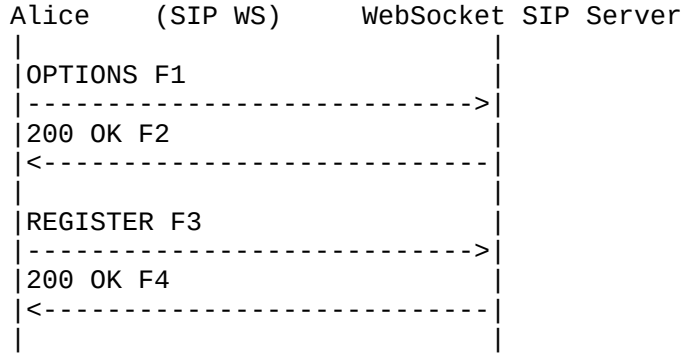
In some cases due to transient network errors, the connection with the WebSocket server could be lost without the WebSocket client being aware of it. The WebSocket client would only realize of the network failure when attempting to send new data over the WebSocket connection.

The authors of this specification have requested the W3C (World Wide Web Consortium) to include a mechanism in the WebSocket API [WS-API] for instructing the WebSocket client to supervise the connection by sending periodical Ping frames at the interval requested by the API user.

9. Examples

The flows depicted in this section describe the behavior of an initial prototype which is currently under development.

9.1. Registration



Alice is a WebSocket client running on a web browser. Alice establishes a plain WebSocket connection with a WebSocket server (also a SIP proxy/registrar) implementing the SIP subprotocol. Upon connection, Alice sends a SIP OPTIONS request including an empty "rport" parameter [RFC3581] in the Via header and obtains its source IP and port from the Via "received" and "rport" parameters in the response. Alice then forms its SIP local URI and constructs a REGISTER request.

Message details (authentication and SDP bodies are omitted for simplicity):

F1 OPTIONS Alice -> WebSocket SIP Server

```
OPTIONS sip:ws-server.atlanta.com SIP/2.0
Via: SIP/2.0/WS 1.2.3.4;branch=z9hG4bKasudf;rport
From: sip:alice@atlanta.com;tag=ux8asodj
To: sip:ws-server.atlanta.com
Call-ID: 87djahs72kjsd
CSeq: 1 OPTIONS
Max-Forwards: 1
Accept: application/sdp
```

F2 200 OK WebSocket SIP Server -> Alice

```
SIP/2.0 200 OK
Via: SIP/2.0/WS 1.2.3.4;branch=z9hG4bKasudf;received=93.12.40.105;
  rport=19465
From: sip:alice@atlanta.com;tag=ux8asodj
To: sip:ws-server.atlanta.com;tag=jcx67hjm
Call-ID: 87djahs72kjsd
CSeq: 1 OPTIONS
Content-Type: application/sdp
```

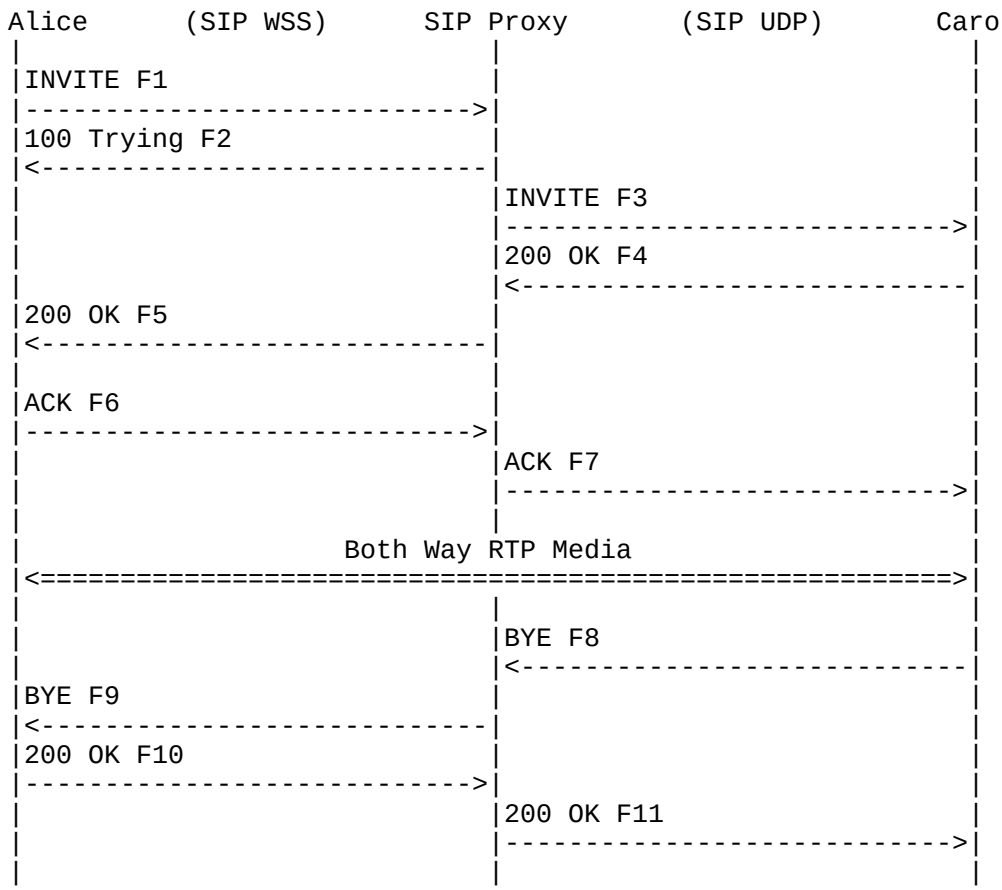
F3 REGISTER Alice -> WebSocket SIP Server

```
REGISTER sip:proxy.atlanta.com SIP/2.0
Via: SIP/2.0/WS 93.12.40.105:19465;branch=z9hG4bKasudf
From: sip:alice@atlanta.com;tag=65bnmj.34asd
To: sip:ws-server.atlanta.com
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Max-Forwards: 70
Contact: <sip:alice@93.12.40.105:19465;transport=ws>
```

F4 200 OK WebSocket SIP Server -> Alice

```
SIP/2.0 200 OK
Via: SIP/2.0/WS 93.12.40.105:19465;branch=z9hG4bKasudf
From: sip:alice@atlanta.com;tag=65bnmj.34asd
To: sip:ws-server.atlanta.com;tag=12isjljn8
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Contact: <sip:alice@93.12.40.105:19465;transport=ws>
```

9.2. INVITE dialog through a proxy



Here the WebSocket server is also a SIP proxy and registrar for the domain atlanta.com. Alice, a WebSocket SIP client, calls Carol's AoR through a secure WebSocket connection. The WebSocket SIP server acts as a SIP proxy routing the INVITE to the UDP location of Carol. The proxy does Loose-Routing. Carol answers the call and terminates it later.

Message details (authentication and SDP bodies are omitted for simplicity):

F1 INVITE Alice -> SIP Proxy (transport WSS)

INVITE sip:carol@atlanta.com SIP/2.0

```
   Via: SIP/2.0/WSS 93.12.40.105:20565;branch=z9hG4bK56sdasks
   From: sip:alice@atlanta.com;tag=asdyka899
   To: sip:carol@atlanta.com
   Call-ID: asidkj3ss
   CSeq: 1 INVITE
   Max-Forwards: 70
   Contact: <sips:alice@93.12.40.105:20565;transport=ws>
   Content-Type: application/sdp
```

F2 100 Trying SIP Proxy -> Alice (transport WSS)

```
   SIP/2.0 100 Trying
   Via: SIP/2.0/WSS 93.12.40.105:20565;branch=z9hG4bK56sdasks
   From: sip:alice@atlanta.com;tag=asdyka899
   To: sip:carol@atlanta.com
   Call-ID: asidkj3ss
   CSeq: 1 INVITE
```

F3 INVITE SIP Proxy -> Carol (transport UDP)

```
   INVITE sip:carol@77.123.45.23:5060 SIP/2.0
   Via: SIP/2.0/UDP 100.100.100.100;branch=z9hG4bKhjhjqw32c
   Via: SIP/2.0/WSS 93.12.40.105:20565;branch=z9hG4bK56sdasks
   Record-Route: <sip:100.100.100.100;transport=udp>,
     <sips:100.100.100.100:9090;transport=ws>
   From: sip:alice@atlanta.com;tag=asdyka899
   To: sip:carol@atlanta.com
   Call-ID: asidkj3ss
   CSeq: 1 INVITE
   Max-Forwards: 69
   Contact: <sips:alice@93.12.40.105:20565;transport=ws>
   Content-Type: application/sdp
```

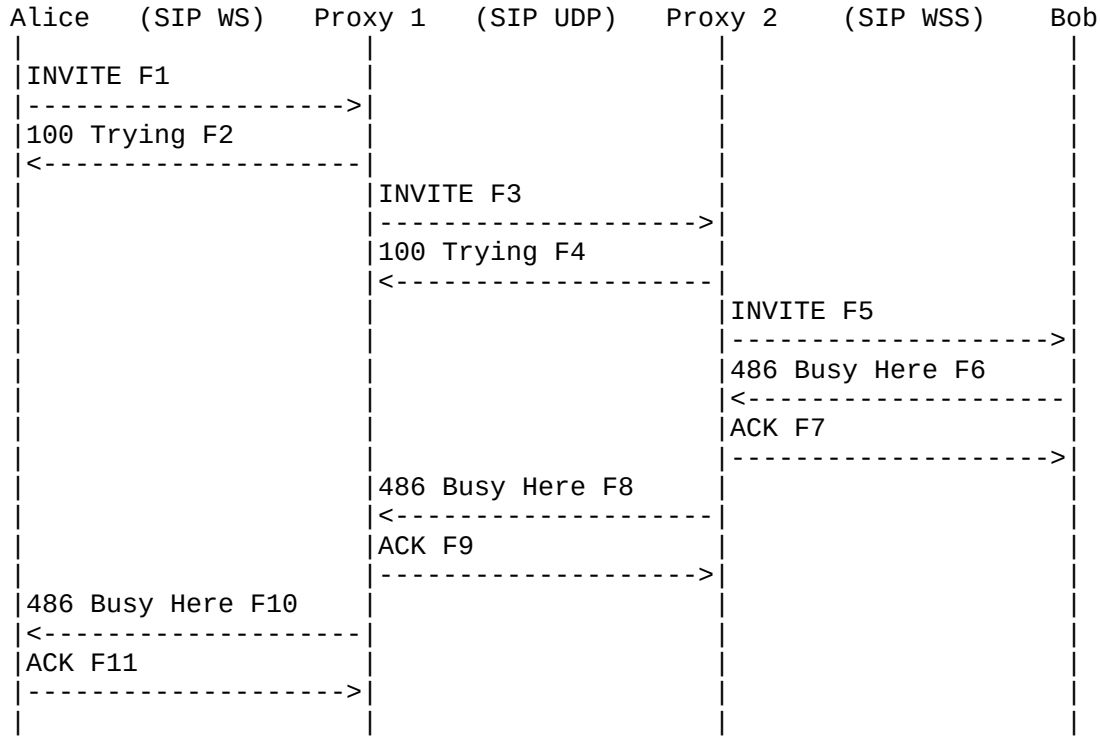
F4 200 OK Carol -> SIP Proxy (transport UDP)

```
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP 100.100.100.100;branch=z9hG4bKhjhjqw32c
   Via: SIP/2.0/WSS 93.12.40.105:20565;branch=z9hG4bK56sdasks
   Record-Route: <sip:100.100.100.100;transport=udp>,
     <sips:100.100.100.100:9090;transport=ws>
   From: sip:alice@atlanta.com;tag=asdyka899
   To: sip:carol@atlanta.com;tag=bmqkjhsd
   Call-ID: asidkj3ss
   CSeq: 1 INVITE
   Max-Forwards: 69
```

```
   Contact: <sip:carol@77.123.45.23:5060;transport=udp>
   Content-Type: application/sdp


   F5 200 OK SIP Proxy -> Alice (transport WSS)

   SIP/2.0 200 OK
   Via: SIP/2.0/WSS 93.12.40.105:20565;branch=z9hG4bK56sdasks
   Record-Route: <sip:100.100.100.100;transport=udp>,
     <sips:100.100.100.100:9090;transport=ws>
   From: sip:alice@atlanta.com;tag=asdyka899
   To: sip:carol@atlanta.com;tag=bmqkjhsd
   Call-ID: asidkj3ss
   CSeq: 1 INVITE
   Max-Forwards: 69
   Contact: <sip:carol@77.123.45.23:5060;transport=udp>
   Content-Type: application/sdp


   F6 ACK Alice -> SIP Proxy (transport WSS)

   ACK sip:carol@77.123.45.23:5060;transport=udp SIP/2.0
   Via: SIP/2.0/WSS 93.12.40.105:20565;branch=z9hG4bKhgqqp090
   Route: <sips:100.100.100.100:9090;transport=ws>,
     <sip:100.100.100.100;transport=udp>
   From: sip:alice@atlanta.com;tag=asdyka899
   To: sip:carol@atlanta.com;tag=bmqkjhsd
   Call-ID: asidkj3ss
   CSeq: 1 ACK
   Max-Forwards: 70


   F7 ACK SIP Proxy -> Carol (transport UDP)

   ACK sip:carol@77.123.45.23:5060;transport=udp SIP/2.0
   Via: SIP/2.0/UDP 100.100.100.100;branch=z9hG4bKhwpoc80zzx
   Via: SIP/2.0/WSS 93.12.40.105:20565;branch=z9hG4bKhgqqp090
   From: sip:alice@atlanta.com;tag=asdyka899
   To: sip:carol@atlanta.com;tag=bmqkjhsd
   Call-ID: asidkj3ss
   CSeq: 1 ACK
   Max-Forwards: 69


   F8 BYE Carol -> SIP Proxy (transport UDP)

   BYE sips:alice@93.12.40.105:20565;transport=ws SIP/2.0
   Via: SIP/2.0/UDP 77.123.45.23;branch=z9hG4bKbiuiansd001
```

```
   Route: <sip:100.100.100.100;transport=udp>,
     <sips:100.100.100.100:9090;transport=ws>
   From: sip:carol@atlanta.com;tag=bmqkjhsd
   To: sip:alice@atlanta.com;tag=asdyka899
   Call-ID: asidkj3ss
   CSeq: 1201 BYE
   Max-Forwards: 70


   F9 BYE SIP Proxy -> Alice (transport WSS)

   BYE sips:alice@93.12.40.105:20565;transport=ws SIP/2.0
   Via: SIP/2.0/WSS 100.100.100.100:9090;branch=z9hG4bKmma01m3r5
   Via: SIP/2.0/UDP 77.123.45.23;branch=z9hG4bKbiuiansd001
   From: sip:carol@atlanta.com;tag=bmqkjhsd
   To: sip:alice@atlanta.com;tag=asdyka899
   Call-ID: asidkj3ss
   CSeq: 1201 BYE
   Max-Forwards: 69


   F10 200 OK Alice -> SIP Proxy (transport WSS)

   SIP/2.0 200 OK
   Via: SIP/2.0/WSS 100.100.100.100:9090;branch=z9hG4bKmma01m3r5
   Via: SIP/2.0/UDP 77.123.45.23;branch=z9hG4bKbiuiansd001
   From: sip:carol@atlanta.com;tag=bmqkjhsd
   To: sip:alice@atlanta.com;tag=asdyka899
   Call-ID: asidkj3ss
   CSeq: 1201 BYE


   F11 200 OK SIP Proxy -> Carol (transport UDP)

   SIP/2.0 200 OK
   Via: SIP/2.0/UDP 77.123.45.23;branch=z9hG4bKbiuiansd001
   From: sip:carol@atlanta.com;tag=bmqkjhsd
   To: sip:alice@atlanta.com;tag=asdyka899
   Call-ID: asidkj3ss
   CSeq: 1201 BYE
```

9.3. INVITE dialog through two proxies



Alice and Bob are WebSocket clients running on web browsers. Alice belongs to atlanta.com SIP domain while Bob does to biloxi.com. Each domain has its own SIP proxy. Both proxies are also WebSocket servers. Alice calls Bob's AoR through a WebSocket connection. Bob responds the INVITE with a 486 Busy Here. Communication through proxies is made via UDP transport protocol.

Message details (authentication and SDP bodies are omitted for simplicity):

F1 INVITE Alice -> Proxy 1 (transport WS)

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
From: Alice <sip:alice@atlanta.com>;tag=lxtyr
To: Bob <sip:bob@biloxi.com>
Call-ID: aslke3dkj
CSeq: 1 INVITE
Max-Forwards: 70
Contact: <sip:alice@93.12.40.105:21324;transport=ws>
```

```
   Content-Type: application/sdp


   F2 100 Trying Proxy 1 -> Alice (transport WS)

   SIP/2.0 100 Trying
   Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
   From: Alice <sip:alice@atlanta.com>;tag=lxtyr
   To: Bob <sip:bob@biloxi.com>
   Call-ID: aslke3dkj
   CSeq: 1 INVITE


   F3 INVITE Proxy 1 -> Proxy 2 (transport UDP)

   INVITE sip:bob@biloxi.com SIP/2.0
   Via: SIP/2.0/UDP 101.101.101.101;branch=z9hG4bKdkej
   Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
   Record-Route: <sip:101.101.101.101;transport=udp>
   Record-Route: <sip:101.101.101.101:80;transport=ws>
   From: Alice <sip:alice@atlanta.com>;tag=lxtyr
   To: Bob <sip:bob@biloxi.com>
   Call-ID: aslke3dkj
   CSeq: 1 INVITE
   Max-Forwards: 69
   Contact: <sip:alice@93.12.40.105:21324;transport=ws>
   Content-Type: application/sdp


   F4 100 Trying Proxy 2 -> Proxy 1 (transport UDP)

   SIP/2.0 100 Trying
   Via: SIP/2.0/UDP 101.101.101.101;branch=z9hG4bKdkej
   Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
   From: Alice <sip:alice@atlanta.com>;tag=lxtyr
   To: Bob <sip:bob@biloxi.com>
   Call-ID: aslke3dkj
   CSeq: 1 INVITE


   F5 INVITE Proxy 2 -> Bob (transport WSS)

   INVITE sips:bob@85.84.123.222:30142;transport=ws SIP/2.0
   Via: SIP/2.0/WSS 102.102.102.102:443;branch=z9hG4bKqowin
   Via: SIP/2.0/UDP 101.101.101.101;branch=z9hG4bKdkej
   Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
   Record-Route: <sips:102.102.102.102:443;transport=ws>
   Record-Route: <sip:102.102.102.102;transport=udp>
```

```
   Record-Route: <sip:101.101.101.101;transport=udp>
   Record-Route: <sip:101.101.101.101:9090;transport=ws>
   From: Alice <sip:alice@atlanta.com>;tag=lxtyr
   To: Bob <sip:bob@biloxi.com>
   Call-ID: aslke3dkj
   CSeq: 1 INVITE
   Max-Forwards: 68
   Contact: <sip:alice@93.12.40.105:21324;transport=ws>
   Content-Type: application/sdp
```

F6 486 Busy Here Bob -> Proxy 2 (transport WSS)

```
   SIP/2.0 486 Busy Here
   Via: SIP/2.0/WSS 102.102.102.102:443;branch=z9hG4bKqowin
   Via: SIP/2.0/UDP 101.101.101.101;branch=z9hG4bKdkej
   Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
   From: Alice <sip:alice@atlanta.com>;tag=lxtyr
   To: Bob <sip:bob@biloxi.com>;tag=dskfjd
   Call-ID: aslke3dkj
   CSeq: 1 INVITE
```

F7 ACK Proxy 2 -> Bob (transport WSS)

```
   ACK sips:bob@85.84.123.222:30142;transport=ws SIP/2.0
   Via: SIP/2.0/WSS 102.102.102.102:443;branch=z9hG4bKqowin
   From: Alice <sip:alice@atlanta.com>;tag=lxtyr
   To: Bob <sip:bob@biloxi.com>;tag=dskfjd
   Call-ID: aslke3dkj
   CSeq: 1 ACK
```

F8 486 Busy Here Proxy 2 -> Proxy 1 (transport UDP)

```
   SIP/2.0 486 Busy Here
   Via: SIP/2.0/UDP 101.101.101.101;branch=z9hG4bKdkej
   Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
   From: Alice <sip:alice@atlanta.com>;tag=lxtyr
   To: Bob <sip:bob@biloxi.com>;tag=dskfjd
   Call-ID: aslke3dkj
   CSeq: 1 INVITE
```

F9 ACK Proxy 1 -> Proxy 2 (transport UDP)

```
   ACK sip:bob@biloxi.com SIP/2.0
   Via: SIP/2.0/UDP 101.101.101.101;branch=z9hG4bKdkej
```

```
From: Alice <sip:alice@atlanta.com>;tag=lxtyr
To: Bob <sip:bob@biloxi.com>;tag=dskfjd
Call-ID: aslke3dkj
CSeq: 1 ACK
```

F10 486 Busy Here Proxy 1 -> Alice (transport WS)

```
SIP/2.0 486 Busy Here
Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
From: Alice <sip:alice@atlanta.com>;tag=lxtyr
To: Bob <sip:bob@biloxi.com>;tag=dskfjd
Call-ID: aslke3dkj
CSeq: 1 INVITE
```

F11 ACK Alice -> Proxy 1 (transport WS)

```
ACK sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/WS 93.12.40.105:21324;branch=z9hG4bKmmuuq
From: Alice <sip:alice@atlanta.com>;tag=lxtyr
To: Bob <sip:bob@biloxi.com>;tag=dskfjd
Call-ID: aslke3dkj
CSeq: 1 ACK
```

10. Security Considerations

If the client (typically a web-based application) needs to protect the privacy of the SIP traffic through the WebSocket connection, it is encouraged to use a secure WebSocket connection.

11. IANA Considerations

11.1. Registration of new Via transports

This specification registers two new transport identifiers for Via headers:

WS: MUST be used when constructing a SIP request to be sent over a plain WebSocket connection.

WSS: MUST be used when constructing a SIP request to be sent over a secure WebSocket connection (tunneled over TLS [RFC5246]).

11.2. Registration of new SIP URI transport

This specification registers a new value for the "transport" parameter in a SIP URI:

ws: Identifies a SIP URI to be contacted using a WebSocket (plain or secure) connection.

11.3. Registration of the WebSocket SIP subprotocol

If a registry is created for WebSocket subprotocols, the SIP subprotocol defined in this specification will be registered.

12. References

12.1. Normative References

[I-D.ietf-hybi-thewebsocketprotocol]
Fette, I. and A. Melnikov, "The WebSocket protocol", draft-ietf-hybi-thewebsocketprotocol-14 (work in progress), September 2011.

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

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

12.2. Informative References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

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

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4168] Rosenberg, J., Schulzrinne, H., and G. Camarillo, "The Stream Control Transmission Protocol (SCTP) as a Transport for the Session Initiation Protocol (SIP)", RFC 4168, October 2005.

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

[RFC5658] Froment, T., Lebel, C., and B. Bonnaerens, "Addressing Record-Route Issues in the Session Initiation Protocol

(SIP)", RFC 5658, October 2009.

[RTC-Web] IETF and W3C, "Real Time Collaboration on the World Wide Web", October 2010.

[WS-API] Hickson, I., "The Web Sockets API", September 2010.

Authors' Addresses

Inaki Baz Castillo
XtraTelecom S.A.
Barakaldo, Basque Country
Spain

Email: ibc@aliax.net

Jose Luis Millan
XtraTelecom S.A.
Bilbao, Basque Country
Spain

Email: jmillan@aliax.net

Victor Pascual
Acme Packet
Anabel Segura 10
Madrid, Madrid 28108
Spain

Email: vpascual@acmepacket.com

SIPCORE Working Group C. Holmberg
Internet-Draft I. Sedlacek
Intended status: Standards Track Ericsson
Expires: April 23, 2012 October 21, 2011

Requirements for indication of features supported by a SIP proxy
draft-ietf-sipcore-proxy-feature-reqs-02.txt

Abstract

The Session Initiation Protocol (SIP) "Caller Preferences" extension defined in RFC 3840 provides a mechanism that allows a SIP message to convey information relating to the originator's supported features/ capabilities. This document defines requirements for a mechanism that would allow SIP proxies to convey information relating to the proxy's supported features/capabilities.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 23, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) "Caller Preferences" extension defined in RFC 3840 [RFC3840] provides a mechanism that allows a SIP message to convey information relating to the originator's supported features/capabilities.

It can be useful for SIP proxies to indicate supported feature/ capabilities, that might trigger actions and enable functions in other SIP entities.

This document defines requirements for a mechanism that would allow SIP proxies to convey information related to the proxy's supported features/capabilities.

1.1. Use-case: IMS Service Continuity, handover of session in alerting state

The 3rd Generation Partnership Project (3GPP) defines a IP Multimedia Subsystem (IMS) Service Continuity mechanism [3GPP.23.237] for handover of Packet Switched (PS) sessions to Circuit Switched (CS) calls.

The handover is controlled by a Service Centralization and Continuity Application Server (SCC AS). When a session is established the User Equipment (UE) needs to determine whether SCC AS in signalling path of the session supports handover of session in alerting state (i.e. 180 Ringing response has already been sent or received but the dialog is not confirmed dialog yet) or not.

When handover occurs and a session in alerting state exists and both UE and SCC AS indicated support of the handover of session in alerting state, then the UE and SCC AS perform handover for the session in alerting state.

NOTE: The UE indicates the support of the handover of session in alerting state by the feature tag included in Contact header field.

1.2. Use-case: IMS Enhanced Service Continuity

The 3rd Generation Partnership Project (3GPP) defines a IP Multimedia Subsystem (IMS) Service Continuity mechanism [3GPP.23.237] for handover of Packet Switched (PS) sessions to Circuit Switched (CS) calls. The handover can be performed by a Service Centralization and Continuity Application Server (SCC AS), or by a SCC AS together with an Access Transfer Control Function (ATCF), that acts as a SIP proxy. Delegating part of the session handover functionality to an ATCF provides advantages related to voice interruption during session

handover etc, since the ATCF is located in the same network as the user.

1.2.1. Use-case: IMS Enhanced Service Continuity, ATCF discovery

In order for an SCC AS to delegate part of the session handover functionality to an ATCF, when the SCC AS is informed by the registrar about an accepted REGISTER transaction, the SCC AS needs to determine whether a proxy supporting the ATCF functionality is in the registration path.

1.2.2. Use-case: IMS Enhanced Service Continuity, identifying sessions subject to handover

In order for ATCF to perform the delegated part of the session handover functionality, when a session is set up, the ATCF needs to determine whether a SIP proxy supporting the SCC AS functionality is in the signalling path of the session.

1.2.3. Use-case: IMS Enhanced Service Continuity, indicating handover subfeature support

As the session handover functionality has been specified over several 3GPP releases, some subfeatures of the handover functionality are optional. Examples are:

- The handover of sessions with audio on hold (called the MSC server assisted mid-call feature); and

- The handover of sessions where a 180 Ringing response to the initial SIP INVITE request has already been sent or received but a final response has not been sent or received yet (called the SRVCC for calls in alerting phase).

The SCC AS needs to be aware of support of those subfeatures in ATCF, in order for the UE and the SCC AS to execute the correct handling when the handover occurs.

When ATCF receives a SIP REGISTER request, the ATCF indicates the support of those subfeatures along the indication of ATCF functionality.

When SCC AS is informed about the new/updated binding where a proxy indicated support of ATCF functionality along with support of those subfeatures, the SCC AS discovers the support of those subfeatures in the ATCF.

1.3. Use-case: IMS Inter-UE Transfer

The 3rd Generation Partnership Project (3GPP) defines inter-UE transfer enhancements [3GPP.24.837] which enhance delivery of media of a session to several User Equipments (UE).

The Service Centralization and Continuity Application Server (SCC AS) serving one of the UEs acts as local hub for the session. The UE controls the media of the session and is called controller UE.

Triggered by requests from the controller UE, the SCC AS serving the controller UE transfers media of the session to other UEs, called controlee UEs, by sending INVITE request offering the media to be transferred.

When an INVITE request is routed to the UE, the SCC AS serving the UE needs to determine whether a SIP proxy supporting the inter-UE transfer enhancements functionality (i.e. SCC AS of the controller UE) is already in the signalling path.

If so, the SCC AS proxies the signalling without further handling as there is already an existing local hub for the session.

If not, the SCC AS acts as local hub for the session.

2. 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 BCP 14, RFC 2119 [RFC2119].

3. Requirements

REQ-1: It MUST be possible for a SIP proxy to indicate, and convey to other SIP entities in the signalling path of a registration request, support of a particular feature/capability.

REQ-2: It MUST be possible for a SIP proxy to indicate, and convey to other SIP entities in the signalling path of a dialog-forming request, support of a particular feature/capability.

REQ-3: It MUST be possible for a SIP proxy to indicate that the indicated support of a feature/capability only applies to other SIP entities in the direction towards one of the SIP endpoints in the signalling path.

REQ-4: A SIP proxy MUST NOT, when indicating support of a feature/ capability, make any assumptions that SIP entities in the signalling path that receive the indicator will support, or understand the meaning of, the feature/capability, or even support the proxy feature/capability indication mechanism as a whole.

REQ-5: A SIP proxy MUST be able to indicate support of a feature/ capability to other SIP entities in the signaling path, even if some SIP entities in the signaling path (possibly including the UAC and/or UAS) do not support, or understand the meaning of, the feature/ capability, or even support the proxy feature/capability indication mechanism as a whole.

REQ-6: It MUST be possible to indicate whether indicated support of a feature/capability applies to specific registration, to a specific dialog, or to all dialogs created as part of INVITE transaction.

NOTE: This requirement might be fully implemented as part of the protocol mechanism, or parts might be left to be specified in a feature/capability specification, or it might be left to be specified in a feature/capability specification completely.

REQ-7: It MUST be possible to assign additional parameters (either as a single value, or a list of values) to a feature/capability indicator, in order to provide additional information about the feature/capability.

REQ-8: If a SIP entity receives a feature support indication that it does not understand, it MUST act as if it hadn’t received the indication.

REQ-9: If a SIP entity that does not support the proxy feature/ capability indication mechanism receives a feature support indication, it MUST act as if it hadn’t received the indication.

REQ-10: Other SIP entities MUST be able to make routing decisions based on received feature/capability support indications.

REQ-11: A feature/capability support indicator MUST only be used to indicate support of a feature/capability, and MUST NOT be used to indicate whether procedures associated with the feature/capability have been applied or not.

REQ-12: It MUST be possible to determine which features/capabilities are supported by the same proxy

REQ-13: A procedure for registering feature/capability indication values with IANA MUST be defined.

4. Security Considerations

Feature/capability support indications can provide sensitive information about a SIP entity. RFC 3840 cautions against providing sensitive information to another party. Once this information is given out, any use may be made of it.

5. IANA Considerations

None identified.

6. Acknowledgements

Thanks to Paul Kyzivat and Robert Sparks for their comments and guidance on the mailing list. Thanks to Andrew Allen and Dale Worley for providing text on additional use-cases. Thanks to Cullen Jennings for providing text on additional requirements. Thanks to Dale Worley for providing comments and text improvement suggestions. Thanks to Hadriel Kaplan for telling us how SBCs mess up the mechanisms we try to specify.

Thanks to Robert Sparks, Adam Roach and Paul Kyzivat for giving working procedure guidance.

7. Change Log

[RFC EDITOR NOTE: Please remove this section when publishing]

Changes from draft-ietf-sipcore-proxy-feature-reqs-01
- New REQ-12 added (old REQ-12 becomes REQ-13).
- New use-case added (section 1.2.3).

Changes from draft-ietf-sipcore-proxy-feature-reqs-00
- New REQ-5 added (IETF#81).
- New REQ-9 added (Dale Worley).
- Text added to REQ-4 and REQ-5, indicating that the requirement applies also in cases where an entity does not support the mechanism as a whole (Dale Worley).
- Usage of "session establishment transactions" terminology in REQ-6, in order to avoid misunderstanding of "session" (Dale Worley).
- Editorial correction in REQ-7: "additional parameter"->"additional parameters"
- Editorial clarifications to use-cases.

8. References

8.1. Normative References

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

8.2. Informative References

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

[3GPP.23.237] 3GPP, "IP Multimedia Subsystem (IMS) Service Continuity; Stage 2", 3GPP TS 23.237 10.7.0, September 2011.

[3GPP.24.837] 3GPP, "IP Multimedia (IM) Core Network (CN) subsystem inter-UE transfer enhancements; Stage 3", 3GPP TR 24.837 10.0.0, April 2011.

Authors' Addresses

Christer Holmberg
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: christer.holmberg@ericsson.com

Ivo Sedlacek
Ericsson
Scheelevaegen 19C
Lund 22363
Sweden

Email: ivo.sedlacek@ericsson.com

Network Working Group
Internet-Draft
Obsoletes: 3265 (if approved)
Updates: 3261, 4660
(if approved)
Intended status: Standards Track
Expires: November 1, 2012

A. B. Roach
Tekelec

April 30, 2012

SIP-Specific Event Notification
draft-ietf-sipcore-rfc3265bis-09

Abstract

This document describes an extension to the Session Initiation Protocol (SIP) defined by RFC 3261. The purpose of this extension is to provide an extensible framework by which SIP nodes can request notification from remote nodes indicating that certain events have occurred.

Note that the event notification mechanisms defined herein are NOT intended to be a general-purpose infrastructure for all classes of event subscription and notification.

This document represents a backwards-compatible improvement on the original mechanism described by RFC 3265, taking into account several years of implementation experience. Accordingly, this document obsoletes RFC 3265. This document also updates RFC 4660 slightly to accommodate some small changes to the mechanism that were discussed in that document.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 1, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The ability to request asynchronous notification of events proves useful in many types of SIP services for which cooperation between end-nodes is required. Examples of such services include automatic callback services (based on terminal state events), buddy lists (based on user presence events), message waiting indications (based on mailbox state change events), and PSTN and Internet Internetworking (PINT) [RFC2848] status (based on call state events).

The methods described in this document provide a framework by which notification of these events can be ordered.

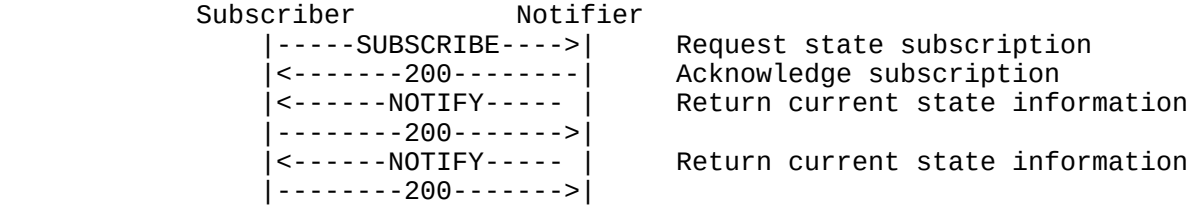
The event notification mechanisms defined herein are NOT intended to be a general-purpose infrastructure for all classes of event subscription and notification. Meeting requirements for the general problem set of subscription and notification is far too complex for a single protocol. Our goal is to provide a SIP-specific framework for event notification which is not so complex as to be unusable for simple features, but which is still flexible enough to provide powerful services. Note, however, that event packages based on this framework may define arbitrarily elaborate rules which govern the subscription and notification for the events or classes of events they describe.

This document does not describe an extension which may be used directly; it must be extended by other documents (herein referred to as "event packages"). In object-oriented design terminology, it may be thought of as an abstract base class which must be derived into an instantiatable class by further extensions. Guidelines for creating these extensions are described in Section 5.

1.1. Overview of Operation

The general concept is that entities in the network can subscribe to resource or call state for various resources or calls in the network, and those entities (or entities acting on their behalf) can send notifications when those states change.

A typical flow of messages would be:



Subscriptions are expired and must be refreshed by subsequent SUBSCRIBE requests.

1.2. Documentation Conventions

There are several paragraphs throughout this document which provide motivational or clarifying text. Such passages are non-normative, and are provided only to assist with reader comprehension. These passages are set off from the remainder of the text by being indented thus:

> This is an example of non-normative explanatory text. It does not form part of the specification, and is used only for clarification.

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

In particular, implementors need to take careful note of the meaning of "SHOULD" defined in RFC 2119. To rephrase: violation of SHOULD-strength requirements requires careful analysis and clearly enumerable reasons. It is a protocol violation to fail to comply with "SHOULD"-strength requirements whimsically or for ease of implementation.

The use of quotation marks next to periods and commas follows the convention used by the American Mathematical Society; although contrary to traditional American English convention, this usage lends clarity to certain passages.

2. Definitions

Event Package: An event package is an additional specification which defines a set of state information to be reported by a notifier to a subscriber. Event packages also define further syntax and semantics based on the framework defined by this document required to convey such state information.

Event Template-Package: An event template-package is a special kind of event package which defines a set of states which may be applied to all possible event packages, including itself.

Notification: Notification is the act of a notifier sending a NOTIFY request to a subscriber to inform the subscriber of the state of a resource.

Notifier: A notifier is a user agent which generates NOTIFY requests for the purpose of notifying subscribers of the state of a resource. Notifiers typically also accept SUBSCRIBE requests to create subscriptions.

Subscriber: A subscriber is a user agent which receives NOTIFY requests from notifiers; these NOTIFY requests contain information about the state of a resource in which the subscriber is interested. Subscribers typically also generate SUBSCRIBE requests and send them to notifiers to create subscriptions.

Subscription: A subscription is a set of application state associated with a dialog. This application state includes a pointer to the associated dialog, the event package name, and possibly an identification token. Event packages will define additional subscription state information. By definition, subscriptions exist in both a subscriber and a notifier.

Subscription Migration: Subscription migration is the act of moving a subscription from one notifier to another notifier.

3. SIP Methods for Event Notification

3.1. SUBSCRIBE

The SUBSCRIBE method is used to request current state and state updates from a remote node. SUBSCRIBE requests are target refresh requests, as that term is defined in [RFC3261].

3.1.1. Subscription Duration

SUBSCRIBE requests SHOULD contain an "Expires" header field (defined in [RFC3261]). This expires value indicates the duration of the subscription. In order to keep subscriptions effective beyond the duration communicated in the "Expires" header field, subscribers need to refresh subscriptions on a periodic basis using a new SUBSCRIBE request on the same dialog as defined in [RFC3261].

If no "Expires" header field is present in a SUBSCRIBE request, the implied default MUST be defined by the event package being used.

200-class responses to SUBSCRIBE requests also MUST contain an "Expires" header field. The period of time in the response MAY be

shorter but MUST NOT be longer than specified in the request. The notifier is explicitly allowed to shorten the duration to zero. The period of time in the response is the one which defines the duration of the subscription.

An "expires" parameter on the "Contact" header field has no semantics for the SUBSCRIBE method and is explicitly not equivalent to an "Expires" header field in a SUBSCRIBE request or response.

A natural consequence of this scheme is that a SUBSCRIBE request with an "Expires" of 0 constitutes a request to unsubscribe from the matching subscription.

> In addition to being a request to unsubscribe, a SUBSCRIBE request with "Expires" of 0 also causes a fetch of state; see Section 4.4.3.

Notifiers may also wish to cancel subscriptions to events; this is useful, for example, when the resource to which a subscription refers is no longer available. Further details on this mechanism are discussed in Section 4.2.2.

3.1.2. Identification of Subscribed Events and Event Classes

Identification of events is provided by three pieces of information: Request URI, Event Type, and (optionally) message body.

The Request URI of a SUBSCRIBE request, most importantly, contains enough information to route the request to the appropriate entity per the request routing procedures outlined in [RFC3261]. It also contains enough information to identify the resource for which event notification is desired, but not necessarily enough information to uniquely identify the nature of the event (e.g., "sip:adam@example.com" would be an appropriate URI to subscribe to for my presence state; it would also be an appropriate URI to subscribe to the state of my voice mailbox).

Subscribers MUST include exactly one "Event" header field in SUBSCRIBE requests, indicating to which event or class of events they are subscribing. The "Event" header field will contain a token which indicates the type of state for which a subscription is being requested. This token will be registered with the IANA and will correspond to an event package which further describes the semantics of the event or event class.

If the event package to which the event token corresponds defines behavior associated with the body of its SUBSCRIBE requests, those semantics apply.

Event packages may also define parameters for the Event header field; if they do so, they must define the semantics for such parameters.

3.1.3. Additional SUBSCRIBE Header Field Values

Because SUBSCRIBE requests create a dialog usage as defined in [RFC3261], they MAY contain an "Accept" header field. This header field, if present, indicates the body formats allowed in subsequent NOTIFY requests. Event packages MUST define the behavior for SUBSCRIBE requests without "Accept" header fields; usually, this will connote a single, default body type.

Header values not described in this document are to be interpreted as described in [RFC3261].

3.2. NOTIFY

NOTIFY requests are sent to inform subscribers of changes in state to which the subscriber has a subscription. Subscriptions are created using the SUBSCRIBE method. In legacy implementations, it is possible that other means of subscription creation have been used. However, this specification does not allow the creation of subscriptions except through SUBSCRIBE requests and (for backwards-compatibility) REFER requests [RFC3515].

NOTIFY is a target refresh request, as that term is defined in [RFC3261].

A NOTIFY request does not terminate its corresponding subscription; in other words, a single SUBSCRIBE request may trigger several NOTIFY requests.

3.2.1. Identification of Reported Events, Event Classes, and Current State

Identification of events being reported in a notification is very similar to that described for subscription to events (see Section 3.1.2).

As in SUBSCRIBE requests, NOTIFY request "Event" header fields MUST contain a single event package name for which a notification is being generated. The package name in the "Event" header field MUST match the "Event" header field in the corresponding SUBSCRIBE request.

Event packages may define semantics associated with the body of their NOTIFY requests; if they do so, those semantics apply. NOTIFY request bodies are expected to provide additional details about the nature of the event which has occurred and the resultant resource

state.

When present, the body of the NOTIFY request MUST be formatted into one of the body formats specified in the "Accept" header field of the corresponding SUBSCRIBE request (or the default type according to the event package description, if no Accept header field was specified). This body will contain either the state of the subscribed resource or a pointer to such state in the form of a URI (see Section 5.4.13).

4. Node Behavior

4.1. Subscriber Behavior

4.1.1. Detecting Support for SIP Events

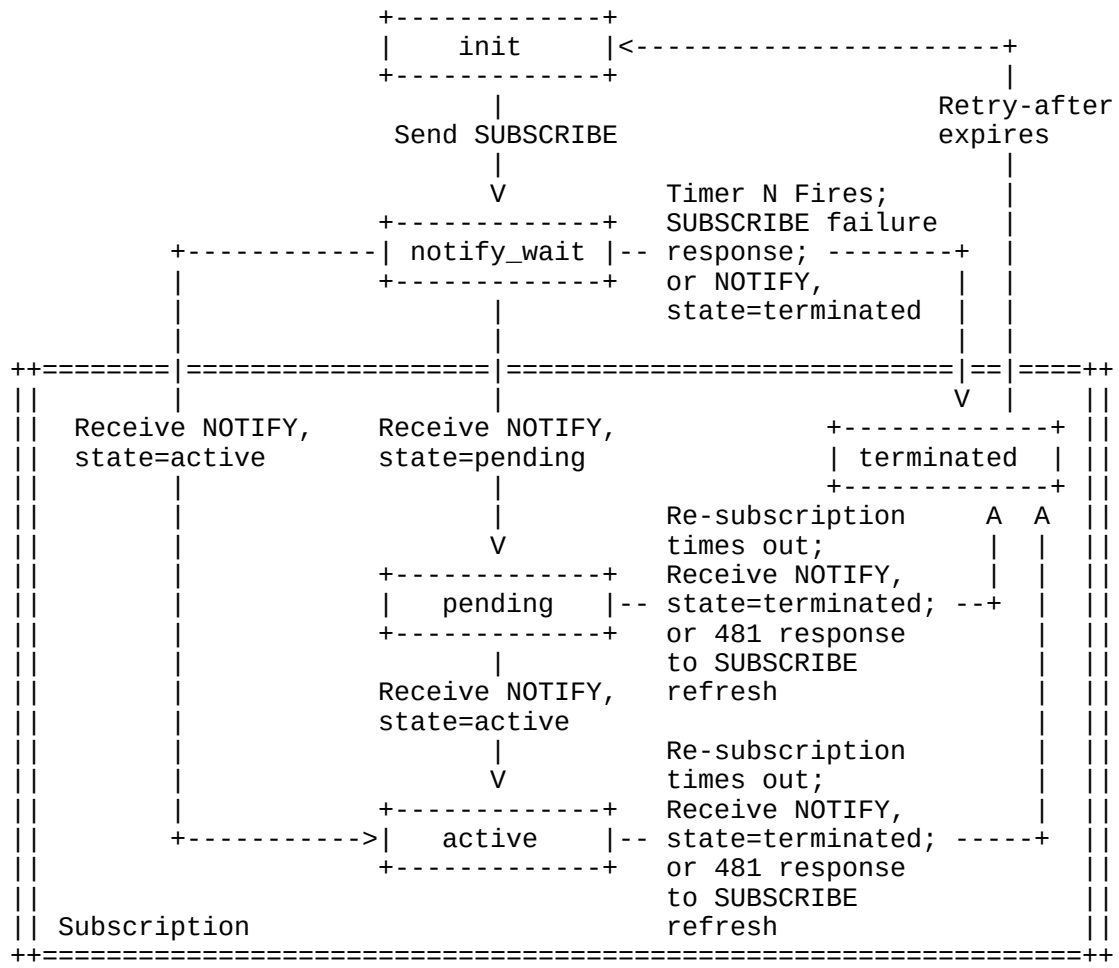
The extension described in this document does not make use of the "Require" or "Proxy-Require" header fields; similarly, there is no token defined for "Supported" header fields. Potential subscribers may probe for the support of SIP Events using the OPTIONS request defined in [RFC3261].

The presence of "SUBSCRIBE" in the "Allow" header field of any request or response indicates support for SIP Events; further, in the absence of an "Allow" header field, the simple presence of an "Allow-Events" header field is sufficient to indicate that the node that sent the message is capable of acting as a notifier (see Section 4.4.4).

> The "methods" parameter for Contact may also be used to specifically announce support for SUBSCRIBE and NOTIFY requests when registering. (See [RFC3840] for details on the "methods" parameter).

4.1.2. Creating and Maintaining Subscriptions

From the subscriber’s perspective, a subscription proceeds according to the following state diagram. Events which result in a transition back to the same state are not represented in this diagram.



In the state diagram, "Re-subscription times out" means that an attempt to refresh or update the subscription using a new SUBSCRIBE request does not result in a NOTIFY request before the corresponding Timer N expires.

Any transition from "notify_wait" into a "pending" or "active" state results in a new subscription. Note that multiple subscriptions can be generated as the result of a single SUBSCRIBE request (see Section 4.4.1). Each of these new subscriptions exists in its own independent state machine, and runs its own set of timers.

4.1.2.1. Requesting a Subscription

SUBSCRIBE is a dialog-creating method, as described in [RFC3261].

When a subscriber wishes to subscribe to a particular state for a resource, it forms a SUBSCRIBE request. If the initial SUBSCRIBE request represents a request outside of a dialog (as it typically will), its construction follows the procedures outlined in [RFC3261] for UAC request generation outside of a dialog.

This SUBSCRIBE request will be confirmed with a final response. 200-class responses indicate that the subscription has been accepted, and that a NOTIFY request will be sent immediately.

The "Expires" header field in a 200-class response to SUBSCRIBE request indicates the actual duration for which the subscription will remain active (unless refreshed). The received value might be smaller than the value indicated in the SUBSCRIBE request, but cannot be larger; see Section 4.2.1 for details.

Non-200 class final responses indicate that no subscription or new dialog usage has been created, and no subsequent NOTIFY request will be sent. All non-200 class responses (with the exception of "489", described herein) have the same meanings and handling as described in [RFC3261]. For the sake of clarity: if a SUBSCRIBE request contains an "Accept" header field, but that field does not indicate a media type that the notifier is capable of generating in its NOTIFY requests, then the proper error response is 406 (Not Acceptable).

4.1.2.2. Refreshing of Subscriptions

At any time before a subscription expires, the subscriber may refresh the timer on such a subscription by sending another SUBSCRIBE request on the same dialog as the existing subscription. The handling for such a request is the same as for the initial creation of a subscription except as described below.

If a SUBSCRIBE request to refresh a subscription receives a 404, 405, 410, 416, 480-485, 489, 501, or 604 response, the subscriber MUST consider the subscription terminated. (See [RFC5057] for further details and notes about the effect of error codes on dialogs and usages within dialog, such as subscriptions). If the subscriber wishes to re-subscribe to the state, he does so by composing an unrelated initial SUBSCRIBE request with a freshly-generated Call-ID and a new, unique "From" tag (see Section 4.1.2.1.)

If a SUBSCRIBE request to refresh a subscription fails with any error code other than those listed above, the original subscription is

still considered valid for the duration of the most recently known "Expires" value as negotiated by the most recent successful SUBSCRIBE transaction, or as communicated by a NOTIFY request in its "Subscription-State" header field "expires" parameter.

> Note that many such errors indicate that there may be a problem with the network or the notifier such that no further NOTIFY requests will be received.

When refreshing a subscription, a subscriber starts Timer N, set to 64*T1, when it sends the SUBSCRIBE request. If this Timer N expires prior to the receipt of a NOTIFY request, the subscriber considers the subscription terminated. If the subscriber receives a success response to the SUBSCRIBE request that indicates that no NOTIFY request will be generated -- such as the 204 response defined for use with the optional extension described in [RFC5839] -- then it MUST cancel Timer N.

4.1.2.3. Unsubscribing

Unsubscribing is handled in the same way as refreshing of a subscription, with the "Expires" header field set to "0". Note that a successful unsubscription will also trigger a final NOTIFY request.

The final NOTIFY request may or may not contain information about the state of the resource; subscribers need to be prepared to receive final NOTIFY requests both with and without state.

4.1.2.4. Confirmation of Subscription Creation

The subscriber can expect to receive a NOTIFY request from each node which has processed a successful subscription or subscription refresh. To ensure that subscribers do not wait indefinitely for a subscription to be established, a subscriber starts a Timer N, set to 64*T1, when it sends a SUBSCRIBE request. If this Timer N expires prior to the receipt of a NOTIFY request, the subscriber considers the subscription failed, and cleans up any state associated with the subscription attempt.

Until Timer N expires, several NOTIFY requests may arrive from different destinations (see Section 4.4.1). Each of these requests establish a new dialog usage and a new subscription. After the expiration of Timer N, the subscriber SHOULD reject any such NOTIFY requests that would otherwise establish a new dialog usage with a "481" response code.

Until the first NOTIFY request arrives, the subscriber should consider the state of the subscribed resource to be in a neutral

state. Event package specifications MUST define this "neutral state" in such a way that makes sense for their application (see Section 5.4.7).

Due to the potential for out-of-order messages, packet loss, and forking, the subscriber MUST be prepared to receive NOTIFY requests before the SUBSCRIBE transaction has completed.

Except as noted above, processing of this NOTIFY request is the same as in Section 4.1.3.

4.1.3. Receiving and Processing State Information

Subscribers receive information about the state of a resource to which they have subscribed in the form of NOTIFY requests.

Upon receiving a NOTIFY request, the subscriber should check that it matches at least one of its outstanding subscriptions; if not, it MUST return a "481 Subscription does not exist" response unless another 400- or 500-class response is more appropriate. The rules for matching NOTIFY requests with subscriptions that create a new dialog usage are described in Section 4.4.1. Notifications for subscriptions which were created inside an existing dialog match if they are in the same dialog and the "Event" header fields match (as described in Section 8.2.1).

If, for some reason, the event package designated in the "Event" header field of the NOTIFY request is not supported, the subscriber will respond with a "489 Bad Event" response.

To prevent spoofing of events, NOTIFY requests SHOULD be authenticated, using any defined SIP authentication mechanism, such as those described in sections 22.2 and 23 of [RFC3261].

NOTIFY requests MUST contain "Subscription-State" header fields which indicate the status of the subscription.

If the "Subscription-State" header field value is "active", it means that the subscription has been accepted and (in general) has been authorized. If the header field also contains an "expires" parameter, the subscriber SHOULD take it as the authoritative subscription duration and adjust accordingly. The "retry-after" and "reason" parameters have no semantics for "active".

If the "Subscription-State" value is "pending", the subscription has been received by the notifier, but there is insufficient policy information to grant or deny the subscription yet. If the header field also contains an "expires" parameter, the subscriber SHOULD

take it as the authoritative subscription duration and adjust accordingly. No further action is necessary on the part of the subscriber. The "retry-after" and "reason" parameters have no semantics for "pending".

If the "Subscription-State" value is "terminated", the subscriber MUST consider the subscription terminated. The "expires" parameter has no semantics for "terminated" -- notifiers SHOULD NOT include an "expires" parameter on a "Subscription-State" header field with a value of "terminated," and subscribers MUST ignore any such parameter, if present. If a reason code is present, the client should behave as described below. If no reason code or an unknown reason code is present, the client MAY attempt to re-subscribe at any time (unless a "retry-after" parameter is present, in which case the client SHOULD NOT attempt re-subscription until after the number of seconds specified by the "retry-after" parameter). The reason codes defined by this document are:

deactivated: The subscription has been terminated, but the subscriber SHOULD retry immediately with a new subscription. One primary use of such a status code is to allow migration of subscriptions between nodes. The "retry-after" parameter has no semantics for "deactivated".

probation: The subscription has been terminated, but the client SHOULD retry at some later time (as long as the resource's state is still relevant to the client at that time). If a "retry-after" parameter is also present, the client SHOULD wait at least the number of seconds specified by that parameter before attempting to re-subscribe.

rejected: The subscription has been terminated due to change in authorization policy. Clients SHOULD NOT attempt to re-subscribe. The "retry-after" parameter has no semantics for "rejected".

timeout: The subscription has been terminated because it was not refreshed before it expired. Clients MAY re-subscribe immediately. The "retry-after" parameter has no semantics for "timeout". This reason code is also associated with polling of resource state, as detailed in Section 4.4.3

giveup: The subscription has been terminated because the notifier could not obtain authorization in a timely fashion. If a "retry-after" parameter is also present, the client SHOULD wait at least the number of seconds specified by that parameter before attempting to re-subscribe; otherwise, the client MAY retry immediately, but will likely get put back into pending state.

noresource: The subscription has been terminated because the resource state which was being monitored no longer exists. Clients SHOULD NOT attempt to re-subscribe. The "retry-after" parameter has no semantics for "noresource".

invariant: The subscription has been terminated because the resource state is guaranteed not to change for the foreseeable future. This may be the case, for example, when subscribing to the location information of a fixed-location land-line telephone. When using this reason code, notifiers are advised to include a "retry-after" parameter with a large value (for example, 31536000 -- or one year) to prevent older, RFC 3265-compliant clients from periodically resubscribing. Clients SHOULD NOT attempt to resubscribe after receiving a reason code of "invariant," regardless of the presence of or value of a "retry-after" parameter.

Other specifications may define new reason codes for use with the "Subscription-State" header field.

Once the notification is deemed acceptable to the subscriber, the subscriber SHOULD return a 200 response. In general, it is not expected that NOTIFY responses will contain bodies; however, they MAY, if the NOTIFY request contained an "Accept" header field.

Other responses defined in [RFC3261] may also be returned, as appropriate. In no case should a NOTIFY transaction extend for any longer than the time necessary for automated processing. In particular, subscribers MUST NOT wait for a user response before returning a final response to a NOTIFY request.

4.1.4. Forking of SUBSCRIBE Requests

In accordance with the rules for proxying non-INVITE requests as defined in [RFC3261], successful SUBSCRIBE requests will receive only one 200-class response; however, due to forking, the subscription may have been accepted by multiple nodes. The subscriber MUST therefore be prepared to receive NOTIFY requests with "From:" tags which differ from the "To:" tag received in the SUBSCRIBE 200-class response.

If multiple NOTIFY requests are received in different dialogs in response to a single SUBSCRIBE request, each dialog represents a different destination to which the SUBSCRIBE request was forked. Subscriber handling in such situations varies by event package; see Section 5.4.9 for details.

4.2. Notifier Behavior

4.2.1. Subscription Establishment and Maintenance

Notifiers learn about subscription requests by receiving SUBSCRIBE requests from interested parties. Notifiers MUST NOT create subscriptions except upon receipt of a SUBSCRIBE request. However, for historical reasons, the implicit creation of subscriptions as defined in [RFC3515] is still permitted.

> [RFC3265] allowed the creation of subscriptions using means other than the SUBSCRIBE method. The only standardized use of this mechanism is the REFER method [RFC3515]. Implementation experience with REFER has shown that the implicit creation of a subscription has a number of undesirable effects, such as the inability to signal the success of a REFER request while signaling a problem with the subscription; and difficulty performing one action without the other. Additionally, the proper exchange of dialog identifiers is difficult without dialog re-use (which has its own set of problems; see Section 4.5).

4.2.1.1. Initial SUBSCRIBE Transaction Processing

In no case should a SUBSCRIBE transaction extend for any longer than the time necessary for automated processing. In particular, notifiers MUST NOT wait for a user response before returning a final response to a SUBSCRIBE request.

> This requirement is imposed primarily to prevent the non-INVITE transaction timeout timer F (see [RFC3261]) from firing during the SUBSCRIBE transaction, since interaction with a user would often exceed 64*T1 seconds.

The notifier SHOULD check that the event package specified in the "Event" header field is understood. If not, the notifier SHOULD return a "489 Bad Event" response to indicate that the specified event/event class is not understood.

The notifier SHOULD also perform any necessary authentication and authorization per its local policy. See Section 4.2.1.3.

The notifier MAY also check that the duration in the "Expires" header field is not too small. If and only if the expiration interval is greater than zero AND smaller than one hour AND less than a notifier-configured minimum, the notifier MAY return a "423 Interval Too Brief" error which contains a "Min-Expires" header field field. The "Min-Expires" header field is described in [RFC3261].

Once the notifier determines that it has enough information to create the subscription (i.e., it understands the event package, the subscription pertains to a known resource, and there are no other barriers to creating the subscription), it creates the subscription and a dialog usage, and returns a 200 (OK) response.

When a subscription is created in the notifier, it stores the event package name as part of the subscription information.

The "Expires" values present in SUBSCRIBE 200-class responses behave in the same way as they do in REGISTER responses: the server MAY shorten the interval, but MUST NOT lengthen it.

> If the duration specified in a SUBSCRIBE request is unacceptably short, the notifier may be able to send a 423 response, as described earlier in this section.

200-class responses to SUBSCRIBE requests will not generally contain any useful information beyond subscription duration; their primary purpose is to serve as a reliability mechanism. State information will be communicated via a subsequent NOTIFY request from the notifier.

The other response codes defined in [RFC3261] may be used in response to SUBSCRIBE requests, as appropriate.

4.2.1.2. Confirmation of Subscription Creation/Refreshing

Upon successfully accepting or refreshing a subscription, notifiers MUST send a NOTIFY request immediately to communicate the current resource state to the subscriber. This NOTIFY request is sent on the same dialog as created by the SUBSCRIBE response. If the resource has no meaningful state at the time that the SUBSCRIBE request is processed, this NOTIFY request MAY contain an empty or neutral body. See Section 4.2.2 for further details on NOTIFY request generation.

Note that a NOTIFY request is always sent immediately after any 200-class response to a SUBSCRIBE request, regardless of whether the subscription has already been authorized.

4.2.1.3. Authentication/Authorization of SUBSCRIBE Requests

Privacy concerns may require that notifiers apply policy to determine whether a particular subscriber is authorized to subscribe to a certain set of events. Such policy may be defined by mechanisms such as access control lists or real-time interaction with a user. In general, authorization of subscribers prior to authentication is not particularly useful.

SIP authentication mechanisms are discussed in [RFC3261]. Note that, even if the notifier node typically acts as a proxy, authentication for SUBSCRIBE requests will always be performed via a "401" response, not a "407". Notifiers always act as a user agents when accepting subscriptions and sending notifications.

> Of course, when acting as a proxy, a node will perform normal proxy authentication (using 407). The foregoing explanation is a reminder that notifiers are always UAs, and as such perform UA authentication.

If authorization fails based on an access list or some other automated mechanism (i.e., it can be automatically authoritatively determined that the subscriber is not authorized to subscribe), the notifier SHOULD reply to the request with a "403 Forbidden" or "603 Decline" response, unless doing so might reveal information that should stay private; see Section 6.2.

If the notifier owner is interactively queried to determine whether a subscription is allowed, a 200 (OK) response is returned immediately. Note that a NOTIFY request is still formed and sent under these circumstances, as described in the previous section.

If subscription authorization was delayed and the notifier wishes to convey that such authorization has been declined, it may do so by sending a NOTIFY request containing a "Subscription-State" header field with a value of "terminated" and a reason parameter of "rejected".

4.2.1.4. Refreshing of Subscriptions

When a notifier receives a subscription refresh, assuming that the subscriber is still authorized, the notifier updates the expiration time for subscription. As with the initial subscription, the server MAY shorten the amount of time until expiration, but MUST NOT increase it. The final expiration time is placed in the "Expires" header field in the response. If the duration specified in a SUBSCRIBE request is unacceptably short, the notifier SHOULD respond with a "423 Interval Too Brief" response.

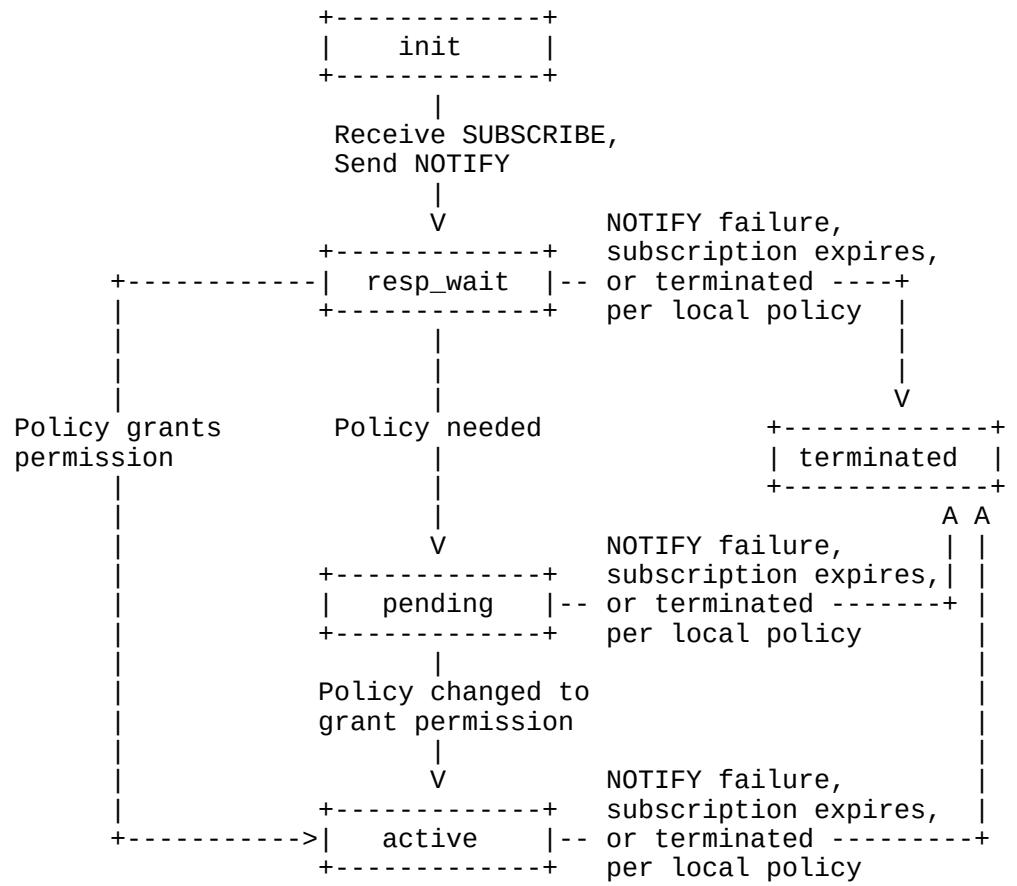
If no refresh for a notification address is received before its expiration time, the subscription is removed. When removing a subscription, the notifier SHOULD send a NOTIFY request with a "Subscription-State" value of "terminated" to inform it that the subscription is being removed. If such a request is sent, the "Subscription-State" header field SHOULD contain a "reason=timeout" parameter.

Clients can cause a subscription to be terminated immediately by sending a SUBSCRIBE request with an "Expires" header field set to '0'. Notifiers largely treat this the same way as any other subscription expiration: they send a NOTIFY request containing a "Subscription-State" of "terminated", with a reason code of "timeout." For consistency with state polling (see Section 4.4.3) and subscription refreshes, the notifier may choose to include resource state in this final NOTIFY request. However, in some cases, including such state makes no sense. Under such circumstances, the notifier may choose to omit state information from the terminal NOTIFY request.

The sending of a NOTIFY request when a subscription expires allows the corresponding dialog usage to be terminated, if appropriate.

4.2.2. Sending State Information to Subscribers

Notifiers use the NOTIFY method to send information about the state of a resource to subscribers. The notifier's view of a subscription is shown in the following state diagram. Events which result in a transition back to the same state are not represented in this diagram.



When a SUBSCRIBE request is answered with a 200-class response, the notifier MUST immediately construct and send a NOTIFY request to the subscriber. When a change in the subscribed state occurs, the notifier SHOULD immediately construct and send a NOTIFY request, unless the state transition is caused by a NOTIFY transaction failure. The sending of this NOTIFY message is also subject to authorization, local policy, and throttling considerations.

If the NOTIFY request fails due to expiration of SIP Timer F (transaction timeout), the notifier SHOULD remove the subscription.

This behavior prevents unnecessary transmission of state information for subscribers who have crashed or disappeared from the network. Because such transmissions will be sent multiple times, per the retransmission algorithm defined in [RFC3261] (instead of the typical single transmission for functioning clients), continuing to service them when no client is available

to acknowledge them could place undue strain on a network. Upon client restart or reestablishment of a network connection, it is expected that clients will send SUBSCRIBE requests to refresh potentially stale state information; such requests will re-install subscriptions in all relevant nodes.

If the NOTIFY transaction fails due to the receipt of a 404, 405, 410, 416, 480-485, 489, 501, or 604 response to the NOTIFY request, the notifier MUST remove the corresponding subscription. See [RFC5057] for further details and notes about the effect of error codes on dialogs and usages within dialog (such as subscriptions).

A notify error response would generally indicate that something has gone wrong with the subscriber or with some proxy on the way to the subscriber. If the subscriber is in error, it makes the most sense to allow the subscriber to rectify the situation (by re-subscribing) once the error condition has been handled. If a proxy is in error, the periodic sending of SUBSCRIBE requests to refresh the expiration timer will re-install subscription state once the network problem has been resolved.

NOTIFY requests MUST contain a "Subscription-State" header field with a value of "active", "pending", or "terminated". The "active" value indicates that the subscription has been accepted and has been authorized (in most cases; see Section 6.2). The "pending" value indicates that the subscription has been received, but that policy information is insufficient to accept or deny the subscription at this time. The "terminated" value indicates that the subscription is not active.

If the value of the "Subscription-State" header field is "active" or "pending", the notifier MUST also include in the "Subscription-State" header field an "expires" parameter which indicates the time remaining on the subscription. The notifier MAY use this mechanism to shorten a subscription; however, this mechanism MUST NOT be used to lengthen a subscription.

Including expiration information for active and pending subscriptions is necessary in case the SUBSCRIBE request forks, since the response to a forked SUBSCRIBE request may not be received by the subscriber. [RFC3265] allowed the notifier some discretion in the inclusion of this parameter, so subscriber implementations are warned to handle the lack of an "expires" parameter gracefully. Note well that this "expires" value is a parameter on the "Subscription-State" header field, NOT an "Expires" header field.

> The period of time for a subscription can be shortened to zero by the notifier. In other words, it is perfectly valid for a SUBSCRIBE request with a non-zero expires to be answered with a NOTIFY request that contains "Subscription-Status: terminated;reason=expired". This merely means that the notifier has shortened the subscription timeout to zero, and the subscription has expired instantaneously. The body may contain valid state, or it may contain a neutral state (see Section 5.4.7).

If the value of the "Subscription-State" header field is "terminated", the notifier SHOULD also include a "reason" parameter. The notifier MAY also include a "retry-after" parameter, where appropriate. For details on the value and semantics of the "reason" and "retry-after" parameters, see Section 4.1.3.

4.2.3. PINT Compatibility

The "Event" header field is considered mandatory for the purposes of this document. However, to maintain compatibility with PINT (see [RFC2848]), notifiers MAY interpret a SUBSCRIBE request with no "Event" header field as requesting a subscription to PINT events. If a notifier does not support PINT, it SHOULD return "489 Bad Event" to any SUBSCRIBE requests without an "Event" header field.

4.3. Proxy Behavior

Proxies need no additional behavior beyond that described in [RFC3261] to support SUBSCRIBE and NOTIFY transactions. If a proxy wishes to see all of the SUBSCRIBE and NOTIFY requests for a given dialog, it MUST add a Record-Route header field to the initial SUBSCRIBE request and all NOTIFY requests. It MAY choose to include Record-Route in subsequent SUBSCRIBE requests; however, these requests cannot cause the dialog’s route set to be modified.

Proxies that did not add a Record-Route header field to the initial SUBSCRIBE request MUST NOT add a Record-Route header field to any of the associated NOTIFY requests.

> Note that subscribers and notifiers may elect to use S/MIME encryption of SUBSCRIBE and NOTIFY requests; consequently, proxies cannot rely on being able to access any information that is not explicitly required to be proxy-readable by [RFC3261].

4.4. Common Behavior

4.4.1. Dialog Creation and Termination

Dialogs usages are created upon completion of a NOTIFY transaction for a new subscription, unless the NOTIFY request contains a "Subscription-State" of "terminated."

Because the dialog usage is established by the NOTIFY request, the route set at the subscriber is taken from the NOTIFY request itself, as opposed to the route set present in the 200-class response to the SUBSCRIBE request.

NOTIFY requests are matched to such SUBSCRIBE requests if they contain the same "Call-ID", a "To" header field "tag" parameter which matches the "From" header field "tag" parameter of the SUBSCRIBE request, and the same "Event" header field. Rules for comparisons of the "Event" header fields are described in Section 8.2.1.

A subscription is destroyed after a notifier sends a NOTIFY request with a "Subscription-State" of "terminated," or in certain error situations described elsewhere in this document. The subscriber will generally answer such final requests with a "200 OK" response (unless a condition warranting an alternate response has arisen). Except when the mechanism described in Section 4.5.2 is used, the destruction of a subscription results in the termination of its associated dialog.

> A subscriber may send a SUBSCRIBE request with an "Expires" header field of 0 in order to trigger the sending of such a NOTIFY request; however, for the purposes of subscription and dialog lifetime, the subscription is not considered terminated until the NOTIFY transaction with a "Subscription-State" of "terminated" completes.

4.4.2. Notifier Migration

It is often useful to allow migration of subscriptions between notifiers. Such migration may be effected by sending a NOTIFY request with a "Subscription-State" header field of "terminated", and a reason parameter of "deactivated". This NOTIFY request is otherwise normal, and is formed as described in Section 4.2.2.

Upon receipt of this NOTIFY request, the subscriber SHOULD attempt to re-subscribe (as described in the preceding sections). Note that this subscription is established on a new dialog, and does not re-use the route set from the previous subscription dialog.

The actual migration is effected by making a change to the policy (such as routing decisions) of one or more servers to which the

SUBSCRIBE request will be sent in such a way that a different node ends up responding to the SUBSCRIBE request. This may be as simple as a change in the local policy in the notifier from which the subscription is migrating so that it serves as a proxy or redirect server instead of a notifier.

Whether, when, and why to perform notifier migrations may be described in individual event packages; otherwise, such decisions are a matter of local notifier policy, and are left up to individual implementations.

4.4.3. Polling Resource State

A natural consequence of the behavior described in the preceding sections is that an immediate fetch without a persistent subscription may be effected by sending a SUBSCRIBE with an "Expires" of 0.

Of course, an immediate fetch while a subscription is active may be effected by sending a SUBSCRIBE request with an "Expires" equal to the number of seconds remaining in the subscription.

Upon receipt of this SUBSCRIBE request, the notifier (or notifiers, if the SUBSCRIBE request was forked) will send a NOTIFY request containing resource state in the same dialog.

Note that the NOTIFY requests triggered by SUBSCRIBE requests with "Expires" header fields of 0 will contain a "Subscription-State" value of "terminated", and a "reason" parameter of "timeout".

Polling of event state can cause significant increases in load on the network and notifiers; as such, it should be used only sparingly. In particular, polling SHOULD NOT be used in circumstances in which it will typically result in more network messages than long-running subscriptions.

When polling is used, subscribers SHOULD attempt to cache authentication credentials between polls so as to reduce the number of messages sent.

> Due to the requirement on notifiers to send a NOTIFY request immediately upon receipt of a SUBSCRIBE request, the state provided by polling is limited to the information that the notifier has immediate local access to when it receives the SUBSCRIBE request. If, for example, the notifier generally needs to retrieve state from another network server, then that state will be absent from the NOTIFY request that results from polling.

4.4.4. Allow-Events header field usage

The "Allow-Events" header field, if present, MUST include a comprehensive and inclusive list of tokens which indicates the event packages for which the User Agent can act as a notifier. In other words, a user agent sending an "Allow-Events" header field is advertising that it can process SUBSCRIBE requests and generate NOTIFY requests for all of the event packages listed in that header field.

Any user agent that can act as a notifier for one or more event packages SHOULD include an appropriate "Allow-Events" header field indicating all supported events in all methods which initiate dialogs and their responses (such as INVITE) and OPTIONS responses.

> This information is very useful, for example, in allowing user agents to render particular interface elements appropriately according to whether the events required to implement the features they represent are supported by the appropriate nodes.
> On the other hand, it doesn’t necessarily make much sense to indicate supported events inside a dialog established by a NOTIFY request if the only event package supported is the one associated with that subscription.

Note that "Allow-Events" header fields MUST NOT be inserted by proxies.

The "Allow-Events" header field does not include a list of the event template packages supported by an implementation. If a subscriber wishes to determine which event template packages are supported by a notifier, it can probe for such support by attempting to subscribe to the event template packages it wishes to use.

> For example: to check for support for the templatized package "presence.winfo", a client may attempt to subscribe to that event package for a known resource, using an "Expires" header value of 0. if the response is a 489 error code, then the client can deduce that "presence.winfo" is unsupported.

4.5. Targeting Subscriptions at Devices

[RFC3265] defined a mechanism by which subscriptions could share dialogs with invite usages and with other subscriptions. The purpose of this behavior was to allow subscribers to ensure that a subscription arrived at the same device as an established dialog. Unfortunately, the re-use of dialogs has proven to be exceedingly confusing. [RFC5057] attempted to clarify proper behavior in a variety of circumstances; however, the ensuing rules remain confusing

and prone to implementation error. At the same time, the mechanism described in [RFC5627] now provides a far more elegant and unambiguous means to achieve the same goal.

Consequently, the dialog re-use technique described in RFC 3265 is now deprecated.

This dialog-sharing technique has also historically been used as a means for targeting an event package at a dialog. This usage can be seen, for example, in certain applications of the REFER method [RFC3515]. With the removal of dialog re-use, an alternate (and more explicit) means of targeting dialogs needs to be used for this type of correlation. The appropriate means of such targeting is left up to the actual event packages. Candidates include the "Target-Dialog" header field [RFC4538], the "Join" header field [RFC3911], and the "Replaces" header field [RFC3891], depending on the semantics desired. Alternately, if the semantics of those header fields do not match the event package's purpose for correlation, event packages can devise their own means of identifying dialogs. For an example of this approach, see the Dialog Event Package [RFC4235].

4.5.1. Using GRUUs to Route to Devices

Notifiers MUST implement the Globally Routable User-Agent URI (GRUU) extension defined in [RFC5627], and MUST use a GRUU as their local target. This allows subscribers to explicitly target desired devices.

If a subscriber wishes to subscribe to a resource on the same device as an established dialog, it should check whether the remote contact in that dialog is a GRUU (i.e., whether it contains a "gr" URI parameter). If so, the subscriber creates a new dialog, using the GRUU as the request URI for the new SUBSCRIBE request.

> Because GRUUs are guaranteed to route to a a specific device, this ensures that the subscription will be routed to the same place as the established dialog.

4.5.2. Sharing Dialogs

For compatibility with older clients, subscriber and notifier implementations may choose to allow dialog sharing. The behavior of multiple usages within a dialog are described in [RFC5057].

Subscribers MUST NOT attempt to re-use dialogs whose remote target is a GRUU.

Note that the techniques described in this section are included for backwards compatibility purposes only. Because subscribers cannot re-use dialogs with a GRUU for their remote target, and because notifiers must use GRUUs as their local target, any two implementations that conform to this specification will automatically use the mechanism described in Section 4.5.1.

Further note that the prohibition on re-using dialogs does not exempt implicit subscriptions created by the REFER method. This means that implementations complying with this specification are required to use the "Target-Dialog" mechanism described in [RFC4538] when the remote target is a GRUU.

If a subscriber wishes to subscribe to a resource on the same device as an established dialog and the remote contact is not a GRUU, it MAY revert to dialog sharing behavior. Alternately, it MAY choose to treat the remote party as incapable of servicing the subscription (i.e., the same way it would behave if the remote party did not support SIP events at all).

If a notifier receives a SUBSCRIBE request for a new subscription on an existing dialog, it MAY choose to implement dialog sharing behavior. Alternately, it may choose to fail the SUBSCRIBE request with a 403 response. The error text of such 403 responses SHOULD indicate that dialog sharing is not supported.

To implement dialog sharing, subscribers and notifiers perform the following additional processing:

- When subscriptions exist in dialogs associated with INVITE-created application state and/or other subscriptions, these sets of application state do not interact beyond the behavior described for a dialog (e.g., route set handling). In particular, multiple subscriptions within a dialog are expire independently, and require independent subscription refreshes.

- If a subscription’s destruction leaves no other application state associated with the dialog, the dialog terminates. The destruction of other application state (such as that created by an INVITE) will not terminate the dialog if a subscription is still associated with that dialog. This means that, when dialog are re-used, then a dialog created with an INVITE does not necessarily terminate upon receipt of a BYE. Similarly, in the case that several subscriptions are associated with a single dialog, the dialog does not terminate until all the subscriptions in it are destroyed.

- Subscribers MAY include an "id" parameter in SUBSCRIBE request "Event" header field to allow differentiation between multiple subscriptions in the same dialog. This "id" parameter, if present, contains an opaque token which identifies the specific subscription within a dialog. An "id" parameter is only valid within the scope of a single dialog.

- If an "id" parameter is present in the SUBSCRIBE request used to establish a subscription, that "id" parameter MUST also be present in all corresponding NOTIFY requests.

- When a subscriber refreshes a the subscription timer, the SUBSCRIBE request MUST contain the same "Event" header field "id" parameter as was present in the SUBSCRIBE request that created the subscription. (Otherwise, the notifier will interpret the SUBSCRIBE request as a request for a new subscription in the same dialog).

- When a subscription is created in the notifier, it stores any "Event" header field "id" parameter as part of the subscription information (along with the event package name).

- If an initial SUBSCRIBE request is sent on a pre-existing dialog, a matching NOTIFY request merely creates a new subscription associated with that dialog.

4.6. CANCEL Requests for SUBSCRIBE and NOTIFY Transactions

Neither SUBSCRIBE nor NOTIFY requests can be canceled. If a UAS receives a CANCEL request that matches a known SUBSCRIBE or NOTIFY transaction, it MUST respond to the CANCEL request, but otherwise ignore it. In particular, the CANCEL request MUST NOT affect processing of the SUBSCRIBE or NOTIFY request in any way.

UACs SHOULD NOT send CANCEL requests for SUBSCRIBE or NOTIFY transactions.

5. Event Packages

This section covers several issues which should be taken into consideration when event packages based on the SUBSCRIBE and NOTIFY methods are proposed.

5.1. Appropriateness of Usage

When designing an event package using the methods described in this document for event notification, it is important to consider: is SIP an appropriate mechanism for the problem set? Is SIP being selected because of some unique feature provided by the protocol (e.g., user mobility), or merely because "it can be done?" If you find yourself defining event packages for notifications related to, for example, network management or the temperature inside your car's engine, you may want to reconsider your selection of protocols.

> Those interested in extending the mechanism defined in this document are urged to follow the development of "Guidelines for Authors of SIP Extensions" [RFC4485] for further guidance regarding appropriate uses of SIP.

Further, it is expected that this mechanism is not to be used in applications where the frequency of reportable events is excessively rapid (e.g., more than about once per second). A SIP network is generally going to be provisioned for a reasonable signaling volume; sending a notification every time a user's GPS position changes by one hundredth of a second could easily overload such a network.

5.2. Event Template-packages

Normal event packages define a set of state applied to a specific type of resource, such as user presence, call state, and messaging mailbox state.

Event template-packages are a special type of package which define a set of state applied to other packages, such as statistics, access policy, and subscriber lists. Event template-packages may even be applied to other event template-packages.

To extend the object-oriented analogy made earlier, event template-packages can be thought of as templatized C++ packages which must be applied to other packages to be useful.

The name of an event template-package as applied to a package is formed by appending a period followed by the event template-package name to the end of the package. For example, if a template-package called "winfo" were being applied to a package called "presence", the event token used in the "Event" header field would be "presence.winfo".

This scheme may be arbitrarily extended. For example, application of the "winfo" package to the the "presence.winfo" state of a resource would be represented by the name "presence.winfo.winfo". It naturally follows from this syntax that the order in which templates are specified is significant.

For example: consider a theoretical event template-package called "list". The event "presence.winfo.list" would be the application of the "list" template to "presence.winfo", which would presumably be a list of winfo state associated with presence. On the other hand, the event "presence.list.winfo" would represent the application of winfo to "presence.list", which would be represent the winfo state of a list of presence information.

Event template-packages must be defined so that they can be applied to any arbitrary package. In other words, event template-packages cannot be specifically tied to one or a few "parent" packages in such a way that they will not work with other packages.

5.3. Amount of State to be Conveyed

When designing event packages, it is important to consider the type of information which will be conveyed during a notification.

A natural temptation is to convey merely the event (e.g., "a new voice message just arrived") without accompanying state (e.g., "7 total voice messages"). This complicates implementation of subscribing entities (since they have to maintain complete state for the entity to which they have subscribed), and also is particularly susceptible to synchronization problems.

There are two possible solutions to this problem that event packages may choose to implement.

5.3.1. Complete State Information

In general, event packages need to be able to convey a well-defined and complete state, rather than just a stream of events. If it is not possible to describe complete system state for transmission in NOTIFY requests, then the problem set is not a good candidate for an event package.

For packages which typically convey state information that is reasonably small (on the order of 1 KB or so), it is suggested that event packages are designed so as to send complete state information whenever an event occurs.

In some circumstances, conveying the current state alone may be

insufficient for a particular class of events. In these cases, the event packages should include complete state information along with the event that occurred. For example, conveying "no customer service representatives available" may not be as useful as conveying "no customer service representatives available; representative sip:46@cs.xyz.int just logged off".

5.3.2. State Deltas

In the case that the state information to be conveyed is large, the event package may choose to detail a scheme by which NOTIFY requests contain state deltas instead of complete state.

Such a scheme would work as follows: any NOTIFY request sent in immediate response to a SUBSCRIBE request contains full state information. NOTIFY requests sent because of a state change will contain only the state information that has changed; the subscriber will then merge this information into its current knowledge about the state of the resource.

Any event package that supports delta changes to states MUST include a version number that increases by exactly one for each NOTIFY transaction in a subscription. Note that the state version number appears in the body of the message, not in a SIP header field.

If a NOTIFY request arrives that has a version number that is incremented by more than one, the subscriber knows that a state delta has been missed; it ignores the NOTIFY request containing the state delta (except for the version number, which it retains to detect message loss), and re-sends a SUBSCRIBE request to force a NOTIFY request containing a complete state snapshot.

5.4. Event Package Responsibilities

Event packages are not required to reiterate any of the behavior described in this document, although they may choose to do so for clarity or emphasis. In general, though, such packages are expected to describe only the behavior that extends or modifies the behavior described in this document.

Note that any behavior designated with "SHOULD" or "MUST" in this document is not allowed to be weakened by extension documents; however, such documents may elect to strengthen "SHOULD" requirements to "MUST" strength if required by their application.

In addition to the normal sections expected in standards-track RFCs and SIP extension documents, authors of event packages need to address each of the issues detailed in the following subsections.

For clarity: well-formed event package definitions contain sections addressing each of these issues, ideally in the same order and with the same titles as these subsections.

5.4.1. Event Package Name

This section, which MUST be present, defines the token name to be used to designate the event package. It MUST include the information which appears in the IANA registration of the token. For information on registering such types, see Section 7.

5.4.2. Event Package Parameters

If parameters are to be used on the "Event" header field to modify the behavior of the event package, the syntax and semantics of such header fields MUST be clearly defined.

Any "Event" header field parameters defined by an event package MUST be registered in the "Header Field Parameters and Parameter Values" registry defined by [RFC3968]. An "Event" header field parameter, once registered in conjunction with an event package, MUST NOT be re-used with any other event package. Non-event-package specifications MAY define "Event" header field parameters that apply across all event packages (with emphasis on "all", as opposed to "several"), such as the "id" parameter defined in this document. The restriction of a parameter to use with a single event package only applies to parameters that are defined in conjunction with an event package.

5.4.3. SUBSCRIBE Request Bodies

It is expected that most, but not all, event packages will define syntax and semantics for SUBSCRIBE request bodies; these bodies will typically modify, expand, filter, throttle, and/or set thresholds for the class of events being requested. Designers of event packages are strongly encouraged to re-use existing media types for message bodies where practical. See [RFC4288] for information on media type specification and registration.

This mandatory section of an event package defines what type or types of event bodies are expected in SUBSCRIBE requests (or specify that no event bodies are expected). It should point to detailed definitions of syntax and semantics for all referenced body types.

5.4.4. Subscription Duration

It is RECOMMENDED that event packages give a suggested range of times considered reasonable for the duration of a subscription. Such packages MUST also define a default "Expires" value to be used if

none is specified.

5.4.5. NOTIFY Request Bodies

The NOTIFY request body is used to report state on the resource being monitored. Each package MUST define what type or types of event bodies are expected in NOTIFY requests. Such packages MUST specify or cite detailed specifications for the syntax and semantics associated with such event body.

Event packages also MUST define which media type is to be assumed if none are specified in the "Accept" header field of the SUBSCRIBE request.

5.4.6. Notifier processing of SUBSCRIBE requests

This section describes the processing to be performed by the notifier upon receipt of a SUBSCRIBE request. Such a section is required.

Information in this section includes details of how to authenticate subscribers and authorization issues for the package.

5.4.7. Notifier generation of NOTIFY requests

This section of an event package describes the process by which the notifier generates and sends a NOTIFY request. This includes detailed information about what events cause a NOTIFY request to be sent, how to compute the state information in the NOTIFY, how to generate neutral or fake state information to hide authorization delays and decisions from users, and whether state information is complete or deltas for notifications; see Section 5.3. Such a section is required.

This section may optionally describe the behavior used to process the subsequent response.

5.4.8. Subscriber processing of NOTIFY requests

This section of an event package describes the process followed by the subscriber upon receipt of a NOTIFY request, including any logic required to form a coherent resource state (if applicable).

5.4.9. Handling of forked requests

Each event package MUST specify whether forked SUBSCRIBE requests are allowed to install multiple subscriptions.

If such behavior is not allowed, the first potential dialog-

establishing message will create a dialog. All subsequent NOTIFY requests which correspond to the SUBSCRIBE request (i.e., match "To", "From", "From" header field "tag" parameter, "Call-ID", "Event", and "Event" header field "id" parameter) but which do not match the dialog would be rejected with a 481 response. Note that the 200-class response to the SUBSCRIBE request can arrive after a matching NOTIFY request has been received; such responses might not correlate to the same dialog established by the NOTIFY request. Except as required to complete the SUBSCRIBE transaction, such non-matching 200-class responses are ignored.

If installing of multiple subscriptions by way of a single forked SUBSCRIBE request is allowed, the subscriber establishes a new dialog towards each notifier by returning a 200-class response to each NOTIFY request. Each dialog is then handled as its own entity, and is refreshed independent of the other dialogs.

In the case that multiple subscriptions are allowed, the event package MUST specify whether merging of the notifications to form a single state is required, and how such merging is to be performed. Note that it is possible that some event packages may be defined in such a way that each dialog is tied to a mutually exclusive state which is unaffected by the other dialogs; this MUST be clearly stated if it is the case.

5.4.10. Rate of notifications

Each event package is expected to define a requirement (SHOULD or MUST strength) which defines an absolute maximum on the rate at which notifications are allowed to be generated by a single notifier.

Each package MAY further define a throttle mechanism which allows subscribers to further limit the rate of notification.

5.4.11. State Aggregation

Many event packages inherently work by collecting information about a resource from a number of other sources -- either through the use of PUBLISH [RFC3903], by subscribing to state information, or through other state gathering mechanisms.

Event packages that involve retrieval of state information for a single resource from more than one source need to consider how notifiers aggregate information into a single, coherent state. Such packages MUST specify how notifiers aggregate information and how they provide authentication and authorization.

5.4.12. Examples

Event packages SHOULD include several demonstrative message flow diagrams paired with several typical, syntactically correct, and complete messages.

It is RECOMMENDED that documents describing event packages clearly indicate that such examples are informative and not normative, with instructions that implementors refer to the main text of the document for exact protocol details.

5.4.13. Use of URIs to Retrieve State

Some types of event packages may define state information which is potentially too large to reasonably send in a SIP message. To alleviate this problem, event packages may include the ability to convey a URI instead of state information; this URI will then be used to retrieve the actual state information.

[RFC4483] defines a mechanism that can be used by event packages to convey information in such a fashion.

6. Security Considerations

6.1. Access Control

The ability to accept subscriptions should be under the direct control of the notifier's user, since many types of events may be considered sensitive for the purposes of privacy. Similarly, the notifier should have the ability to selectively reject subscriptions based on the subscriber identity (based on access control lists), using standard SIP authentication mechanisms. The methods for creation and distribution of such access control lists is outside the scope of this document.

6.2. Notifier Privacy Mechanism

The mere act of returning certain 4xx and 6xx responses to SUBSCRIBE requests may, under certain circumstances, create privacy concerns by revealing sensitive policy information. In these cases, the notifier SHOULD always return a 200 (OK) response. While the subsequent NOTIFY request may not convey true state, it MUST appear to contain a potentially correct piece of data from the point of view of the subscriber, indistinguishable from a valid response. Information about whether a user is authorized to subscribe to the requested state is never conveyed back to the original user under these circumstances.

Individual packages and their related documents for which such a mode of operation makes sense can further describe how and why to generate such potentially correct data. For example, such a mode of operation is mandated by [RFC2779] for user presence information.

6.3. Denial-of-Service attacks

The current model (one SUBSCRIBE request triggers a SUBSCRIBE response and one or more NOTIFY requests) is a classic setup for an amplifier node to be used in a smurf attack.

Also, the creation of state upon receipt of a SUBSCRIBE request can be used by attackers to consume resources on a victim's machine, rendering it unusable.

To reduce the chances of such an attack, implementations of notifiers SHOULD require authentication. Authentication issues are discussed in [RFC3261].

6.4. Replay Attacks

Replaying of either SUBSCRIBE or NOTIFY requests can have detrimental effects.

In the case of SUBSCRIBE requests, attackers may be able to install any arbitrary subscription which it witnessed being installed at some point in the past. Replaying of NOTIFY requests may be used to spoof old state information (although a good versioning mechanism in the body of the NOTIFY requests may help mitigate such an attack). Note that the prohibition on sending NOTIFY requests to nodes which have not subscribed to an event also aids in mitigating the effects of such an attack.

To prevent such attacks, implementations SHOULD require authentication with anti-replay protection. Authentication issues are discussed in [RFC3261].

6.5. Man-in-the middle attacks

Even with authentication, man-in-the-middle attacks using SUBSCRIBE requests may be used to install arbitrary subscriptions, hijack existing subscriptions, terminate outstanding subscriptions, or modify the resource to which a subscription is being made. To prevent such attacks, implementations SHOULD provide integrity protection across "Contact", "Route", "Expires", "Event", and "To" header fields of SUBSCRIBE requests, at a minimum. If SUBSCRIBE request bodies are used to define further information about the state of the call, they SHOULD be included in the integrity protection

scheme.

Man-in-the-middle attacks may also attempt to use NOTIFY requests to spoof arbitrary state information and/or terminate outstanding subscriptions. To prevent such attacks, implementations SHOULD provide integrity protection across the "Call-ID", "CSeq", and "Subscription-State" header fields and the bodies of NOTIFY requests.

Integrity protection of message header fields and bodies is discussed in [RFC3261].

6.6. Confidentiality

The state information contained in a NOTIFY request has the potential to contain sensitive information. Implementations MAY encrypt such information to ensure confidentiality.

While less likely, it is also possible that the information contained in a SUBSCRIBE request contains information that users might not want to have revealed. Implementations MAY encrypt such information to ensure confidentiality.

To allow the remote party to hide information it considers sensitive, all implementations SHOULD be able to handle encrypted SUBSCRIBE and NOTIFY requests.

The mechanisms for providing confidentiality are detailed in [RFC3261].

7. IANA Considerations

(This section is not applicable until this document is published as an RFC.)

With the exception of Section 7.2, the subsections here are for current reference, carried over from the original specification. The only IANA actions requested here are updating all registry references that point to RFC 3265 to instead indicate this document, and creating the new "reason code" registry described in Section 7.2.

7.1. Event Packages

This document defines an event-type namespace which requires a central coordinating body. The body chosen for this coordination is the Internet Assigned Numbers Authority (IANA).

There are two different types of event-types: normal event packages,

and event template-packages; see Section 5.2. To avoid confusion, template-package names and package names share the same namespace; in other words, an event template-package are forbidden from sharing a name with a package.

Policies for registration of SIP event packages and SIP event package templates are defined in section 4.1 of [RFC5727].

Registrations with the IANA are required to include the token being registered and whether the token is a package or a template-package. Further, packages must include contact information for the party responsible for the registration and/or a published document which describes the event package. Event template-package token registrations are also required to include a pointer to the published RFC which defines the event template-package.

Registered tokens to designate packages and template-packages are disallowed from containing the character ".", which is used to separate template-packages from packages.

7.1.1. Registration Information

As this document specifies no package or template-package names, the initial IANA registry for event types will be empty. The remainder of the text in this section gives an example of the type of information to be maintained by the IANA; it also demonstrates all five possible permutations of package type, contact, and reference.

The table below lists the event packages and template-packages defined in "SIP-Specific Event Notification" [RFC xxxx]. Each name is designated as a package or a template-package under "Type".

Package Name	Type	Contact	Reference
example1	package	[Roach]	
example2	package	[Roach]	[RFC xxxx]
example3	package		[RFC xxxx]
example4	template	[Roach]	[RFC xxxx]
example5	template		[RFC xxxx]

PEOPLE

[Roach] Adam Roach <adam.roach@tekelec.com>

REFERENCES

[RFC xxxx] A.B. Roach, "SIP-Specific Event Notification", RFC XXXX,

7.1.2. Registration Template

To: ietf-sip-events@iana.org
Subject: Registration of new SIP event package

Package Name:

(Package names must conform to the syntax described in Section 8.2.1.)

Is this registration for a Template Package:

(indicate yes or no)

Published Specification(s):

(Template packages require a published RFC. Other packages may reference a specification when appropriate).

Person & email address to contact for further information:

7.2. Reason Codes

This document further defines "reason" codes for use in the "Subscription-State" header field (see Section 4.1.3).

Following the policies outlined in "Guidelines for Writing an IANA Considerations Section in RFCs" [RFC5226], new reason codes require a Standards Action.

Registrations with the IANA include the reason code being registered and a reference to a published document which describes the event package. Insertion of such values takes place as part of the RFC publication process or as the result of inter-SDO liaison activity, the result of which will be publication of an associated RFC. New reason codes must conform to the syntax of the ABNF "token" element defined in [RFC3261].

[RFC4660] defined a new reason code prior to the establishment of an IANA registry. We include its reason code ("badfilter") in the initial list of reason codes to ensure a complete registry.

The IANA registry for reason code will be initialized with the following values:

Reason Code	Reference
deactivated	[RFC xxxx]
probation	[RFC xxxx]
rejected	[RFC xxxx]
timeout	[RFC xxxx]
giveup	[RFC xxxx]
noresource	[RFC xxxx]
invariant	[RFC xxxx]
badfilter	[RFC 4660]

REFERENCES

[RFC xxxx] A.B. Roach, "SIP-Specific Event Notification", RFC XXXX, Monthname 20XX

[RFC 4660] Khartabil, H., Leppanen, E., Lonnfors, M., and J. Costa-Requena, "Functional Description of Event Notification Filtering", September 2006.

7.3. Header Field Names

This document registers three new header field names, described elsewhere in this document. These header fields are defined by the following information, which is to be added to the header field sub-registry under http://www.iana.org/assignments/sip-parameters.

Header Name: Allow-Events
Compact Form: u

Header Name: Subscription-State
Compact Form: (none)

Header Name: Event
Compact Form: o

7.4. Response Codes

This document registers two new response codes. These response codes are defined by the following information, which is to be added to the method and response-code sub-registry under http://www.iana.org/assignments/sip-parameters.

Response Code Number: 202
Default Reason Phrase: Accepted

Response Code Number: 489
Default Reason Phrase: Bad Event

8. Syntax

This section describes the syntax extensions required for event notification in SIP. Semantics are described in Section 4. Note that the formal syntax definitions described in this document are expressed in the ABNF format used in [RFC3261], and contain references to elements defined therein.

8.1. New Methods

This document describes two new SIP methods: SUBSCRIBE and NOTIFY.

8.1.1. SUBSCRIBE method

"SUBSCRIBE" is added to the definition of the element "Method" in the SIP message grammar.

Like all SIP method names, the SUBSCRIBE method name is case sensitive. The SUBSCRIBE method is used to request asynchronous notification of an event or set of events at a later time.

8.1.2. NOTIFY method

"NOTIFY" is added to the definition of the element "Method" in the SIP message grammar.

The NOTIFY method is used to notify a SIP node that an event which has been requested by an earlier SUBSCRIBE method has occurred. It may also provide further details about the event.

8.2. New Header Fields

8.2.1. "Event" Header Field

Event is added to the definition of the element "message-header field" in the SIP message grammar.

For the purposes of matching NOTIFY requests with SUBSCRIBE requests, the event-type portion of the "Event" header field is compared byte-by-byte, and the "id" parameter token (if present) is compared byte-by-byte. An "Event" header field containing an "id" parameter never matches an "Event" header field without an "id" parameter. No other parameters are considered when performing a comparison. SUBSCRIBE responses are matched per the transaction handling rules in [RFC3261].

> Note that the forgoing text means that "Event: foo; id=1234" would match "Event: foo; param=abcd; id=1234", but not "Event: foo" (id does not match) or "Event: Foo; id=1234" (event portion does not match).

This document does not define values for event-types. These values will be defined by individual event packages, and MUST be registered with the IANA.

There MUST be exactly one event type listed per event header field. Multiple events per message are disallowed.

The "Event" header field is defined only for use in SUBSCRIBE and NOTIFY requests, and other requests whose definition explicitly calls for its use. It MUST NOT appear in any other SIP requests, and MUST NOT appear in responses.

8.2.2. "Allow-Events" Header Field

Allow-Events is added to the definition of the element "general-header field" in the SIP message grammar. Its usage is described in Section 4.4.4.

User Agents MAY include the "Allow-Events" header field in any request or response, as long as its contents comply with the behavior described in Section 4.4.4.

8.2.3. "Subscription-State" Header Field

Subscription-State is added to the definition of the element "request-header field" in the SIP message grammar. Its usage is described in Section 4.1.3. "Subscription-State" header fields are defined for use in NOTIFY requests only. They MUST NOT appear in other SIP requests or responses.

8.3. New Response Codes

8.3.1. "202 Accepted" Response Code

For historical purposes, the 202 (Accepted) response code is added to the "Success" header field definition.

This document does not specify the use of the 202 response code in conjunction with the SUBSCRIBE or NOTIFY methods. Previous versions of the SIP Events Framework assigned specific meaning to the 202 response code.

Due to response handling in forking cases, any 202 response to a

SUBSCRIBE request may be absorbed by a proxy, and thus it can never be guaranteed to be received by the UAC. Furthermore, there is no actual processing difference for a 202 as compared to a 200; a NOTIFY request is sent after the subscription is processed, and it conveys the correct state. SIP interoperability tests found that implementations were handling 202 differently from 200, leading to incompatibilities. Therefore, the 202 response is being deprecated to make it clear there is no such difference and 202 should not be handled differently than 200.

Implementations conformant with the current specification MUST treat an incoming 202 response as identical to a 200 response, and MUST NOT generate 202 response codes to SUBSCRIBE or NOTIFY requests.

This document also updates [RFC4660], which reiterates the 202-based behavior in several places. Implementations compliant with the present document MUST NOT send a 202 response to a SUBSCRIBE request, and will send an alternate success response (such as 200) in its stead.

8.3.2. "489 Bad Event" Response Code

The 489 event response is added to the "Client-Error" header field field definition. "489 Bad Event" is used to indicate that the server did not understand the event package specified in a "Event" header field.

8.4. Augmented BNF Definitions

The Augmented BNF definitions for the various new and modified syntax elements follows. The notation is as used in [RFC3261], and any elements not defined in this section are as defined in SIP and the documents to which it refers.

```
   SUBSCRIBEm        = %x53.55.42.53.43.52.49.42.45 ; SUBSCRIBE in caps
   NOTIFYm           = %x4E.4F.54.49.46.59 ; NOTIFY in caps
   extension-method = SUBSCRIBEm / NOTIFYm / token

   Event             =  ( "Event" / "o" ) HCOLON event-type
                        *( SEMI event-param )
   event-type        =  event-package *( "." event-template )
   event-package     =  token-nodot
   event-template    =  token-nodot
   token-nodot       =  1*( alphanum / "-"  / "!" / "%" / "*"
                          / "_" / "+" / "`" / "'" / "~" )

   ; The use of the "id" parameter is deprecated; it is included
   ; for backwards compatibility purposes only.
   event-param      =  generic-param / ( "id" EQUAL token )

   Allow-Events =  ( "Allow-Events" / "u" ) HCOLON event-type
                  *(COMMA event-type)

   Subscription-State   = "Subscription-State" HCOLON substate-value
                          *( SEMI subexp-params )
   substate-value       = "active" / "pending" / "terminated"
                          / extension-substate
   extension-substate   = token
   subexp-params        =   ("reason" EQUAL event-reason-value)
                          / ("expires" EQUAL delta-seconds)
                          / ("retry-after" EQUAL delta-seconds)
                          / generic-param
   event-reason-value   =   "deactivated"
                          / "probation"
                          / "rejected"
                          / "timeout"
                          / "giveup"
                          / "noresource"
                          / "invariant"
                          / event-reason-extension
   event-reason-extension = token
```

9. References

9.1. Normative References

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

[RFC2848] Petrack, S. and L. Conroy, "The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call

Services", RFC 2848, June 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

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

[RFC4483] Burger, E., "A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", RFC 4483, May 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

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

[RFC5727] Peterson, J., Jennings, C., and R. Sparks, "Change Process for the Session Initiation Protocol (SIP) and the Real-time Applications and Infrastructure Area", BCP 67, RFC 5727, March 2010.

9.2. Informative References

[RFC2779] Day, M., Aggarwal, S., Mohr, G., and J. Vincent, "Instant Messaging / Presence Protocol Requirements", RFC 2779, February 2000.

[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

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

[RFC3891] Mahy, R., Biggs, B., and R. Dean, "The Session Initiation Protocol (SIP) "Replaces" Header", RFC 3891, September 2004.

[RFC3903] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.

[RFC3911] Mahy, R. and D. Petrie, "The Session Initiation Protocol (SIP) "Join" Header", RFC 3911, October 2004.

[RFC4235] Rosenberg, J., Schulzrinne, H., and R. Mahy, "An INVITE-Initiated Dialog Event Package for the Session Initiation Protocol (SIP)", RFC 4235, November 2005.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[RFC4485] Rosenberg, J. and H. Schulzrinne, "Guidelines for Authors of Extensions to the Session Initiation Protocol (SIP)", RFC 4485, May 2006.

[RFC4538] Rosenberg, J., "Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)", RFC 4538, June 2006.

[RFC4660] Khartabil, H., Leppanen, E., Lonnfors, M., and J. Costa-Requena, "Functional Description of Event Notification Filtering", RFC 4660, September 2006.

[RFC5057] Sparks, R., "Multiple Dialog Usages in the Session Initiation Protocol", RFC 5057, November 2007.

[RFC5839] Niemi, A. and D. Willis, "An Extension to Session Initiation Protocol (SIP) Events for Conditional Event Notification", RFC 5839, May 2010.

Appendix A. Acknowledgements

Thanks to the participants in the Events BOF at the 48th IETF meeting in Pittsburgh, as well as those who gave ideas and suggestions on the SIP Events mailing list. In particular, I wish to thank Henning Schulzrinne of Columbia University for coming up with the final three-tiered event identification scheme, Sean Olson for miscellaneous guidance, Jonathan Rosenberg for a thorough scrubbing of the -00 draft, and the authors of the "SIP Extensions for Presence" document for their input to SUBSCRIBE and NOTIFY request semantics.

I also owe a debt of gratitude to all the implementors who have provided feedback on areas of confusion or difficulty in the original specification. In particular, Robert Sparks' Herculean efforts

organizing, running, and collecting data from the SIPit events have proven invaluable in shaking out specification bugs. Robert Sparks is also responsible for untangling the dialog usage mess, in the form of RFC 5057 [RFC5057].

Appendix B. Changes from RFC 3265

This document represents several changes from the mechanism originally described in RFC 3265. This section summarizes those changes. Bug numbers refer to the identifiers for the bug reports kept on file at http://bugs.sipit.net/.

B.1. Bug 666: Clarify use of expires=xxx with terminated

Strengthened language in Section 4.1.3 to clarify that expires should not be sent with terminated, and must be ignored if received.

B.2. Bug 667: Reason code for unsub/poll not clearly spelled out

Clarified description of "timeout" in Section 4.1.3. (n.b., the text in Section 4.4.3 is actually pretty clear about this).

B.3. Bug 669: Clarify: SUBSCRIBE for a duration might be answered with a NOTIFY/expires=0

Added clarifying text to Section 4.2.2 explaining that shortening a subscription to zero seconds is valid. Also added sentence to Section 3.1.1 explicitly allowing shortening to zero.

B.4. Bug 670: Dialog State Machine needs clarification

The issues associated with the bug deal exclusively with the handling of multiple usages with a dialog. This behavior has been deprecated and moved to Section 4.5.2. This section, in turn, cites [RFC5057], which addresses all of the issues in Bug 670.

B.5. Bug 671: Clarify timeout-based removal of subscriptions

Changed Section 4.2.2 to specifically cite Timer F (so as to avoid ambiguity between transaction timeouts and retransmission timeouts).

B.6. Bug 672: Mandate expires= in NOTIFY

Changed strength of including of "expires" in a NOTIFY from SHOULD to MUST in Section 4.2.2.

B.7. Bug 673: INVITE 481 response effect clarification

This bug was addressed in [RFC5057].

B.8. Bug 677: SUBSCRIBE response matching text in error

Fixed Section 8.2.1 to remove incorrect "...responses and..." -- explicitly pointed to SIP for transaction response handling.

B.9. Bug 695: Document is not explicit about response to NOTIFY at subscription termination

Added text to Section 4.4.1 indicating that the typical response to a terminal NOTIFY is a "200 OK".

B.10. Bug 696: Subscription state machine needs clarification

Added state machine diagram to Section 4.1.2 with explicit handling of what to do when a SUBSCRIBE never shows up. Added definition of and handling for new Timer N to Section 4.1.2.4. Added state machine to Section 4.2.2 to reinforce text.

B.11. Bug 697: Unsubscription behavior could be clarified

Added text to Section 4.2.1.4 encouraging (but not requiring) full state in final NOTIFY request. Also added text to Section 4.1.2.3 warning subscribers that full state may or may not be present in the final NOTIFY.

B.12. Bug 699: NOTIFY and SUBSCRIBE are target refresh requests

Added text to both Section 3.1 and Section 3.2 explicitly indicating that SUBSCRIBE and NOTIFY are target refresh methods.

B.13. Bug 722: Inconsistent 423 reason phrase text

Changed reason code to "Interval Too Brief" in Section 4.2.1.1 and Section 4.2.1.4, to match 423 reason code in SIP [RFC3261].

B.14. Bug 741: guidance needed on when to not include Allow-Events

Added non-normative clarification to Section 4.4.4 regarding inclusion of Allow-Events in a NOTIFY for the one-and-only package supported by the notifier.

B.15. Bug 744: 5xx to NOTIFY terminates a subscription, but should not

Issue of subscription (usage) termination versus dialog termination is handled in [RFC5057]. The text in Section 4.2.2 has been updated to summarize the behavior described by 5057, and cites it for additional detail and rationale.

B.16. Bug 752: Detection of forked requests is incorrect

Removed erroneous "CSeq" from list of matching criteria in Section 5.4.9.

B.17. Bug 773: Reason code needs IANA registry

Added Section 7.2 to create and populate IANA registry.

B.18. Bug 774: Need new reason for terminating subscriptions to resources that never change

Added new "invariant" reason code to Section 4.1.3, ABNF syntax.

B.19. Clarify handling of Route/Record-Route in NOTIFY

Changed text in Section 4.3 mandating Record-Route in initial SUBSCRIBE and all NOTIFY requests, and adding "MAY" level statements for subsequent SUBSCRIBE requests.

B.20. Eliminate implicit subscriptions

Added text to Section 4.2.1 explaining some of the problems associated with implicit subscriptions, normative language prohibiting them. Removed language from Section 3.2 describing "non-SUBSCRIBE" mechanisms for creating subscriptions. Simplified language in Section 4.2.2, now that the soft-state/non-soft-state distinction is unnecessary.

B.21. Deprecate dialog re-use

Moved handling of dialog re-use and "id" handling to Section 4.5.2. It is documented only for backwards-compatibility purposes.

B.22. Rationalize dialog creation

Section 4.4.1 has been updated to specify that dialogs should be created when the NOTIFY arrives. Previously, the dialog was established by the SUBSCRIBE 200, or by the NOTIFY transaction. This was unnecessarily complicated; the newer rules are easier to implement (and result in effectively the same behavior on the wire).

B.23. Refactor behavior sections

Reorganized Section 4 to consolidate behavior along role lines (subscriber/notifier/proxy) instead of method lines.

B.24. Clarify sections that need to be present in event packages

Added sentence to Section 5 clarifying that event packages are expected to include explicit sections covering the issues discussed in this section.

B.25. Make CANCEL handling more explicit

Text in Section 4.6 now clearly calls out behavior upon receipt of a CANCEL. We also echo the "...SHOULD NOT send..." requirement from [RFC3261].

B.26. Remove State Agent Terminology

As originally planned, we anticipated a fairly large number of event packages that would move back and forth between end-user devices and servers in the network. In practice, this has ended up not being the case. Certain events, like dialog state, are inherently hosted at end-user devices; others, like presence, are almost always hosted in the network (due to issues like composition, and the ability to deliver information when user devices are offline). Further, the concept of State Agents is the most misunderstood by event package authors. In my expert review of event packages, I have yet to find one that got the concept of State Agents completely correct -- and most of them start out with the concept being 100% backwards from the way RFC 3265 described it.

Rather than remove the ability to perform the actions previously attributed to the widely misunderstood term "State Agent," we have simply eliminated this term. Instead, we talk about the behaviors required to create state agents (state aggregation, subscription notification) without defining a formal term to describe the servers that exhibit these behaviors. In effect, this is an editorial change to make life easier for event package authors; the actual protocol does not change as a result.

The definition of "State Agent" has been removed from Section 2. Section 4.4.2 has been retooled to discuss migration of subscription in general, without calling out the specific example of state agents. Section 5.4.11 has been focused on state aggregation in particular, instead of state aggregation as an aspect of state agents.

B.27. Miscellanous Changes

The following changes are relatively minor revisions to the document that resulted primarily from review of this document in the working group and IESG, rather than implementation reports.

- Clarified scope of Event header field parameters. In RFC3265, the scope is ambiguous, which causes problems with the RFC3968 registry. The new text ensures that Event header field parameters are unique across all event packages.

- Removed obsoleted language around IANA registration policies for event packages. Instead, we now cite RFC5727, which supersedes RFC3265, and is authoritative on event package registration policy.

- Several editorial updates after input from working group, including proper designation of "dialog usage" rather than "dialog" where appropriate.

- Clarified two normative statements about subscription termination by changing from plain English prose to RFC2119 language.

- Removed "Table 2" expansions, per WG consensus on how SIP table 2 is to be handled.

- Removed 202 response code.

- Clarified that "Allow-Events" does not list event template packages.

- Added clarification about proper response when the SUBSCRIBE indicates an unknown media type in its Accept header field.

- Minor clarifications to Route and Record-Route behavior.

- Added non-normative warning about the limitations of state polling.

- Added information about targeting subscriptions at specific dialogs.

- Added RFC 3261 to list of documents updated by this one (rather than the "2543" indicated by RFC3265).

- Clarified text in Section 3.1.1 explaining the meaning of "Expires: 0".

- Changed text in definition of "probation" reason code to indicate that subscribers don't need to re-subscribe if the associated state is no longer of use to them.

- Specified that the termination of a subscription due to a NOTIFY transaction failure does not require sending another NOTIFY message.

- Clarified how order of template application affects the meaning of an Event header field value. (e.g., "foo.bar.baz" is different than "foo.baz.bar").

Author's Address

Adam Roach
Tekelec
17210 Campbell Rd.
Suite 250
Dallas, TX 75252
US

Email: adam@nostrum.com

Network Working Group
Internet-Draft
Obsoletes: 4244 (if approved)
Intended status: Standards Track
Expires: April 4, 2014

M. Barnes
Polycom
F. Audet
Skype
S. Schubert
NTT
J. van Elburg
Detecon International Gmbh
C. Holmberg
Ericsson
Oct 2013

An Extension to the Session Initiation Protocol (SIP) for Request History Information

draft-ietf-sipcore-rfc4244bis-12.txt

Abstract

This document defines a standard mechanism for capturing the history information associated with a Session Initiation Protocol (SIP) request. This capability enables many enhanced services by providing the information as to how and why a SIP request arrives at a specific application or user. This document defines an optional SIP header field, History-Info, for capturing the history information in requests. The document also defines SIP header field parameters for the History-Info and Contact header fields to tag the method by which the target of a request is determined. In addition, this specification defines a value for the Privacy header field that directs the anonymization of values in the History-Info header field. This document obsoletes RFC 4244.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 4, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

Many services that SIP is anticipated to support require the ability to determine why and how a SIP request 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 Uniform Resource Locators (URLs) on a web page, "call history/ logging" style services within intelligent "call management" software for SIP User Agents (UAs), and calls to voicemail servers. Although SIP implicitly provides the retarget capabilities that enable SIP requests to be routed to chosen applications, there is a need for a standard mechanism within SIP for communicating the retargeting history of the requests. This "request history" information allows the receiving application to obtain information about how and why the SIP request arrived at the application/user.

This document defines a SIP header field, 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. SIP header field parameters are defined for the History-Info and Contact header fields to tag the method by which the target of a request is determined. This specification also defines a value, "history", for the Privacy header field. In addition a SIP option tag, "histinfo", is defined.

The History-Info header field provides a building block for development of SIP based applications and services. The requirements for the solution described in this specification are included in Appendix A. Example scenarios using the History-Info header field are available in [I-D.ietf-sipcore-rfc4244bis-callflows].

2. Conventions and Terminology

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

The term "retarget" is used in this specification to refer to the process of a SIP entity changing the request-URI [RFC3261, section 7.1] in a request based on the rules for determining request targets as described in Section 16.5 of [RFC3261] and of the subsequent forwarding of that request as described in step 2 in section 16.6 of [RFC3261]. This includes changing the Request-URI due to a location service lookup and redirect processing. This also includes internal (to a Proxy/SIP intermediary) changes of the URI prior to forwarding of the request.

The terms "location service", "forward", "redirect" and "AOR" are used consistently with the terminology in [RFC3261].

The terms "target user" is used in this specification as the human user associated with particular AoR or AoRs (in case the human user has multiple alias).

The references to "domain for which the SIP entity/Proxy/Intermediary is responsible" are consistent with and intended to convey the same context as the usage of that terminology in [RFC3261]. The applicability of History-Info to architectures or models outside the context of [RFC3261] is outside the scope of this specification.

3. Background

SIP implicitly provides retargeting capabilities that enable SIP requests to be routed to specific applications as defined in [RFC3261]. The motivation for capturing the request history is that in the process of retargeting a request, old routing information can be forever lost. This lost information may be important history that allows elements to which the request is retargeted to process the request in a locally defined, application-specific manner. This document defines a mechanism for transporting the request history. Application-specific behavior is outside the scope of this specification.

Current network applications for other protocols provide the ability for elements involved with the request to obtain additional information relating to how and why the request 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 that will receive some "referral" commission for generating this traffic

2. Email relaying whereby the recipient obtains a detailed "trace of the path" of the message from originator to receiver, including the time of each relay.

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:

- Some diagnostic information for debugging SIP requests.
- Capturing aliases and Globally Routable User Agent URIs (GRUUs) [RFC5627], which can be overwritten by a registrar or a "home proxy" (a proxy serving as the terminal point for routing an address-of-record) upon receipt of the initial request.
- Facilitating the use of limited use addresses (minted on demand) and sub-addressing.
- Preserving service specific URIs that can be overwritten by a downstream proxy, such as those defined in [RFC3087], and control of network announcements and IVR with SIP URI [RFC4240].

4. Overview

The fundamental functionality provided by the request history information is the ability to inform proxies and user agents (UAs) involved in processing a request about the history or progress of that request. The solution is to capture the Request-URIs as a request is retargeted, in a SIP header field: History-Info. This allows for the capturing of the history of a request that would be lost with the normal SIP processing involved in the subsequent retargeting of the request.

The History-Info header field is added to a Request when a new request is created by a user agent client (UAC) or forwarded by a Proxy, or when the target of a request is changed. It is possible for the target of a request to be changed by the same proxy/SIP intermediary multiple times (referred to as 'internal retargeting'). A SIP entity changing the target of a request in response to a redirect also propagates any History-Info header field from the initial request in the new request. The ABNF and detailed description of the History-Info header field parameters along with examples, is provided in Section 5. Section 6, Section 7 and Section 8 provide the detailed handling of the History-Info header field by SIP User Agents, Proxies and Redirect Servers respectively.

This specification also defines three new SIP header field parameters, "rc", "mp" and "np", for the History-Info and Contact header fields, to tag the method by which the target of a request is determined. Further detail on the use of these header field parameters is provided in Section 5.

This specification also defines a priv-value for the Privacy header, "history", that requires anonymization of all the History-Info header field entries in a Request or to a specific History-Info header field hi-entry as described above. Further detail is provided in Section 10.1.

In addition a SIP option tag, "histinfo", is defined. The use of this option tag is described in Section 6.1.

5. History-Info Header Field Protocol Structure

The History-Info header field defined in this specification defines the usage in out-of-dialog requests or initial requests for a dialog (e.g., INVITE, REGISTER, MESSAGE, REFER and OPTIONS, PUBLISH and SUBSCRIBE, etc.) and any non-100 provisional or final responses to these requests.

The following provides details for the information that is captured in the History-Info header field entries for each target used for forwarding a request:

- hi-targeted-to-uri: A mandatory parameter for capturing the Request-URI for the specific request as it is forwarded.

- hi-index: A mandatory parameter for History-Info reflecting the chronological order of the information, indexed to reflect the forking and retargeting of requests. The format for this parameter is a sequence of non-negative integers, 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 leaf. By adding the new entries in chronological order (i.e., following existing entries per the details in Section 10.3), including the index and sending the messages using a secure transport, the ordering of the History-Info header fields in the request is assured. In addition, applications may extract a variety of metrics (total number of retargets, total number of retargets from a specific branch, etc.) based upon the index values.

- hi-target-param: An optional parameter reflecting the mechanism by which the Request URI captured in the hi-targeted-to-uri in the History-Info header field value (hi-entry) was determined. This parameter is either an "rc", "mp" or "np" header field parameter, which is interpreted as follows:

"rc": The hi-targeted-to-URI represents a change in Request-URI while the target user remains the same. This occurs for example when user has multiple AoRs as an alias. The "rc" header field parameter contains the value of the hi-index in the hi-entry with an hi-targeted-to-uri that reflects the Request-URI that was retargeted

"mp": The hi-targeted-to-URI represents a user other than the target user associated with the Request-URI in the incoming request that was retargeted. This occurs when a request is statically or dynamically retargeted to another user represented by an AoR unassociated with the AoR of the original target user. The "mp" header field parameter contains the value of the hi-index in the hi-entry with an hi-targeted-to-uri that reflects the Request-URI that was retargeted, thus identifying the "mapped from" target.

"np": The hi-targeted-to-URI represents that there was no change in Request-URI. This would apply for example when a proxy merely forwards a request to a next hop proxy and loose routing is used. The "np" header field parameter contains the value of the hi-index in the hi-entry with an hi-targeted-to-uri that reflects the Request-URI that was copied unchanged into the request represented by this hi-entry. That value will usually be the hi-index of the parent hi-entry of this hi-entry.

- Extension (hi-extension): A parameter to allow for future optional extensions. As per [RFC3261], any implementation not understanding an extension MUST ignore it.

The ABNF syntax [RFC5234] for the History-Info header field and header field parameters is as follows:

```
History-Info = "History-Info" HCOLON hi-entry *(COMMA hi-entry)

hi-entry = hi-targeted-to-uri *(SEMI hi-param)

hi-targeted-to-uri = name-addr

hi-param = hi-index / hi-target-param / hi-extension

hi-index = "index" EQUAL index-val

index-val =  number *("." number)

number =  [ %31-39 *DIGIT ] DIGIT

hi-target-param = rc-param / mp-param / np-param

rc-param = "rc" EQUAL index-val

mp-param = "mp" EQUAL index-val

np-param = "np" EQUAL index-val

hi-extension = generic-param
```

The ABNF definitions for "generic-param", "name-addr", "HCOLON", "COMMA", "SEMI" and "EQUAL" are from [RFC3261].

This document also extends the "contact-params" for the Contact header field as defined in [RFC3261] with the "rc", "mp" and "np" header field parameters defined above.

In addition to the parameters defined by the ABNF, an hi-entry may also include a Reason header field and/or a Privacy header field, which are both included in the "headers" component of the hi-targeted-to-uri as described below:

- Reason: An optional parameter for History-Info, reflected in the History-Info header field by including the Reason header field [RFC3326] included in the hi-targeted-to-uri. A reason is included in the hi-targeted-to-uri of an hi-entry to reflect information received in a response to the request sent to that URI.

- Privacy: An optional parameter for History-Info, reflected in the History-Info header field values by including the Privacy Header [RFC3323] with a priv-value of "history", as defined in this document, included in the hi- targeted-to-uri or by adding the

Privacy header field with a priv-value of "history" to the request. The latter case indicates that the History-Info entries for all History-Info entries whose hi-targeted-to-uri has the same domain as the domain for which the SIP entity processing the message is responsible MUST be anonymized prior to forwarding, whereas the use of the Privacy header field included in the hi-targeted-to-uri means that a specific hi-entry MUST be anonymized.

Note that since both the Reason and Privacy parameters are included in the hi-targeted-to-uri, these fields will not be available in the case that the hi-targeted-to-uri is a Tel-URI [RFC3966].

The following provides examples of the format for the History-Info header field. Note that the backslash, CRLF and whitespace between the lines in the examples below are inserted for readability purposes only. Note, however, that History-Info can be broken into multiple lines due to the SWS (sep whitespace) that is part of HCOLON, COMMA and SEMI, and there can be multiple History-Info header fields due to the rule of section 7.3 [RFC3261]. Additional detailed examples are available in [I-D.ietf-sipcore-rfc4244bis-callflows].

```
History-Info: <sip:UserA@ims.example.com>;index=1;foo=bar

History-Info: <sip:UserA@ims.example.com?Reason=SIP%3B\
              cause%3D302>;index=1.1,\
              <sip:UserB@example.com?Privacy=history&Reason=SIP%3B\
              cause%3D486>;index=1.2;mp=1.1,\
              <sip:45432@192.168.0.3>;index=1.3;rc=1.2
```

5.1. History-Info Header Field Example Scenario

The following is an illustrative example of usage of History-Info.

In this example, Alice (sip:alice@atlanta.example.com) calls Bob (sip:bob@biloxi.example.com). Alice’s proxy in her home domain (sip: atlanta.example.com) forwards the request to Bob’s proxy (sip: biloxi.example.com). When the request arrives at sip: biloxi.example.com, it does a location service lookup for bob@biloxi.example.com and changes the target of the request to Bob’s Contact URIs provided as part of normal SIP registration. In this example, Bob is simultaneously contacted on a PC client and on a phone, and Bob answers on the PC client.

One important thing illustrated by this call flow is that without History-Info, Bob would "lose" the original target information or the initial request-URI, including any parameters in the request URI. Bob can recover that information by locating the last hi-entry with

an "rc" header field parameter. This "rc" header field parameter contains the index of the hi-entry containing the lost target information - i.e., the sip:bob@biloxi.example.com hi-entry with index=1.1. Note that in the 200 response to Alice, an hi-entry is not included for the fork to sip:bob@192.0.2.7 (index 1.1.1) since biloxi.example.com had not received a response from that fork at the time it sent the 200 OK that ultimately reached Alice.

Additional detailed examples are available in [I-D.ietf-sipcore-rfc4244bis-callflows].

Note: This example uses loose routing procedures.

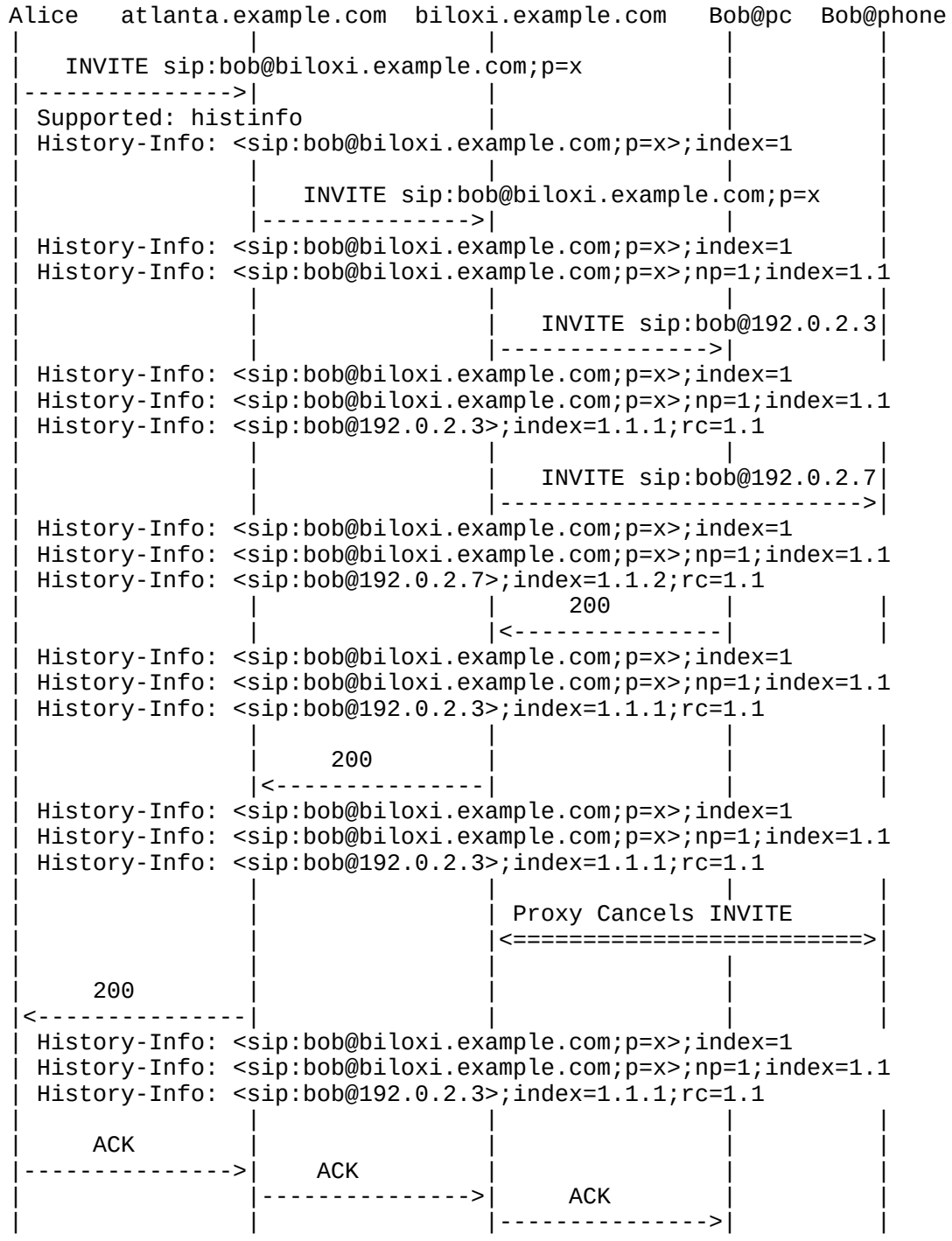

Alice
atlanta.example.com
biloxi.example.com
Bob@pc
Bob@phone
INVITE sip:bob@biloxi.example.com;p=x
Supported: histinfo
History-Info: <sip:bob@biloxi.example.com;p=x>;index=1
INVITE sip:bob@biloxi.example.com;p=x
History-Info: <sip:bob@biloxi.example.com;p=x>;index=1
History-Info: <sip:bob@biloxi.example.com;p=x>;np=1;index=1.1
INVITE sip:bob@192.0.2.3
History-Info: <sip:bob@biloxi.example.com;p=x>;index=1
History-Info: <sip:bob@biloxi.example.com;p=x>;np=1;index=1.1
History-Info: <sip:bob@192.0.2.3>;index=1.1.1;rc=1.1
INVITE sip:bob@192.0.2.7
History-Info: <sip:bob@biloxi.example.com;p=x>;index=1
History-Info: <sip:bob@biloxi.example.com;p=x>;np=1;index=1.1
History-Info: <sip:bob@192.0.2.7>;index=1.1.2;rc=1.1
200
History-Info: <sip:bob@biloxi.example.com;p=x>;index=1
History-Info: <sip:bob@biloxi.example.com;p=x>;np=1;index=1.1
History-Info: <sip:bob@192.0.2.3>;index=1.1.1;rc=1.1
200
History-Info: <sip:bob@biloxi.example.com;p=x>;index=1
History-Info: <sip:bob@biloxi.example.com;p=x>;np=1;index=1.1
History-Info: <sip:bob@192.0.2.3>;index=1.1.1;rc=1.1
Proxy Cancels INVITE
200
History-Info: <sip:bob@biloxi.example.com;p=x>;index=1
History-Info: <sip:bob@biloxi.example.com;p=x>;np=1;index=1.1
History-Info: <sip:bob@192.0.2.3>;index=1.1.1;rc=1.1
ACK
ACK
ACK

Figure 1: Basic Call

6. User Agent Handling of the History-Info Header Field

This section describes the processing specific to UAs(UACs, UASs and B2BUAs) for the History-Info header."

6.1. User Agent Client (UAC) Behavior

The UAC MUST include the "histinfo" option tag in the Supported header field in any out-of-dialog requests or initial requests for a dialog for which the UAC would like the History-Info header field in the response. When issuing a request, the UAC MUST follow the procedures in Section 9.2. In the case of an initial request, except where the UAC is part of a B2BUA, there is no cache of hi- entries with which to populate the History-Info header field and the hi-index is set to 1 per Section 10.3. When receiving a response the UAC MUST follow the procedures in Section 9.3.

If the UAC generates further forks of the initial request (either due to acting on a 3xx response or internally-directed forking to multiple destinations), the successive requests will add hi-entries with hi-indexes of 2, 3, etc.

6.2. User Agent Server (UAS) Behavior

When receiving a request, a UAS MUST follow the procedures defined in Section 9.2. When sending a response other than a 3xx response, a UAS MUST follows the procedures as defined in Section 9.4. When sending a 3xx response, the UAS MUST follow the procedures defined for a redirect server per Section 8. An application at the UAS can make use of the cached hi-entries as described in Section 11.

6.3. Back-2-Back User Agent (B2BUA) Behavior

A back-to-back user agent (B2BUA) MAY follow the behavior of a SIP intermediary, per Section 7, as an alternative to following the behavior of a user agent server (UAS) per Section 6.2 and a UAC per Section 6.1. In behaving as an intermediary, a B2BUA carries forward hi-entries received in requests at the UAS to requests being forwarded by the UAC, as well as carrying forward hi-entries in responses received at the UAC to the responses forwarded by the UAS, subject to privacy considerations per Section 10.1.

7. Proxy/Intermediary Handling of History-Info Header Fields

This section describes the procedures for proxies and other SIP intermediaries for the handling of the History-Info header fields for each of the following scenarios:

Receiving a Request: An intermediary MUST follow the procedures in Section 9.1 for the handling of hi-entries in incoming SIP requests.

Sending a Request: For each outgoing request relating to a target in the target set, the intermediary MUST follow the procedures of Section 9.2.

Receiving a Response or Timeout: An intermediary MUST follow the procedures of Section 9.3 when a SIP response is received or a request times out.

Sending a Response: An intermediary MUST follow the procedures of Section 9.4 for the handling of the hi-entries when sending a SIP response.

In some cases, an intermediary may retarget a request more than once before forwarding - i.e., a request is retargeted to a SIP entity that is "internal" to the intermediary before the same intermediary retargets the request to an external target . A typical example would be a proxy that retargets a request first to a different user (i.e., it maps to a different AOR) and then forwards to a registered contact bound to the same AOR. In this case, the intermediary MUST add a hi-entry for (each of) the internal target(s) per the procedures in Section 9.2. The intermediary MAY include a Reason header field in the hi-entry with the hi-targeted-to-uri that has been retargeted. Note, that this is shown in the INVITE (F6) in the example entitled "Sequentially Forking (History-Info in Response)" in [I-D.ietf-sipcore-rfc4244bis-callflows].

8. Redirect Server Handling of History-Info Header Fields

A redirect server MUST follow the procedures in Section 9.1 when it receives a SIP Request. A redirect server MUST follow the procedures in Section 9.4 when it sends a SIP Response. When generating the Contact header field in a 3xx response, the redirect server MUST add the appropriate "mp", "np" or "rc" header field parameter to each Contact header field as described in Section 10.4, if applicable.

9. Handling of History-Info Header Fields in Requests and Responses

This section describes the procedures for SIP entities for the handling of the History-Info header field in SIP requests and responses.

9.1. Receiving a Request

When receiving a request, a SIP entity MUST keep a copy of the hi-entries from the incoming request. This document describes this copy in terms of a cache containing the hi-entries associated with the request. The hi-entries MUST be added to the cache in the order in which they were received in the request.

If the Request-URI of the incoming request does not match the hi-targeted-to-uri in the last hi-entry (i.e., the previous SIP entity that sent the request did not include a History-Info header field), the SIP entity MUST add a hi-entry to end of the cache, on behalf of the previous SIP entity before proceeding to Section 9.2, as follows:

The SIP entity MUST set the hi-targeted-to-uri to the value of the Request-URI in the incoming request. If the Request-URI is a Tel-URI, it SHOULD be transformed into a SIP URI per section 19.1.6 of [RFC3261] before being added as a hi-targted-to-uri.

If privacy is required, the SIP entity MUST follow the procedures of Section 10.1.

The SIP entity MUST set the hi-index parameter as described in Section 10.3.

The SIP entity MUST NOT include an "rc", "mp" or "np" header field parameter.

9.2. Sending a Request with History-Info

When sending a request, a SIP entity MUST include all the hi-entries from the cache that was created per Section 9.1. In addition, the SIP entity MUST add a new hi-entry to the outgoing request, but the SIP entity MUST NOT add the hi-entry to the cache at this time. The hi-entries in the outgoing request's History-Info header field is the preorder of the tree of hi-entries, that is, by the lexicographic ordering of the hi-indexes. The new hi-entry is populated as follows:

hi-targeted-to-uri: The hi-targeted-to-uri MUST be set to the value of the Request-URI of the current (outgoing) request.

privacy: If privacy is required, the procedures of Section 10.1 MUST be followed.

hi-index: The SIP entity MUST include an hi-index for the hi-entry as described in Section 10.3.

rc/mp/np: The SIP entity MUST include an "rc", "mp" or "np" header field parameter in the hi-entry, if applicable, per the procedures in Section 10.4.

9.3. Receiving a Response with History-Info or Request Timeouts

When a SIP entity receives a non-100 response or a request times out, the SIP entity performs the following steps:

Step 1: Add hi-entry to cache

The SIP entity MUST add the hi-entry that was added to the request that received the non-100 response or timed out to the cache, if it was not already cached. The hi-entry MUST be added to the cache in ascending order as indicated by the values in the hi-index parameters of the hi-entries (e.g., 1.2.1 comes after 1.2 but before 1.2.2 or 1.3).

Step 2: Add Reason header field

If the response is not a 100 or 2xx response, the SIP entity adds one or more Reason header fields to the hi-targeted-to-uri in the (newly) cached hi-entry reflecting the SIP response code in the non-100 or non-2xx response, per the procedures of Section 10.2.

Step 3: Add additional hi-entries

The SIP entity MUST also add to the cache any hi-entries received in the response that are not already in the cache. This situation can occur when the entity that generated the non-100 response retargeted the request before generating the response. As per Step 1, the hi-entries MUST be added to the cache in ascending order as indicated by the values in the hi-index parameters of the hi-entries

It is important to note that the cache (and the request or response) does not contain hi-entries for requests that have not yet received a non-100 response, so there can be gaps in indices (e.g., 1.2 and 1.4 could be present but not 1.3).

9.4. Sending History-Info in Responses

When sending a response other than a 100, a SIP entity MUST include all the cached hi-entries in the response, subject to the privacy consideration in Section 10.1.2, and with the following exception: If the received request contained no hi-entries and there is no "histinfo" option tag in the Supported header field, the SIP entity MUST NOT include History-Info in the response.

10. Processing the History-Info Header Field

The following sections describe the procedures for processing the History-Info header field. These procedures are applicable to SIP entities such as Proxies/Intermediaries, Redirect Servers or User Agents.

10.1. Privacy in the History-Info Header Field

The privacy requirements for this document are described in Appendix A.2. Section 10.1.1 describes the insertion of the Privacy header field defined in [RFC3323] to indicate the privacy to be applied to the History-Info header field entries. Section 10.1.2 describes how to apply privacy to a request or response that is being forwarded, based on the presence of the Privacy header field.

10.1.1. Indicating Privacy

As with other SIP headers described in [RFC3323], the hi-targeted-to-uris in the History-Info header field can inadvertently reveal information about the initiator of the request. Thus, the UAC needs a mechanism to indicate that the hi-targeted-to-uris in the hi-entries need to be privacy protected. The Privacy header field is used by the UAC to indicate that privacy is to be applied to all the hi-entries in the request as follows:

- If the UAC is including a Privacy header field with a priv-value of "header" in the request, then the UAC SHOULD NOT include a priv-value of "history" in the Privacy header field in the Request.

- If the UAC is including any priv-values other than "header" in the Privacy header field, then the UAC MUST also include a priv-value of "history" in the Privacy header field in the Request.

- If the UAC is not including any priv-values in the Privacy header field in the request, then the UAC MUST add a Privacy header field, with a priv-value of "history", to the request. The UAC

MUST NOT include a priv-value of "critical" in the Privacy header field in the Request in this case.

In addition, the History-Info header field can reveal general routing and diverting information within an intermediary, which the intermediary wants to privacy protect. In this case, the intermediary MUST construct a Privacy header field with the single priv-value of "history" and include the Privacy header field in the hi-targeted-to-uri, for each new hi-entry created by the intermediary whose hi-targeted-to-uri it wishes to privacy protect. Note that the priv-value in the Privacy header for the incoming request does not necessarily influence whether the intermediary includes a Privacy header field in the hi-entries. For example, even if the Privacy header for the incoming request contained a priv-value of "none", the Proxy can still set a priv-value of "history" in the Privacy header field included in the hi-targeted-to-uri.

Finally, the UAS may not want to reveal the final reached target to the originator. In this case, the UAS MUST include a Privacy header field with a priv-value of "history" in the hi-targeted-to-uri in the last hi-entry, in the response. As noted above, the UAS of the request MUST NOT use any other priv-values in the Privacy header field included in the hi-entry.

10.1.2. Applying Privacy

When a SIP message is forwarded to a domain for which the SIP intermediary is not responsible, a Privacy Service at the boundary of the domain applies the appropriate privacy based on the value of the Privacy header field in the message header or in the "headers" component of the hi-targeted-to-uri in the individual hi-entries.

If there is a Privacy header field in the message header of a request or response, with a priv-value of "header" or "history", then all the hi-targeted-to-uris in the hi-entries, associated with the domain for which the SIP intermediary is responsible, are anonymized by the Privacy Service. The Privacy Service MUST change any hi-targeted-to-uris in these hi-entries that have not been anonymized (evidenced by their domain not being "anonymous.invalid") to anonymous URIs containing a domain of anonymous.invalid as recommended in section 4.1.1.3 of [RFC3323]. As defined in section 4.1.1.2 of [RFC3323] the recommendations of [RFC3261] for anonymyzing the URI Username SHOULD be followed (i.e., "anonymous" in the user portion of the URI). If there is a Privacy header field in the "headers" component of the hi-targeted-to-uri in the hi-entries, then the Privacy header field value MUST be removed from the hi- entry. Once all the appropriate hi-entries have been anonymized, the Privacy Service MUST remove the priv-value of "history" from the Privacy header field in the message

header of the request or response. If there are no remaining priv-values in the Privacy header field, the Privacy Service MUST remove the Privacy header field from the request or response per [RFC3323].

If there is not a Privacy header field in the message header of the request or response that is being forwarded, but there is a Privacy header field with a priv-value of "history" in the "headers" component in any of the hi-targeted-uris in the hi-entries associated with the domain for which a SIP intermediary is responsible, then the Privacy Service MUST update those hi-targeted-to-uris as described above. Any other priv-values in the Privacy header field in the "headers" component of the hi-targeted-to-uris in the hi-entries MUST be ignored. In any case, the Privacy Service MUST remove the Privacy header field from the "headers" compenent of the hi-targeted-to-uris in the hi-entries prior to forwarding.

10.2. Reason in the History-Info Header Field

A Reason header field is added to the "headers" component in an hi-targeted-to-uri when the hi-entry is added to the cache based upon the receipt of a SIP response that is neither a 100 nor a 2xx response, as described in Section 9.3. If the Reason header field is being added due to receipt of an explicit SIP response and the response contains any Reason header fields (see [RFC3326]), then the SIP entity MUST include the Reason header fields in the "headers" component of the hi-targeted-to-uri in the last hi-entry added to the cache, unless the hi-targeted-to-uri is a Tel-URI. If the SIP response does not contain a Reason header field, the SIP entity MUST include a Reason header field, containing the SIP Response Code, in the "headers" component of the hi-targeted-to-uri in the last hi-entry added to the cache, unless the hi-targeted-to-uri is a Tel-URI.

If a request has timed out (instead of being explicitly rejected), the SIP entity MUST update the cache as if the request received a SIP error response code of 408 "Request Timeout".

A request can receive multiple responses, that are neither 100 nor 2xx responses, which carry or imply (for responses without Reason headers, and for timeouts) multiple, possibly duplicated, reason-values to be applied to an hi- targeted-to-uri. In these situations, the SIP entity creating History-Info header value would choose the appropriate Reason header field value.

A SIP entity MAY also include a Reason header field in the "headers" component of an hi-targeted-to-uri containing the URI of a request that was retargeted as a result of internal retargeting.

If additional Reason header field parameters are defined in the

future per [RFC3326], the use of these Reason header field parameters for the History-Info header field MUST follow the same rules as described above.

10.3. Indexing in the History-Info Header Field

In order to maintain ordering and accurately reflect the retargeting of the request, the SIP entity MUST add a hi-index to each hi-entry. Per the syntax in Section 5, the hi-index consists of a series of nonnegative integer separated by dots (e.g., 1.1.2). Each dot reflects a SIP forwarding hop. The nonnegative integer following each dot reflects the order in which a request was retargeted at the hop. The highest nonnegative integer at each hop reflects the number of entities to which the request has been retargeted at the specific hop (i.e., the number of branches) at the time that the request represented by this hi-entry was generated. Thus, the indexing results in a logical tree representation for the history of the request and the hi-entries are given in the preorder of the tree.

The first index in a series of History-Info entries MUST be set to 1. In the case that a SIP entity (intermediary or UAS) adds a first hi-entry on behalf of the previous hop, the hi-index MUST be set to 1. For each forward hop (i.e., each new level of indexing), the last integers of the hi-indexes of the new requests MUST be generated starting at 1 and incrementing by 1 for each additional request.

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

1. Forwarding a request without changing the target: In the case of a request that is being forwarded without changing the target, the hi-index reflects the increasing length of the branch. In this case, the SIP entity MUST read the value from the History-Info header field in the received request and MUST add another level of indexing by appending the dot delimiter followed by an initial value of 1 for the new level. For example, if the hi-index in the last History-Info header field in the received request is 1.1, a proxy would add a hi-entry with an hi-index of 1.1.1 and forward the request.

2. Retargeting within a processing entity - 1st instance: For the first instance of retargeting within a processing entity, the SIP entity MUST calculate the hi-index as prescribed for basic forwarding.

3. Retargeting within a processing entity - subsequent instance: For each subsequent retargeting of a request by the same SIP entity, the SIP entity MUST calculate and add the hi-index for each new branch by incrementing the rightmost value from the hi-index in

the last hi-entry. Per the example above, the hi-index in the next request forwarded by this same SIP entity would be 1.1.2.

4. Retargeting based upon a Response: In the case of retargeting due to a specific response (e.g., 302), the SIP entity MUST calculate the hi-index calculated per rule 3. That is, the rightmost value of the hi-index MUST be incremented (i.e., a new branch is created). For example, if the hi-index in the History-Info header field of the sent request is 1.2 and the response to the request is a 302, then the hi-index in the History-Info header field for the new hi-targeted-to-URI would be 1.3.

5. Forking requests: If the request forwarding is done in multiple forks (sequentially or in parallel), the SIP entity MUST set the hi-index for each hi-entry for each forked request per the rules above, with each new request having a unique index. Each index MUST be sequentially assigned. For example, if the index in the last History-Info header field in the received request is 1.1, this processing entity would initialize its index to 1.1.1 for the first fork, 1.1.2 for the second, and so forth (see Figure 1 for an example). Note, that in the case of parallel forking, only the hi-entry corresponding to the fork is included in the request because no response can yet have been received for any of the parallel forked requests.

6. Missing entry: If the request clearly has a gap in the hi-entry (i.e., the last hi-entry and Request-URI differ), the entity adding an hi-entry MUST add a single index with a value of "0" (i.e., the non-negative integer zero) prior to adding the appropriate index for the action to be taken. For example, if the index of the last hi-entry in the request received was 1.1.2 and there was a missing hi-entry and the request was being forwarded to the next hop, the resulting index will be 1.1.2.0.1. In the case of requests which are forked by a proxy that does not support History-Info, it is possible for hi-entries generated by different entities to have the same index - i.e., each entity supporting History-Info would receive a forked request with the same hi-index to which they would add the value of ".0" prior to adding the appropriate index. Thus, in the previous example, each of the next hop entities would generate an hi-index of 1.1.2.0.1.

10.4. Mechanism for Target Determination in the History-Info Header Field

This specification defines three header field parameters, "rc", "mp" and "np". The header field parameters "rc" and "mp" indicate the mechanism by which a new target for a request is determined. The

header field "np" reflects that the target has not changed. All parameters contain an index whose value is the hi-index of the hi-entry with an hi-targeted-to-uri that represents the Request-URI that was retargeted.

The SIP entity MUST determine the specific parameter field to be included in the hi-target-param, in the History-Info header field, as the targets are added to the target set per the procedures in section 16.5 of [RFC3261] or per section 8.1.3.4 [RFC3261] in the case of retargeting to a contact URI received in a 3xx response. In the latter case, the specific header field parameter in the Contact header field becomes the header field parameter that is used in the hi-entry when the request is retargeted. If the Contact header field does not contain an "rc" or "mp" header field parameter, then the SIP entity MUST NOT include an "rc" or "mp" header field parameter in the hi-target-param in the hi-entry when the request is retargeted to a contact URI received in a 3xx response. This is because the redirect server is the only element with any knowledge on how the target was determined. Note, that the "np" header field parameter is not applicable in the case of redirection.

The SIP entity (intermediary or redirect server) determines the specific header field parameter ("rc", "mp" or "np") to be used based on the following criteria:

- "rc": The Request-URI has changed while retaining the target user associated with the original Request-URI prior to retargeting.

- "mp": The target was determined based on a mapping to a user other than the target user associated with the Request-URI being retargeted.

- "np": The target hasn’t changed and the associated Request-URI remained the same.

Note that there are two scenarios by which the "mp" header field parameter can be derived.

- The mapping was done by the receiving entity on its own authority, in which case the mp-value is the parent index of the hi-entry’s index.

- The mapping was done due to receiving a 3xx response, in which case the mp-value is an earlier sibling or descendant of an earlier sibling of the hi-entry’s index, that of the downstream request which received the 3xx response.

11. Application Considerations

History-Info provides a very flexible building block that can be used by intermediaries and UAs for a variety of services. Prior to any application usage of the History-Info header field parameters, the SIP entity that processes the hi-entries MUST evaluate the hi-entries. The SIP entity MUST be prepared to process effectively messages whose hi-entries show evidence of "gaps", that is, situations that reveal that not all of the forks of the request have been recorded in the hi-entries. Gaps are possible if the request is forwarded through intermediaries that do not support the History-Info header field and are reflected by the existence of hi-entries with a nonnegative integer of "0" e.g. "1.1.0.1". Gaps are also possible in the case of parallel forking if there is an outstanding request at the time the SIP entity sends a message. In addition, gaps may introduce the possibility of duplicate values for the hi-index in the case that a proxy that does not support History-Info forks a request. If gaps are detected, the SIP entity MUST NOT treat this as an error, but SHOULD indicate to any applications that there are gaps. The interpretation of the information in the History-Info header field depends upon the specific application; an application might need to provide special handling in some cases where there are gaps.

The following describes some categories of information that applications can use:

1. Complete history information - e.g., for debug or other operational and management aspects, optimization of determining targets to avoid retargeting to the same URI, etc. This information is relevant to proxies, UACs and UASs.

2. Hi-entry with the index that matches the value of the "rc" header field parameter in the last hi-entry with a "rc" header field parameter in the Request received by a UAS - i.e., the last AOR that was retargeted to a contact based on an AOR-to-contact binding.

3. Hi-entry with the index that matches the value of the "mp" header field parameter in the last hi-entry with a "mp" header field parameter in the hi-target-param in the Request received by a UAS - i.e., the last Request URI that was mapped to reach the destination.

4. Hi-entry with the index that matches the value of the "rc" header field parameter in the first hi-entry with a "rc" header field parameter in the Request received by a UAS. Note, this would be the original AoR if all the entities involved support the History-Info header field and there is absence of an "mp" header

field parameter prior to the "rc" header field parameter in the hi-target-param in the History-Info header field. However, there is no guarantee that all entities will support History-Info, thus the hi-entry that matches the value of the "rc" header field parameter of the first hi-entry with an "rc" header field parameter in the hi-target-param within the domain associated with the target URI at the destination is more likely to be useful.

5. Hi-entry with the index that matches the value of "mp" header field parameter in the first hi-entry with an "mp" header field parameter in the Request received by a UAS. Note, this would be the original mapped URI if all entities supported the History-Info header field. However, there is no guarantee that all entities will support History-Info, thus the hi-entry that matches the value of the "mp" header field parameter of the first hi-entry with an "mp" header field parameter within the domain associated with the target URI at the destination is more likely to be useful.

In many cases, applications are most interested in the information within a particular domain(s), thus only a subset of the information is required.

Some applications may use multiple types of information. For example, an Automatic Call Distribution (ACD)/Call center application that utilizes the hi-entry which index matches the value of the "mp" header field parameter of the first hi-entry with an "mp" header field parameter, may also display other agents, reflected by other hi-entries prior to entries with hi-target value of "rc" header field parameter, to whom the call was targeted prior to its arrival at the current agent. This could allow the agent the ability to decide how they might forward or reroute the call if necessary (avoiding agents that were not previously available for whatever reason, etc.).

Since support for History-Info header field is optional, a service MUST define default behavior for requests and responses not containing History-Info header fields. For example, an entity may receive an incomplete set of hi-entries or hi-entries which are not tagged appropriately with an hi-target-param in the case of entries added by entities that are only compliant to RFC4244. This may not impact some applications (e.g., debug), however, it could require some applications to make some default assumptions in this case. For example, in an ACD scenario, the application could select the oldest hi-entry with the domain associated with the ACD system and display that as the original called party. Depending upon how and where the request may have been retargeted, the complete list of agents to whom the call was targeted may not be available.

12. Application Specific Usage

The following are possible (non-normative) application-specific usages of History-Info.

12.1. PBX Voicemail

A voicemail system typically requires the original called party information to determine the appropriate mailbox so an appropriate greeting can be provided and the appropriate party notified of the message.

The original target is determined by finding the first hi-entry tagged with "rc" and using the hi-entry referenced by the index of "rc" header field parameter as the target for determining the appropriate mailbox. This hi-entry is used to populate the "target" URI parameter as defined in [RFC4458] The VMS can look at the last hi-entry and find the target of the mailbox by looking at the URI entry in the "target" URI parameter in the hi-entry.

This example usage does not work properly in the presence of forwarding that takes place before the call reaches the company in that case not the first hi-entry with an rc value, but the first hi-entry with an rc value following an mp entry needs to be picked. Further detail for this example can be found in the call flow entitled "PBX Voicemail Example" in [I-D.ietf-sipcore-rfc4244bis-callflows].

Note that in the case where there is no entry tagged with "rc", a VMS can follow the procedures, as defined in [RFC4458], for the "Interaction with Request History Information".

12.2. Consumer Voicemail

The voicemail system in these environment typically requires the last called party information to determine the appropriate mailbox so an appropriate greeting can be provided and the appropriate party notified of the message.

The last target is determined by finding the hi-entry referenced by the index of last hi-entry tagged with "rc" for determining the appropriate mailbox. This hi-entry is used to populate the "target" URI parameter as defined in [RFC4458]. The VMS can look at the last hi-entry and find the target of the mailbox by looking for the "target" URI parameter in the hi-entry. Further detail for this example can be found in the call flow entitled "Consumer Voicemail Example" in [I-D.ietf-sipcore-rfc4244bis-callflows].

In the case where there is no entry tagged with "rc", a VMS can follow the procedures, as defined in [RFC4458], for the "Interaction with Request History Information".

13. Security Considerations

The security requirements for this specification are specified in Appendix A.1.

This document defines a header field for SIP. The use of the Transport Layer Security (TLS) protocol [RFC5246] as a mechanism to ensure the overall confidentiality of the History-Info header fields (SEC-req-4) is strongly RECOMMENDED. If TLS is NOT used, the intermediary MUST ensure that the messages are only sent within an environment that is secured by other means or that the messages don't leave the intermediary's domain. This results in History-Info having at least the same level of security as other headers in SIP that are inserted by intermediaries. With TLS, History-Info header fields are no less, nor no more, secure than other SIP header fields, which generally have even more impact on the subsequent processing of SIP sessions than the History-Info header field.

Note that while using the SIPS scheme (as per [RFC5630]) protects History-Info from tampering by arbitrary parties outside the SIP message path, all the intermediaries on the path are trusted implicitly. A malicious intermediary could arbitrarily delete, rewrite, or modify History-Info. This specification does not attempt to prevent or detect attacks by malicious intermediaries.

In terms of ensuring the privacy of hi-entries, the same security considerations as those described in [RFC3323] apply. The privacy service that's defined in [RFC3323] MUST also support the new privacy header field priv-value of "history" and anonymize hi-entries in the case of a priv-value of "header" as described in Section 10.1.2.

14. IANA Considerations

This document requires several IANA registrations detailed in the following sections.

This document obsoletes [RFC4244] but uses the same SIP header field name, Privacy header field and Option tag. The IANA registry needs to update the references to [RFC4244] with [RFC XXXX], where XXXX is the RFC number for this document.

14.1. Registration of New SIP History-Info Header Field

This document defines a SIP header field name: History-Info and an option tag: histinfo. The following updates have been made to http:///www.iana.org/assignments/sip-parameters.

The following row has been updated in the header field section:

Header Name	Compact Form	Reference
History-Info	none	[RFC XXXX]

The following has been updated in the Options Tags section:

Name	Description	Reference
histinfo	When used with the Supported header field, this option tag indicates the UAC supports 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.	[RFC XXXX]

Note to RFC Editor: Please replace RFC XXXX with the RFC number of this specification.

14.2. Registration of "history" for SIP Privacy Header Field

This document defines a priv-value for the SIP Privacy header field: history. The following updates have been made to http://www.iana.org/assignments/sip-priv-values. The following has been updated in the registration for the SIP Privacy header field:

Name	Description	Registrant	Reference
history	Privacy requested for History-Info header fields(s)	Mary Barnes mary.ietf.barnes@gmail.com	[RFC XXXX]

Note to RFC Editor: Please replace RFC XXXX with the RFC number of this specification.

14.3. Registration of Header Field Parameters

This specification defines the following new SIP header field parameters in the SIP Header Field parameter sub-registry in the SIP Parameter Registry, http:/www.iana.org/assignments/sip-parameters.

Header Field	Parameter Name	Predefined Values	Reference
History-Info	mp	No	[RFC xxxx]
History-Info	rc	No	[RFC xxxx]
History-Info	np	No	[RFC xxxx]
Contact	mp	No	[RFC xxxx]
Contact	rc	No	[RFC xxxx]
Contact	np	No	[RFC xxxx]

Note to RFC Editor: Please replace RFC XXXX with the RFC number of this specification.

15. Acknowledgements

Jonathan Rosenberg et al produced the document that provided additional use cases precipitating the requirement for the new header parameters to capture the method by which a Request URI is determined. The authors would like to acknowledge the constructive feedback provided by Ian Elz, Paul Kyzivat, John Elwell, Hadriel Kaplan, Marianne Mohali, Brett Tate, and Dale Worley. John Elwell also provided excellent suggestions in terms of document structure. Dan Romascanu performed the Gen-ART review.

Mark Watson, Cullen Jennings and Jon Peterson provided significant input into the initial work that resulted in the development of of [RFC4244]. The editor would like to acknowledge the constructive feedback provided by Robert Sparks, Paul Kyzivat, Scott Orton, John Elwell, Nir Chen, Palash Jain, Brian Stucker, Norma Ng, Anthony Brown, Jayshree Bharatia, Jonathan Rosenberg, Eric Burger, Martin Dolly, Roland Jesske, Takuya Sawada, Sebastien Prouvost, and Sebastien Garcin in the development of [RFC4244].

The editor would like to acknowledge the significant input from Rohan Mahy on some of the normative aspects of the ABNF for [RFC4244], particularly around the need for and format of the index and around the security aspects.

16. Changes from RFC 4244

This RFC replaces [RFC4244].

Deployment experience with [RFC4244] over the years has shown a number of issues, warranting an update:

- In order to make [RFC4244] work in "real life", one needs to make "assumptions" on how History-Info is used. For example, many implementations filter out many entries, and only leave specific entries corresponding, for example, to first and last redirection. Since vendors uses different rules, it causes significant interoperability issues.
- [RFC4244] is overly permissive and evasive about recording entries, causing interoperability issues.
- The examples in the call flows had errors, and confusing because they often assume "loose routing".
- [RFC4244] has lots of repetitive and unclear text due to the combination of requirements with solution.
- [RFC4244] gratuitously mandates the use of TLS on every hop. No existing implementation enforces this rule, and instead, the use of TLS or not is a general SIP issue, not an [RFC4244] issue per se.
- [RFC4244] does not include clear procedures on how to deliver current target URI information to the UAS when the Request-URI is replaced with a contact.
- [RFC4244] does not allow for marking History-Info entries for easy processing by User Agents.

The following summarizes the functional changes between this specification and [RFC4244]:

1. Added header field parameters to capture the specific method by which a target is determined to facilitate processing by users of the History-Info header field entries. A specific header field parameter is captured for each of the target URIs as the target set is determined (per section 16.5 of [RFC3261]). The header field parameter is used in both the History-Info and the Contact header fields.
2. Added a way to indicate a gap in History-Info by adding a non-negative integer of "0".

3. Rather than recommending that entries be removed in the case of certain values of the Privacy header field, the entries are anonymized.

4. Updated the security section to be equivalent to the security recommendations for other SIP header fields inserted by intermediaries.

5. Removed Appendix B since a separate call flow document is being published as a companion to this document.

The first 2 changes are intended to facilitate application usage of the History-Info header field and eliminate the need to make assumptions based upon the order of the entries and ensure that the most complete set of information is available to the applications.

In addition, editorial changes were done to both condense and clarify the text, moving the requirements to an appendix and removing the inline references to the requirements. The examples were simplified and updated to reflect the protocol changes. Several of the call flows in the appendix were removed and put into a separate document that includes additional use cases that require the new header field parameters.

16.1. Backwards compatibility

This specification is backwards compatible since [RFC4244] allows for the addition of new optional parameters. This specification adds an optional SIP header field parameter to the History-Info and Contact header fields. Entities that have not implemented this specification will ignore these parameters, however, per [RFC4244] an entity will not remove these parameters from an hi-entry. While entities compliant to this document and [RFC4244] must be able to recognize gaps in the hi-entries, this document requires that an index of "0" be used in this case. Whereas [RFC4244] recommended (but did not require) the use of "1". However, since the ABNF in [RFC4244] defines the index as a DIGIT, "0" would be a valid value, thus an [RFC4244] implementation should not have an issue if it receives hi-entries added by intermediaries compliant to this document.

As for the behavior of the UACs, UASs and intermediaries, the following additional normative changes have been made:

UAC behavior

1. Inclusion of option tag by UAC has changed from SHOULD to MUST.

2. Inclusion of hi-target-entry along with hi-index has changed from MAY/RECOMMEND to MUST/MUST.

3. Behavior surrounding the addition of hi-target-entry based on 3xx response has changed from MAY/SHOULD to MUST.

None of the behavior changes would cause any backward or forward compatibility issues.

UAS behavior

1. Inclusion of hi-entry in response has changed from SHOULD to MUST.

As the entity receiving response with hi-entry expected it with SHOULD, this change will not cause any backward compatibility issues.

Proxy/Redirect Server behavior

1. Inclusion of H-I as forwarding the request has changed from SHOULD to MUST.

2. Association of Reason with time-out/internal reason has changed from MAY to MUST.

3. Inclusion of hi-index has changed from RECOMMENDED to MUST.

4. Inclusion of hi-entries in response has changed from SHOULD to MUST.

None of above behavior changes impact backwards compatibility since they only strengthen normative behavior to improve interoperability.

In cases where an entity that is compliant to this document, receives a request that contains hi-entries compliant only to RFC4244 (i.e, the hi-entries do not contain any of the new header field parameters), the entity MUST NOT add any of the new header field parameters to the hi-entries. The hi-entries MUST be cached and forwarded as any other entries are as specified in Section 9.1. As with RFC4244 compliant entities, applications must be able to function in cases of missing information, as specified in Section 11.

17. References

17.1. Normative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

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

[RFC3323] Peterson, J., "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", BCP 14, RFC 2119, March 1997.

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

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

[RFC4244] Barnes, M., "An Extension to the Session Initiation Protocol (SIP) for Request History Information", RFC 4244, November 2005.

17.2. Informative References

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

[RFC5630] Audet, F., "The Use of the SIPS URI Scheme in the Session Initiation Protocol (SIP)", RFC 5630, October 2009.

[RFC3087] Campbell, B. and R. Sparks, "Control of Service Context using SIP Request-URI", RFC 3087, April 2001.

[RFC4240] Burger, E., Van Dyke, J., and A. Spitzer, "Basic Network Media Services with SIP", RFC 4240, December 2005.

[RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.

[RFC4458] Jennings, C., Audet, F., and J. Elwell, "Session Initiation Protocol (SIP) URIs for Applications such as Voicemail and Interactive Voice Response (IVR)", RFC 4458,

April 2006.

[I-D.ietf-sipcore-rfc4244bis-callflows]
Barnes, M., Audet, F., Schubert, S., Elburg, H., and C. Holmberg, "Session Initiation Protocol (SIP) History-Info Header Call Flow Examples", draft-ietf-sipcore-rfc4244bis-callflows-06 (work in progress), July 2013.

Appendix A. 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. Although 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. GENERATION-req: "Request History" information is generated when the request is retargeted.

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

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

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

4. CONTENT-req: The "Request History" information for each occurrence of retargeting shall include the following:

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

   B. The URI or address from which the request was retargeted, and whether the retarget URI was an AOR

   C. The mechanism by which the new URI or address was determined,

D. The reason for the Request-URI or address modification,

E. Chronological ordering of the Request History information.

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

6. 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.

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

A.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-Info entry either by adding an additional hi-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.

3. A rogue application could delete some or all of the Request History 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 entity providing the request.

4. SEC-req-4: To ensure the confidentiality of the Request History information, only entities that 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.

A.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 [RFC3323]) 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 shall not be included in out going messages unless it is protected as described in [RFC3323].

Authors' Addresses

Mary Barnes
Polycom
TX
US

Email: mary.ietf.barnes@gmail.com

Francois Audet
Skype

Email: francois.audet@skype.net

Shida Schubert
NTT

Email: shida@ntt-at.com

Hans Erik van Elburg
Detecon International Gmbh
Sternengasse 14-16
Cologne,
Germany

Email: ietf.hanserik@gmail.com

Christer Holmberg
Ericsson
Hirsalantie 11, Jorvas
Finland

Email: christer.holmberg@ericsson.com