SIPPING Working Group
Internet-Draft
Expires: December 27, 2005

A. Allen, Ed.
Research in Motion (RIM)
J. Holm
Ericsson
T. Hallin
Motorola

June 25, 2005

The P-Answer-State Header Extension to the Session Initiation Protocol (SIP) for the Open Mobile Alliance (OMA) Push to talk over Cellular (PoC)

draft-allen-sipping-poc-p-answer-state-header-00

Status of this Memo

Copyright Notice

Abstract

This document describes a private Session InitiationProtocol(SIP) header (P-header) used by the Open Mobile Alliance (OMA),For Push to talk over Cellular (PoC) along with its applicability, which is

limited to the OMA PoC application. The P-Answer-State header is used for indicating the answering mode of the handset which is particular to the PoC application.

Requirements Language

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 [1].

Table of Contents

1. Overall Applicability

The SIP extension specified in this document makes certain assumptions regarding network topology, and the availability of transitive trust. These assumptions are generally NOT APPLICABLE in the Internet as a whole. The mechanism specified here was designed to satisfy the requirements specified by the Open Mobile Alliance for Push-to-talk over cellular for which either no general-purpose solution was found, where insufficient operational experience was available to understand if a general solution is needed, or where a more general solution is not yet mature. For more details about the assumptions made about this extension, consult the Applicability subsection for the extension.

2. Introduction

The Open Mobile Alliance (OMA) (http://www.openmobilealliance.org) is specifying the Push-to-talk Over Cellular (PoC) service where SIP is the protocol used to establish half duplex media sessions across different participants. This document describes a private extension to address specific requirements of the PoC service and may not be applicable to the general Internet.

The PoC service allows a SIP UA (PoC terminal) to establish a session to one or more SIP UAs simultaneously, usually initiated by the initiating user pushing a button.

OMA has defined a collection of very stringent requirements in support of the PoC service. In order to provide the user with a satisfactory experience the initial session establishment from the time the user presses the button to the time they get an indication to speak must be minimized.

3. Overview

The PoC terminal may support such hardware capabilities as a speaker phone and/or headset and software that provide the capability for the user to configure the PoC terminal to accept the session invitations immediately and play out the media as soon as it is received without requiring the intervention of the called user. This mode of operation is known as Automatic Answer mode. The user may alternatively configure the PoC terminal to first alert the user and require the user to manually accept the session invitation before media is accepted. This mode of operation is known as Manual Answer mode. The PoC terminal may support both or only one of these modes of operation. The user may change the Answer Mode (AM) configuration of the PoC terminal frequently based on their current circumstances and preference,(perhaps because the user is busy, or in a public area

where she cannot use a speaker phone, etc).

The OMA PoC Architecture utilizes SIP servers within the network that may perform such roles as a conference focus [5], a RTP translator or a network policy enforcement server. A possible optimization to minimize the delay in the providing of the caller with an indication to speak is for the SIP network server to perform buffering of media packets in order to provide an early or unconfirmed indication back to the caller and allow the caller to start speaking before the called PoC terminal has answered. An event package and mechanisms for a SIP UA to indicate its current answer mode to a SIP Server in order to enable buffering are defined in [6]. In addition, particularly when multiple domains are involved in the session more than one intermediate SIP server may be involved in the signaling path for the session and the server that performs the buffering may not be the server that has knowledge of the current answer mode of the SIP UA that is the final destination for the SIP INVITE request. A mechanism is to allow a terminal that acts as a SIP UA or a network based server that acts as a SIP UA to indicate a preference to the final destination SIP UAS to answer in a particular mode is defined in [7]. However a mechanism is required for an intermediary SIP UAS or proxy to relay the unconfirmed indication in a response back towards the originating SIP UAC.

This document proposes a new SIP header field to support this unconfirmed indication. The extension may be optionally included in a response to a SIP INVITE request or in a NOTIFY sent as a result of a REFER that requests an INVITE to be sent to provide an indication from an intermediate node acting as a SIP proxy or back-to-back UA that it has information that hints that the terminating UA will likely answer automatically and therefore provides an unconfirmed indication back towards the inviting SIP UA to transmit media prior to receiving a final response from the final destination of the SIP INVITE request. The extension is described below.

4. The P-Answer-State header

The purpose of the P-Answer-State header field is to provide an indication from an intermediate node acting as a SIP proxy or back-to-back UA that is has information that hints that the terminating UA identified in the Request-URI of the request will likely answer automatically and therefore provides an unconfirmed indication back towards the inviting SIP UA to transmit media prior to receiving a final response from the final destination of the SIP INVITE request. If a Provisional response contains the P-Answer-State header with the value "Unconfirmed" and does not contain SDP then a receiving intermediate node may send a 200 OK response containing SDP and a P-Answer-State header with the value "Unconfirmed" if the

intermediate node is willing to perform media buffering. If the response containing the P-Answer-State header with the value "Unconfirmed" also contains SDP the intermediate node that inserted the header and SDP in the response is also indicating that it is willing to buffer the media until a final confirmed indication is received.

The P-Answer-State header field MAY be included in a provisional or final response to a SIP INVITE request or in a NOTIFY request sent as a result of a REFER request to send an INVITE request. If the P-Answer-State header field with value "Unconfirmed" is included in a provisional response that contains SDP the intermediate node is leaving the decision where to do buffering to other nodes upstream and will forward upstream a "Confirmed indication" in a 200 OK response when the final response is received from the destination UA.

The P-Answer-State header is only included in a provisional response when the node that sends the response has knowledge that there is a B2BUA node that understands this extension in the signaling path between itself and the originating UAC that will only pass the header on in either a 200 OK response or in the sipfrag of a NOTIFY request. Such a situation only occurs with specific network topologies which is another reason why use of this header is not relevant to the general internet. The originating UAC will only receive the P-Answer-state header in a 200 OK response or in the sipfrag of a NOTIFY request.

4.1 Requirements

The OMA PoC service has initial setup performance requirements that can be met by an intermediate server (SIP B2BUA) spooling media from the inviting PoC subscriber until one or more invited PoC subscribers have accepted the session. The specific requirements are

REQ-1: An intermediate server MAY spool media from the inviting SIP UA until one or more invited PoC SIP UAs has accepted the invitation.

REQ-2: An intermediate server that is capable of spooling media MAY accept an invite request from an inviting SIP UAC even if no invited SIP UAS has accepted the invite request if it has a hint that the invited SIP UAC is likely to accept the request without requiring user intervention.

REQ-3: An intermediate server or proxy that is incapable of spooling media or does not wish to, but has a hint that the invited SIP UAC is likely to automatically accept the session invitation MUST be able to indicate back to another intermediate server that can spool media that it has some hint that the invited UAC is likely to automatically accept the session invitation.

REQ-4: An intermediate server that is willing to spool media from the inviting SIP UA until one or more invited SIP UAs have accepted the invite SHOULD indicate that it is spooling media to the inviting SIP UAC.

4.2 Alternatives Considered

In order to meet REQ-3, an intermediate server needs to receive an indication back that the invited SIP UA is likely to accept the invite request without requiring user intervention. In this case, the intermediate server or proxy that has a hint that the invited SIP UAC is likely to accept the request can include an answer state indication in the 183 Session Progress or 200 OK response.

A number of alternatives were considered for the intermediate server to inform the another intermediate server or the inviting SIP UAC of the invited PoC SIP UAs answer mode settings.

One proposal was to create a unique reason-phrase in the 183 and 200 OK response. This was rejected because the reason phrases are normally intended for human readers and not meant to be parsed by servers for special syntactic and semantic meaning.

Another proposal was to use a Reason header [8] in the 183 and 200 OK response. This was rejected because this would be inconsistent with the intended use of the reason header and its usage is not defined for these response codes and would have required creating and registering a new protocol identifier.

Another proposal was to use a feature-tag in the returned Contact header as defined in [9]. This was rejected because it was not a different feature, but is an attribute of the session and can be applied to many different features.

Another proposal was to use a new SDP attribute. The choice of an SDP parameter was rejected because the answer state applies to the session and not to a media stream.

The P-Answer-State header was chosen to give additional information about the state of the SIP session progress and acceptance. Even

though the UAC sees that its SDP offer has been answered and accepted, the header lets the UAC know whether invited PoC subscriber has accepted the invite or just an intermediary has done the acceptance.

4.3 Applicability statement for the P-Answer-State header

The P-Answer-State header is applicable in the following circumstances:

- o In networks where there are UAs that engage in half-duplex communication where there is not the possibility for the invited user to verbally acknowledge the answering of the session as is normal in full duplex communication;
- o Where the invited UA may automatically accept the session without manual acceptance;
- o The network also contains intermediate network SIP servers that are trusted;
- o The intermediate network SIP servers have knowledge of the current answer mode setting of the terminating UAS; and,
- o The intermediate network SIP servers can provide buffering of the media in order to reduce the time for the inviting user to send media.
- o The intermediate network SIP servers assume knowledge of the network topology and the existence of similar intermediate network SIP servers in the signaling path.

Such configurations are generally not applicable to the internet as a whole where such trust relationships do not exist.

In addition security issues have only been considered for networks which are trusted and use hop by hop security mechanisms and security issues with usage of this mechanism in the general internet have not been evaluated.

4.4 Usage of the P-Answer-State header

A UAS B2BUA or proxy MAY insert a P-Answer-State header field in any 1XX or 2XX response that is allowed to contain an SDP answer in response to an SDP offer contained in an INVITE as specified in [2]. Typically the P-Answer-State header field is inserted in either a 183 Session Progress or a 200 OK response. A UA that receives a REFER request to send an INVITE MAY also insert a P-Answer-State header

field in a NOTIFY request it sends as a result of the implicit subscription created by the REFER request.

When the P-Answer-State header field contains the parameter "Unconfirmed" the UAC or proxy is indicating that it has information that hints that the final destination UAS for the INVITE request is likely to automatically accept the session but that this is unconfirmed and it is possible that the final destination UAS will first alert the user and require manual acceptance of the session or not accept the session request. This is referred to here as an "unconfirmed response". When the P-Answer-State header field contains the parameter "Confirmed" the UAC or proxy is indicating that the destination UAS has accepted the session and is ready to receive media. The parameter value of "Confirmed" has the usual semantics of an SDP answer and is included for completeness. The usual end to end SDP answer response semantics are referred to here as a "confirmed response".

4.4.1 Procedures at the UA (terminal)

A UAC (terminal) that receives a 1XX or 2XX response containing a P-Answer-State header field containing the parameter "Unconfirmed" and an SDP answer MAY send media as specified in [2], however there is no guarantee that the media will be received by the final recipient. How a UAC confirms whether the media was or was not received by the final destination when it his received a 2XX "unconfirmed response" is application specific and outside of the scope of this document. If the application is a conference then the mechanism specified in [2] could be used to determine that the invited user joined. Alternatively a BYE request could be received or the media could be placed on hold if the final destination UAS does not accept the session.

A UAC (terminal) that receives a 1XX or 2XX response without a P-Answer-State Header or containing a P-Answer-State header field containing the parameter "Confirmed" SHALL treat it as a "confirmed response". A UAC (terminal) that receives in response to a REFER request a NOTIFY request containing a P-Answer-State header field containing the parameter "Unconfirmed" as either a SIP header or contained in a sipfrag in the body of the NOTIFY request received on a pre-existing dialog that was established by an INVITE request and for which there has been a successful SDP offer-answer exchange according to [2] then the UAC MAY send media, however there is no guarantee that the media will be received by the final recipient that was indicated in the Refer-To header in the original REFER request.

There are no P-Answer-State procedures for a terminal acting in the UAS role.

4.4.2 Procedures at the UA (Intermediate Node)

A UAS (Intermediate Node) MAY insert a P-Answer-State header field in any 1XX or 2XX response that is allowed to contain an SDP answer in response to an SDP offer contained in an INVITE request as specified in [2]. A response containing the P-Answer-State header field containing the parameter "Unconfirmed" MAY or MAY NOT contain an SDP answer. If the response contains an SDP answer then the sending UA MUST be ready to receive media as specified in [2].

An intermediate node that acts as a back-to-back UA and returns a 1XX or 2XX response in response to an INVITE request MAY insert a P-Answer-State header field containing the parameter "Unconfirmed" in the response if it has not yet received a "confirmed response" from the final destination UA. If the intermediate node UAS also includes SDP in the response along with a P-Answer-State header field containing the parameter "Unconfirmed" the intermediate node MUST be ready to receive media as specified in [2] and MAY buffer any media it receives until it receives a "confirmed response" from the final destination UA or until the buffer is full. Such an intermediate node may insert an SDP answer in the response it generates even if the "unconfirmed response" it received did not contain an SDP answer.

An intermediate node that acts as a back-to-back UA and receives a REFER request to send an INVITE request to another UA as specified in [11] MAY insert a P-Answer-State header field containing the parameter "Unconfirmed" in the initial NOTIFY request sent in response to the REFER request if it has not yet received a "confirmed response" from the final destination UA and it has information that hints that the final destination UAS for the INVITE is likely to automatically accept the session. If the REFER was sent as part of an existing dialog established by an INVITE request and for which there has been a successful SDP offer-answer exchange according to [2] the intermediate node MUST be ready to receive media as specified in [2] and MAY buffer any media it receives until it receives a "confirmed response" from the final destination UA or until its buffer is full.

An intermediate node that acts as a back-to-back UA and receives a 1XX or 2XX response in response to an INVITE request containing a P-Answer-State header field in the response SHOULD include the P-Answer-State header field unmodified in the 1XX or 2XX response it sends as a result of receiving that response. If the intermediate node that acts as a back-to-back UA sends a NOTIFY request according to [11] then the intermediate node UAC SHOULD include the P-Answer-State header field unmodified in the sipfrag of the response included in the body of the NOTIFY request.

A UAC (Intermediate Node) that receives a 1XX or 2XX response without a P-Answer-State Header or containing a P-Answer-State header field containing the parameter "Confirmed" SHALL treat it as a "confirmed response". If the UAS (Intermediate Node) knows that the final destination UA is now ready to accept media and the UAS previously sent an "Unconfirmed response" the UAS SHOULD insert a P-Answer-State header field containing the parameter "Confirmed" in the response.

An intermediate node that acts as a back-to-back UA that previously sent an initial NOTIFY request containing a P-Answer-State header field containing the parameter "Unconfirmed" that subsequently receives a "confirmed response" without a P-Answer-State header field in response to the INVITE request sent as a result of the REFER request SHOULD include a P-Answer-State header containing the parameter "Confirmed" in the subsequent NOTIFY request generated as a result of the "confirmed response".

If the UAS knows that the final destination UA is ready to accept media and the UAS did not previously send an "Unconfirmed response" the UAS MAY insert a P-Answer-State header field containing the parameter "Confirmed" in the response.

If an intermediate node that acts as a back-to-back UA and sends an INVITE request in response to a REFER request learns by receiving a "confirmed response" that the final destination UA is ready to accept media and the back-to-back UA did not previously include a P-Answer-State header containing the parameter "Unconfirmed" in the initial NOTIFY request sent in response to the REFER request then the back-to-back UA MAY insert a P-Answer-State header field containing the parameter "Confirmed" in the response if the "confirmed response" does not contain a P-Answer-State header.

4.4.3 Procedures at the proxy server

SIP proxy servers do not need to understand the semantics of the P-Answer-State header field. As part of the regular SIP rules for unknown headers, a proxy will forward unknown headers. A proxy MAY insert a P-Answer-State header field in a 1XX response that it originates compliant with [3] or add it to a 2XX response that contains an SDP answer in response to an SDP offer contained in an INVITE request as specified in [2].

A proxy that returns a 1XX response in response to an INVITE request MAY insert a P-Answer-State header field containing the parameter "Unconfirmed" in the response if it has not yet received a "confirmed response" from the final destination UA.

A proxy that receives a 1XX or 2XX response without a P-Answer-State

Header or containing a P-Answer-State header field containing the parameter "Confirmed" SHALL for the purposes of this document treat it as a "confirmed response".

If the proxy knows that the final destination UA is now ready to accept media and the proxy previously sent an "Unconfirmed response" the proxy SHOULD insert a P-Answer-State header field containing the parameter "Confirmed" in the response.

If the proxy knows that the final destination UA is ready to accept media and the proxy did not previously send an "Unconfirmed response" the proxy MAY insert a P-Answer-State header field containing the parameter "Confirmed" in the response.

5. Formal Syntax

The mechanisms specified in this document is described in both prose and an augmented Backus-Naur Form (BNF) defined in RFC 2234 [3]. Further, several BNF definitions are inherited from SIP and are not repeated here. Implementers need to be familiar with the notation and contents of SIP [3] and RFC 2234 [3] to understand this document.

5.1 P-Answer-State header syntax

The syntax of the P-Answer-State header is described as follows:

```
P-Answer-State = "P-Answer-State" HCOLON answer-type
answer-type  = "Confirmed" / "Unconfirmed"
```

5.2 Table of the new header

Table 1 extends the headers defined in this document to Table 2 in SIP [3], section 7.1 of the SIP-specific event notification [10] tables 1 and 2 in the SIP INFO method [11], tables 1 and 2 in Reliability of provisional responses in SIP [12], tables 1 and 2 in the SIP UPDATE method [13], tables 1 and 2 in the SIP extension for Instant Messaging [14], table 1 in the SIP REFER method [15], and table 2 in the SIP PUBLISH method [16]:

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	SUB
P-Answer-State	1xx,2xx	ar	-	-	-	o	-	-	-

Header field		NOT	PRA	INF	UPD	MSG	REF	PUB
P-Answer-State	R	o	-	-	-	-	-	-

Figure 1

6. Example Usage Session Flows

For simplicity some details such as intermediate proxies and 100 Trying responses are not shown in the following example flows. The term "policy server" is used here to mean a policy enforcement server.

6.1 Pre-arranged Group Call using On-demand Session

The following flow shows Alice making a Pre-arranged Group Call using a Conference URI which has Bob on the member list. The session initiation uses the On-demand Session establishment mechanism where a SIP INVITE containing SDP is sent by Alices's terminal when Alice pushes her push to talk button.

In this example Alice's Policy Server acts a Call Stateful SIP Proxy and Bob's Policy Server which is aware that the current Answer Mode setting of Bob's terminal is set to Auto Answer acts as a B2BUA.

For simplicity the invitations by the Conference Focus to the other members of the group are not shown in this example.

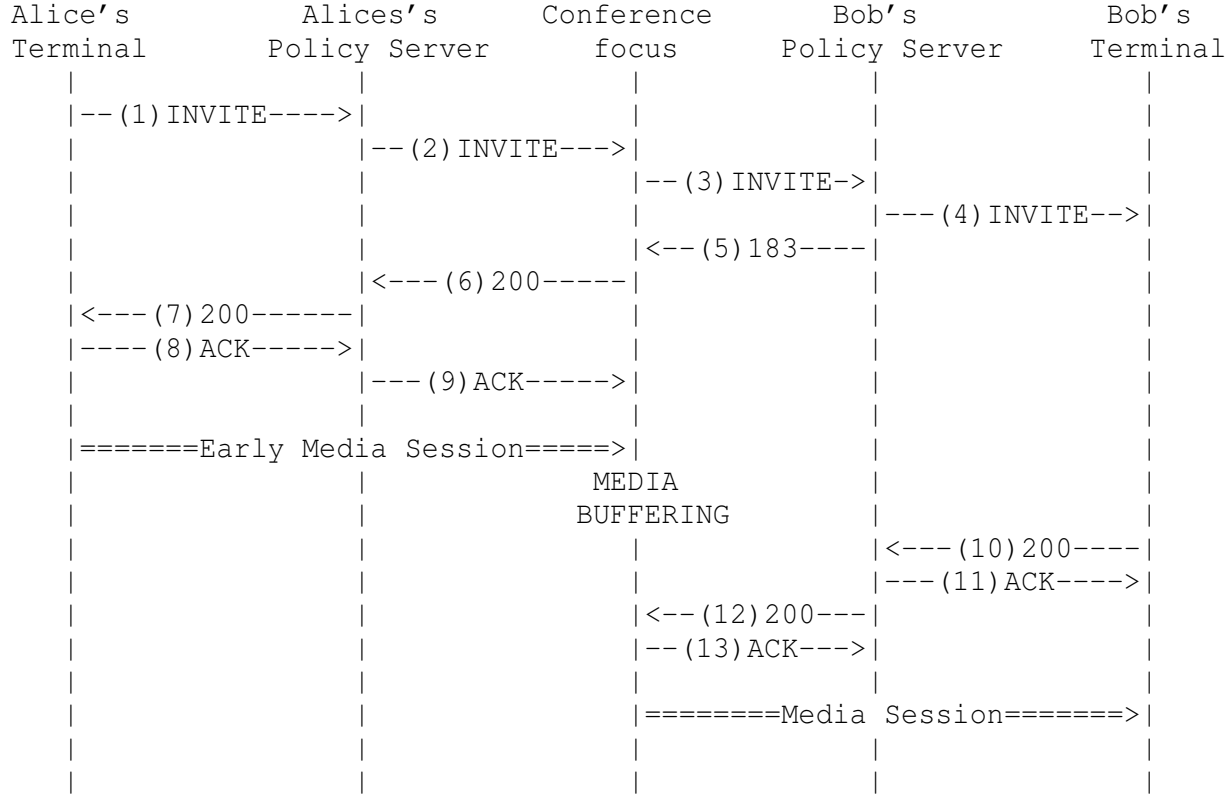


Figure 2

F1 INVITE Alice -> Alices's Policy Server

```
INVITE sip:FriendsOfAlice@atlanta.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: "Alice's Friends" <sip:FriendsOfAlice@atlanta.com>
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(SDP not shown)
```

F2 INVITE Alice's Policy Server -> Conference Focus

```
INVITE sip:FriendsOfAlice@atlanta.com SIP/2.0
Via: SIP/2.0/UDP
AlicesPolicyServer.atlanta.com;branch=z9hG4bK77ef4c2312983.1
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
Record-Route: <sip:AlicesPolicyServer.atlanta.com>
Max-Forwards: 69
To: "Alice's Friends" <sip:FriendsOfAlice@atlanta.com>
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(SDP not shown)
```

The Conference Focus explodes the Conference URI and Invites Bob

F3 INVITE Conference Focus -> Bob's Policy Server

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP
AlicesConferenceFocus.atlanta.com;branch=z9hG4bK4721d8
Max-Forwards: 70
To: "Bob" <sip:bob@biloxi.com>
From: "Alice's Friends"
<sip:FriendsOfAlice@atlanta.com>;tag=2178309898
Call-ID: e60a4c784b6716
CSeq: 301166605 INVITE
Contact: <sip:AlicesConferenceFocus.atlanta.com>
Content-Type: application/sdp
```

```
    Content-Length: 142

   (SDP not shown)

   F4 INVITE Bob's Policy Server -> Bob

   INVITE sip:bob@biloxi.com SIP/2.0
   Via: SIP/2.0/UDP BobsPolicyServer.biloxi.com;branch=z9hG4bKa27bc93
   Max-Forwards: 70
   To: "Bob" <sip:bob@biloxi.com>
   From: "Alice's Friends"
   <sip:FriendsOfAlice@atlanta.com>;tag=781299330
   Call-ID: 6eb4c66a847710
   CSeq: 478209 INVITE
   Contact: <sip:BobsPolicyServer.biloxi.com>
   Content-Type: application/sdp
   Content-Length: 142

   (SDP not shown)

   F5 183 Session Progress Bob's Policy Server -> Conference Focus

   SIP/2.0 183 Session Progress
   Via: SIP/2.0/UDP
   AlicesConferenceFocus.atlanta.com;branch=z9hG4bK4721d8
   To: "Bob" <sip:bob@biloxi.com>;tag=a6c85cf
   From: "Alice's Friends"
   <sip:FriendsOfAlice@atlanta.com>;tag=2178309898
   Call-ID: e60a4c784b6716
   Contact: <sip:BobsPolicyServer.biloxi.com>
   CSeq: 301166605 INVITE
   P-Answer-State: Unconfirmed
   Content-Length: 0

   F6 200 OK Conference Focus -> Alice's Policy Server

   SIP/2.0 200 OK
   Via: SIP/2.0/UDP
   AlicesPolicyServer.atlanta.com;branch=z9hG4bK77ef4c2312983.1
   Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
   Record-Route: <sip:AlicesPolicyServer.atlanta.com>
   To: "Alice's Friends" <sip:FriendsOfAlice@atlanta.com>;tag=c70ef99
   From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
   Call-ID: a84b4c76e66710
   CSeq: 314159 INVITE
   Contact: <sip:AlicesConferenceFocus.atlanta.com>
   P-Answer-State: Unconfirmed
   Content-Type: application/sdp
```

```
    Content-Length: 131

   (SDP not shown)
```

F7 200 OK Alice's Policy Server -> Alice

```
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
   Record-Route: <sip:AlicesPolicyServer.atlanta.com>
   To: "Alice's Friends" <sip:FriendsOfAlice@atlanta.com>;tag=c70ef99
   From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
   Call-ID: a84b4c76e66710
   CSeq: 314159 INVITE
   Contact: <sip:AlicesConferenceFocus.atlanta.com>
   P-Answer-State: Unconfirmed
   Content-Type: application/sdp
   Content-Length: 131

   (SDP not shown)
```

F8 ACK Alice -> Alice's Policy Server

```
   ACK sip:AlicesConferenceFocus.atlanta.com SIP/2.0
   Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9
   Route: <sip:AlicesPolicyServer.atlanta.com>
   Max-Forwards: 70
   To: "Alice's Friends" <sip:FriendsOfAlice@atlanta.com>;tag=c70ef99
   From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
   Call-ID: a84b4c76e66710
   CSeq: 314159 ACK
   Content-Length: 0
```

F9 ACK Alice's Policy Server -> Conference Focus

```
   ACK sip:AlicesConferenceFocus.atlanta.com SIP/2.0
   Via: SIP/2.0/UDP
   AlicesPolicyServer.atlanta.com;branch=z9hG4bK77ef4c2312983.1
   Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9
   Max-Forwards: 69
   To: "Alice's Friends" <sip:FriendsOfAlice@atlanta.com>;tag=c70ef99
   From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
   Call-ID: a84b4c76e66710
   CSeq: 314159 ACK
   Content-Length: 0
```

The early half duplex media session between Alice and the Conference Focus is now established and the Conference Focus buffers the media it receives from Alice.

F10 200 OK Bob -> Bob's Policy Server

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP BobsPolicyServer.biloxi.com;branch=z9hG4bKa27bc93
To: "Bob" <sip:bob@biloxi.com>;tag=d28119a
From: "Alice's Friends"
<sip:FriendsOfAlice@atlanta.com>;tag=781299330
Call-ID: 6eb4c66a847710
CSeq: 478209 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(SDP not shown)
```

F11 ACK Bob's Policy Server -> Bob

```
ACK sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP BobsPolicyServer.biloxi.com;branch=z9hG4bKa27bc93
Max-Forwards: 70
To: "Bob" <sip:bob@biloxi.com>;tag=d28119a
From: "Alice's Friends"
<sip:FriendsOfAlice@atlanta.com>;tag=781299330
Call-ID: 6eb4c66a847710
CSeq: 478209 ACK
Content-Length: 0
```

F12 200 OK Bob's Policy Server -> Conference Focus

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP
AlicesConferenceFocus.atlanta.com;branch=z9hG4bK4721d8
To: "Bob" <sip:bob@biloxi.com>;tag=a6670811
From: "Alice's Friends"
<sip:FriendsOfAlice@atlanta.com>;tag=2178309898 Call-ID:
e60a4c784b6716
Contact: <sip:BobsPolicyServer.biloxi.com>
CSeq: 301166605 INVITE
P-Answer-State: Confirmed
Content-Type: application/sdp
Content-Length: 131

(SDP not shown)
```

F13 ACK Conference Focus -> Bob's Policy Server

```
ACK sip:BobsPolicyServer.biloxi.com SIP/2.0
Via: SIP/2.0/UDP
```

```
AlicesConferenceFocus.atlanta.com;branch=z9hG4bK4721d8
Max-Forwards: 70
To: "Bob" <sip:bob@biloxi.com>;tag=a6670811
From: "Alice's Friends"
<sip:FriendsOfAlice@atlanta.com>;tag=2178309898
Call-ID: e60a4c784b6716
CSeq: 301166605 ACK
Content-Length: 0
```

The media session between Alice and Bob is now established and the Conference Focus forwards the buffered media to Bob.

6.2 1-1 Call using Pre-established Session

The following flow shows Alice making a 1-1 Call to Bob using a pre-established session. A pre-established session is where a dialog is established with Alices's Policy Server using a SIP INVITE SDP offer answer exchange to pre-negotiate the codecs and other media Parameters to be used for media sessions ahead of Alice initiating a Communication. When Alice initiates a communication to Bob a SIP REFER is used to Request Alice's Policy Server to send an INVITE to Bob. In this example Bob's Terminal does not use the Pre-established Session mechanism.

In this example Alice's Policy Server acts a B2BUA and also performs the Conference Focus function. Bob's Policy Server which is aware that the current Answer Mode setting of Bob's terminal is set to Auto Answer acts as a B2BUA.

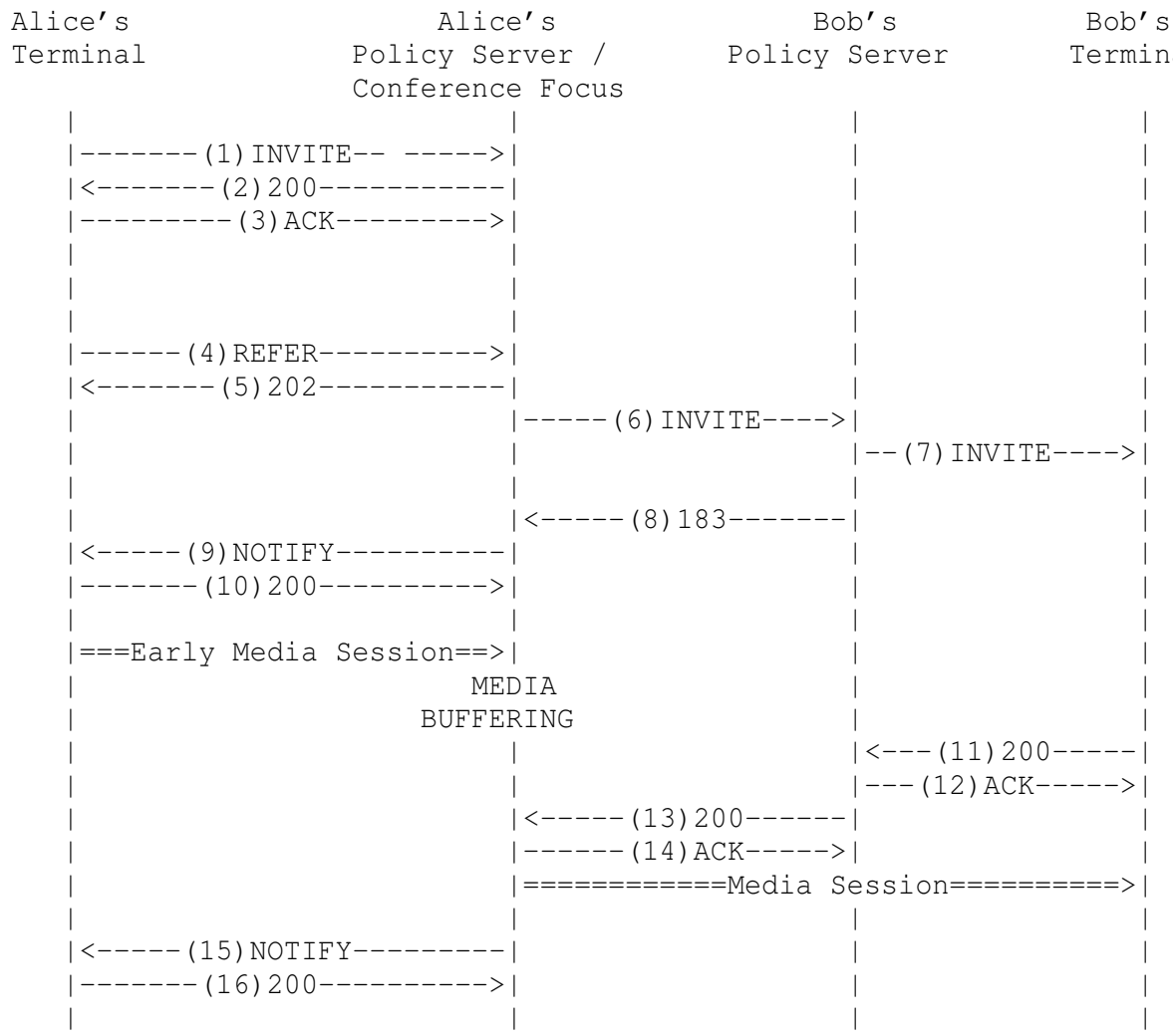


Figure 3

F1 INVITE Alice -> Alices's Policy Server

```
INVITE sip: AlicesConferenceFactoryURI.atlanta.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: <sip:AlicesConferenceFactoryURI.atlanta.com>
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(SDP not shown)
```

F2 200 OK Alice's Policy Server -> Alice

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: <sip:AlicesConferenceFactoryURI.atlanta.com>;tag=c70ef99
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:AlicesPre-establishesSession@
AlicesPolicyServer.atlanta.com>
Content-Type: application/sdp
Content-Length: 131

(SDP not shown)
```

F3 ACK Alice -> Alice's Policy Server

```
ACK sip:AlicesPre-establishesSession@AlicesPolicyServer.atlanta.com
SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9
Max-Forwards: 70
To: <sip:AlicesConferenceFactoryURI.atlanta.com>;tag=c70ef99
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 ACK
Content-Length: 0
```

Alices's terminal has established a Pre-established Session with Alice's Policy Server. All the media parameters are pre-negotiated for use at communication time.

Alice initiates a Communication to Bob

F4 REFER Alice -> Alices's Policy Server

```
REFER sip:AlicesPre-establishesSession@AlicesPolicyServer.atlanta.com
SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: <sip:AlicesConferenceFactoryURI.atlanta.com>;tag=c70ef99
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314160 REFER
Refer-To: "Bob" <sip:bob@biloxi.com>
Contact: <sip:alice@pc33.atlanta.com>
```

F5 202 ACCEPTED Alice's Policy Server -> Alice

```
SIP/2.0 202 ACCEPTED
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: <sip:AlicesConferenceFactoryURI.atlanta.com>;tag=c70ef99
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314160 REFER
Contact: <sip:AlicesPre-establishesSession@
AlicesPolicyServer.atlanta.com>
```

F6 INVITE Conference Focus -> Bob's Policy Server

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP
AlicesConferenceFocus.atlanta.com;branch=z9hG4bk4721d8
Max-Forwards: 70
To: "Bob" <sip:bob@biloxi.com>
From: "Alice" <sip:Alice@atlanta.com>;tag=2178309898
Call-ID: e60a4c784b6716
CSeq: 301166605 INVITE
Contact: <sip:AlicesConferenceFocus.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(SDP not shown)
```

F7 INVITE Bob's Policy Server -> Bob

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP BobsPolicyServer.biloxi.com;branch=z9hG4bKa27bc93
Max-Forwards: 70
To: "Bob" <sip:bob@biloxi.com>
From: "Alice" <sip:Alice@atlanta.com>;tag=781299330
Call-ID: 6eb4c66a847710
CSeq: 478209 INVITE
Contact: <sip:BobsPolicyServer.biloxi.com>
Content-Type: application/sdp
Content-Length: 142

(SDP not shown)
```

F8 183 Session Progress Bob's Policy Server -> Conference Focus

```
SIP/2.0 183 Session Progress
Via: SIP/2.0/UDP
AlicesConferenceFocus.atlanta.com;branch=z9hG4bK4721d8
To: "Bob" <sip:bob@biloxi.com>;tag=a6c85cf
From: "Alice" <sip:Alice@atlanta.com>;tag=2178309898
Call-ID: e60a4c784b6716
```

```
 Contact: <sip:BobsPolicyServer.biloxi.com>
CSeq: 301166605 INVITE
P-Answer-State: Unconfirmed
Content-Length: 0
```

F9 NOTIFY Alices's Policy Server -> Alice

```
NOTIFY sip:alice@pc33.atlanta.com SIP/2.0
Via: SIP/2.0/UDP AlicesPre-establishesSession@
AlicesPolicyServer.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: <sip:AlicesConferenceFactoryURI.atlanta.com>;tag=c70ef99
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314161 NOTIFY
Contact: <sip:AlicesPre-establishesSession@
AlicesPolicyServer.atlanta.com>
Event: refer
Subscription-State: Active;Expires=60
Content-Type: message/sipfrag;version=2.0
Content-Length: 99

SIP/2.0 183 Session Progress
To: "Bob" <sip:bob@biloxi.com>;tag=d28119a
P-Answer-State: Unconfirmed
```

F10 202 ACCEPTED Alice -> Alice's Policy Server

```
SIP/2.0 202 ACCEPTED
Via: SIP/2.0/UDP AlicesPre-establishesSession@
AlicesPolicyServer.atlanta.com;branch=z9hG4bKnashds8
To: <sip:AlicesConferenceFactoryURI.atlanta.com>;tag=c70ef99
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314161 NOTIFY
```

The early half duplex media session between Alice and the Conference Focus is now established and the Conference Focus buffers the media it receives from Alice.

F11 200 OK Bob -> Bob's Policy Server

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP BobsPolicyServer.biloxi.com;branch=z9hG4bK927bc93
To: "Bob" <sip:bob@biloxi.com>;tag=d28119a
From: "Alice's Friends"
<sip:FriendsOfAlice@atlanta.com>;tag=781299330
Call-ID: 6eb4c66a847710
```

```
 CSeq: 478209 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(SDP not shown)
```

F12 ACK Bob's Policy Server -> Bob

```
ACK sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP BobsPolicyServer.biloxi.com;branch=z9hG4bK927bc93
Max-Forwards: 70
To: "Bob" <sip:bob@biloxi.com>;tag=d28119a
From: "Alice" <sip:Alice@atlanta.com>;tag=781299330
Call-ID: 6eb4c66a847710
CSeq: 478209 ACK
Content-Length: 0
```

F13 200 OK Bob's Policy Server -> Conference Focus

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP
AlicesConferenceFocus.atlanta.com;branch=z9hG4bK4721d8
To: "Bob" <sip:bob@biloxi.com>;tag=a6670811
From: "Alice's Friends"
<sip:FriendsOfAlice@atlanta.com>;tag=2178309898
Call-ID: e60a4c784b6716
Contact: <sip:BobsPolicyServer.biloxi.com>
CSeq: 301166605 INVITE
P-Answer-State: Confirmed
Content-Type: application/sdp
Content-Length: 131

(SDP not shown)
```

F14 ACK Conference Focus -> Bob's Policy Server

```
ACK sip:BobsPolicyServer.biloxi.com SIP/2.0
Via: SIP/2.0/UDP
AlicesConferenceFocus.atlanta.com;branch=z9hG4bK4721d8
Max-Forwards: 70
To: "Bob" <sip:bob@biloxi.com>;tag=a6670811
From: "Alice" <sip:Alice@atlanta.com>;tag=1928301774
Call-ID: e60a4c784b6716
CSeq: 301166605 ACK
Content-Length: 0
```

The media session between Alice and Bob is now established and the

Conference Focus forwards the buffered media to Bob.

F15 NOTIFY Alices's Policy Server -> Alice

```
NOTIFY sip:alice@pc33.atlanta.com SIP/2.0
Via: SIP/2.0/UDP AlicesPre-establishesSession@
AlicesPolicyServer.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: <sip:AlicesConferenceFactoryURI.atlanta.com>;tag=c70ef99
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314162 NOTIFY
Contact: <sip:AlicesPre-establishesSession@
AlicesPolicyServer.atlanta.com>
Event: refer
Subscription-State: Active;Expires=60
Content-Type: message/sipfrag;version=2.0
Content-Length: 83

SIP/2.0 200 OK
To: "Bob" <sip:bob@biloxi.com>;tag=d28119a
P-Answer-State: Confirmed
```

F16 202 ACCEPTED Alice -> Alice's PolicyServer

```
SIP/2.0 202 ACCEPTED
Via: SIP/2.0/UDP AlicesPre-establishesSession@
AlicesPolicyServer.atlanta.com;branch=z9hG4bKnashds8
To: <sip:AlicesConferenceFactoryURI.atlanta.com>;tag=c70ef99
From: "Alice" <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314162 NOTIFY
```

7. Security Considerations

The information returned in the P-Answer-State header is not viewed as particularly sensitive. Rather, it is informational in nature, providing an indication to the UAC that delivery of any media sent as a result of an answer in this response is not guaranteed. An eavesdropper cannot gain any useful information by obtaining the contents of this header.

If end-to-end protection is not used at the SIP layer, it is possible for proxies between the UAs to remove the header or modify the contents of the header value. However end-to-end protection has not been considered as the P-Answer-State header is normally added by an intermediate node that acts either as a B2BUA or proxy. This attack either denies the caller the knowledge that the callee has yet to be

contacted or falsely indicates that the callee has yet to be contacted when they have already answered. It is therefore RECOMMENDED that this extension is used in a secured trusted environment where transitive trust exists between the proxies and UAs.

If end-to-end security mechanisms are to be used issues such as key exchange between endpoints and intermediate network nodes need to be considered."

8. IANA Considerations

8.1 Registration of Header Fields

This document defines a private SIP extension header field (beginning with the prefix "P-") based on the registration procedures defined in RFC 3427 [17].

The following rows shall be added to the "Header Fields" section of the SIP parameter registry:

Header Name	Compact Form	Reference
P-Answer-State		[RFCXXXX]

Editor Note: [RFCXXXX] should be replaced with the designation of this document.

8.2 Registration of Header Field Parameters

This document defines parameters for the header fields defined in the preceding section. The header field named "P-Answer-State" may take the values "Unconfirmed", or "Confirmed".

The following rows shall be added to the "Header Field Parameters and Parameter Values" section of the SIP parameter registry:

Header Field	Parameter Name	Predefined Values	Reference
P-Answer-State	Unconfirmed	Yes	[RFCXXXX]
P-Answer-State	Confirmed	Yes	[RFCXXXX]

Editor Note: [RFCXXXX] should be replaced with the designation of this document.

9. Changes since previous version

This document is based upon the definition of the P-Answer-State header in [18] with the following changes:

The P-Alerting-Mode header definition has been removed as this extension will be progressed separately as a standards track RFC [7].

The text on security considerations has been improved.

PUBLISH method has been added to figure 1.

The procedures at the UA text has been split into UA-terminal and UA- intermediate-node for better clarity.

A session flow example section has been added.

Various nits and editorial corrections have been made.

10. Acknowledgements

The authors would like to thank Dean Willis, Rohan Mahay, Christian Schmidt, Mike Hammer, and Miguel Garcia-Martin for their comments that contributed to the progression of this work. The authors would also like to thank the OMA POC Working Group members for their support of this document and in particular Tom Hiller for presenting the concept of the P-Answer-State header to SIPPING at IETF#62.

11. References

11.1 Normative References

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

[2] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

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

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

11.2 Informative References

[5] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-05 (work in progress), May 2005.

[6] Garcia-Martin, M., "A Session Initiation Protocol (SIP) Event Package and Data Format for various settings in support for the Push-to-talk Over Cellular (PoC) service", draft-garcia-sipping-poc-isb-am-02 (work in progress), June 2005.

[7] Willis, D. and A. Allen, "Requesting Answering and Alerting Modes for the Session Initiation Protocol (SIP)", draft-willis-sip-answeralert-00 (work in progress), June 2005.

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

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

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

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

[12] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

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

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

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

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

[17] Mankin, A., Bradner, S., Mahy, R., Willis, D., Ott, J., and B.

Rosen, "Change Process for the Session Initiation Protocol (SIP)", BCP 67, RFC 3427, December 2002.

[18] Allen, A., "Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the Open Mobile Alliance (OMA) Push to talk over Cellular (PoC)", draft-allen-sipping-poc-p-headers-01 (work in progress), February 2005.

Authors' Addresses

Andrew Allen (editor)
Research in Motion (RIM)
122 West John Carpenter Parkway, Suite 430
Irving, Texas 75039
USA

Phone: unlisted
Fax: unlisted
Email: aallen@rim.com

Jan Holm
Ericsson
Gotalandsvagen 220
Stockholm 612526
Sweden

Phone: unlisted
Fax: unlisted
Email: Jan.Holm@ericsson.com

Tom Hallin
Motorola
1501 W Shure Drive
Arlington Heights 60004
USA

Phone: unlisted
Fax: unlisted
Email: thallin@motorola.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING Working Group
Internet-Draft
Expires: January 19, 2006

J. Hautakorpi, Ed.
G. Camarillo
Ericsson
M. Bhatia
NexTone Communications
R. Penfield
Acme Packet
A. Hawrylyshen
Ditech Communications Corporation
July 18, 2005

SIP (Session Initiation Protocol)-Unfriendly Functions in Current Communication Architectures

draft-camarillo-sipping-sbc-funcs-01.txt

Status of this Memo

Copyright Notice

Abstract

This document gives an overview to SIP-unfriendly functions in

current communication architectures. These functions are generally implemented in SBCs (Session Border Controllers). The purpose of this document is to help the IETF community understand what SIP-unfriendly functions can be found in existing networks so that the appropriate working groups can decide whether or not new standard solutions need to be developed to provide such functionality (or a subset of it) in a standard, SIP-friendly way. Working groups may also develop recommendations on how to use existing standard mechanisms to provide such functionality.

Table of Contents

1. Introduction

This document gives an overview to SIP [1] unfriendly functions in current communication architectures. These functions are generally implemented in Session Border Controllers (SBCs). The reason for this is that network policies are typically enforced at the edge of the network, and SBCs are typically located there.

Of course, a typical SBC does not only implement SIP-unfriendly functions. They usually implement a set of functions, some of which are perfectly legal from the SIP point of view. However, this document focuses on those functions that break SIP somehow.

Many existing SBCs use proprietary solutions because there is no standard solution for a given issue or because the standard solution does not fully meet the requirements of the network operator. This document is intended to be taken as input by the IETF so that the appropriate working groups can decide whether or not new standard solutions need to be developed to provide the same functionality (or a subset of it) in a SIP-friendly way. Working groups may also decide to develop recommendations on how to use existing standard mechanisms to provide such functionality.

2. Background on SBCs

The term SBC is pretty vague, since it is not standardized or defined anywhere. Nodes that may be referred to as SBCs but do not implement SIP are outside the scope of this document.

Even though many SBCs currently break things like end-to-end security and can impact feature negotiations, there is clearly a market for them. Network operators need many of the features current SBCs provide and many times there are no standard mechanisms available to provide them in a better way.

SIP-based SBCs typically handle both signaling and media, and often modify the session descriptions contained in SIP messages. Consequently, they are, by definition, B2BUAs (Back-to-Back User Agents). The transparency of these B2BUAs varies depending on the functions they perform. For example, some SBCs modify the session description carried in the message and insert a Record-Route entry. Other SBCs replace the value of the Contact header field with the SBCs address, and generate a new Call-ID and new To and From tags.

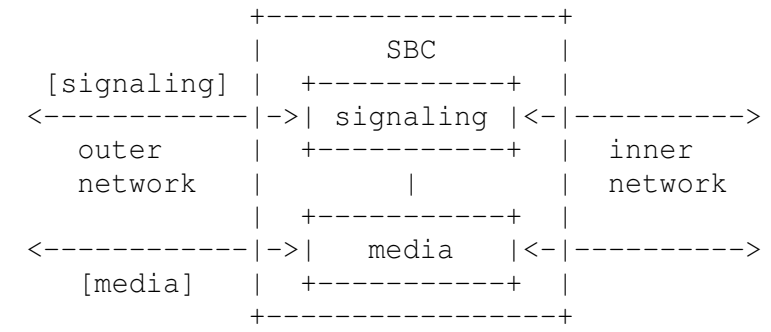


Figure 1: SBC architecture

Figure 1 shows the logical architecture of an SBC, which includes a signaling and a media component. Typically SBCs are located between two networks. In this document, the terms outer and inner network are used for describing these two networks.

There are two typical deployment scenarios where SBCs are used. The first one of them is a peering scenario, where the SBC is located between different-operators' networks. The second deployment scenario is an access scenario, where the SBC is located between the access network and the operator's network.

3. SIP-Unfriendly Functions

This section examines SIP-unfriendly functions that are used in current communication networks. Each subsection describes a particular function or feature, what is the operator's motivation for having it, how it is currently implemented, and why it breaks SIP. Each section also discusses the problems associated to that particular way of implementing it. Providing suggestions for alternative, more SIP-friendly ways of implementing each of the functions is outside the scope of this document.

3.1 Topology Hiding

Topology hiding consists of limiting the amount of topology information given to external parties. Operators want to have this functionality because they do not want the IP addresses of their equipment (proxies, gateways, application servers, etc) to be exposed to outside parties. This may be because they do not want to expose their equipment to DoS (Denial of Service) attacks or because they may use other carriers for certain traffic and do not want their customers to be aware of it. In some environments, the operator's customers may wish to hide the addresses of their equipment or the SIP messages may contain private, non-routable addresses.

The current way of implementing topology consists of having an SBC act as a B2BUA (Back-to-Back User Agents) and remove all traces of topology information (e.g., Via and Record-Route entries) from outgoing messages.

Like a regular proxy server that inserts a Record-Route entry, the SBC handles every single message of a given SIP dialog. However, unlike the proxy server, if the SBC loses state (e.g., the SBC restarts for some reason), it will not be able to route messages properly. For example, if the SBC removes Via entries from a request and then restarts losing state, the SBC will not be able to route responses to that request.

3.2 Media Traffic Shaping

Media traffic shaping is the act of controlling media traffic. Operators have several reasons for having this functionality. Operators want to control the traffic they carry on their network. Traffic shaping helps them create different kinds of billing models (e.g., video telephony can be priced differently than voice-only calls). Additionally, traffic shaping can be used to implement intercept capabilities (e.g., lawful intercept).

Some operators do not actually want to reshape the traffic, but only to monitor it. However, the SIP techniques needed for monitoring media traffic are the same as for reshaping media traffic.

Currently, traffic shaping is performed in the following way. The SBC behaves as a B2BUA and inserts itself, or some other entity under the operator's control, in the media path. In practice, the SBC modifies the session descriptions carried in the SIP messages. As a result, the SBC receives media from one user agent and relays it to the other in both directions.

An example of traffic shaping is codec restriction. The SBC restricts the codec set negotiated in offer/answer [2] exchange between the user agents. After modifying the session descriptions, the SBC can check whether or not the media stream corresponds to what was negotiated in the offer/answer exchange. If it differs, the SBC has the ability to terminate the media stream.

Note SBCs can terminate media streams and SIP dialogs for a number of different reasons (e.g., in a situation where the subscriber runs out of credits or when a user agent loses radio coverage).

The current way of implementing traffic shaping has some SIP-unfriendly aspects. The SBC needs to access and modify the session descriptions (i.e., offers and answers) exchanged between the user

agents. Consequently, this approach does not work if user agents encrypt or integrity-protect their message bodies end-to-end. User agents do not have any way to distinguish the SBC actions from an attack by a MitM (Man-in-the-Middle).

Furthermore, the SBC needs to understand the session description protocol and all the extensions used by the user agents. This means that in order to use a new extension (e.g., an extension to implement a new service) or a new session description protocol, it is not enough with upgrading the user agents; SBCs in the network need also to be upgraded. This fact may slow down service innovation.

Additionally, the SBC may need to generate requests on its own (e.g., a BYE request to terminate a SIP dialog). This does not work when end-to-end authentication is in use.

3.3 Fixing Capability Mismatches

SBCs fixing capability mismatches enable communications between user agents with different capabilities. For example, user agents that support different IP versions, different codecs, or that are in different address realms.

SBCs fixing capability mismatches insert a media element in the media path using the procedures described in Section 3.2. Therefore, these SBCs have the same problems as SBCs performing traffic shaping: the SBC modifies SIP messages without explicit consent from any of the user agents. This breaks end-to-end security and extension negotiation.

Additionally, SBCs may make wrong assumptions about the capabilities of the user agents. When this happens, user agents with compatible capabilities may end up communicating via the SBC instead of doing it directly between them (e.g., the SBC assumes that a dual-stack user agent only supports IPv6).

3.4 NAT Traversal

An SBC performing a NAT (Network Address Translator) traversal function for a user agent behind a NAT sits between the user agent and the registrar of the domain. When the registrar receives a REGISTER request from the user agent and responds with a 200 (OK) response, the SBC modifies such a response decreasing the validity of the registration (i.e., the registration expires sooner). This forces the user agent to send a new REGISTER to refresh the registration sooner that it would have done on receiving the original response from the registrar. The REGISTER requests sent by the user agent refresh the binding of the NAT before the binding expires.

Note that the SBC does not need to relay all the REGISTER requests received from the user agent to the registrar. The SBC can generate responses to REGISTER requests received before the registration is about to expire at the registrar. Moreover, the SBC needs to deregister the user agent if this fails to refresh its registration in time, even if the registration at the registrar would still be valid.

Operators implement this functionality in an SBC instead of in the registrar because the SBC is supposed to have more information about the access the user agent is using. Additionally, distributing this functionality helps offload the registrar.

This approach to NAT traversal does not work when end-to-end confidentiality or integrity-protection is used. The SBC would be seen as a MitM modifying the messages between the user agent and the registrar.

4. Security Considerations

Many of the functions this document describes have important security and privacy implications. If the IETF decides to develop standard mechanisms to address those functions, security and privacy-related aspects will need to be taken into consideration.

5. IANA Considerations

This document has no IANA considerations.

6. Acknowledgements

The ad-hoc meeting about SBCs, held on Nov 9th 2004 at Washington DC during the 61st IETF meeting, provided valuable input to this document. Special thanks goes also to Sridhar Ramachandran, Gaurav Kulshreshtha, and to Rakendu Devdhar.

7. References

7.1 Normative References

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

[2] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

7.2 Informational References

Authors' Addresses

Jani Hautakorpi (editor)
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Jani.Hautakorpi@ericsson.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com

Medhavi Bhatia
NexTone Communications
101 Orchard Ridge Drive
Gaithersburg, MD 20878
US

Email: mbhatia@nextone.com

Robert F. Penfield
Acme Packet
71 Third Avenue
Burlington, MA 01803
US

Email: bpenfield@acmepacket.com

Alan Hawrylyshen
Ditech Communications Corporation
310, 602 - 11 Ave SW
Calgary, Alberta T2R 1J8
Canada

Email: alan@jasomi.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING Working Group
Internet-Draft
Expires: December 8, 2005

V. Hilt
Bell Labs/Lucent Technologies
G. Camarillo
Ericsson
June 6, 2005

Use Cases for Session-Specific Session Initiation Protocol (SIP) Session Policies

draft-hilt-sipping-policy-usecases-00

Status of this Memo

Copyright Notice

Abstract

This draft describes use cases for session-specific Session Initiation Protocol (SIP) sessions policies.

Table of Contents

1. Introduction

Some domains have policies in place, which impact the sessions established using the Session Initiation Protocol (SIP) [3]. These policies are often needed to support the network infrastructure or for the execution of services. For example, wireless networks usually have limited resources for media traffic. A wireless network provider may want to restrict codec usage on the network to lower rate codecs or disallow the use of high bandwidth media types such as video.

Session policies provide a mechanism for a network domain to communicate these policies to user agents. With session policies, user agents know about the policies in the network and can adjust their sessions so that they comply with these policies or simply connect to a domain with less stringent policies.

There has been much discussion in the IETF about the most suitable protocol for session-specific Session Initiation Protocol (SIP) policies [2]. The goal of this draft is to describe common use cases in which session-specific policies are expected to be used. Particularly controversial in this discussion is the mechanism for conveying session information from the user agent to the policy server. This draft will therefore describe the session information a policy server needs to know for each of the discussed use case scenarios.

This document focuses on session-specific policies. In some of the use cases it might also be possible to use session-independent policies [1]. However, session-independent policies are outside of the scope of this document and their use will not be discussed here.

2. Use Cases

Most of the use cases for session-specific policies are based on the insertion of a media intermediary or the limitation of certain aspects of a session. The two categories of use cases are discussed separately in the sections below.

2.1 Media Intermediary

This section provides a general overview over the insertion of a media intermediary with session policies. It then discusses use cases that are based on intermediaries.

2.1.1 General Overview

In this scenario it is assumed that a UA A is located in a policy

enabled domain (see Figure 1), which has an outbound proxy (P A), a policy server (PS A) and a media intermediary (M A). UA A places a call to a UA in another domain (UA B).

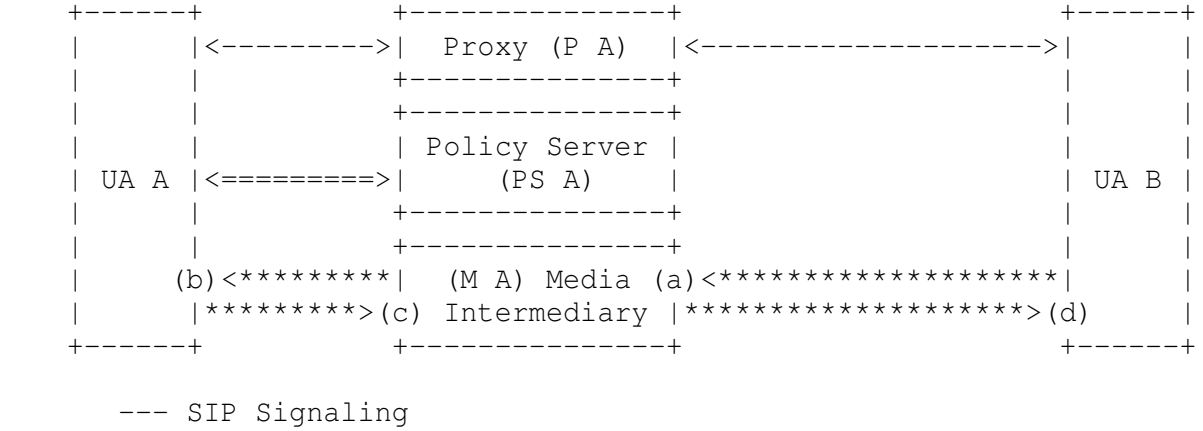


Figure 1

In step (1) in Figure 2 UA A sends out an INVITE and receives the address of the policy server PS A in step (2). It discloses session information (from the offer) to policy server PS A in step (3). The information disclosed includes the IP address and port (b) UA A is going to use for inbound streams.

The policy server uses this information to create an address binding for inbound media streams on M A. The binding connects a port on M A (port (a) in Figure 2) to the address and port provided by UA A (i.e. port (b)). With this binding, M A forwards all incoming traffic on port (a) to port (b) on UA A.

UA A must use the address of M A and port (a) in the offer (instead of its own address and port). PS A therefore returns the policy shown in Figure 3 to UA A in step (4). This policy contains the address of M A (192.0.2.0) and port (a) (6000/6001). Using this policy, UA A creates a new offer' and sends it to UA B in step (5). UA B will now send its media to port (a) on M A. From there it will be forwarded to port (a) on UA A. Thus, M A has been inserted into the inbound media stream.

UA A receives an answer in step (6) and discloses the address of UA B and port (d) (extracted from the answer) to the policy server PS A in step (7). PS A sets up a second binding now for outbound streams on the M A. This binding connects port (c) on M A to the address and port that was in the answer (i.e. the address of UA B and port(d)).

The address of M A and port (c) is returned to UA A in a policy in step (8). UA A applies this policy to the answer. Thus, UA A will now send its outbound traffic to M A, which in turn forwards it to UA B. M A has also been inserted into the outbound media stream.

Proxy P A stays in the signaling path and removes the address bindings on M A when the session is terminated.

Media streams can be encrypted by the UAs. In many cases, media intermediaries need to decrypt the encrypted streams. UA A may therefore need to provide an intermediary with the encryption keys for the current session.

Information the UA needs to disclose to the policy server on the policy channel:

- offer: the UA discloses the IP addresses and ports of all media streams in the offer. The UA may also need to disclose the encryption keys used for the current session.
- answer: the UA discloses the IP addresses and ports of all media streams in the answer. The UA may also need to disclose the encryption keys used for the current session.

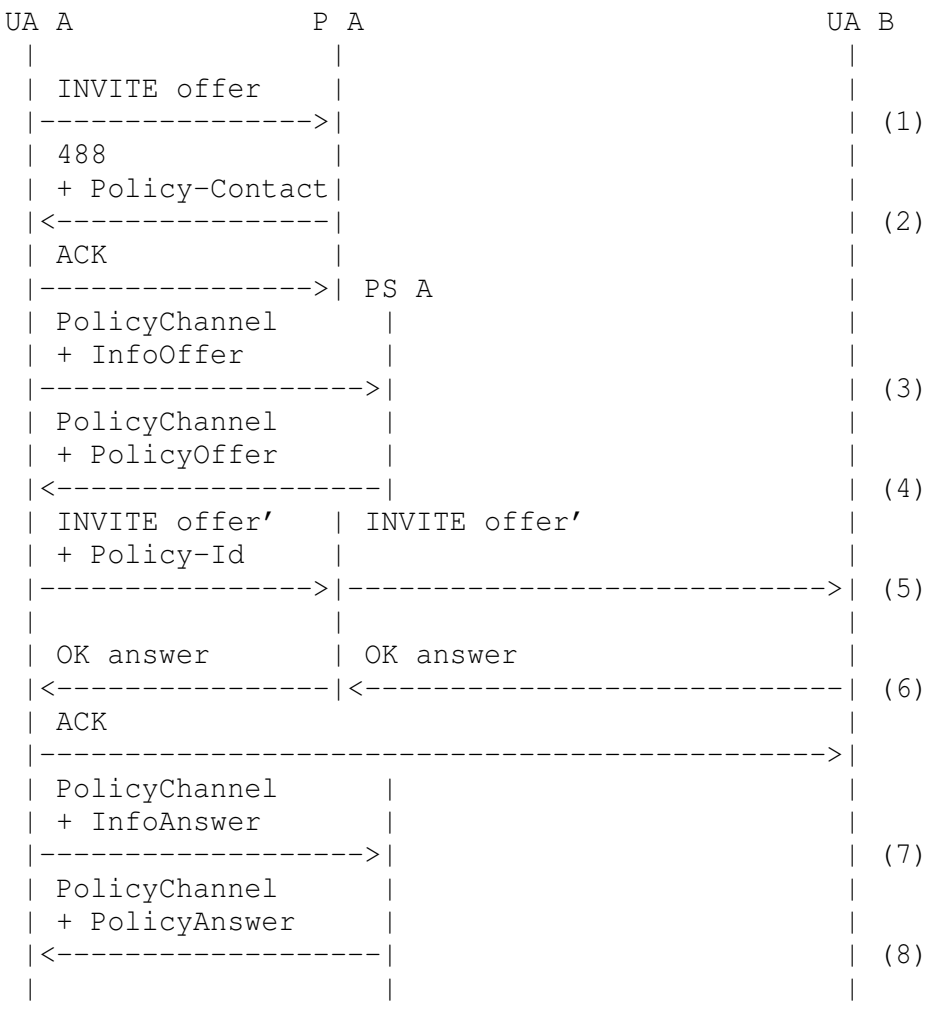


Figure 2

```
<session-policy>
  <media-intermediary>
    <int-uri>192.0.2.0:6000</int-uri>
    <int-addl-port>6001</int-addl-port>
    <int-lroute>none</int-lroute>
  </media-intermediary>
</session-policy>
```

Figure 3

In the above call flow, intermediaries are inserted by modifying the address and ports of media streams. Other techniques for inserting a media intermediary such as using IP tunneling or TURN are also feasible but not discussed in this draft.

2.1.2 Traffic Monitoring

Traffic monitoring generally requires that an entity in the network can examine the media packets of a flow. It may also require that media streams can be associated with SIP sessions.

A media intermediary that has been inserted into a media stream as described above can examine media packets as required. Media streams can be associated with the session for which the policy was created.

2.1.3 Enforcing SLA/Access Control

It is often desirable to enforce policies on media streams, for example, according to a Service Level Agreement (SLA). Enforcing policies requires that a network entity can monitor traffic and, in case policies are violated, block traffic as needed.

Traffic monitoring can be performed by a media intermediary. The intermediary can also decide whether packets are compliant to a given policy or not. It can block streams if policies are violated by dropping the respective packets.

An intermediary can enforce many types of media-level policies. For example, it can enforce limitations session aspects (e.g., codecs, bandwidth, media types), ensure that the media streams correspond with what has been announced in the session description and it can reject traffic that is sent outside of a SIP session. The intermediary can also terminate streams for other reasons, for example, if the user runs out of credit.

2.1.4 Load Balancing and Traffic Shaping

Load balancing and traffic shaping typically requires that the network can determine the route of media streams independent of the path that would be chosen by IP routing. This type of route selection can take multiple criteria into account such as current traffic conditions or peering agreements. For example, a service provide may be connected to another service provider through two links and may want to selectively route calls over one or the other link. In another example, a service provide wants to route traffic through a domain that would otherwise not be traversed by the media streams.

A media intermediary can perform traffic shaping and load balancing. The intermediary receives packets from the UA and can determine the next hop destination for these packets.

A service provider may have multiple media intermediaries at

strategic locations in the network. By selecting one of these intermediaries for a session, it forces the corresponding media streams to go through the chosen intermediary. This way, media traffic can be directed through a specific link, to a certain part of the network or through a certain domain. The routing decision is made by simply including a specific intermediary into the path.

2.1.5 QoS Marking

Two approaches for QoS marking on media streams are feasible with session policies:

- Hosted QoS marking: a media intermediary is inserted as described above. The intermediary performs QoS marking.
- Endsystem-based QoS marking: the policy server returns a session policy that contains the desired DSCP value (following the flow described in Section 2.2). The endpoint itself performs QoS marking using this DSCP value.

2.1.6 NAT/Firewall Traversal

NAT/firewall traversal requires that the NAT/firewall is inserted into the media path. Each endpoint sends its traffic to the NAT/ firewall, which then forwards it to the destination on the other side of the NAT/firewall as permitted by NAT/firewall policies.

An media intermediary that is inserted into the media path as described above can act as NAT/firewall.

2.1.7 Media-level Topology Hiding

Topology hiding is mostly done at the signaling level. However, media-level topology may be used to hide the addresses of media endpoints inside the network.

A media intermediary inserted into streams as described above can perform media-level topology hiding. Such a media intermediary will usually act as RTP mixer/translator. It can remove all internal IP addresses and possibly other sensitive information carried in RTCP reports.

2.1.8 IPv4/IPv6 Interworking

IPv4/v6 translation on media streams can also be performed by a media intermediary.

2.1.9 Media Encryption

A media intermediary can encrypt/decrypt media traffic before forwarding it. Media encryption/decription requires that the intermediary has the encryption keys for the current session.

Media intermediaries may also need to decrypt encrypted media streams in order to perform other functionalities that require a deep inspection of RTP packets.

2.2 Limitations and Restrictions

This section provides a general overview over the use of session policies to restrict certain aspects of a session. It provides example use cases for some of these policies.

2.2.1 General Overview

The scenario is similar to Figure 1 except that there is no media intermediary (in a real architecture, it may still be present to perform other functionalities such as policy enforcement).

Steps (1) to (3) in Figure 4 are the same as above (see Figure 2).

After receiving offer information in step (3), the policy server determines whether a policy is needed or not. If a policy is needed, it is returned to UA A in step (4) (see policy examples below). The UA A creates a modified offer' according to the received policies (step (5)) and completes the session setup in step (6).

Policies that define limitations or restrictions reduce available choices for session parameters. These policies only need to be applied to an offer because the answer can't extend the choices available in an offer.

The policy server PS A can asynchronously update the policy provided to UA A during session setup. For example, a policy server may whish to disallow a codec that was allowed by the initial session policy. In step (7) PS A sends an updated policy to UA A. This policy requires a change in the current session. UA A therefore updates the session by sending a re-INVITE with a modified offer in step (8).

The information the UA needs to disclose to the policy server depends on the individual use case and are described below.

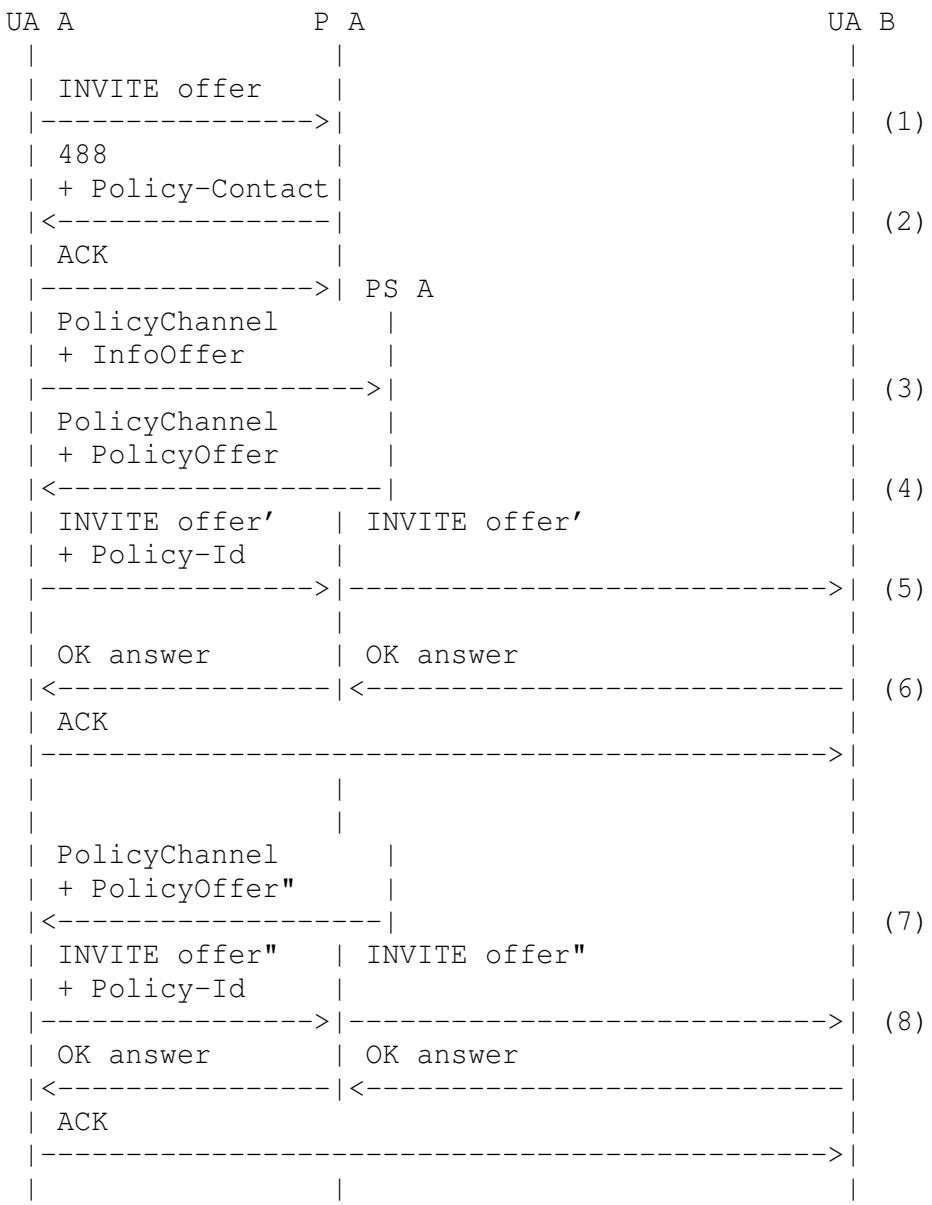


Figure 4

2.2.2 Limit Bandwidth

In some environments there is only a limited bandwidth available to each user agent (e.g. in a wireless network). Communicating bandwidth limitations to the user agent enables the user agent to make an informed decisions, for example, about the codecs or media types to be used in a session.

The following example policy informs the UA of a 80 kbit/s bandwidth limit. In the context of session-specific policies, this policy is returned if the offer can result in a session that exceeds the allowed maximum bandwidth. The offer' that is created based on this policy should not contain choices that may exceed the maximum bandwidth (e.g. it could consist of one audio stream and the codecs G.711 and G.729).

```
<session-policy>
  <max-bandwidth>80</max-bandwidth>
</session-policy>
```

Information the UA needs to disclose to the policy server about the offer on the policy channel:

- The UA must disclose all aspects of a session that may affect the media bandwidth used. These are typically the number of streams together with the media type and the codecs available on a stream. Alternatively, the UA can disclose a maximum bandwidth value.

2.2.3 Restrict Codec Usage

Networks may want to impose restrictions on the codecs that can be used. With session-specific policies it is possible tell the UA that some of the codecs in the offer are prohibited.

The following example policy informs the UA that the codecs G.729 and G.723 are not allowed. Offer' should therefore not include G.729 and G.723.

```
<session-policy>
  <codecs default-policy="allow">
    <codec policy="disallow">G729</codec>
    <codec policy="disallow">G723</codec>
  </codecs>
</session-policy>
```

Information the UA needs to disclose to the policy server about the offer on the policy channel:

- The UA must disclose all codecs it has included in the offer.

2.2.4 Restrict Media Type Usage

Similar to codecs, it is possible to restrict the use of media types using session-specific policies.

The example policy below defines that audio is required, video is allowed and all other media types are disallowed.

```
<session-policy>
  <media-types default-policy="disallow">
    <media-type policy="mandatory">audio</media-type>
    <media-type policy="allow">video</media-type>
  </media-types>
</session-policy>
```

Information the UA needs to disclose to the policy server about the offer on the policy channel:

o The UA must disclose all media types it has included in the offer.

3. Security Considerations

This draft describes the use of mechanisms defined in other drafts and does not introduce additional security issues.

4. IANA Considerations

This draft does not introduce new identifiers or namespaces.

5. Informative References

[1] Hilt, V., Camarillo, G., and J. Rosenberg, "Session-Independent Session Initiation Protocol (SIP) Policies - Profile Data and Mechanisms", draft-ietf-sipping-session-indep-policy-01 (work in progress), October 2004.

[2] Hilt, V., Camarillo, G., and J. Rosenberg, "A Delivery Mechanism for Session-Specific Session Initiation Protocol (SIP) Session Policies", draft-ietf-sipping-session-spec-policy-03 (work in progress), April 2005.

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

Authors' Addresses

Volker Hilt
Bell Labs/Lucent Technologies
101 Crawfords Corner Rd
Holmdel, NJ 07733
USA

Email: volkerh@bell-labs.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING Working Group
Internet-Draft
Expires: January 13, 2006

V. Hilt
Bell Labs/Lucent Technologies
G. Camarillo
Ericsson
J. Rosenberg
Cisco Systems
July 12, 2005

A Delivery Mechanism for Session-Specific Session Initiation Protocol (SIP) Session Policies
draft-hilt-sipping-session-spec-policy-03

Status of this Memo

Copyright Notice

Abstract

This specification defines a delivery mechanism for session-specific Session Initiation Protocol (SIP) sessions policies.

Table of Contents

1. Introduction

Some domains have policies in place, which impact the sessions established using the Session Initiation Protocol (SIP) [4]. These policies are often needed to support the network infrastructure or the execution of services. For example, wireless networks usually have limited resources for media traffic. During periods of high activity, a wireless network provider wants to restrict the bandwidth that is available in a session. With session policies, the user agent is able to learn about the current bandwidth limit and can make an informed decision about the number of streams, the media types, and the codecs it can use in that session. Similarly, a user agent can be informed that certain codecs or media types are disallowed and may not be used in the current session.

In another example, a SIP user agent is using a network which connects to the public Internet through a firewall or a network border device. The provider would like to tell the user agent to direct the media streams to the appropriate IP addresses and ports of that firewall or border device. Knowing this policy enables the user agent to setup sessions with other user agents across the firewall or the network border.

In a third example, a domain wants to perform QoS marking and traffic shaping on media streams. This functionality is implemented in a media intermediary. With session policies, such a media intermediary can be inserted into the media path. In contrast to other methods, the use of session policies does not require the inspection or modification of SIP message bodies by intermediaries (a discussion of this and other design aspects can be found in [8]).

Domains sometimes enforce policies they have in place. For example, a domain might have a configuration in which all packets containing a certain audio codec are dropped. Unfortunately, enforcement mechanisms usually do not inform the user about the policies they are enforcing and silently keep the user from doing anything against them. This may lead to the malfunctioning of devices that is incomprehensible to the user. With session policies, the user knows about the restricted codecs and can use a different codec or simply connect to a domain with less stringent policies. Session policies provide an important combination of consent coupled with enforcement. That is, the user becomes aware of the policy and needs to act on it, but the provider still retains the right to enforce the policy.

Session-policies can be set up in two different ways: specifically for a session or independent of a session. Session-specific policies are created for one particular session, usually under consideration of certain aspects of this session (e.g. the IP addresses and ports

that are used for media). Since session-specific policies are tailored to a session, they only apply to the session they are created for. These policies require a delivery mechanism that enables the exchange of session policy information at the time a session is established. This document defines such a delivery mechanism. It enables user agents to submit session details to a policy server and allows the policy server to provide policies for this session in response.

Session-independent policies on the other hand are independent of a specific session and generally apply to the sessions set up by a user agent. An example is a policy which generally prohibits the use of high-bandwidth codecs. In principle, these policies could also be delivered to user agents individually for each session, using the session-specific delivery mechanism. However, since these policies apply to many sessions, it is more efficient to deliver them to user agents only when the user agent is initialized or a policy changes. The framework for session-independent policies [6] defines a delivery mechanism for session-independent policies. It also defines a minimal session policy format aimed at achieving interoperability between different user agents and policy servers. This policy format is independent of the policy delivery mechanism and can be used for session-independent as well as for session-specific policies.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant implementations.

3. Architecture

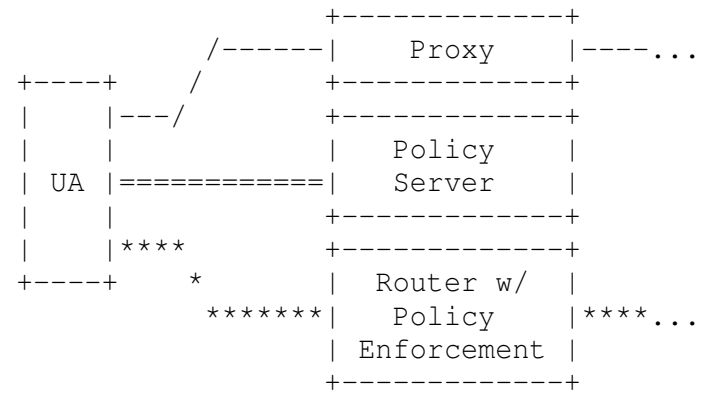


Figure 1

The following entities are involved in setting up session-specific policies (see Figure 1): a user agent (UA), a proxy, a policy server and possibly a router with policy enforcement functionality.

The proxy's role is to provide a rendezvous mechanism for UA and policy server. It conveys the URI of the policy server in its domain to UAs and ensures that UAs know where to retrieve policies from. It does not deliver the actual policies to UAs.

The policy server is a separate logical entity that may be physically co-located with the proxy. Each domain has at most one policy server. The role of the policy server is to generate session policies for a session. It receives session information from a UA, generates a policy and returns that policy back to the UA. The way policies are generated is outside the scope of this specification. A policy server could, for example, use local rules, query external sources for additional information or retrieve policies from a separate policy infrastructure.

A UA receives the URI of a policy server from the proxy. It uses this URI to establish a policy channel to the policy server. It provides information about the current session to the policy server and receives session policies in response. The UA may also receive policy updates from the policy server during the course of a session.

A network may have a policy enforcement infrastructure in place. However, this specification does not make any assumptions about the

enforcement of session policies and the mechanisms defined here are orthogonal a policy enforcement infrastructure. Their goal is to provide a means for the UA to convey session information to a policy server and to receive the policies that apply to this session in response.

The protocol defined in this specification follows a separate channel model. SIP signaling is only used to rendezvous the UA with the policy server. From this point on, UA and policy server communicate directly with each other over a separate policy channel. This is opposed to a piggyback model, where the exchange of session and policy information between the user agent and the policy server is piggybacked onto SIP signaling messages exchanged between the two user agents.

A disadvantage of the separate channel model is that it requires additional messages for the exchange of policy information. The advantages of using a separate policy channel is that it decouples the exchange of signaling messages between endpoints from the exchange of policy information between endpoint and policy server. This decoupling enables the use of encryption on the signaling path (to secure the communication between endpoints) and on the policy channel (to secure the communication between endpoint and policy server). Existing schemes for authorization, authentication, signing and encryption can be used on the policy channel. This is not possible if policies are piggybacked onto the signaling messages. Another advantage of the separate channel model is that policies do not travel along the signaling path possibly crossing may domains. If policy server and UA are in the same network, policy information never leaves this network. In addition, endpoints can specifically decide which aspects of a session they want to disclose to a certain policy server. Finally, a policy server does not rely on a SIP signaling message flowing by to provide a session policy to an endpoint. A policy server can use the separate channel at any time to update session policies as needed.

The communication on the policy channel between a UA and a policy server involves two main operations:

1. The UA discloses information about the current session and the offer/answer exchange to the policy server.
2. The policy server sends policy instructions to the UA.

Some types of policies do not involve sending policy instructions, but only information disclosure. Still, a general session-specific policy mechanism needs to support both operations.

The same way, some policy servers only need to inspect the offer, but

not the answer. Nevertheless, a general mechanism needs to consider policy servers which need to inspect both.

Finally, some policy servers need to update the session policies that have been sent to a UA. Again, a general mechanism should provide this capability.

4. Overview of Operation

This section provides example call flows to illustrate the establishment of session-specific policies. It does not contain a normative protocol definition.

In the following scenario, there are two domains (domain A and domain B), which both have session-specific policies for UAs in their domain. Both domains do not provide policies to UAs outside of their domain. The two domains have a proxy (P A and P B) and a policy server (PS A and PS B). The policies in both domains involve the session description offer and answer.

4.1 Offer in Request

The first call flow depicts an INVITE transaction with the offer in the request. It is assumed that the UAC does not have previous knowledge about the policy server in its domain.

(1) UA A sends an INVITE to proxy P A. P A knows that policies apply to this session and (2) returns a 488 to UA A. P A includes the URI of PS A in the 488 response. (3) UA A contacts PS A, discloses the session description offer to PS A and (4) receives policies for the offer. (5) UA A reformulates the INVITE request under consideration of the received policies and includes a Policy-Id header to indicate that it has already contacted PS A. P A does not reject the INVITE this time and removes the Policy-Id header when forwarding the INVITE. P B adds a Policy-Contact header containing the URI of PS B. (6) UA B uses this URI to contact PS B and discloses the offer and the answer it is about to send. (7) UA B receives policies from PS B and applies them to the offer and answer respectively. (8) UA B returns the updated answer in the 200 OK. (9) UA A contacts PS A with the answer and (10) retrieves answer policies from PS A.

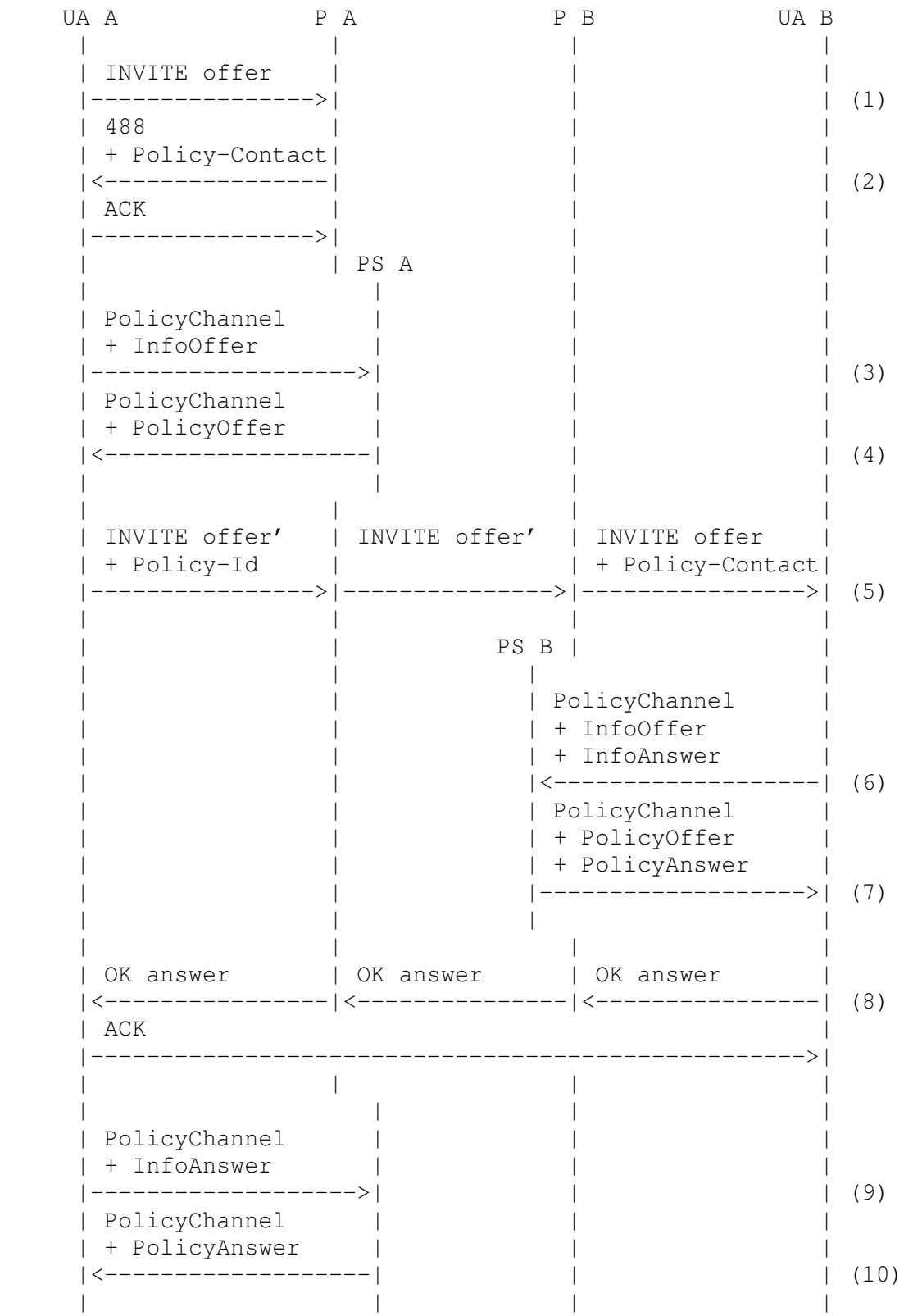
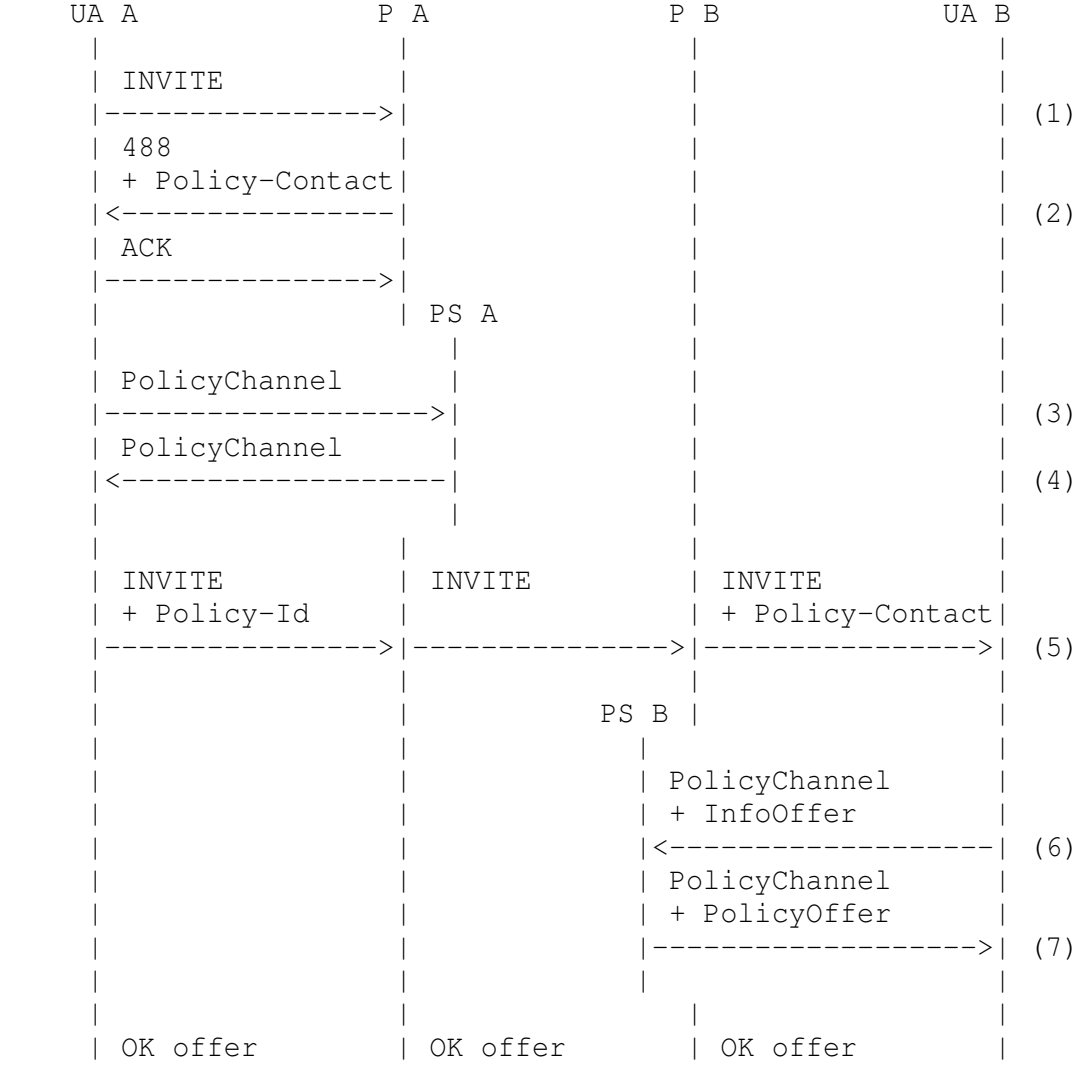

UA A
P A
P B
UA B
INVITE offer
488
+ Policy-Contact
ACK
PS A
PolicyChannel
+ InfoOffer
PolicyChannel
+ PolicyOffer
INVITE offer'
+ Policy-Id
INVITE offer'
INVITE offer
+ Policy-Contact
PS B
PolicyChannel
+ InfoOffer
+ InfoAnswer
PolicyChannel
+ PolicyOffer
+ PolicyAnswer
OK answer
OK answer
OK answer
ACK
PolicyChannel
+ InfoAnswer
PolicyChannel
+ PolicyAnswer
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)

Figure 2

4.2 Offer in Response

This call flow depicts an INVITE transaction with the offer in the response.

Steps (1) - (8) are analogous to steps (1) - (8) in the above flow. An important difference is that in steps (9) and (10) UA A contacts PS A after receiving the offer in the 200 OK but before returning the answer in step (11). This enables UA A to return the final answer, which includes all applicable policies, in the ACK. However, it requires that PS A immediately returns a policy to avoid a delay in the transmission of the ACK. This is similar to Flow I in [7].



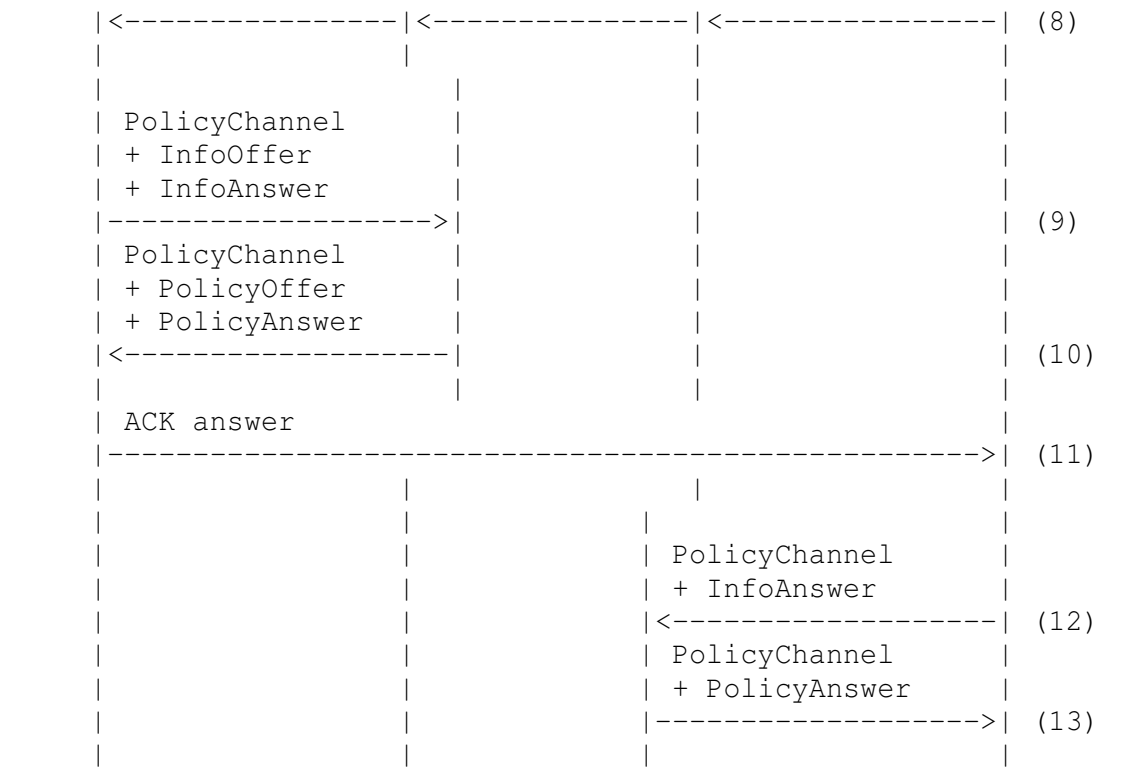


Figure 3

5. UA/Policy Server Rendezvous

The first step in setting up session-specific policies is to rendezvous the UAs with the relevant policy servers. This is achieved by providing the URIs of all policy servers relevant for a session to the UAs.

5.1 UAC Behavior

When a UA compliant to this specification generates an INVITE or UPDATE request, it MUST include a Supported header field with the option tag "policy" in the request.

A UAC may receive a 488 in response to an INVITE or UPDATE request, which contains a Policy-Contact header field. This is a new header that contains the URI of a policy server. A 488 response with this header is generated by a proxy to convey the URI of the local policy server to the UAC. The UAC SHOULD contact this URI to retrieve the session policies that apply to the current request. If the UAC decides to accept the received policies, it SHOULD apply them to the request and resend the updated request.

If the UAC has applied session policies to a request, it MUST insert a Policy-Id header into that request. The Policy-Id header MUST include the URIs of all policy servers the UAC has contacted during the processing of that request. The Policy-Id header enables a proxy to determine whether the URI of its policy server is already known to the UAC (and thus the request can be passed through) or whether the URI still needs to be conveyed to the UAC in a 488 response.

In some cases, a request may traverse multiple domains with session-policies in place. Each of these domains may return a 488 response containing a policy server URI. Since the UAC contacts the policy server URI received in a 488 response before it resends the request, session policies are always applied to a session in the order in which the request traverses through these domains. Policies of the local network are applied first (since the local proxy is the first proxy that responds with a 488 response), policies of the first policy-enabled transit network are applied next, and so on. The order in which policies are applied to a session may be significant, for example, if a policy inserts media intermediaries into the media path.

Session policies may apply to the offer, the answer or both session descriptions. Depending on the requirements of the policy, a UAC may need to contact the policy server with the offer and with the answer. A UAC MUST always contact the same policy servers for the offer and the answer. If the UAC receives an answer in the response to an INVITE request (i.e. the request contained the offer), it MUST send the ACK before retrieving the policies for the answer from the policy server. If the UAC receives a response with an offer (i.e. the INVITE request did not contain an offer), the UAC MUST first contact the policy server to retrieve session policies and apply these policies before sending the answer in the ACK. The answer in the ACK will therefore already consider the relevant policies.

> This approach assumes that the policy server immediately responds to a policy request and does not require manual intervention to create a policy. A delay in the response from the policy server would delay the transmission of the ACK and could trigger retransmissions of the INVITE response (also see the recommendations for Flow I in [7]).

A UAC SHOULD cache the URI of the local policy server. It receives this URI in a 488 from the proxy in the local domain. The UAC SHOULD use this URI to retrieve session policies for a new INVITE or UPDATE request before it is sent. Caching the local policy server URI avoids the retransmission of this URI for each new INVITE or UPDATE request. Some domains may want to prevent the UAC from caching the local policy server URI. For example, if the policy server does not

need to be involved in all sessions or the policy server URI changes from session to session. A proxy can mark the URI of such a policy server as "non-cacheable". The UA SHOULD NOT cache a non-cacheable policy server URI and SHOULD remove the current URI from its cache when receiving such a URI.

The UAC SHOULD NOT cache policy server URIs it has received from proxies outside of the local domain. These policy servers may not be relevant for subsequent sessions, which may go to a different destination and may traverse different domains.

The UAC SHOULD maintain a list of policy server URIs for each dialog. This list SHOULD include all policy server URIs that were contacted for the initial INVITE that created the dialog. The UAC should keep this list until the dialog is terminated. The UAC SHOULD contact the policy server URIs in this list before sending an INVITE or UPDATE request within that dialog. This avoids the retransmission of policy server URIs for mid-dialog requests. Contacting policy servers for mid-dialog INVITE or UPDATE requests is needed to enable policy servers to keep track of the session description and to update policies accordingly.

5.2 UAS Behavior

An incoming INVITE or UPDATE request may contain a Policy-Contact header with a list of policy server URIs. The UAS SHOULD use these URIs to retrieve session policies. The UAS MUST use the policy server URIs in the order in which they were contained in the Policy-Contact header, starting with the topmost value.

If the UAS receives an ACK with an answer, it may need to contact the policy servers again depending on the policy. In this case, it MUST contact the same policy servers it has contacted for the offer.

5.3 Proxy Behavior

A proxy may provide the URI of the local policy server to the UAC or the UAS when processing an INVITE or UPDATE request.

If an INVITE or UPDATE request contains a Supported header field with the option tag "policy", the proxy MAY reject the request with a 488 response to provide the local policy server URI to the UAC. Before rejecting a request, the proxy MUST check whether the request has a Policy-Id header field that already contains this policy server URI. If the request does not have such a header or the local policy server URI is not present in that header, then the proxy MAY reject the request with a 488. The proxy MUST insert a Policy-Contact header in the 488 response that contains the URI of the local policy server.

The proxy MAY add the header field parameter "non-cacheable" to prevent the UAC from caching this policy server URI.

If the local policy server URI is already present in the Policy-Id header of an INVITE or UPDATE request, the proxy MUST NOT reject the request as described above. The proxy SHOULD remove this policy server URI from the Policy-Id header field before forwarding the request.

The proxy MAY insert a Policy-Contact header field into an INVITE or UPDATE request in order to convey the policy server URI to the UAS. If the request already contains a Policy-Contact header field, the proxy MUST insert the URI before of all existing values at the beginning of the list. A proxy MUST NOT change the order of existing Policy-Contact header values.

5.4 Header Definition and Syntax

The Policy-Id header field is inserted into an INVITE or UPDATE request by the UAC. It identifies all policy servers the UAC has contacted for the request. A Policy-Id header value is the URI of a policy server.

The syntax of the Policy-Id header field is:

```
Policy-Id        = "Policy-Id" HCOLON absoluteURI
                   *(COMMA absoluteURI)
```

The Policy-Contact header field can be inserted into INVITE and UPDATE requests by a proxy. It contains an ordered list of policy server URIs that need to be contacted by the UAS. The UAS starts to process the header field at the topmost value of this list. New header field values are inserted at the top. The Policy-Contact header field effectively forms a stack. The "non-cacheable" header field parameter MUST NOT be used in a request.

The Policy-Contact header field can also be inserted into a 488 response to an INVITE or UPDATE request by a proxy. It contains a policy server URI that needs to be contacted by the UAC. A proxy MAY add the "non-cacheable" header field parameter to indicate that the UAC should not cache the policy server URI.

The syntax of the Policy-Contact header field is:

```
Policy-Contact   = "Policy-Contact" HCOLON policyURI
                   *(COMMA policyURI)
policyURI        = absoluteURI [ SEMI "non-cacheable" ]
```

The BNF for absoluteURI is defined in [4].

Table 1 is an extension of Tables 2 and 3 in [4]. The column 'UPD' is for the UPDATE method [3].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	UPD
Policy-Id	R	rd	-	-	-	o	-	-	o
Policy-Contact	R	a	-	-	-	o	-	-	o
Policy-Contact	488	a	-	-	-	o	-	-	o

Table 1: Policy-Id and Policy-Contact Header Fields

Figure 6

6. Policy Channel

The policy channel is set up between the UA and the policy server. This channel is needed to accomplish two tasks: first, to convey information about the current session from the UA to the policy server and, second, to return the policies for that session from the policy server back to the UA.

6.1 Session Information

OPEN ISSUE: Which method should be used to convey session information from the UA to the policy server? Use cases for session-specific policies that may help resolving this issue are discussed in [6]. The following proposals have been made:

1. SIP SUBSCRIBE: session information is conveyed to the policy server in the body of SUBSCRIBE requests. The UA subscribes to session policies for each session. Semantically, session information is a filter criteria that selects the policies, which apply to the current session from the pool of all available policies. This semantics may not be applicable to all policies, in particular, if they are generated dynamically based on session information. Also, session information can only be provided by the subscriber and not by a third party.
2. SIP PUBLISH: the UA submits session information to the policy server in the body of a SIP PUBLISH request. The policy server uses this information to generate policies and makes these policies available for subscriptions. The UA can subscribe to these policies and will receive all policies (new or updated) via NOTIFY requests. The subscription can be established at the same time the PUBLISH request is sent. The UA may even use a single subscription to receive the policies for all sessions it sets up. In this case, each NOTIFY would cover all policies from that

server (content indirection may be used).
3. HTTP: Similar to PUBLISH. Instead of using a PUBLISH request, the UA submits session information in the body of a HTTP request. The policies are received through a subscription. The UA needs two policy server URIs: a SIP URI (for the subscription) and a HTTP URI (to upload session information).
4. XCAP: Similar to HTTP. Instead of using plain HTTP, XCAP is used to upload session information.

OPEN ISSUE: Which information should be disclosed to the policy server. Is this policy specific? Or should the UA generally disclose the session description?

6.2 Session Policies

The UA accesses the policies that apply to the current session through the policy server URIs it has received during session establishment (see Section 5). The UA subscribes to these URIs and receives the current session policies. The policies for a session may change while the session is in progress. The UA is notified about updates to policies through the subscription.

The session policy documents may be contained directly in the body of a NOTIFY message or they may be retrieved from an URI contained in the NOTIFY via content indirection.

The subscription to session-specific policies is based on the Framework for SIP User Agent Profile Delivery [2]. It uses the new profile-type 'policy', which is defined in this document. Defining a new profile type for session-policies enables the decoupling of session-specific policies from other sources of profile information, such as user, device, or local profiles. 'Policy' profiles are provided by domains that have session-specific policies in place.

6.2.1 Event Header Parameters

The new token 'policy' is defined for the 'profile-type' event header parameter. This extends the syntax of the profile-type event header parameter [2] as follows:

```
profile-types      =  "device" / "user" / "application" /
                      "local" / "policy"
```

A SUBSCRIBE request for the policy profile-type may contain the 'network-user' parameter with the user's AoR. This parameter may be needed since the subscription URI does not reveal the user's AOR. Knowing the user's AOR may help a policy server to decide whether or not to accept a subscription and to determine which policies are

applicable.

6.2.2 The Use of URIs

The SUBSCRIBE request URI for the 'policy' profile is a policy server URI the user agent has received through mechanisms described in Section 5.

A policy server URI MAY contain a 'document' URI parameter when it is received by the UA. This parameters can be used to identify a specific document on the policy server, to which the UA should subscribe. If this parameter is present in a policy server URI, it MUST be copied into the 'document' event header parameter of the SUBSCRIBE request. The 'document' parameter MUST be removed from the policy server URI before it is used in the SUBSCRIBE request URI.

6.2.3 Subscriber Behavior

The 'policy' profile SHOULD be used when subscribing to a policy server URI. The UA SHOULD establish a separate subscription to each policy server URI it has received. It may receive session-specific policies through each of these subscriptions. The subscriber SHOULD include the 'network-user' parameter in the SUBSCRIBE request.

6.2.4 Notifier Behavior

A notifier (i.e. a policy server) MUST immediately respond to SUBSCRIBE requests and MUST immediately send a NOTIFY in case it accepts the subscription. If the notifier cannot respond with a session policy right away, it must send an empty policy document and later update this policy. Note that updating a policy may require the subscriber to re-negotiate session parameters and should be avoided if possible.

> The timely transmission of session policies is in particular important if the UA (i.e. the subscriber) is requesting policies for an offer in an INVITE response (see Section 4.2).

The policy server SHOULD authenticate the user before submitting a policy that grants additional privileges to the user.

6.2.5 Example

The following example contains a policy server URI and a SUBSCRIBE message.

```
Policy Server URI:
  sip:policy@ps.example.com;document=session-id/48ei48rj474k


SUBSCRIBE sip:policy@ps.example.com SIP/2.0
Via: SIP/2.0/TCP terminal.example.com;branch=z9hG4bK6d6d35b6
Event: sip-profile;profile-type="policy";
  document="session-id/48ei48rj474k";
  vendor="vendor.example.com";model="Z100";version="1.2.3";
  network-user="alice@example.com"
To: sip:policy@ps.example.com
From: sip:alice@example.com;tag=1234
Call-ID: ue8K743jRhr83@terminal.example.com
CSeq: 1 SUBSCRIBE
Contact: <sip:alice@terminal.example.com>
Accept: message/external-body, application/session-policy+xml
Content-Length: 0
```

7. Updating Policies

A UA may receive policy updates through a policy channel. The UA SHOULD apply these policies to the current session. It MUST generate a re-INVITE or UPDATE request if the updated policies modify aspects of the session that need to be communicated to the peer UA.

8. Security Considerations

In particular authentication and authorization are critical issues that need to be addressed here.

[TBD.]

9. IANA Considerations

[TBD.]

10. References

10.1 Normative References

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

[2] Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-06 (work in progress), February 2005.

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

[4] 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.

10.2 Informative References

[5] Hilt, V. and G. Camarillo, "Use Cases for Session-Specific Session Initiation Protocol (SIP) Session Policies", draft-hilt-sipping-policy-usecases-00 (work in progress), June 2005.

[6] Hilt, V., Camarillo, G., and J. Rosenberg, "Session Initiation Protocol (SIP) Session Policies - Document Format and Session-Independent Delivery Mechanism", draft-ietf-sipping-session-indep-policy-03 (work in progress), June 2005.

[7] Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)", BCP 85, RFC 3725, April 2004.

[8] Rosenberg, J., "Requirements for Session Policy for the Session Initiation Protocol (SIP)", draft-ietf-sipping-session-policy-req-02 (work in progress), July 2004.

Authors' Addresses

Volker Hilt
Bell Labs/Lucent Technologies
101 Crawfords Corner Rd
Holmdel, NJ 07733
USA

Email: volkerh@bell-labs.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com

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

Email: jdrosen@cisco.com

Appendix A. Acknowledgements

Many thanks to Allison Mankin for the discussions and the suggestions for this draft. A big thanks also to everyone who contributed by providing feedback on the mailing list and in IETF meetings.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING
Internet-Draft
Expires: January 19, 2006

J. Rosenberg
Cisco Systems
G. Camarillo, Ed.
Ericsson
D. Willis
Cisco Systems

July 18, 2005

A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP)

draft-ietf-sipping-consent-framework-02.txt

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) supports communications across many media types, including real-time audio, video, text, instant messaging, and presence. In its current form, it allows session invitations, instant messages, and other requests to be delivered

from one party to another without requiring explicit consent of the recipient. Without such consent, it is possible for SIP to be used for malicious purposes, including spam and denial-of-service attacks. This document identifies a framework for consent-based communications in SIP.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] supports communications across many media types, including real-time audio, video, text, instant messaging and presence. This communication is established by the transmission of various SIP requests (such as INVITE and MESSAGE [2]) from an initiator to the recipient, with whom communication is desired. Although a recipient of such a SIP request can reject the request, and therefore decline the session, a SIP network will deliver a SIP request to the recipient without their explicit consent.

Receipt of these requests without explicit consent can cause a number of problems in SIP networks. These include spam and DoS (Denial of Service) attacks. These problems are described in more detail in a companion requirements document [5].

This specification defines a basic framework for adding consent-based communication to SIP.

2. Definitions

Recipient URI: The request-URI of an outgoing request sent by an entity (e.g., a user agent or a proxy). The sending of such request may have been the result of a translation operation.

Target URI: The request-URI of an incoming request that arrives to an entity (e.g., a proxy) that will perform a translation operation.

Translation operation: Operation by which an entity (e.g., a proxy) translates the request URI of an incoming request (i.e., the target URI) into one or more URIs (i.e., recipient URIs) which are used as the request URIs of one or more outgoing requests.

3. Consent between User Agents

The simplest type of consent occurs between user agents. Given a set of user agents using consent-based communications, any particular user agent needs to obtain permission to communicate with any other user agent. That is, if user agent A wants to communicate with user agent B, user agent A needs to obtain permission from user agent B in order to do so.

This situation can be found in many current instant messaging systems, which do not allow sending an instant message unless the receiving user has explicitly given permission to the sender.

4. Relays

In addition to the simple scenario described in Section 3 where user agents obtain permissions to communicate directly between them. this framework covers scenarios that involve relays between the user agents.

A relay is defined as any SIP server, be it a proxy, B2BUA (Back-to-Back User Agent), or some hybrid, which receives a request and translates the request URI into one or more next hop URIs to which it then delivers a request. The request URI of the incoming request is referred to as 'target URI' and the destination URI of the outgoing requests is referred to as 'recipient URIs', as shown in Figure 1.

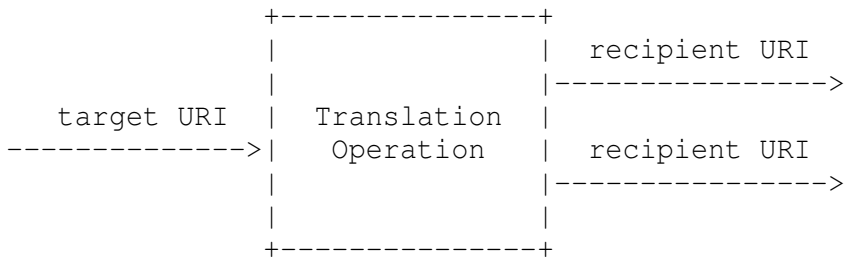


Figure 1: Translation operation

Thus, an essential aspect of a relay is that of translation. When a relay receives a request, it translates the request URI into one or more additional URIs. Or, more generally, it can create outgoing requests to one or more additional URIs. The translation operation is what creates the consent problem.

Additionally, since the translation operation can result in more than one URI, it is also the source of amplification. Servers that do not perform translations, such as outbound proxy servers, do not cause amplification.

Since the translation operation is based on local policy or local data (such as registrations), it is the vehicle by which a request is delivered directly to an endpoint, when it would not otherwise be possible to. In other words, if a spammer has the address of a user, 'sip:user@example.com', it cannot deliver a MESSAGE request to the UA (User Agent) of that user without having access to the registration data that maps 'sip:user@example.com' to the user agent on which that user is present. Thus, it is the usage of this registration data, and more generally, the translation logic, which must be authorized in order to prevent undesired communications.

The reference architecture is shown in Figure 2. In this architecture, a user agent client (UAC) wishes to send a message to a request URI representing a resource in domain A (sip:resource@A). This request may pass through a local outbound proxy (not shown), but eventually arrives at a server authoritative for domain A. This server, which acts as a relay, performs a translation operation, translating the target URI into one or more recipient URIs, which may or may not belong to domain A. This relay may be, for instance, a proxy server or a URI-list service [7].

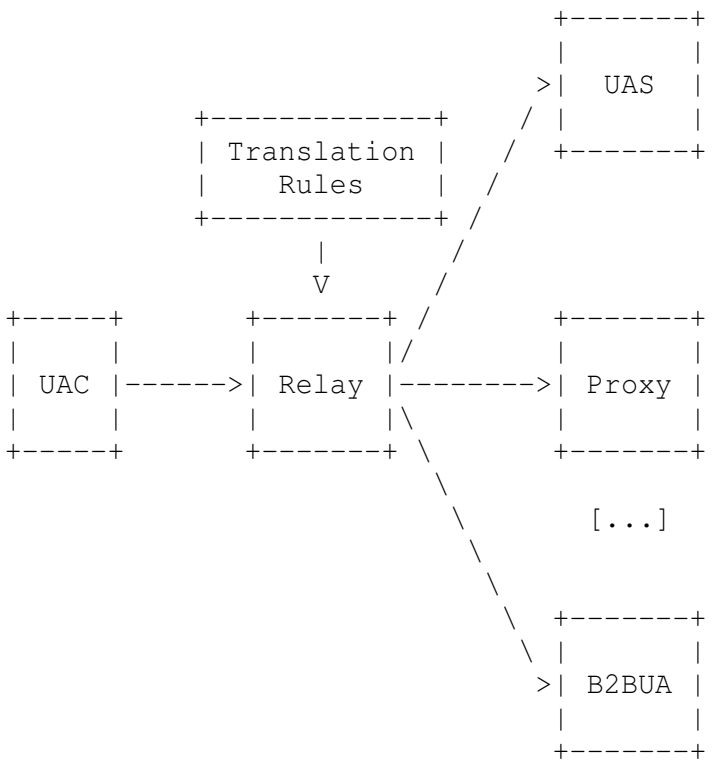


Figure 2: Relay performing a translation

5. Structure of a Permission

This framework centers on the idea that a relay will only perform a translation if a permission is in place authorizing that translation. Thus, the translation rules associated to a translation include a set of recipient URIs and the permissions associated with each of these URIs. For example, one recipient may have given permission for the translation while another recipient may not have given it. In this case, the relay would only be authorized to perform the translation towards the recipient that gave permission.

A permission is an object, represented in XML, that contains several pieces of data:

Identity of the Sender: A URI representing the identity of the sender for whom permissions are granted.

Identity of the Original Recipient: A URI representing the identity of the original recipient, which is used as the input for the translation operation. This is also called the target URI.

Identity of the Final Recipient: A URI representing the result of the translation. The permission grants ability for the sender to send requests to the target URI, and for a relay receiving those requests to forward them to this URI. This is also called the recipient URI.

Operations Permitted: A set of specific methods or qualifiers for which the permission applies. For example, the permission may only grant relaying for INVITE requests and not for MESSAGE requests.

Signature: A digital signature over the rest of the permission, signed by an entity that can identify itself as the recipient URI. The signature is not always present.

Permission documents may contain wildcards. For example, a permission document may authorize any relay to forward INVITE requests coming from a particular sender to a particular recipient. Such a permission document would apply to any target URI. That is, the field containing the identity of the original recipient would match any URI.

The format for permission documents is defined in...

OPEN ISSUE: the common policy format [3] has elements that we may want to reuse (e.g., identity). However, we probably want to define our own format instead of extending that one because we do not need actions or transformations, and we need more than one identity per document. The new format would be something like the one in Figure 3.

Figure 3 contains an example of a permission document that authorizes the relay handling the URI list 'sip:bobs-friends@example.com' to relay INVITE requests to Alice no matter who the sender is.

```
<target>
  <identity>
    <id>bobs-friends@example.com</id>
  </identity>
</target>
<sender>
  <identity>
    <any-identity/>
  </identity>
</sender>
<recipient>
  <identity>
    <id>alice@example.com</id>
  </identity>
</recipient>
<operations>
  <method>INVITE</method>
</operations>
```

Figure 3: Permission document

6. Two-party Scenario

This section describes the fundamental operations of this framework in a two-party scenario. The descriptions are illustrated with an example (see Figure 4).

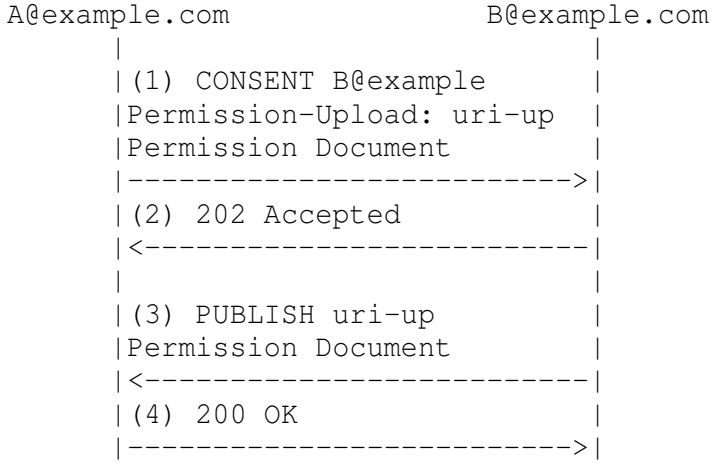


Figure 4: Two-party Scenario

A creates a CONSENT request which contains the permission document in Figure 5 in its body:

```
<target>
  <identity>
    <any-identity/>
  </identity>
</target>
<sender>
  <identity>
    <id>A@example.com</id>
  </identity>
</sender>
<recipient>
  <identity>
    <id>B@example.com</id>
  </identity>
</recipient>
<operations>
  <method>INVITE</method>
</operations>
```

Figure 5: Permission document

This document describes the permissions that A is requesting from B. Note that the permission applies to any target URI. Therefore, the permission is not specific to any particular relay.

Additionally, the CONSENT request contains the URI where B is requested to upload the permission document. This URI is carried in a Permission-Upload header field.

On receiving the CONSENT request, B signs the permission document and uploads it to the URI in the Permission-Upload header field using a PUBLISH request. At this point, A has permission to send INVITE requests to B.

7. Permission Servers

Section 6 described how a user agent receiving a CONSENT request can use a PUBLISH request to grant certain permissions. Nevertheless, users are not on-line all the time and, so, sometimes are not able to receive CONSENT requests.

This issue is also found in presence, where a user's status is reported by a presence server instead of by the user's user agents, which can go on and off-line. Similarly, we define permission servers. Permission servers are network elements that act as SIP user agents and handle CONSENT requests for a user.

Permission servers inform users about new CONSENT requests using the

"grant-permission" event package. Figure Figure 6 illustrates this point.

The user associated with the target URI SUBSCRIBEs (1) to the "grant-permission" event package at the permission server. This event package models the state of all pending CONSENT requests for a particular resource, for which permissions do not yet exist. When a new CONSENT request (3) arrives for which permissions have not been granted, a NOTIFY (5) is sent to the user. This informs them that permission is needed for a particular sender. The NOTIFY contains the permissions requested and the URI to upload the document.

> There is a strong similarity between the watcherinfo event package and the grant-permission event package. Indeed, the grant-permission package is effectively a superset of watcherinfo. Once in place, presentities could use the grant-permission event package for presence in addition to all other services for which opt-in is being provided.

When a user is notified of a new pending CONSENT request, the user follows regular procedures to upload the permissions that were requested (7).

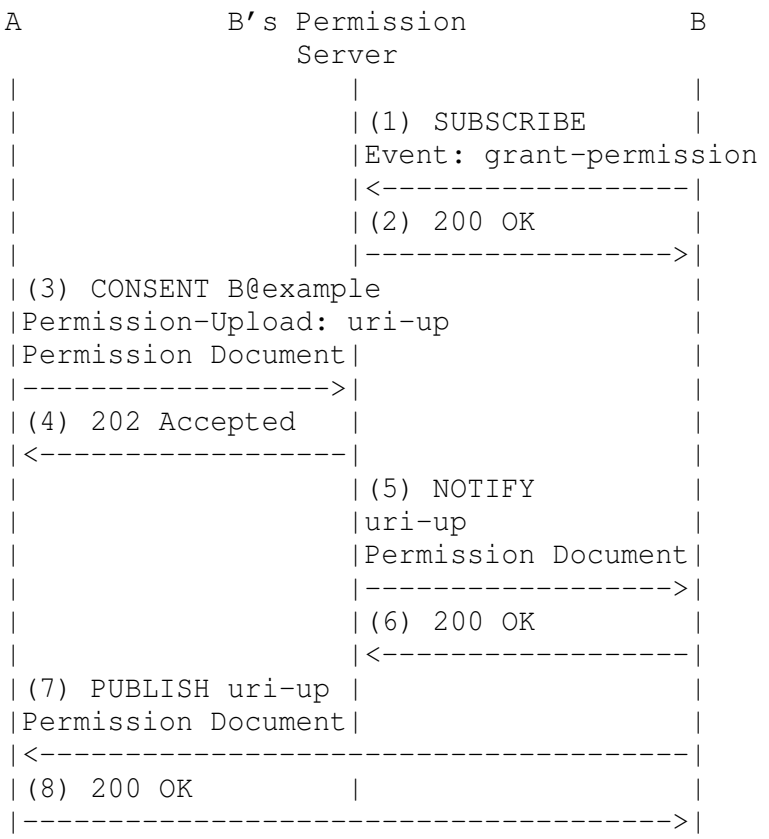


Figure 6: Permission server operation

8. Relay Scenario

Manipulating translation rules at a relay may involve obtaining permissions from some users. For example, if a new recipient URI is added to a URI-list service, the URI-list service will need to obtain permission to send request to that URI. Thus, when a new recipient URI is added to a set of translation rules, the URI will be in the "Permission Pending" state until permissions are obtained for it. Relays do not send requests to recipient URIs in this state.

Therefore, effectively, adding a new recipient URI to a set of translation rules involves two operations: adding the new URI to the list of recipient URIs and obtaining permissions to send requests to it.

The addition of the new recipient URI can be performed using different methods (e.g., XCAP). All these methods provide the entity adding the recipient URI with a URI to upload the permission document associated with the new recipient URI. Such an entity can go off

obtain the permissions and then upload them into the relay. Figure 7 shows an example of this process. A adds B's URI to the relay's list of recipient URIs, obtains permissions from B, and uploads them to the relay.

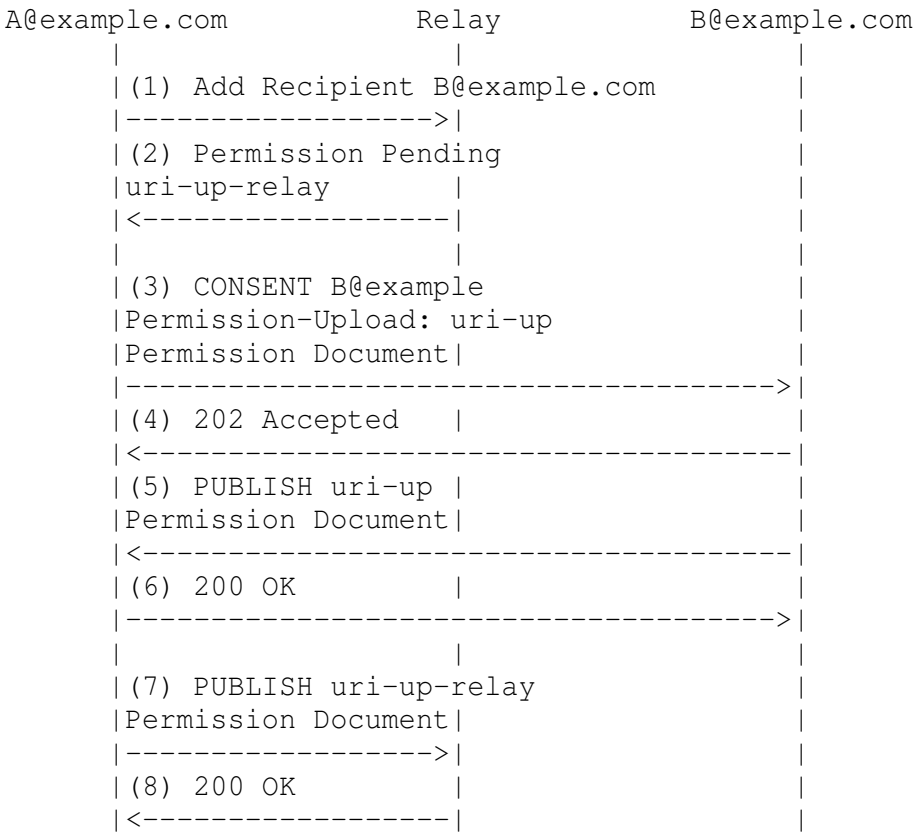


Figure 7: Relay Scenario

9. Relays Obtaining Permissions

Section 8 shows how a user can add a recipient URI to a relay's translation rules, obtain permissions to send requests to it, and upload them to the relay. This works well when there is an infrastructure that allows users to sign permission documents. This way, the relay knows that the permission document was generated by the owner of the recipient URI. However, such infrastructure is not always available.

Additionally, some architectures prevent users from communicating directly between them forcing them to always communicate via a relay.

In this case, a user cannot contact directly the owner of the recipient URI to obtain permissions to send requests to the URI.

This framework handles the previous situations by having relays request permissions directly from the recipient URIs. The relay sends a CONSENT request to the recipient URI. As usual, the CONSENT request carries a permission document describing the permissions being requested and a URI where the permission document needs to be uploaded. The recipient uses a PUBLISH request to upload the permission document to that URI.

9.1 Permission Document Authentication

A relay obtaining permissions from a recipient needs to make sure that the permission document received was generated by the recipient. If the infrastructure does not allow signing permission documents, the relay can use two methods to authenticate the permission document: SIP identity or a return routability test.

The SIP identity mechanism can be used to authenticate the sender of the PUBLISH request uploading the permission document. This way, the relay ensures that the entity uploading the permission document is the owner of the recipient URI.

Return routability tests do not provide the same level of security as SIP identity, but they provide a good-enough security level in architectures where the SIP identity mechanism is not available. The relay generates an unguessable URI (e.g., with a long and random-looking user part) and places it in the CONSENT request. The recipient needs to upload the permission document to that URI.

Using unguessable URIs ensures that the entities that have handled the CONSENT request are the only ones that know the URI. If the CONSENT request is sent to a SIPS URI, the only entities able to upload a forged permission document are the proxies that may handle the CONSENT request between the relay and the recipient.

9.2 Amplification Prevention

Having relays contact directly recipients to obtain documents creates a potential amplification attack. A user adds a large number of URIs to a relay's translation rules and has the relay request permissions for all of them. In this case, the relay would generate a large number of CONSENT requests and send them to the URIs provided by the user. These URIs are the victims of the attack.

To prevent this attack, a user adding URIs to a relays translation rules is requested to generate an amount of bandwidth that is

comparable with the bandwidth the relay will generate to request permissions for those URIs. The user needs to send a REFER request to the relay for each recipient URI. Each REFER request requests the relay to generate a CONSENT request towards one of the recipient URIs. Figure 8 illustrates this mechanism. Note that the sender of the REFER request uses the norefersub extension, which supreses the implicit subscription that is associated with REFER tranl

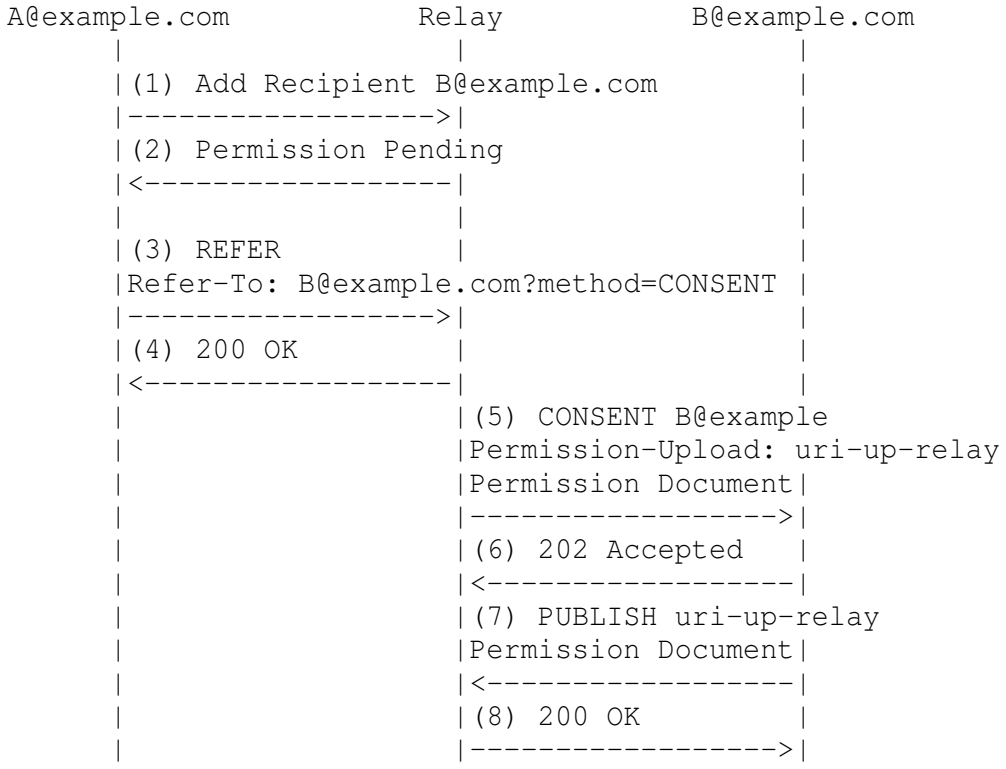


Figure 8: Amplification Attack Prevention

Generally, the mechanism to add new recipient URIs provides the user adding the new recipients with information on the status of the recipient URIs (i.e., whether or not permissions have been obtained for them). This way, the user knows when all the permissions have been successfully uploaded to the relay by the recipients. One mechanism to provide such information is the wait-permission event package.

10. Attemping Communication

In the scenarios described so far, a user adds recipient URIs to the translation rules of a relay. However, the relay does not perform translations towards those URIs until permissions are obtained. If a

user wants to know which recipient URIs are active at a given point, the user contacts the relay to obtain this information.

URI-list services using request-contained URI lists are a special case because the addition of recipient URIs is performed at the same time as the communication attempt. A user places a set of recipient URIs in a request and sends it to a relay so that the relay sends a similar request to all those recipient URIs. If the relay cannot send the request to a URI because it does not have permission to do so, the user needs to be informed.

The relay can inform the user with a 470 (Consent Needed) response. Such a response contains the URIs for which there is not permission and a URI where the user can subscribe to get information about the status of the permissions for those URIs. On receiving such a response, the user sends a REFER for each URI for which there is no permission. Figure 9 illustrates the use of 470 (Consent Needed) responses.

Figure 9: Communication attempt

11. Registrations

Registrations are a special type of translations. The user registering has a trust relationship with the registrar in its home domain. This is not the case when a user gives any type of permissions to a relay in a different domain.

Traditionally, REGISTER transactions have performed two operations at the same time: setting up a translation and authorizing the use of that translation. For example, a user registering its current contact URI is giving permission to the registrar to forward traffic sent to the user's AoR (Address of Records) to the registered contact URI. This works fine when the entity registering is the same as the one that will be receiving traffic at a later point (e.g., over the same connection as the registration). However, this schema creates some potential attacks which relate to third-party registrations.

An attacker binds, via a registration, his or her AoR with the contact URI of a victim. Now, the victim will receive unsolicited traffic that was originally addressed to the attacker.

The process of authorizing registration is shown in Figure 10.

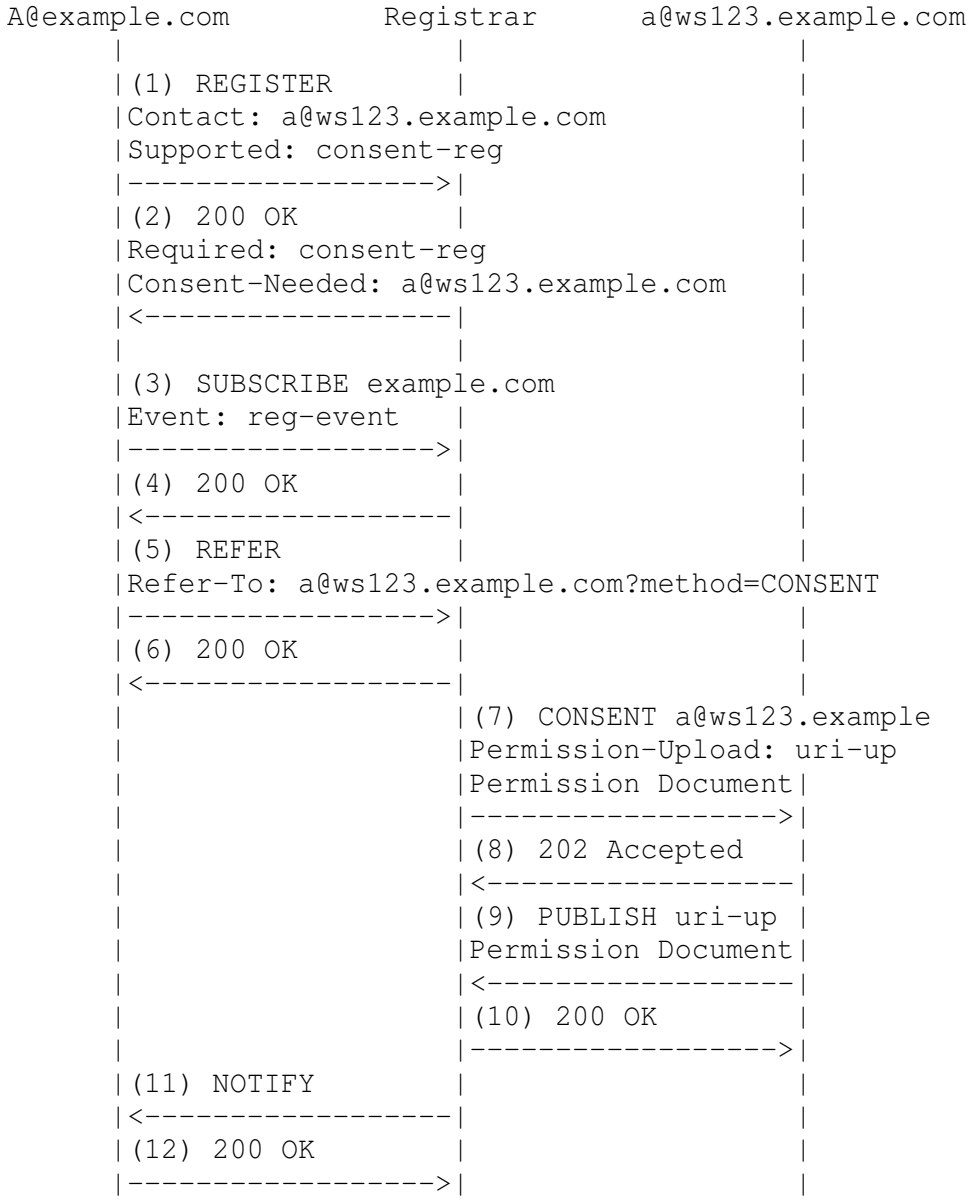


Figure 10: Registration

The permission document uploaded to the registrar in (9) is shown in Figure 11. Note that this permission document is very general. That is, it authorizes the registrar to forward any request from any sender. This is the type of granularity that this framework intends to provide for registrations. Users who want to define how incoming requests are treated with a finer granularity should use other mechanisms such as CPL.

```
    <target>
      <identity>
             <id>A@example.com</id>
      </identity>
    </target>
    <sender>
      <identity>
        <any-identity/>
      </identity>
    </sender>
    <recipient>
      <identity>
        <id>a@ws123.example.com</id>
      </identity>
    </recipient>
    <operations>
        <any-method/>
    </operations>
```

Figure 11: Permission document uploaded to the registrar

12. Permission Revocation

A user that wants to revoke a permission needs to wait until it receives a new request using that permission. Such request which will contain a Permission-Used header field. The Permission-Used header field contains a URI where the permission document used for the translation can be downloaded and a URI where the user can upload a new permission document (e.g., a permission document that does not allow a particular translation any longer).

When permission document authorization is based on a return routability test, requests with Permission-Used header fields need to be sent to a SIPS URI.

OPEN ISSUE: do we want to force all the traffic from the translation to be sent using TLS so that every request carries a Permission-USer header field or do we want to come up with a mechanism whereby the client can request the relay to send it a TLS-protected request with the URI to upload the new permission document? In the latter case, regular traffic from the relay to the user needs not be TLS-protected.

OPEN ISSUE: we may want to define a validity element so that permission documents are not valid for ever.

13. IANA Considerations

TBD.

14. Security Considerations

TBD.

Editor's note: we have to avoid that attackers provide permissions for translations that apply to other users (e.g., allow everyone to send traffic to a victim) and that attackers provide permissions for a translation that apply to them but routes to a victim (e.g., 3rd party registration that binds attacker@relay to victim@somewhere). For the former we need authentication (e.g., SIP identity) and for the latter we relay on the routing infrastructure to route CONSENTs to the same place the traffic will be sent to once permissions are obtained (i.e., a return routability test).

15. References

15.1 Normative References

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

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

15.2 Informative References

[3] Schulzrinne, H., "A Document Format for Expressing Privacy Preferences", draft-ietf-geopriv-common-policy-04 (work in progress), February 2005.

[4] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[5] Rosenberg, J., "Requirements for Consent-Based Communications in the Session Initiation Protocol (SIP)", draft-ietf-sipping-consent-reqs-00 (work in progress), October 2004.

[6] Rosenberg, J., "Presence Authorization Rules", draft-ietf-simple-presence-rules-02 (work in progress), February 2005.

[7] Camarillo, G. and A. Roach, "Requirements and Framework for Session Initiation Protocol (SIP)Uniform Resource Identifier (URI)-List Services", draft-ietf-sipping-uri-services-03 (work in progress), April 2005.

Authors' Addresses

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

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

Gonzalo Camarillo (editor)
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com

Dean Willis
Cisco Systems
2200 E. Pres. George Bush Turnpike
Richardson, TX 75082
USA

Email: dean.willis@softarmor.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING
Internet-Draft
Expires: January 19, 2006

J. Rosenberg
Cisco Systems
G. Camarillo, Ed.
Ericsson
D. Willis
Cisco Systems

July 18, 2005

Requirements for Consent-Based Communications in the Session Initiation Protocol (SIP)

draft-ietf-sipping-consent-reqs-01.txt

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) supports communications across many media types, including real-time audio, video, text, instant messaging, and presence. In its current form, it allows session invitations, instant messages, and other requests to be delivered

from one party to another without requiring explicit consent of the recipient. Without such consent, it is possible for SIP to be used for malicious purposes, including spam and denial-of-service attacks. This document identifies a set of requirements for extensions to SIP that add consent-based communications.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] supports communications across many media types, including real-time audio, video, text, instant messaging, and presence. This communication is established by the transmission of various SIP requests (such as INVITE and MESSAGE [4]) from an initiator to the recipient, with whom communication is desired. Although a recipient of such a SIP request can reject the request, and therefore decline the session, a SIP network will deliver a SIP request to the recipient without their explicit consent.

Receipt of these requests without explicit consent can cause a number of problems in SIP networks. These include spam and DoS (Denial of Service) attacks. These problems have plagued email. Fortunately, most SIP networks, at time of writing, were not interconnected with each other, and so the incidences of such problems have been lower. However, once such broad interconnection occurs, these problems will arise. Therefore, it is important to address them proactively, before it is too late.

This document elaborates on the problems posed by the current open model in which SIP was designed, and then goes on to define a set of requirements for adding a consent framework to SIP.

2. Problem Statement

In SIP networks designed according to the principles of RFC 3261 [1] and RFC 3263 [2], anyone on the Internet can create and send a SIP request to any other SIP user, by identifying that user with a SIP URI. The SIP network will usually deliver this request to the user identified by that URI. It is possible, of course, for network services, such as call screening, to block such messaging from occuring, but this is not widespread and certainly not a systematic solution to the problem under consideration here.

Once the SIP request is received by the recipient, the user agent typically takes some kind of automated action to alert the user about receipt of the message. For INVITE requests, this usually involves "ringing the phone", or creating a screen pop. These indicators frequently convey the subject of the call and the identity of the caller. Due to the real-time nature of the session, these alerts are typically disruptive in nature, so as to get the attention of the user.

For MESSAGE requests, the content of the message is usually rendered to the user.

SUBSCRIBE [3] requests do not normally get delivered to the user agents residing on a user's devices. Rather, they are normally processed by network-based state agents. The watcher information event package allows a user to find out that such requests were generated for them, affording the user the opportunity to approve or deny the request. As a result, SUBSCRIBE processing, and most notably presence, already has a consent-based operation. Nevertheless, this already-existing consent mechanism for SIP subscriptions does not protect network agents against DoS attacks.

There are two principal problems that arise when MESSAGE and INVITE requests can be delivered to user agents directly, without their consent. The first is spam. For INVITE requests, this takes the form of typical "telemarketer" calls. A user might receive a stream of never-ending requests for communications, each of them disrupting the user and demanding their attention. For MESSAGE requests, the problem is even more severe. The user might receive a never-ending stream of screen pops that deliver unwanted, malicious, or otherwise undesired content.

The second problem is DoS attacks. SIP proxies provide a convenient relay point for targeting a message to a particular user or IP address, and in particular, relaying to a recipient which is often not directly reachable without usage of the proxy. Worse, some proxies or back to back user agents generate multiple outgoing requests upon receipt of an incoming request. This occurs in forking proxies, and in URI-list services. Examples of URI-list services are subscriptions to resource lists, dial out conference servers, and MESSAGE URI-list services. These SIP elements can be used as an amplifier, allowing the transmission of a single SIP request to flood packets to a single recipient or network. For example, a user can create a buddy list with 100 entries, each of which is a URI of the form "sip:identifier@target-IP", where target-IP is the IP address to which the attack is to be directed. Sending a single SIP SUBSCRIBE request to such a list will cause the resource list server to generate 100 SUBSCRIBE requests, each to the IP address of the target, which does not even need to be a SIP node.

> Note that the target-IP does not need to be the same in all the URIs in order to attack a single machine. For example, the target-IP addresses may all belong to the same subnetwork, in which case the target of the attack would be the access router of the subnetwork.

Though the spam and DoS problems are not quite the same, both can be alleviated by adding a consent-based communications framework to SIP. Such a framework keeps servers from relaying messages to users without their consent.

The framework for SIP URI-list services [5] identifies these two problems (spam and DoS attacks) in the context of URI-list services. That framework mandates the use of opt-in lists, which are a form of consent-based communications. The reader can find an analysis on how a consent-based framework help alleviating spam-related problems in [6].

3. Requirements

The following identify requirements for a solution that provides consent-based communications in SIP.

REQ 1: The solution must keep relays from delivering a SIP message to a recipient unless the recipient has explicitly granted permission for receipt of that type of message.

REQ 2: The solution shall prevent SIP servers from generating more than one outbound request in response to an inbound request, unless permission to do so has been granted by the resource to whom the outbound request was to be targeted.

REQ 3: The permissions shall be capable of specifying that messages from a specific user, identified by a SIP AoR, are permitted.

REQ 4: It shall be possible for a user with a particular AoR to specify permissions separately for each resource that wishes to relay requests to that AOR.

REQ 5: The permissions shall be capable of specifying that only certain types of messages, such as INVITE or MESSAGE request, are permitted from a user.

REQ 6: It shall be possible for a user to revoke permissions at any time.

REQ 7: It shall be possible for the users to specify that permissions are time limited, and must be refreshed after expiration.

REQ 8: It shall not be required for a user or user agent to store information in order to be able to revoke permissions that were previously granted for a relay resource.

REQ 9: The solution shall work in an inter-domain context, without requiring pre-established relationships between domains.

REQ 10: The solution shall work for all current and future SIP methods.

REQ 11: The solution shall be applicable to forking proxies.

REQ 12: The solution shall be applicable to URI-list services, such as resource list servers, MESSAGE URI-list services, and conference servers performing dial-out functions.

REQ 13: The solution shall be applicable to both stored and request-contained URI-list services.

REQ 14: The solution shall allow anonymous communications, as long as the recipient is willing to accept anonymous communications.

REQ 15: If the recipient of requests wishes to be anonymous, it shall be possible for them to grant permissions without a sender knowing their identity.

REQ 16: The solution shall prevent against attacks that seek to undermine the underlying goal of consent. That is, it should not be possible to "fool" the system into delivering a request for which permission was not, in fact, granted.

REQ 17: The solution shall not require the recipient of the communications to be connected to the network at the time communications is attempted.

REQ 18: The solution shall note require the sender of a communications to be connected at the time that a recipient provides permission.

REQ 19: The solution should not, in and of itself, create substantial additional messaging. Doing so defeats some of the purpose of the solution.

REQ 20: The solution should scale to Internet-wide deployment.

4. Security Considerations

Security has been discussed throughout this specification.

5. References

5.1 Normative References

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

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

5.2 Informational References

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

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

[5] Camarillo, G. and A. Roach, "Requirements and Framework for Session Initiation Protocol (SIP)Uniform Resource Identifier (URI)-List Services", draft-ietf-sipping-uri-services-03 (work in progress), April 2005.

[6] Rosenberg, J. and C. Jennings, "The Session Initiation Protocol (SIP) and Spam", draft-rosenberg-sipping-spam-01 (work in progress), October 2004.

Authors' Addresses

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

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

Gonzalo Camarillo (editor)
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com

Dean Willis
Cisco Systems
2200 E. Pres. George Bush Turnpike
Richardson, TX 75082
USA

Email: dean.willis@softarmor.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING Working Group
Internet-Draft
Expires: January 17, 2006

V. Hilt
Bell Labs/Lucent Technologies
G. Camarillo
Ericsson
J. Rosenberg
Cisco Systems
July 16, 2005

Session Initiation Protocol (SIP) Session Policies - Document Format and Session-Independent Delivery Mechanism
draft-ietf-sipping-session-indep-policy-03

Status of this Memo

Copyright Notice

Abstract

This draft defines a document format for media-related SIP session policies. The format extends the Profile Data Set Schema by specifying a data set for media properties. This draft also defines a delivery mechanism for session policies that is independent of a

SIP session.

Table of Contents

1. Introduction

Some domains have policies in place, which impact the sessions established using the Session Initiation Protocol (SIP) [15]. These policies are often needed to support the network infrastructure or for the execution of services. For example, wireless networks usually have limited resources for media traffic. A wireless network provider may therefore restrict the bandwidth that is available to a single user. Knowing about the bandwidth limit enables an user agent to make an informed decision about the number of streams, codecs and media types it can use in a session.

In another example, a service provider wants to specifically restrict the set of codecs and media types that can be used in the network. These restrictions may change depending on network conditions. With session policies, the current set of restrictions can be conveyed to user agents to prevent them from inadvertently violating any of the network policies.

In a third example, a network provides quality of service (QoS) for media streams through differentiated services. By knowing that differentiated services are available and knowing the service class assigned to media streams, a user agent can mark the packets of media streams accordingly and therefore benefit from the QoS infrastructure.

Domains sometimes enforce policies they have in place. For example, a domain might have a configuration in which all packets containing a certain audio codec are dropped. Unfortunately, enforcement mechanisms usually do not inform the user about the policies they are enforcing and silently keep the user from doing anything against them. This may lead to the malfunctioning of devices that is incomprehensible to the user. With session policies, the user knows about the restricted codecs and can use a different codec or simply connect to a domain with less stringent policies. Session policies provide an important combination of consent coupled with enforcement. That is, the user becomes aware of the policy and needs to act on it, but the provider still retains the right to enforce the policy.

Session-policies can be set up in two different ways: specifically for a session or independent of a session. Session-specific policies are created for one particular session, usually under consideration of certain aspects of this session (e.g. the IP addresses and ports that are used for media). Since session-specific policies are tailored to a session, they only apply to the session they are created for. These policies require a delivery mechanism that enables the exchange of session policy information at the time a session is established. The framework for session-specific policies

[17] defines such a delivery mechanism for session-specific policies.

Session-independent policies on the other hand are independent of a specific session and generally apply to the sessions set up by a user agent. In principle, these policies could also be delivered to user agents individually for each session, using the session-specific policy framework. However, since these policies apply to many sessions, it is more efficient to deliver them to user agents only when the user agent is initialized or a policy changes. This draft defines a delivery mechanism for session-independent policies.

This draft also defines a document format for media-related session policies. This format is based on XML [16]. It extends the Profile Data Set Schema [13] by specifying a data set for media properties. The format defines a minimal set of media-related properties [18] and is aimed at achieving interoperability between different user agents and profile delivery/policy servers. The format can be extended through the XML extension mechanisms if additional media properties are needed. The XML document format is independent of the delivery mechanism and can be used with session-independent and session-specific session policies.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, [1] and indicate requirement levels for compliant implementations.

3. Session-Independent Policy Mechanism

Session-independent policies can be delivered to UAs using the mechanism defined in the Framework for SIP User Agent Profile Delivery [12]. Session-independent policies can reside on the same server as other configuration information and they can be delivered to UAs in conjunction with this information. Session-independent policies can also reside on a separate policy server, which is independent of a configuration server. A UA may receive session-independent policies from multiple servers.

In this draft, the terms policy server and profile delivery server are used interchangeably. A policy server is a profile delivery server that provides session policies.

3.1 Subscriber Behavior

A UA can express interest in session-independent policies by

subscribing to session policies as described in [12]. If the UA already has the URIs of policy servers (e.g., through provisioning) it may directly use these URIs to subscribe to session-independent policies.

Session-independent policies are frequently provided to a UA by the following two network domains: the domain a user registers at (i.e., the domain in the address-of-record (AoR)) and the domain the UA is physically connected to (i.e. the local network domain). A policy server in the AoR-domain may, for example, provide policies needed for services the user has subscribed to. The domain that provides the physical network connection may have policies needed to ensure the operativeness of the network, e.g., by limiting the bandwidth available to a UA. A UA SHOULD attempt to subscribe to the policy servers in both domains. These subscriptions are established using the "user" (for subscriptions to the AoR-domains) and the "localnetwork" (for subscriptions to the network domain) profile-types [12].

A UA SHOULD create a SUBSCRIBE request in the following events:

- The UA registers a AoR for the first time or removes a AoR from the set of AoRs it has registered. This occurs, for example, when a UA starts up (and registers AoRs) and when it shuts down (and deregisters AoRs). This event also occurs when a new AoR is added to a UA or a AoR is removed. In these cases, the UA SHOULD establish subscriptions for each new AoR using the "user" and the "localnetwork" profile-types. It SHOULD terminate all subscriptions for the AoRs that have been removed.
- The UA changes the domain it is connected to. The UA SHOULD create a new subscription for each AoR using the "localnetwork" profile-type. It SHOULD terminate all existing subscriptions for the "localnetwork" profile-type. It does not need to change the subscriptions for "user" profiles.

If a subscriber is unable to establish a subscription, it SHOULD NOT attempt to re-try this subscription, unless one of the above events occurs again. This is to limit the number of SUBSCRIBE requests sent within domains that do not support session-policies.

A subscriber compliant to this specification SHOULD indicate its support for session-independent session policies by adding the MIME types of supported session policy formats to the Accept header of the SUBSCRIBE request. This specification defines the new MIME type "application/session-policy+xml", which MUST be supported by UAs compliant to this specification. UAs MAY also indicate support for MIME type extensions (e.g. an additional XML namespace) using [3].

4. Basic Media Policy Format

The Basic Media Policy Format (BMPF) is a document format for media-related policies. It extends the Profile Data Set Schema by providing a media data set and is used to define media-related SIP session policies.

A BMPF document is an XML [16] document that MUST be well-formed and MUST be valid according to schemas, including extension schemas, available to the validator and applicable to the XML document. BMPF documents MUST be based on XML 1.0 and MUST be encoded using UTF-8.

A user agent may receive multiple BMPF documents from different sources. These documents need to merged into a single document the user agent can work with. General rules for merging BMPF documents are described in [13]. Specific merging rules for each of the BMPF elements are described below.

4.1 Namespace

This specification makes use of XML namespaces [4]. The namespace URIs for schemas defined in this specification are URNs [7], using the namespace identifier 'ietf' defined by [8] and extended by [5]. The namespace URN for the BMPF schema is:

urn:ietf:params:xml:ns:mediadataset

The MIME type for the Basic Media Policy Format is:

application/session-policy+xml

ISSUE: a separate MIME type might not be needed for BMPF. The MIME type of the Profile Data Set Schema may be sufficient. We still need a separate namespace.

4.2 Extensibility

The BMPF format is an extension of the Profile Data Set Schema [13]. Elements from the BMPF namespace can be used in conjunction with elements from other Profile Data Sets.

The BMPF format itself can also be extended using XML extension mechanisms. In particular, elements from different XML namespaces MAY be present within a BMPF document for the purposes of extensibility; elements or attributes from unknown namespaces MUST be ignored.

4.3 Attributes

The following attributes provide common functionalities, which are generally useful for media properties:

- Per-stream properties: 'stream-label' attribute
- Media-type specific properties: 'media-type' attribute

These attributes are defined in addition to the attributes inherited from the Profile Data Set Schema [13]:

- Property Access Control: 'visibility' attribute
- Policies: 'policy' and 'excluded-policy' attribute
- Unidirectional Properties: 'direction' attribute
- Preferences: 'q' attribute

The use of these attributes is defined individually for each element in the XML format below.

4.3.1 The 'stream-label' Attribute

Some properties only apply to a specific media stream. The stream to which a property applies to must be identifiable through a label [6]. Per-stream properties can be expressed by adding a 'stream-label' attribute to the respective element. Such a property only applies to the identified stream. If there is no stream with this label, the element must be ignored.

Per-stream properties require that the labels of media streams are known to the creator of a document (i.e. the profile delivery/policy server). These labels are, for example, part of the session description. Per-stream properties are therefore typically used for session-specific policies.

4.3.2 The 'media-type' Attribute

Some properties only apply to streams of a certain media type. For example, a property may only apply to audio streams. Media-type specific properties can be defined by adding a 'media-type' attribute to the respective element. Such a property only applies to media streams of that type.

The value of the 'media-type' attribute MUST be the name of a IANA registered media type (see [2]), such as 'audio', 'video', 'text', or 'application'.

4.4 Elements

The following elements are defined for the BMPF format.

4.4.1 The <session-policy> Element

The <session-policy> element is a container for media policy elements. It MAY occur multiple times inside a <property_set> [13] element.

The <session-policy> element MAY contain one optional <context> element and multiple (including zero) <media-types>, <codecs>, <media-intermediary>, <qos-dscp>, and <max-bandwidth> elements as well as elements from other namespaces.

> OPEN ISSUE: the <session-policy> seems to have pretty much the same functionality as the <property_set> element. Maybe it needs to be removed and the context element needs to go into the Profile Data Set Schema.

4.4.2 The <context> Element

The <context> element provides context information about this policy.

The <context> element is optional in a <session-policy> element. It MAY contain a <dialog-ID>, <domain>, multiple <contact> and an <info> element.

> Merging rule: the <context> element is not subject to merging. Information in the context element may be used to assist the user if a conflict occurs during the merging process.
> Policies that affect different sessions (i.e. have different <dialog-ID> values) are not merged.

4.4.3 The <dialog-ID> Element

Session-specific policies only apply to one particular session. The <dialog-ID> element is used to identify this session. If this element is present the <context> element of a <session-policy> container, all properties defined in this container only apply to the identified session. A single document may contain multiple <session-policy> containers, which each contains a different <dialog-ID> element. This way, session-specific policies for different sessions can be contained in one document. If the user agent does not have a session with this dialog-ID, the content of the respective <session-policy> container MUST be ignored.

The <dialog-ID> element is optional in a <context> element. It MUST

contain a <call-ID> and a <local-tag> and MAY contain a <remote-tag> element.

The <call-ID> element contains the call-ID (as defined in [15]) of the session the policies are for.

The <local-tag> element contains the local tag (as defined in [15]) of the session the policies are for.

The <remote-tag> element contains the remote tag (as defined in [15]) of the session the policies are for. If the remote tag element is omitted, the policies apply to all sessions that have the given call-ID and local tag.

Local and remote tags are defined from the viewpoint of the recipient of the document.

4.4.4 The <domain> Element

The <domain> element contains a URI that identifies the domain which has issued this policy.

The <domain> element is optional and MAY occur only once inside a <context> element.

4.4.5 The <contact> Element

The <contact> element contains a contact address (e.g. a SIP URI or email address) under which the issuer of this policy can be reached.

The <contact> element is optional and MAY occur multiple times inside a <context> element.

4.4.6 The <info> Element

The <info> element provides a short textual description of the policy that should be intelligible to the human user.

The <info> element is optional and MAY occur only once inside a <context> element.

4.4.7 The <media-types> Element

The <media-types> element expresses a policy for the use of media types (e.g. audio, video). It defines the media types that must be used, may be used, and must not be used in a session.

This element may have the following attributes (see Section 4.3):

visibility, excluded-policy, direction. The 'excluded-policy' attribute specifies the default policy for all media types that are not listed inside this element.

The <media-types> element is optional in a <session-policy> element and MAY occur multiple times. Multiple <media-types> elements MAY only be present if each element applies to a different set of streams (e.g. one <media-types> for incoming and one for outgoing streams). The <media-types> MUST contain one or more <media-type> elements.

Merging rule: <media-types> containers are merged using the "Multiple Enumerated Value Merging Algorithm" defined in [13].

4.4.8 The <media-type> Element

The <media-type> element defines a policy for the use of the media type identified by this element. The value of this element MUST be the name of a IANA registered media type (see [2]), such as 'audio', 'video', 'text', or 'application'.

This element may have the following attributes (see Section 4.3): policy, q. Media types that have the policy 'mandatory' MUST be used in a session, media types with the policy 'allowed' MAY be used and media types with the policy 'disallowed' MUST NOT be used.

The <media-type> element is mandatory and MAY occur multiple times inside a <media-types> element.

4.4.9 The <codecs> Element

The <codecs> element expresses a policy for the use of codecs. A policy can define that a codec must be used, may be used, or must not be used in a session. A policy MUST allow the use of at least one codec and MUST NOT define more than one mandatory codec for a media type.

This element may have the following attributes (see Section 4.3): visibility, excluded-policy, direction, stream-label. The 'excluded-policy' attribute specifies the default policy for all codecs that are not listed inside this element.

The <codecs> element is optional in a <session-policy> element and MAY occur multiple times. Multiple <codecs> elements MAY only be present if each element applies to a different set of streams (e.g. one <codecs> for incoming and one for outgoing streams). The <codecs> element MUST contain one or more <codec> elements.

Merging rule: <codecs> containers are merged using the "Multiple Enumerated Value Merging Algorithm" defined in [13].

4.4.10 The <codec> Element

The <codec> element defines a policy for the use of the codec identified by this element. The value of this element MUST be the name of a registered MIME type for a encoding (see [2]), such as "PCMA", "G729", or "H263".

This element may have the following attributes (see Section 4.3): policy, q. Codecs that have the policy 'mandatory' MUST be used in a session, codecs with the policy 'allowed' MAY be used and codecs with the policy 'disallowed' MUST NOT be used.

The <codec> element is mandatory and MAY occur multiple times inside a <codecs> element.

4.4.11 The <media-intermediary> Element

The <media-intermediary> element expresses a policy for routing a media stream through a media intermediary. The purpose of the <media-intermediary> element is to tell the UA to send the media for a particular stream through an IP address and port on an intermediary. Instead of merely sending the media there, the UA can instead specify a source route, which touches that intermediary, but also any other intermediaries and then the final recipient. Thus, if there are N hops, including the final recipient, there needs to be a way for the media stream to specify N destinations. The way these N destinations should be identified when sending the media stream is expressed using the <int-lroute> element.

This element may have the following attributes (see Section 4.3): visibility, policy, direction, stream-label.

The <media-intermediary> element is optional in a <session-policy> element and MAY occur multiple times. The order of <media-intermediary> element instances is significant. It defines the order in which the media intermediaries must be traversed. The UA sends the media stream to the intermediary listed first, then to the intermediary listed next and so on. The <media-intermediary> element MUST contain one <int-uri> and one <int-lroute> element.

Merging rule: the intermediaries defined in all policies are traversed. In general, local intermediaries should be traversed before remote intermediaries. During the merging process, <media-intermediary> element values from different servers are ordered using the "Closest Value First Merging Algorithm" [13]. The

intermediaries should be traversed in this order.

4.4.12 The <int-uri> Element

The <int-uri> element contains a URI that identifies the IP address and port number of a media intermediary. The UA uses this URI to send its media streams to the intermediary. If a protocol uses multiple subsequent ports (e.g. RTP), the lowest port number SHOULD be included in the URI. All additional port numbers SHOULD be identified in <int-addl-port> elements.

The <int-uri> element occurs exactly once inside a <media-intermediary> element.

4.4.13 The <int-addl-port> Element

If a protocol uses multiple subsequent ports (e.g. RTP), the lowest port number SHOULD be included in the <int-uri> element. All additional port numbers SHOULD be identified in <int-addl-port> elements.

The <int-addl-port> element is optional and MAY occur multiple times inside a <media-intermediary> element.

4.4.14 The <int-lroute> Element

The <int-lroute> element identifies the loose source routing protocol to be used with this intermediary. The value of this element can be one of the following:

- ip-in-ip: IP-in-IP tunneling is used to specify the hops of media traversal. The ultimate destination is specified in the destination IP address of the innermost packet. Each subsequent hop results in another encapsulation, with the destination of that hop in the destination IP address of the packet.
- ip-loose: IP provides a loose routing mechanism that allows the sender of an IP datagram to specify a set of IP addresses that are to be visited on the way before reaching the final destination.
- turn: TURN provides a mechanism for inserting a media relay into the path. Although the main purpose of TURN is NAT traversal, it is possible for a TURN relay to perform other media intermediary functionalities. The user agent establishes a binding on the TURN server and uses this binding to transmit and receive media.
- media-specific: media protocols can provide their own loose routing mechanism. If that is the case, the loose routing mechanism of that protocol is used. As an example, SIP provides its own loose routing mechanisms with the Route header. It can be used to direct an instant message using the SIP MESSAGE method

through a set of intermediaries.

- o none: if there is no loose-routing mechanism available, the media is just sent to the first media intermediary listed in the header. Note that this requires the intermediary to know where to forward the packets to. Such a route must be set up in the intermediary through other means. For example, with session-specific policies, the policy server can extract the destination address from the session description.

The <int-lroute> element occurs exactly once inside a <media-intermediary> element.

4.4.15 The <max-bandwidth> Element

The <max-bandwidth> element contains the maximum bandwidth in kilobits per second an entity can use for its media streams.

This element may have the following attributes (see Section 4.3): visibility, policy, direction, media-type.

The <max-bandwidth> element is optional and MAY occur multiple times inside a <session-policy> element. If it occurs multiple times, each instance MUST apply to different media streams (i.e. one <max-bandwidth> element for outgoing and one for incoming streams).

Merging rule: the lowest max-bandwidth value is used.

4.4.16 The <qos-dscp> Element

The <qos-dscp> element contains an Differentiated Services Codepoint (DSCP) [10] that should be used to populate the IP DS field of media packets. The <qos-dscp> contains an integer value that represents a 6 bit field and therefore ranges from 0 to 63.

This element may have the following attributes (see Section 4.3): visibility, policy, direction, stream-label, media-type.

The <qos-dscp> element is optional and MAY occur multiple times inside a <session-policy> element. If it occurs multiple times, each instance MUST apply to a different media stream (i.e. one <qos-dscp> element for audio and one for video streams).

Merging rule: the domain that is first traversed by the media stream has precedence and its DSCP value is used. During the merging process, <qos-dscp> element values from different servers are ordered using the "Closest Value First Merging Algorithm" [13]. The DSCP value from the closest server is used.

4.4.17 Other Elements

A number of additional elements have been proposed for a policy language. These elements are deemed to be outside the scope of a basic media policy format. However, they may be defined in extensions of BMPF or other profile data sets.

- maximum number of streams
- maximum number of sessions
- maximum number of streams per session
- maximum bandwidth per session
- maximum bandwidth per stream
- external address and port
- media transport protocol
- outbound proxy
- SIP methods
- SIP option tags
- SIP transport protocol
- body disposition
- body format
- body encryption

4.5 Example

The following example describes a policy that requires the use of audio, allows the use of video and prohibits the use of other media types. It allows the use of any codec except G.723 and G.729. The policy also inserts a media intermediary into outgoing media streams.

```
   <property-set>
     <session-policy>
       <context>
         <domain>example.com</domain>
         <contact>sip:policy_manager@example.com</contact>
         <info>Access network policies</info>
       </context>
       <media-types excluded-policy="disallow">
         <media-type policy="mandatory">audio</media-type>
         <media-type policy="allow">video</media-type>
       </media-types>
       <codecs excluded-policy="allow">
         <codec policy="disallow">G729</codec>
         <codec policy="disallow">G723</codec>
       </codecs>
       <media-intermediary direction="sendonly" policy="mandatory">
         <int-uri>192.0.2.0:6000</int-uri>
         <int-addl-port>6001</int-addl-port>
         <int-lroute>ip-in-ip</int-lroute>
       </media-intermediary>
     </session-policy>
   </property-set>
```

4.6 Schema Definition

```
   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema targetNamespace="urn:ietf:params:xml:ns:mediadataset"
         xmlns:tns="urn:ietf:params:xml:ns:mediadataset"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:spds="http://sipfoundry.org/schema/profile-data-sets-00">


     <xs:attributeGroup name="single_stream_attributes" >
       <xs:attribute name="stream-label"
                     type="xs:string" use="optional"/>
     </xs:attributeGroup>

     <xs:attributeGroup name="media_type_attributes" >
       <xs:attribute name="media-type"
                     type="xs:string" use="optional"/>
     </xs:attributeGroup>


     <xs:element name="session-policy">
       <xs:complexType>
         <xs:sequence>
```

```
          <xs:element ref="tns:context"
            minOccurs="0" maxOccurs="1"/>
          <xs:element ref="tns:media-types"
            minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="tns:codecs"
            minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="tns:media-intermediary"
            minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="tns:max-bandwidth"
            minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="tns:qos-dscp"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>


    <xs:element name="context">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:dialog-ID"
            minOccurs="0" maxOccurs="1"/>
          <xs:element name="domain" type="xs:anyURI" minOccurs="0"
            maxOccurs="1"/>
          <xs:element name="contact" type="xs:anyURI" minOccurs="0"
            maxOccurs="unbounded"/>
          <xs:element name="info" type="xs:string"
             minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="media-types"
                substitutionGroup="spds:setting_container">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:media-type"
            minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attributeGroup ref="spds:directional_setting_attributes" />
      </xs:complexType>
    </xs:element>

    <xs:element name="codecs"
                substitutionGroup="spds:setting_container">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:codec"
```

```
            minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attributeGroup ref="spds:directional_setting_attributes" />
        <xs:attributeGroup ref="tns:single_stream_attributes" />
      </xs:complexType>
    </xs:element>

    <xs:element name="media-intermediary"
                substitutionGroup="spds:setting">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="int-uri" type="xs:anyURI"
            minOccurs="1" maxOccurs="1"/>
          <xs:element name="int-addl-port"
            type="xs:positiveInteger"
            minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="int-lroute" type="tns:int-lroute"
            minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
        <xs:attributeGroup ref="spds:directional_setting_attributes" />
        <xs:attributeGroup ref="tns:single_stream_attributes" />
      </xs:complexType>
    </xs:element>

    <xs:element name="max-bandwidth"
                substitutionGroup="spds:setting">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:positiveInteger" />
        </xs:simpleContent>
        <xs:attributeGroup ref="spds:directional_setting_attributes" />
        <xs:attributeGroup ref="tns:media_type_attributes" />
      </xs:complexType>
    </xs:element>

    <xs:element name="qos-dscp"
                substitutionGroup="spds:setting">
      <xs:complexType>
        <xs:simpleContent>
          <xs:restriction base="xs:integer" >
            <xs:minInclusive value="0" />
            <xs:maxInclusive value="63" />
          </xs:restriction>
        </xs:simpleContent>
        <xs:attributeGroup ref="spds:directional_setting_attributes" />
        <xs:attributeGroup ref="tns:single_stream_attributes" />
        <xs:attributeGroup ref="tns:media_type_attributes" />
      </xs:complexType>
```

```
    </xs:element>


    <xs:element name="dialog-ID">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="call-ID" type="xs:string"
            minOccurs="1" maxOccurs="1"/>
          <xs:element name="local-tag" type="xs:string"
            minOccurs="1" maxOccurs="1"/>
          <xs:element name="remote-tag" type="xs:string"
            minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="media-type"
                substitutionGroup="spds:setting">
      <xs:complexType>
        <xs:simpleContent>
          <xs:restriction base="xs:string" />
        </xs:simpleContent>
        <xs:attributeGroup ref="spds:multi_setting_attributes" />
      </xs:complexType>
    </xs:element>

    <xs:element name="codec"
                substitutionGroup="spds:setting">
      <xs:complexType>
        <xs:simpleContent>
          <xs:restriction base="xs:string" />
        </xs:simpleContent>
        <xs:attributeGroup ref="spds:multi_setting_attributes" />
      </xs:complexType>
    </xs:element>

    <xs:simpleType name="int-lroute">
      <xs:restriction base="xs:string">
        <xs:enumeration value="ip-in-ip"/>
        <xs:enumeration value="ip-loose"/>
        <xs:enumeration value="turn"/>
        <xs:enumeration value="media-specific"/>
        <xs:enumeration value="none"/>
      </xs:restriction>
    </xs:simpleType>

  </xs:schema>
```

5. Security Considerations

Session policy information can be sensitive information. The protocol used to distribute it SHOULD ensure privacy, message integrity and authentication. Furthermore, the protocol SHOULD provide access controls which restrict who can see who else's session policy information.

6. IANA Considerations

This document registers a new MIME type, application/ session-policy+xml, and registers a new XML namespace.

6.1 MIME Registration for application/session-policy+xml

MIME media type name: application

MIME subtype name: session-policy+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [9].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [9].

Security considerations: See Section 10 of RFC 3023 [9] and Section 5 of this specification.

Interoperability considerations: none.

Published specification: This document.

Applications which use this media type: This document type has been used to download the session policy of a domain to SIP user agents.

Additional Information:

Magic Number: None

File Extension: .wif or .xml

Macintosh file type code: "TEXT"

Personal and email address for further information: Volker Hilt, <volkerh@bell-labs.com>

Intended usage: COMMON

Author/Change controller: The IETF.

6.2 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:mediadataset

This section registers a new XML namespace, as per the guidelines in [5]

URI: The URI for this namespace is urn:ietf:params:xml:ns:mediadataset.

Registrant Contact: IETF, SIPPING working group, <sipping@ietf.org>, Volker Hilt, <volkerh@bell-labs.com>

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
        content="text/html;charset=iso-8859-1"/>
  <title>Session Policy Namespace</title>
</head>
<body>
  <h1>Namespace for Session Policy Information</h1>
  <h2>urn:ietf:params:xml:ns:mediadataset</h2>
  <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
</html>
END
```

7. References

7.1 Normative References

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

[2] Handley, M., "SDP: Session Description Protocol", draft-ietf-mmusic-sdp-new-24 (work in progress), February 2005.

[3] Hilt, V., Rosenberg, J., and G. Camarillo, "Media Type Extension Negotiation in the Session Initiation Protocol (SIP) Accept Header Field", draft-hilt-sip-ext-neg-00 (work in progress), January 2005.

[4] Hollander, D., Bray, T., and A. Layman, "Namespaces in XML", W3C REC REC-xml-names-19990114, January 1999.

[5] Mealling, M., "The IETF XML Registry", draft-mealling-iana-xmlns-registry-05 (work in progress), June 2003.

[6] Levin, O. and G. Camarillo, "The SDP (Session Description Protocol) Label Attribute", draft-ietf-mmusic-sdp-media-label-01 (work in progress), January 2005.

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

[8] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

[9] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[10] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[11] Perkins, C., "IP Encapsulation within IP", RFC 2003, October 1996.

[12] Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-06 (work in progress), February 2005.

[13] Petrie, D., Lawrence, S., Dolly, M., and V. Hilt, "A Schema and Guidelines for Defining Session Initiation Protocol User Agent Profile Data Sets", draft-petrie-sipping-profile-datasets-02 (work in progress), April 2005.

[14] Rosenberg, J., "Traversal Using Relay NAT (TURN)", draft-rosenberg-midcom-turn-07 (work in progress), February 2005.

[15] 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.

[16] Yergeau, F., Paoli, J., Sperberg-McQueen, C., Bray, T., and E. Maler, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C REC REC-xml-20040204, February 2004.

7.2 Informative References

[17] Hilt, V., Camarillo, G., and J. Rosenberg, "A Framework for Session-Specific Session Policies in the Session Initiation Protocol (SIP)", draft-hilt-sipping-session-spec-policy-01 (work in progress), October 2004.

[18] Rosenberg, J., "Requirements for Session Policy for the Session Initiation Protocol (SIP)", draft-ietf-sipping-session-policy-req-02 (work in progress), July 2004.

Authors' Addresses

Volker Hilt
Bell Labs/Lucent Technologies
101 Crawfords Corner Rd
Holmdel, NJ 07733
USA

Email: volkerh@bell-labs.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com

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

Email: jdrosen@cisco.com

Appendix A. Acknowledgements

Many thanks to Allison Mankin, Dan Petrie and Martin Dolly for the

great discussions and suggestions. A big thanks also to everyone who contributed by providing feedback on the mailing list and in IETF meetings.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

Internet Engineering Task Force
Internet Draft
Document: <draft-ietf-sipping-toip-01.txt>
July 18 2005
Expires: January 17 2006
Informational

SIPPING WG

A. van Wijk (editor)
Viataal

Framework of requirements for real-time text conversation using SIP.

Status of this Memo

Copyright Notice

Abstract

This document provides the framework of requirements for real-time character-by-character interactive text conversation over the IP network using the Session Initiation Protocol and the Transport Protocol for Real-Time Applications. It discusses requirements for real-time Text-over-IP telephony as well as interworking between Text-over-IP telephony and existing text telephony on the PSTN and other networks.

Table of Contents

1. Introduction

For many years, text has been in use as a medium for conversational, interactive dialogue between users in a similar way as voice telephony is used. Such interactive text is different from messaging and semi-interactive solutions like Instant Messaging in that it offers an equivalent conversational experience to users that cannot, or do not wish to, use voice. It therefore meets a different set of requirements than other text-based solutions already available on IP networks.
Traditionally, deaf, hard of hearing and speech-impaired people are amongst the most proliferate users of conversational, interactive text, but because of its interactivity, it is becoming popular amongst mainstream user groups as well.
This document describes how existing IETF protocols can be used to implement a Text-over-IP solution (ToIP). This ToIP framework is specifically designed to be compatible with Voice-over-IP environments, as well as meeting the userÆs requirements, including those of deaf, hard of hearing and speech-impaired users as described in RFC3351 [21].
The Session Initiation Protocol (SIP) is the protocol of choice for control of Multimedia IP telephony and Voice-over-IP (VoIP) communications. It offers all the necessary control and signaling required for the ToIP framework.
The Real-Time Transport Protocol (RTP) is the protocol of choice for real-time data transmission, and its use for interactive text payloads is described in RFC4103 [5].
This document defines a framework for ToIP to be used either by itself or as part of integrated services, including Total Conversation.

2. Scope

The primary scope of this document is to define a framework for the implementation of ToIP, either stand-alone or as a part of wider services, including Total Conversation. In general, the scope is:

a. Description of ToIP using SIP and RTP;
b. Requirements of Real-time, interactive text;
c. Requirements for ToIP interworking.

The subsequent sections describe those requirements in detail.

3. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [2] and indicate requirement levels for compliant implementations.

4. Definitions

Audio bridging - a function of a gateway or relay service that enables an audio path through the service between the users involved in the call.

Full duplex - media is sent independently in both directions.

Half duplex - media can only be sent in one direction at a time or, if an attempt to send information in both directions is made, errors can be introduced into the presented media.

Interactive text - a term for real time transmission of text in a character-by-character fashion for use in conversational services, often as a text equivalent to voice based conversational services.

TTY û alternative designation for a text telephone, often used in USA, see textphone. Also called TDD, Telecommunication Device for the Deaf.

Textphone û also ôtext telephoneö. A terminal device that allows end-to-end real-time, interactive text communication. A variety of textphone protocols exists world-wide, both in the PSTN and other networks. A textphone can often be combined with a voice telephone, or include voice communication functions for simultaneous or alternating use of text and voice in a call.

Text bridging - a function of a gateway service that enables the flow of text through the service between the users involved in the call.

Text gateway - a multi functional gateway that is able to transcode between different forms of text transport methods, e.g., between ToIP in IP networks and Baudot text telephony in the PSTN.

Text telephony û analog textphone services

Text Relay Service - a third-party or intermediary that enables communications between deaf, hard of hearing and speech-impaired people, and voice telephone users by translating between voice and text in a call.

Transcoding Services - services of a third-party user agent that transcodes one stream into another. Transcoding can be done by human operators, in automated manner or a combination of both methods. Text Relay Services are examples of a transcoding service between text and audio.

Total Conversation - A multimedia service offering real time conversation in video, text and voice according to interoperable standards. All media flow in real time. Further defined in ITU-T F.703 Multimedia conversational services description.

Video Relay Service - A service that enables communications between deaf and hard of hearing people, and hearing persons with voice telephones by translating between sign language and spoken language in a call.

Acronyms:

2G	Second generation cellular (mobile)
2.5G	Enhanced second generation cellular (mobile)
3G	Third generation cellular (mobile)
CDMA	Code Division Multiple Access
CTM	Cellular Text Telephone Modem
GSM	Global System of Mobile Communication
ISDN	Integrated Services Digital Network
ITU-T	International Telecommunications Union-Telecommunications standardisation Sector
PSTN	Public Switched Telephone Network
SIP	Session Initiation Protocol
TDD	Telecommunication Device for the Deaf
TDMA	Time Division Multiple Access
ToIP	Text over Internet Protocol
UTF-8	Universal Transfer Format-8

5. Framework Description

5.1. Background

The main purpose of this document is to provide a framework description for the implementation of real-time, interactive text based conversational services over IP networks, known as Text-over-IP (ToIP).
This framework uses existing standards that are already commonly used for voice based conversational services on IP networks. In particular, the ToIP framework uses the Session Initiation Protocol (SIP) [3] to set up, control and tear down the connections between users.
Media is transported using the Real-Time Transport Protocol (RTP) in the manner described in RFC4103.
This framework allows for implementation of services that meet the requirement of providing a text-based conversational service, equivalent to voice based telephony. In particular, ToIP offers an IP equivalent of text telephony services as used by deaf, hard of hearing and speech-impaired individuals.
In addition, real-time text conversations can be combined with other conversational services using different media like video or voice.
By using SIP, ToIP allows participants to negotiate all media including real-time text conversation[4, 5]. This is a highly desirable function for all IP telephony users, but essential for deaf, hard of hearing, or speech impaired people who have limited or no use of the audio path of the call.
It is important to understand that real-time text conversations are significantly different from other text-based communications

like email or instant messaging. Real-time text conversations deliver an equivalent mode to voice conversations by providing transmission of text character by character as it is entered, so that the conversation can be followed closely and immediate interaction takes place, thus providing the same mode of interaction as voice telephony does for hearing people. Store-and-forward systems like email or messaging on mobile networks or non-streaming systems like instant messaging are unable to provide that functionality.

5.2. Requirements for ToIP

In order to make ToIP the equivalent of what voice is to hearing people, it needs to offer equivalent features in terms of conversationality as voice telephony provides to hearing people. To achieve that, ToIP MUST:

a. Offer real-time presentation of the conversation;
b. Provide simultaneous transmission in both directions;
c. Provide interoperability with text conversation features in other networks, for instance the PSTN, accepting functional limitations that will occur during interoperation.
d. Not prevent other media, like audio and video, to be used in conjunction with ToIP.

Users might want to use multiple modes of communication during the conversation, either at the same time or by switching between modes, e.g., between text and audio for example. Native ToIP services MUST ensure that the text interface is always available.

When communicating via a gateway to other networks and protocols, the service SHOULD support all the functionality for alternating or simultaneous use of modalities as offered by the destination network.

ToIP will often be used to access a relay service [I], allowing text users to communicate with voice users. With relay services, it is crucial that text characters are sent as soon as possible after they are entered. While buffering MAY be done to improve efficiency, the delays SHOULD be kept as small as possible. In particular, buffering of whole lines of text MUST NOT be used.

5.3. Use of SIP and RTP

ToIP services MUST use the Session Initiation Protocol (SIP) [3] for setting up, controlling and terminating sessions for real-time text conversation with one or more participants and possibly including other media like video or audio.
Thus, participants are allowed to negotiate on a set of compatible media types with session descriptions used in SIP invitations. A ToIP service MUST always support at least one Text media type.

ToIP services MUST use the Real-Time Transport Protocol (RTP) according to the specification of RFC4103 for the transport of text between participants, which implements T.140 on IP networks.

The standardized T.140 real-time text conversation [4], in addition to audio and video communications, will be a valuable service to many, including on non-IP networks. Real-time text can be expressed as a part of the session description in SIP and is a useful subset of Total Conversation.

The ToIP specification describes a framework for using the T.140 text conversation in SIP as a part of the multimedia session establishment in real-time over a SIP network.

If the User Agents of different participants indicate that there is an incompatibility between their capabilities to support certain media types, e.g. one terminal only offering T.140 over IP as described in RFC4103 and the other one only supporting audio, the user might want to invoke a transcoding services.

Examples of possible scenarios for including a relay service in the conversation are: speech-to-text (STT), text-to-speech (TTS), text bridging after conversion from speech, audio bridging after conversion from text, etc.

The session description protocol (SDP) [6] used in SIP to describe the session is used to express these attributes of the session (e.g., uniqueness in media mapping for conversion from one media to another for each communicating party).

Real-time text can also be presented in conjunction with other media like video and audio, as for example in Total Conversation services.

User Agents providing ToIP functionality SHOULD provide suitable alerting, specifically offering visual and/or tactile alerting so that deaf and hard of hearing users can use them.

The SIP abilities to set up text conversation sessions from any location, as well as privacy and security provisions SHOULD be implemented in ToIP services.

Where ToIP is used in conjunction with other media, exposure of SIP functions through the User Interface MUST be available in equivalent fashion for all supported media. In other words, where certain SIP call control functions are available for the audio media part of the session, these functions MUST also be supported for the text media part of the same session.

Any ToIP implementation MUST also allow invocation and use of relevant transcoding services where these are available. This can be achieved through application of SIP techniques for different

session establishment models [7]: Third party call control [8] and Conference Bridge model [9].

Both point-to-point and multipoint communication need to be defined for the session establishment using T.140 text conversation. In addition, ToIP services SHOULD support interworking with text telephony [10].

The general framework for ToIP can be described as follows:

a. Session setup, modification and teardown procedures for point-to-point and multimedia calls

b. Registration procedures and address resolutions

c. Registration of user preferences

d. Negotiation procedures for device capabilities

e. Discovery and invocation of transcoding/translation services between the media in the call

f. Different session establishment models for transcoding / translation services invocation: Third party call control and conference bridge model

g. Uniqueness in media mapping to be used in the session for conversion from one media to another by the transcoding / translation server for each communicating party

h. Media bridging services for T.140 real-time text as described in RFC4103, audio, and video for multipoint communications

i. Transparent session setup, modification, and teardown between text conversation capable and voice/video capable devices

j. Support of text media transport using T.140 over RTP as laid out in RFC 4103 [4]

k. Signaling of status information, call progress and the like in a suitable manner, bearing in mind the user may have a hearing impairment

l. T.140 real-time text presentation mixing with voice and video

m. T.140 real-time text conversation sessions using SIP, allowing users to move from one place to another

n. User privacy and security for sessions setup, modification, and teardown as well as for media transfer

o. Interoperability between T.140 conversations and analogue text telephones

p. Routing of emergency calls according to national or regional policy to the same level of a voice call.

5.4. Requirements for ToIP Interworking

Analog text telephony is cumbersome because of incompatible national implementations where interworking was never considered. A large number of these implementations have been documented in ITU-T V.18, which also defines modem detection sequences for the different text terminals. The full modem capability exchange between two wildly different terminals can take more than one minute to complete if both terminals have a common text modulation.

To resolve international analog textphone incompatibilities, text telephone gateways MUST transcode incoming analog signals into T.140 and vice versa. The modem capability exchange time is then also reduced, since V.18 allows the sequence of protocol discovery to be customized. Hence, the text telephone gateways will assume the analog text telephone protocol used in the region the gateway is located. For example, in the USA, Baudot might be tried as the initial protocol. If negotiation for Baudot fails, the full modem capability exchange will then take place. In contrast, in the UK, ITU-T V.21 might be the first choice.

6. Detailed requirements for Text-over-IP

ToIP services MUST use SIP for call control and signaling.

A ToIP user may wish to call another ToIP user, or join a conference call involving several users. He or she may, also, wish to initiate or join a multimedia call, such as a Total Conversation call.

There may be some need for pre-call setup e.g. storing registration information in the SIP registrar to provide information about how a user can be contacted. This will allow calls to be set up rapidly and with proper routing and addressing.

Similarly, there are requirements that need to be satisfied during call set up when other media are preferred by a user. For instance, some users may prefer to use audio while others want to use text as their preferred modality. In this case, transcoding services might be needed for text-to-speech (TTS) and speech-to-text (STT). The requirements for transcoding services need to be negotiated in real-time to set up the session.

The subsequent subsections describe some of these requirements in detail.

6.1. Pre-Call Requirements

The need to use ToIP as a medium of communications can be expressed by users during registration time. Two situations need to be considered in the pre-call setup environment:

a. User Preferences: It MUST be possible for a user to indicate a preference for ToIP by registering that preference with a SIP server that is part of the ToIP service.

b. Server to support User Preferences: SIP servers that are part of ToIP services MUST have the capability to act on users preferences for ToIP to accept or reject the call, based on the user preferences defined during the pre-call setup registration time. For example, if the user is called by another party, and it is determined that a transcoding server is needed, the call MUST be re-directed or otherwise handled accordingly.

6.2 Basic Point-to-Point Call Requirements

The point-to-point call will take place between two parties. The requirements are described in subsequent sub-sections. They assume that one or both of the communicating parties will indicate ToIP as a possible or preferred medium for conversation using SIP in the session setup.

6.2.1 Session Setup

Users will set up a session by identifying the remote party or the service they will want to connect to. However, conversations could be started using a mode other than ToIP. For instance, the conversation might be established using audio and the user could subsequently elect to switch to text, or add text as an additional modality, during the conversation. Systems supporting ToIP MUST allow users to select any of the supported conversation modes at any time, including mid-conversation.

Systems SHOULD allow the user to specify a preferred mode of communication, with the ability to fall back to alternatives that the user has indicated are acceptable.

If the user requests simultaneous use of text and audio, and this is not possible either because the system only supports alternate modalities or because of resource management on the network, the system MUST try to establish a text-only communication. The user MUST be informed of this change throughout the process, either in text or in a combination of modalities that MUST include text.

Session setup, especially through gateways to other networks, MAY require the use of specially formatted addresses or other mechanisms for invoking gateways.

The following features MAY need to be implemented to facilitate the session establishment using ToIP:

a. Caller Preferences: SIP headers (e.g., Contact) can be used to show that ToIP is the medium of choice for communications.

b. Called Party Preferences: The called party being passive can formulate a clear rule indicating how a call should be handled either using ToIP as a preferred medium or not, and whether a designated SIP proxy needs to handle this call or it is handled in the SIP user agent (UA).

c. SIP Server support for User Preferences: SIP servers can also handle the incoming calls in accordance to preferences expressed for ToIP. The SIP Server can also enforce ToIP policy rules for communications (e.g. use of the transcoding server for ToIP).

6.2.2 Addressing

The SIP [3] addressing schemes MUST be used for all entities. For example SIP URL and Tel URL will be used for caller, called party, user devices, and servers (e.g., SIP server, Transcoding server).

The right to include a transcoding service MUST NOT require user registration in any specific SIP registrar, but MAY require authorisation of the SIP registrar in the service.

6.2.3 Alerting and session progress presentation

User Agents supporting ToIP MUST have an alerting method (e.g., for incoming calls) that can be used by deaf and hard of hearing people or provide a range of alternative, but equivalent, alerting methods that are suitable for all users, regardless of their abilities and preferences.

It should be noted that general alerting systems exist, and one common interface for triggering the alerting action is a contact closure between two conductors.

Among the alerting options are alerting by the User AgentÆs User Interface and specific alerting user agents registered to the same registrar as the main user agent.

If present, identification of the originating party (for example in the form of a URL or CLI) MUST be clearly presented to the user in a form suitable for the user BEFORE answering the request. When the invitation to initiate a conversation involving ToIP originates from a gateway, this MAY be signaled to the user.

During a conversation that includes ToIP, status and session progress information MUST be provided in text. That information MUST be equivalent to session progress information delivered in any other format, for example audio. Users MUST be able to manage

the session and perform all session control functions based on the textual session progress information.

The user MUST be informed of any change in modalities.

Session progress information SHOULD use simple language as much as possible so that as many users as possible can understand it. The use of jargon or ambiguous terminology SHOULD be avoided at all times. It is RECOMMENDED to let text information be used together with icons symbolising the items to be reported.

There MUST be a clear indication, both visually as well as audibly whenever a session gets connected or disconnected. The user SHOULD never be in doubt as to what the status of the connection is, even if he/she is not able to use audio feedback or vision.

In summary, it SHOULD be possible to observe visual or tactile indicators about:
- Call progress
- Availability of text, voice and video channels
- Incoming call
- Incoming text
- Typed and transmitted text
- Any loss in incoming text.

6.2.4 Call Negotiations

The Session Description Protocol (SDP) used in SIP [3] provides the capabilities to indicate ToIP as a media in the call setup. RFC 4103 [5] provides the RTP payload type text/t140 for support of ToIP which can be indicated in the SDP as a part of SDP INVITE, OK and SIP/200/ACK for media negotiations. In addition, SIPÆs offer/answer model can also be used in conjunction with other capabilities including the use of a transcoding server for enhanced call negotiations [7,8,9].

6.2.5 Answering

Systems SHOULD provide a best-effort approach to answering invitations for session set-up and users should be kept informed at all times about the progress of session establishment. On all systems that both inform users of session status and support ToIP, this information MUST be available in text, and MAY be provided in other visual media.

6.2.5.1 Answering Machine

Systems for ToIP MAY support an auto-answer function, equivalent to answering machines on telephony networks. If an answering machine function is supported, it MUST support at least 160 characters for the greeting message. It MUST support incoming text message storage of a minimum of 4096 characters, although systems

MAY support much larger storage. It is RECOMMENDED that systems support storage of at least 20 incoming messages of up to 16000 characters.

When the answering machine is activated, user alerting SHOULD still take place. The user SHOULD be allowed to monitor the auto-answer progress and where this is provided the user MUST be allowed to intervene during any stage of the answering machine and take control of the session.

6.2.6 Actions During Calls

Certain actions need to be performed for the ToIP conversation during the call and these actions are described briefly as follows:

a. Text transmission SHALL be done character by character as entered, or in small groups transmitted so that no character is delayed between entry and transmission by more than 300 milliseconds.

b. The text transmission SHALL allow a rate of at least 30 characters per second so that human typing speed as well as speech to text methods of generating conversation text can be supported.

c. After text connection is established, the mean end-to-end delay of characters SHALL be less than two seconds, measured between two ToIP users. This requirement is valid as long as the text input rate is lower or equal to the text reception and display rate.

d. The character corruption rate SHALL be less than 1% in conditions where users experience the quality of voice transmission to be low but useable. This is in accordance with ITU-T F.700 Annex A.3 quality level T1.

e. When interoperability functions are invoked, there may be a need for intermediate storage of characters before transmission to a device receiving slower than the typing speed of the sender. Such temporary storage SHALL be dimensioned to adjust for receiving at 30 characters per second and transmitting at 6 characters per second during at least 4 minutes [less than 3k characters].

f. To enable the use of international character sets the transmission format for text conversation SHALL be UTF-8, in accordance with ITU-T T.140.

g. If text is detected to be missing after transmission, there SHALL be an indication in the text marking the loss. For 7 bit terminals this loss MAY be marked as an apostrophe: Æ.

g. When used from a terminal designed for PSTN text telephony, or in interworking with such a terminal, ToIP shall enable

alternating between text and voice in a similar manner as the PSTN text telephone handles this mode of operation. (This mode is often called VCO/HCO in the USA and the UK).

i. When display of the conversation on end user equipment is included in the design, display of the dialogue SHALL be made so that it is easy to read text belonging to each party in the conversation.

6.2.6.1 Text and other Media Handling Between ToIP User Agents

The following requirements are valid for media handling during calls:

a. When used between User Agents designed for ToIP, it SHALL be possible to send and receive text simultaneously.

b. When used between User Agents that support ToIP, it SHALL be possible to send and receive text simultaneously with the other media (text, audio and/or video) supported by the same terminals.

c. It SHOULD be possible to know during the call that ToIP is available, even if it is not invoked at call setup (only voice and/or video is used for example). To disable this, the user must disable the use of ToIP. This is possible during registration at the REGISTRAR.

6.2.6.2 Call Action with Native ToIP User Agents

a. It SHOULD be possible to answer a call with text capabilities enabled.

b. It MAY be possible to use video simultaneously with the other media in the call.

c. It MUST be possible to answer a call in voice or video without text enabled, and add text later in the call.

d. It MUST be possible to disconnect the call.

e. It SHOULD be possible to invoke multi-party calls.

f. It MUST be possible to transfer the call.

6.2.7 Additional session control

Systems that support additional session control features, for example call waiting, forwarding, hold etc on voice calls, MUST offer equivalent functionality for text calls.

6.2.8 File storage

Systems that support ToIP MAY save the text conversation to a file. This SHOULD be done using a standard file format. For example: UTF8 text file in XML format including record timestamp, party and the text conversation.

6.3 Conference Call Requirements for ToIP User Agents

The conference call requirements deal with multipoint conferencing calls where there will be at least one or more ToIP capable devices along with other end user devices where the total number end user devices will be at least three.

It SHOULD be possible to use the text medium in conference calls, in a similar way as the audio is handled and the video is displayed. Text in conferences can be used both for letting individual participants use the text medium (for example, for sidebar discussions in text while listening to the main conference audio), as well as for central support of the conference with real time text interpretation of speech.

6.4 Transport via RTP

ToIP uses RTP as the default transport protocol for transmission of real-time text via medium text/t140 as specified in RFC 4103 [5].

The redundancy method of RFC 4103 [5] SHOULD be used for making text transmission reliable.

Text capability MUST be announced in SDP by a declaration in line with this example:

```
m=text 11000 RTP/AVP 98 100
a=rtpmap:98 t140/1000
a=rtpmap:100 red/1000
a=fmtp:100 98/98/98
```

Characters SHOULD be buffered for transmission and transmitted every 300 ms.

By having this single coding and transmission scheme for real time text defined, in the SIP call control environment, the opportunity for interoperability is optimized.

However, if good reasons exist, other transport mechanisms MAY be offered and used for the T.140 coded text, provided that proper negotiation is introduced, and RFC 4103 [5] transport MUST be used as both the default as well as the fallback transport.

6.5 Character Set

a. ToIP services MUST use UTF-8 encoding as specified in ITU-T T.140 [12].

b. ToIP SHOULD handle characters with editing effect such as new line, erasure and alerting during session as specified in ITU-T T.140.

6.6 Transcoding

Transcoding of text may need to take place in gateways between ToIP and other forms of text conversation. For example to connect to a PSTN text telephone.

6.7 Relay Services

The relay service acts as an intermediary between two or more callers using different media or different media encoding schemes.

The basic text relay service allows a translation of speech to text and text to speech, which enables hearing and speech impaired callers to communicate with hearing callers. Even though this document focuses on ToIP, we want to remind readers that there exist other relay services like, for example, speech to sign language and vice versa using video.

It is RECOMMENDED that ToIP implementations make the invocation and use of relay services as easy as possible. It MAY happen automatically when the call is being set up based on any valid indication or negotiation of supported or preferred media types. A transcoding framework document using SIP [7] describes invoking relay services, where the relay acts as a conference bridge or uses the third party control mechanism. ToIP implementations SHOULD support this transcoding framework.

Adding or removing a relay service MUST be possible without disrupting the current call.

When setting up a call, the relay service MUST be able to determine the type of service requested (e.g., speech to text or text to speech), to indicate if the caller wants voice carry over, the language of the text, the sign language being used (in the video stream), etc.

It SHOULD be possible to route the call to a preferred relay service even if the user makes the call from another region or network than usually used.

6.8 Emergency services

Access to emergency services using ToIP SHOULD provide an equivalent service to the one offered by other supported media, like audio.

6.9 User Mobility

ToIP User Agents SHOULD use the same mechanisms as other SIP User Agents to resolve mobility issues. It is RECOMMENDED to use a SIP-address for the users, resolved by a SIP REGISTRAR, to enable basic user mobility. Further mechanisms are defined for the 3G IP multimedia systems.

6.10 Confidentiality and Security

User confidentiality and privacy need to be met as described in SIP [3]. For example, nothing should reveal the fact that the user of ToIP is a person with a disability unless the user prefers to make this information public. If a transcoding server is being used, this SHOULD be transparent. Encryption SHOULD be used on end-to-end or hop-by-hop basis as described in SIP [3] and SRTP [19]

Authentication needs to be provided for users in addition to the message integrity and access control.

Protection against Denial-of-service (DoS) attacks needs to be provided considering the case that the ToIP users might need transcoding servers.

7. Interworking Requirements for ToIP

A number of systems for real time text conversation already exist as well as a number of message oriented text communication systems. Interoperability is of interest between ToIP and some of these systems. This section describes requirements on this interoperability, especially for the PSTN text telephony to ensure full backward interoperability with ToIP.

7.1 ToIP Interworking Gateway Services

Interactive texting facilities exist already in various forms and on various networks. On the PSTN, it is commonly referred to as text telephony.

Simultaneous or alternating use of voice and text is used by a large number of users who can send voice, but must receive text or who can hear but must send text due to a speech disability.

7.2 ToIP and PSTN/ISDN Text-Telephony

On PSTN networks, transmission of interactive text takes place using a variety of codings and modulations, including ITU-T V.21 [II], Baudot, DTMF, V.23 [III] and others. Many difficulties have arisen as a result of this variety in text telephony protocols and the ITU-T V.18 [10] standard was developed to address some of these issues.

ITU-T-V.18 [10] offers a native text telephony method plus it defines interworking with current protocols. In the interworking mode, it will recognise one of the older protocols and fall back to that transmission method when required.

In order to allow systems and services based on ToIP to communicate with PSTN text telephones, text gateways are the recommended approach. These gateways MUST use the ITU-T V.18 [10] standard at the PSTN side.

Buffering MUST be used to support different transmission rates. At least 1K buffer MUST be provided. A buffer of at least 2K characters is RECOMMENDED. In addition, the gateway MUST provide a minimum throughput of at least 30 characters/second or the highest speed supported by the PSTN text telephony protocol side, whichever is the lowest.

PSTN-ToIP gateways MUST allow alternating use of text and voice.

PSTN and ISDN to ToIP gateways that receive CLI information from the originating party MUST pass this information to the receiving party as soon as possible.

Priority MUST be given to calls labeled as emergency calls.

7.3 ToIP and Cellular Wireless circuit switched Text-Telephony

Cellular wireless (or Mobile) circuit switched connections provide a digital real-time transport service for voice or data.
The access technologies include GSM, CDMA, TDMA, iDen and various 3G technologies.

Alternative means of transferring the Text telephony data have been developed when TTY services over cellular was mandated by the FCC in the USA. They are a) "No-gain" codec solution, b) the Cellular Text Telephony Modem (CTM) solution and c) "Baudot mode" solution.

The GSM and 3G standards from 3GPP make use of the CTM modem in the voice channel for text telephony.
However, implementations also exist that use the data channel to provide such functionality. Interworking with these solutions SHOULD be done using text gateways that set up the data channel connection at the GSM side and provide ToIP at the other side.

7.3.1 "No-gain"

The "No-gain" text telephone transporting technology uses specially modified EFR [15] and EVR [16] speech vocoders in both mobile terminals used to provide a text telephony call. It provides full duplex operation and supports alternating voice and text.("VCO/HCO"). It is dedicated to the CDMA and TDMA mobile technologies and the US Baudot type of text telephones.

7.3.2 Cellular Text Telephone Modem (CTM)

CTM [17] is a technology independent modem technology that provides the transport of text telephone characters at up to 10 characters/sec using modem signals that are at or below 1 kHz and uses a highly redundant encoding technique to overcome the fading and cell changing losses. On any interface that uses analog transmission, half-duplex operation must be supported as the "send" and "receive" modem frequencies are identical. The use of CTM may have to be modified slightly to support half-duplex operation.

7.3.3 "Baudot mode"

This term is often used by cellular terminal suppliers for a GSM cellular phone mode that allows TTYs to operate into a cellular phone and to communicate with a fixed line TTY.

7.3.4 Data channel mode

Many mobile terminals allow the use of the data channel to transfer data in real-time. Data rates of 9600 bit/s are usually supported on the mobile network. Gateways or the interworking function provides interoperability with PSTN textphones.

7.3.5 Common Text Gateway Functions

Text gateways MUST cover the differences that result from different text protocols. The protocols to be supported will depend on the service requirements of the Gateway.

Different data rates of different protocols MAY require text buffering.

Interoperation of half-duplex and full-duplex protocols MAY require text buffering and some intelligence to determine when to change direction when operating in half-duplex.

Identification may be required of half-duplex operation either at the "user" level (ie. users must inform each other) or at the "protocol" level (where an indication must be sent back to the Gateway).

A text gateway MUST be able to route text calls to emergency service providers when any of the recognised emergency numbers that support text communications for the country or region are called eg. "911" in USA and "112" in Europe. Routing text calls to emergency services MAY require the use of a transcoding service.

A text gateway MUST act as a SIP User Agent on the IP side.

7.4 ToIP and Cellular Wireless ToIP

ToIP MAY be supported over the cellular wireless packet switched service. It interfaces to the Internet. For 3GPP 3G services, the support is described to use ToIP in 3G TS 26.235 [20].

A text gateway with cellular wireless packet switched services MUST be able to route text calls into emergency service providers when any of the recognized emergency numbers that support text communication for the country are called.

7.5 Instant Messaging Support

Many people use Instant Messaging to communicate via the Internet using text. Instant Messaging transfers blocks of text rather than streaming as is used by ToIP. As such, it is not a replacement for ToIP and in particular does not meet the needs for real time conversations of deaf, hard of hearing and speech-impaired users as defined in RFC 3351 [21]. It is unsuitable for communications through a relay service [I]. The streaming character of ToIP provides a better user experience and, when given the choice, users often prefer ToIP.

However, since some users might only have Instant Messaging available, text gateways MAY be developed to allow interworking between Instant Messaging systems and ToIP solutions.

Because Instant Messaging is based on blocks of text, rather than on a continuous stream of characters, such gateways need to transform between these two formats. Text gateways for interworking between Instant Messaging and ToIP MUST concatenate individual characters originating at the ToIP side into blocks of text and:

a. When the length of the concatenated message becomes longer than 50 characters, the buffered text SHOULD be transmitted to the Instant Messaging side as soon as any non-alphanumerical character is received from the ToIP side.

b. When a new line is received from the ToIP side, the buffered characters up to that point, including the carriage return and/or line feed characters, SHOULD be transmitted to the Instant Messaging side.

c. When the ToIP side has been idle for at least 5 seconds, all buffered text up to that point SHOULD be transmitted to the Instant Messaging side.

It is RECOMMENDED that during the session, both users are constantly updated on the progress of the text input.
Many Instant Messaging protocols signal that a user is typing to the other party in the conversation. Text gateways between such Instant Messaging protocols and ToIP MUST provide this signaling to the Instant Messaging side when characters start being received, or at the beginning of the conversation.

At the ToIP side, an indicator of writing the Instant Message MUST be present where the Instant Messaging protocol provides one. For example, the real-time text user MAY see . . . waiting for replying IM. . . And per 5 seconds that pass a . (dot) can be shown.

Those solutions will reduce the difficulties between a streaming versus blocked text.

Even though the text gateway can connect Instant Messaging and ToIP, the best solution is to take advantage of the fact that the user interfaces and the user communities for instant messaging and ToIP telephony are extremely similar. After all, the character input, the character display, Internet connectivity and SIP stack are the same for Instant Messaging (SIMPLE) and ToIP.

Devices that implement Instant Messaging SHOULD implement ToIP as described in this document.

7.6 IP Telephony with Traditional RJ-11 Interfaces

Analogue adapters using SIP based IP communication and RJ-11 connectors for connecting traditional PSTN devices (ATA box) SHOULD enable connection of legacy PSTN text telephones [18]. These adapters SHOULD contain V.18 modem functionality, voice handling functionality, and conversion functions to/from SIP based ToIP with T.140 transported according to RFC 4103 [5], in a similar way as it provides interoperability for voice calls. If a call is set up and text/t140 capability is not declared by the endpoint (by the end-point terminal or the text gateway in the network at the end-point), a method for invoking a transcoding server shall be used. If no such server is available, the signals from the textphone MAY be transmitted in the voice channel as audio with high quality of service.
NOTE: It is preferred that such analogue adaptors do use RFC 4103 [5] on board and thus act as a text gateway. Sending textphone signals over the voice channel is undesirable due to possible filtering and compression and packet loss between the end-points. This can result in dropping characters in the textphone conversation or even not allowing the textphones to connect with each other.

7.7 Multi-functional gateways

In practice many interworking gateways will be implemented as gateways that combine different functions. As such, a text gateway could be build to have modems to interwork with the PSTN and support both Instant Messaging as well as ToIP. Such interworking functions are called Combination gateways.

Combination gateways MUST provide interworking between all of their supported text based functions. For example, a text gateway that has modems to interwork with the PSTN and that support both Instant Messaging and real-time ToIP MUST support the following interworking functions:

- PSTN text telephony to real-time ToIP.
- PSTN text telephony to Instant Messaging.
- Instant Messaging to real-time ToIP.

7.8 ToIP interoperability with PSTN text telephones.

Gateways between the ToIP network and other networks MAY need to transcode text streams. ToIP makes use of the ISO 10646 character set. Most PSTN textphones use a 7-bit character set, or a character set that is converted to a 7-bit character set by the V.18 modem.

When transcoding between character sets and T.140 in gateways, special consideration MUST be given to the national variants of the 7 bit codes, with national characters mapping into different codes in the ISO 10 646 code space. The national variant to be used could be selectable by the user on a per call basis, or be configured as a national default for the gateway.

The missing text indicator in T.140, specified in T.140 amendment 1, cannot be represented in the 7 bit character codes. Therefore these characters SHOULD be transcoded to the ’ (apostrophe) character in legacy text telephone systems, where this character exists. For legacy systems where the character ’ does not exist, the . (full stop) character SHOULD be used instead.

7.9 Gateway Discovery

ToIP requires a method to invoke a text gateway. As described previously in this draft, these text gateways MUST act as User Agents at the IP side. The capabilities of the text gateway during the call will be determined by the call capabilities of the terminal that is using the gateway. For example, a PSTN textphone is only able to receive voice and streaming text, so the text gateway will only allow ToIP and audio.

Examples of possible scenarios for discovery of the text gateway are:

- PSTN textphone users dial a prefix number before dialing out.
- Separate text subscriptions, linked to the phone number or terminal identifier/ IP address.
- Text capability indicators.
- Text preference indicator.
- Listen for V.18 modem modulation text activity in all calls.
- Call transfer request by the called user.
- Placing a call via the web, and using one of the methods described here
- Text gateways with its own telephone number and/or SIP address. (This requires user interaction with the text gateway to place a call).
- ENUM address analysis and number plan
- Number or address analysis leads to the gateway for all PSTN calls.

8. Afterword

The authors want to make it clear that ToIP is a way of allowing real-time, interactive text conversation between all users and is thus not only for the hearing and speech impaired users.

The users may invoke the ToIP services for many different reasons. For example:

- Noisy environment (e.g., in a machine room of a factory where listening is difficult)
- Busy with another call and want to participate in two calls at the same time.
- Text and/or speech recording services (e.g., text documentation/audio recording for legal/clarity/flexibility purposes)
- Overcoming of language barriers through speech translation and/or transcoding services.
- Hearing loss, tinnitus or deafness due to the aging process or any other reason.

NOTE: In many of the above examples, text may accompany speech and could be displayed in a manner similar to subtitling in broadcasting environments or any other suitable manner. This could occur for individuals who are hard of hearing and also for mixed calls with a hearing and deaf person listening to the call.

9. Security Considerations

There are no additional security requirements other than described earlier.

10. Authors Addresses

The following people provided substantial technical and writing contributions to this document, listed alphabetically:

Willem P. Dijkstra
TNO Informatie- en Communicatietechnologie
Postbus 15000
9700 CD Groningen
The Netherlands
Tel: +31 50 585 77 24
Fax: +31 50 585 77 57
Email: willem.dijkstra@tno.nl

Barry Dingle
ACIF, 32 Walker Street
North Sydney, NSW 2060 Australia
Tel +61 (0)2 9959 9111
Fax +61 (0)2 9954 6136
TTY +61 (0)2 9923 1911
Mob +61 (0)41 911 7578
Email barry.dingle@bigfoot.com.au

Guido Gybels
Department of New Technologies
RNID, 19-23 Featherstone Street
London EC1Y 8SL, UK
Tel +44(0)20 7294 3713
Txt +44(0)20 7296 8019
Fax +44(0)20 7296 8069
Email: guido.gybels@rnid.org.uk

Gunnar Hellstrom
Omnitor AB
Renathvagen 2
SE 121 37 Johanneshov
Sweden
Phone: +46 708 204 288 / +46 8 556 002 03
Fax: +46 8 556 002 06
Email: gunnar.hellstrom@omnitor.se

Henry Sinnreich
pulver.com
115 Broadhollow Rd
Suite 225
Melville, NY 11747
USA
Tel: +1.631.961.8950

Gregg C Vanderheiden
University of Wisconsin-Madison
Trace R & D Center
1550 Engineering Dr (Rm 2107)
Madison, Wi 53706
USA
gv@trace.wisc.edu
Phone +1 608 262-6966
FAX +1 608 262-8848

Arnoud A. T. van Wijk
Viataal (Dutch Institute for the Deaf)
Research & Development
Afdeling RDS
Theerestraat 42
5271 GD Sint-Michielsgestel
The Netherlands.
Email: a.vwijk@viataal.nl

11. References

11.1 Normative

1. Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

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

3. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol, RFC 3621, IETF, June 2002.

4. ITU-T Recommendation T.140, "Protocol for Multimedia Application Text Conversation (February 1998) and Addendum 1 (February 2000).

5. G. Hellstrom, "RTP Payload for Text Conversation, RFC 4103, June 2005.

6. G. Camarillo, H. Schulzrinne, and E. Burger, "The Source and Sink Attributes for the Session Description Protocol," IETF, August 2003 - Work in Progress.

7. G.Camarillo, "Framework for Transcoding with the Session Initiation Protocol" IETF June 2005 - Work in progress.

8. G. Camarillo, H. Schulzrinne, E. Burger, and A. van Wijk, "Transcoding Services Invocation in the Session Initiation Protocol (SIP) Using Third Party Call Control (3pcc)" RFC 4117, June 2005.

9. G. Camarillo, "The SIP Conference Bridge Transcoding Model," IETF, August 2003 - Work in Progress.

10. ITU-T Recommendation V.18,"Operational and Interworking Requirements for DCEs operating in Text Telephone Mode," November 2000.

11. "XHTML 1.0: The Extensible HyperText Markup Language: A Reformulation of HTML 4 in XML 1.0", W3C Recommendation. Available at http://www.w3.org/TR/xhtml1.

12. Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

13. TIA/EIA/825 "A Frequency Shift Keyed Modem for Use on the Public Switched Telephone Network." (The specification for 45.45 and 50 bit/s TTY modems.)

14. Bell-103 300 bit/s modem.

15. TIA/EIA/IS-823-A "TTY/TDD Extension to TIA/EIA-136-410 Enhanced Full Rate Speech Codec (must used in conjunction with TIA/EIA/IS-840)"

16. TIA/EIA/IS-127-2 "Enhanced Variable Rate Codec, Speech Service Option 3 for Wideband Spread Spectrum Digital Systems. Addendum 2."

17. 3GPP TS26.226 "Cellular Text Telephone Modem Description" (CTM).

18. I. Butcher, S. Lass, D. Petrie, H. Sinnreich, and C. Stredicke, "SIP Telephony Device Requirements, Configuration and Data," IETF, February 2004 - Work in Progress.

19. Baugher, McGrew, Carrara, Naslund, Norrman, "The Secure Real Time Transport Protocol (SRTP)", RFC 3711, IETF, March 2004.

20. IP Multimedia default codecs. 3GPP TS 26.235

21. Charlton, Gasson, Gybels, Spanner, van Wijk, "User Requirements for the Session Initiation Protocol (SIP) in Support of Deaf, Hard of Hearing and Speech-impaired Individuals", RFC 3351, IETF, August 2002.

22. J. Rosenberg, H. Schulzrinne, "An Offer/Answer Model with the Session Description Protocol (SDP)", RFC 3624, IETF, June 2002.

11.2 Informative

I. A relay service allows the users to transcode between different modalities or languages. In the context of this document, relay services will often refer to text relays that transcode text into voice and vice-versa. See for example http://www.typetalk.org.

II. International Telecommunication Union (ITU), "300 bits per second duplex modem standardized for use in the general switched telephone network". ITU-T Recommendation V.21, November 1988.

III. International Telecommunication Union (ITU), "600/1200-baud modem standardized for use in the general switched telephone network. ITU-T Recommendation V.23, November 1988.

IV. Third Generation Partnership Project (3GPP), "Technical Specification Group Services and System Aspects; Cellular Text Telephone Modem; General Description (Release 5)". 3GPP TS 26.226 V5.0.0.

Intellectual Property Statement

Disclaimer of Validity

ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Acknowledgment

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

Sipping Working Group
Internet Draft
Expires: January 5, 2006

Roland Jesske
Deutsche Telekom

July 2005

Use of the Reason header filed in Session Initiation Protocol (SIP) responses

draft-jesske-sipping-etsi-ngn-reason-00.txt

Status of this Memo

Abstract

This document proposes the use of the Reason header field in SIP responses.

Table of Contents

1. Overview

The European Telecommunication Standards Institute (ETSI) is defining a Next Generation Network (NGN) where a substantial part of it is based on the IP Multimedia Subsystem (IMS) defined by the Third-Generation Partnership Project (3GPP). IMS is largely based on the Session Initiation Protocol [1].

ETSI has developed a number of requirements draft-jesske-sipping-tispan-requirements [5] to support the usage of SIP in Next Generation Networks that interoperate, at the service level, with the Public Switched Telephone Network (PSTN), the Integrated Services Digital Network (ISDN), the 3GPP IP Multimedia Subsystem (IMS), and SIP networks and terminals that implement the service logic.

In order to provide full support in SIP of existing services, extensions to SIP are needed.
This document proposes the use of the Reason header field in responses. This is needed for creating services that must be interoperable with the PSTN/ISDN network and the interoperability of traversing communications through SIP not using SIP-I.
An example is the ACR services described within draft-jesske-sipping-tispan-requirements [2]. Here it is needed to send the cause value #24 back to the PSTN/ISDN to identify why the communication was restricted and serve the user a correct announcement.
The described solution fulfils the requirements [REQ-GEN-1] and [REQ-ACR-1].

2. Overall Applicability

The SIP procedures specified in this document are foreseen for networks providing simulation services and/or interwoking to the PSTN/ISDN.
The document is describing the use of the Reason header in SIP responses. These procedures are only valuable if the reason contained in the element "protocol" is "Q.850" or "Redirection". " Redirection"

is a redirecting reason as described in [7] draft-elwell-sipping-redirection-reason-02.txt.
A inclusion of a SIP reason (protocol="SIP") is not helpful due to the fact that the response already provides the SIP reason.
The Release Causes are described within ETSI EN300 485 [5]

3. Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [RFC2119].

4. Procedures

For providing services and PSTN/ISDN interoperability it MUST be possible to include Reason header fields with Q.850 Cause values or redirecting reasons as described in [7] draft-elwell-sipping-redirection-reason-02.txt in responses.

4.1. Procedures at the UA

A UA that supports the Reason header field can process the Q.850 Cause Value and display it or an equivalent text. The inclusion of a Reason header field by UA is only for 2B2 UA interworking with the PSTN/ISDN or providing services foreseen.

4.2. Procedures at a SIP proxy

SIP proxies that receive a response containing a Reason header field is forwarding the response without changing the reason.

A SIP proxy receiving a request that includes a Reason header field can route the request to an application server for further analysis and base services on it.

Based on network policy a Proxy can remove a Reason header field send from a UAC.

4.3 Procedures at an application server

An application server that receives a SIP request that contains a response including a Reason header MAY analyze the SIP Reason and base further procedures on this analyses.
For Example the application server could use the reason for sending a announcement towards the originating entity of the Session.

If the application server provides a Service it can provide also Responses include a Q.850 reason or redirecting reasons as described in [7] draft-elwell-sipping-redirection-reason-02.txt in responses.

As an example the Anonymous Communication Rejection (ACR) service defined by ETSI Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN)

5 Example

Figure 1 shows the example of the ACR service interworking with the PSTN/ISDN

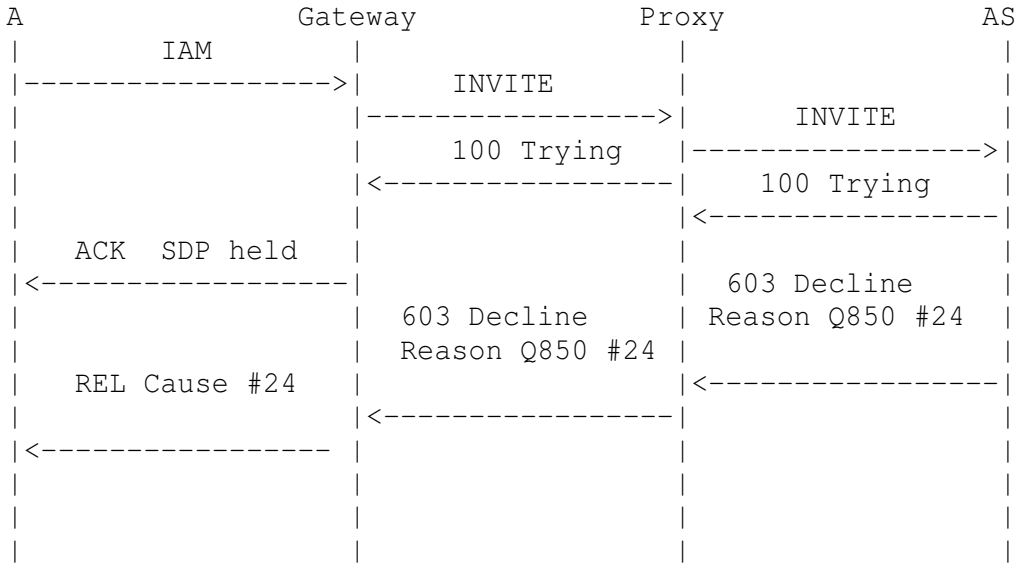


Figure 1: Third Party Call Control

6. Security Considerations

The presence of the Reason header in a response does not affect the treatment of the response.
Including such a header by an untrusted entity could adulterate the reactions of the originating entitys. E.G. sending back a cause value "24" can cause an announcement within the PSTN/ISDN saying that the call was rejected due to the ACR service.
Therefore it is RECOMMENDED to include the Reason header in Responses only by trusted entities as it is described within RFC3325 [8]

7. IANA Considerations

This document describes the use of the Reason header field described within RFC 3326 [2]. No additional SIP elements are defined within this document. Therefore, this document does not provide any action to IANA.

References

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

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

[3] Jesske, R., Alexeitsev, D., Garcia-Martin, M. "Input Requirements for the Session Initiation Protocol (SIP) in support for the European Telecommunications Standards Institute (ETSI) Next Generation Network (NGN) simulation services", draft-jesske-sipping-tispan-requirements-01.txt (work in progress), June 2005.

[4] 3GPP TS 24.229: "IP Multimedia Call Control Protocol based on SIP and SDP".

[5] ETSI EN 300 485: "Integrated Services Digital Network (ISDN); Definition and usage of cause and location in Digital Subscriber Signalling System No. one (DSS1) and Signalling System No. 7 (SS7) ISDN User Part (ISUP) [ITU-T Recommendation Q.850 (1998) with addendum modified] ".

[6] Jesske, R., Alexeitsev, D., Garcia-Martin, M. "Analysis of the Input Requirements for the Session Initiation Protocol (SIP) in support for the European Telecommunications Standards Institute (ETSI) Next Generation Networks (NGN) simulation services" (work in progress) June 2005

[7] Elwell, J., "SIP Reason header extension for indicating redirection reasons", draft-elwell-sipping-redirection-reason-02.txt (work in progress), June 2005.

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

Note: The ETSI specifications can be downloaded under http://pda.etsi.org/pda/queryform.asp free of charge.

Contributors

Denis Alexeitsev
Deutsche Telekom
Am Kavalleriesand 3
64307 Darmstadt
Germany
Phone: +496151832130
Email: d.alexeitsev@t-com.net

Acknowledgments

The author would like to thank the members of the ETSI TISPAN WG3 for their comments to this memo.

Author's Addresses

Roland Jesske
Deutsche Telekom
Am Kavalleriesand 3
64307 Darmstadt
Germany
Phone: +496151835940
Email: r.jesske@t-com.net

Full Copyright Statement

Acknowledgement

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

Sipping Working Group
Internet Draft
Expires: December 28, 2005

Roland Jesske
Denis Alexeitsev
Deutsche Telekom
Miguel Garcia-Martin
Nokia

June 2005

Analysis of the Input Requirements for the Session Initiation Protocol (SIP) in support for the European Telecommunications Standards Institute (ETSI) Next Generation Networks (NGN) simulation services

draft-jesske-sipping-tispan-analysis-00.txt

Status of this Memo

Abstract

This document analyzes the requirements generated by ETSI to support NGN simulation services implemented with SIP. The document analyzes standard solutions that can meet the requirements. Where a standard solution is not available, the document proposes one or more solutions. The aim is to provoke discussion within the Internet community and get early feedback.

Table of Contents

1. Overview

This document is a companion document of the Internet Draft "Input Requirements for the Session Initiation Protocol (SIP) in support for the European Telecommunications Standards Institute (ETSI) Next Generation Network (NGN) simulation services" [3]. We analyze each requirement trying to find an available standard solution that meets the requirement. When such solution is not known, we analyze different alternatives that will meet the requirement. The document’s intention is to provoke discussion in the Internet community in order to find suitable mechanisms that guarantee the implementation of the requirements within the ETSI NGN timeframe.

2 Analysis of the TISPAN Simulation Services reqirements

The following section could be seen as collected thoughts what possibilities are given to fulfill the above mentioned requirements.

Some of these ideas are still under discussion in ETSI and thus, do not even represent a firm proposal, but just a collection of alternatives that require further exploration.

2.1 General Requirements[REQ-GEN-1]

This requirement, since it is generally applicable to all the requirements, does not require a particular behavior. However, solutions that meet other requirements should also meet the constraint of seamlessly interwork with a similar service in the PSTN.

2.2 Anonymus Communication Rejection (ACR)[ACR-REQ-1] and [ACR-REQ-2]

Requirement [ACR-REQ-1] requires informing the caller that her call has been rejected due to anonymity. We thought that an application server that implements the ACR service can either send an instant message to the caller (if supported) or divert the call to an announcement player that plays the appropriate audio message, however, this will be hard to be processed by an automata or a PSTN gateway. For example, in a PSTN originated call the PSTN should indicated an appropriate Release Cause (cause 24) due to interaction with the ACR service. Thus, any solution based on these two proposals would make difficult to meet REQ-GEN-1. The Release Causes are described within ETSI EN300 485 [13]

A more sophisticated solution proposes to add a Reason header with an appropriate Release Cause 24 as described in ETSI EN300 485 [13] to a 603 Decline response. This will allow interworking with the PSTN. Yet another alternative is to create a new response code, but we believe it is not really necessary to go into that solution.

As the current Reason header extension header is limited to requests, some extension is needed to state that the Reason header is valid for either the 603 (Decline) response, or some other status code as determined by this discussion.

Requirement [ACR-REQ-2] reads about allowing certain authorized callers to bypass the ACR service of a given callee. For that we envision that an application or SIP proxy server that has access to the caller’s user profile is able to add indicate in SIP that the user is authorized to bypass a potential callee’s ACR service. We propose to add a new P-ACR header field that proxies or application servers can insert. This header would be subject to the same security concerns and trusted domain considerations as the P-Asserted-Identity header field [8].

2.2.1 The P-ACR header field

In support for the ACR service [REQ-ACR-2], there is a need to indicate that in some circumstances the ACR service provided towards the terminating UE should not apply.

One of these cases is when the originating user has requested privacy of his asserted identity, but because the user belongs to an authorized group (e.g., policeman), all SIP request should go through to the called UA, no matter whether the called user has activated ACR or not.

In another scenario, a PSTN originated call does not deliver the asserted identity of the called party (e.g., if the call is originated in an analogue switch). In this case the PSTN gateway is not able to provide an asserted identity of the calling party. If the call is routed to a user who has the ACR service activated, the call shouldn't be rejected due to ACR.

To tackle this problem we suggest creating a new P-ACR header, which can be populated by a trusted entity (e.g., an Application Server or Media Gateway Control Function). The contents of the header will indicate that willingness of bypassing a potential ACR service in the called party side, and the motivation for it.

It is the same restrictions for the P-ACR as in RFC3325 for the P-Asserted-ID described must apply.

Proposed syntax for this header:

```
P-ACR          = "P-ACR" HCOLON p-acr-spec
                 *(COMMA p-acr-spec)
p-acr-spec     = "bypass" *(SEMI due-to-param)
due-to-param   = "due-to" EQUAL reason-token
reason-token   = "interworking" / "is-authorized" / "network"/
                  token
```

Example: P-ACR = bypass;due-to=is-authorized

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
P-ACR		adr	-	-	-	o	-	-

SUB	NOT	REF	INF	UPD	PRA
o	-	o	-	-	-

MESSAGE	PUBLISH
---	---

o o

The draft-ietf-sip-identity was not considered due to the fact that this draft recommends practices and conventions for identifying end users in SIP messages, and proposes a way to distribute cryptographically-secure authenticated identities. This is only for Requests specified. The P-ACR is especially used for indicating bypass mechanisms for ACR and the indication how P-Asserted-Identity was set.

2.3 Terminating Indication Presentation (TIP)/Terminating Indication Restriction (TIR). [REQ-TIP-1] and [REQ-TIP-2]

The implementation of these requirements need some header to convey the callee's URI, which might be different from the To header field or Request-URI, due to, e.g., redirections, call transfer, usage of PBX extensions, etc.

We believe that the P-Asserted-Identity [8] header field inserted in responses meets these requirements.
An additional Identity header is needed that is send from the connected SIP user like the From header from the originating user.

2.3.1 The P-Additional-Identity Header

The P-Additional-Identity header field is used among SIP entities (typically intermediaries) to carry the identity of the user sending a SIP response as it was not verified by authentication.

This additional header is needed for example a user calling a PBX desk (e.g., +49 6151 83 0000 for Deutsche Telekom in Darmstadt) will be forwarded to an Extension (e.g., +49 6151 83 5940 for Roland Jesske). It is required that the caller gets the latter identity, which is allocated to the callee.

```
PAdditionalID = "P-Additional-Identity" HCOLON PAdditionalID-value
                *(COMMA PAdditionalID-value)
PAdditionalID-value = name-addr / addr-spec
```

A P-Additional-Identity header field value MUST consist of exactly one name-addr or addr-spec. There may be one or two P-Additional-Identity values. If there is one value, it MUST be a sip, sips, or tel URI.
If there are two values, one value MUST be a sip or sips URI and the other MUST be a tel URI. It is worth noting that proxies can (and will) add and remove this header field.

This document adds the following entry to Table 2 of [1]:

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
P-Additional-Identity	r	adr	-	o	-	o	o	-

SUB	NOT	REF	INF	UPD	PRA
o	o	o	-	-	-

MESSAGE	PUBLISH
o	o

Note: Here is also an ongoing discussion if a new header is needed or if a existing header could be used to identify the ôrealö connected user. The Contact or Reply-to header were identified as not usable.

2.4 Advice of Charge (AOC) [REQ-AoC-1] and [REQ-AoC-2]

The Advice of Charge simulation service requires a caller to give an indication to the network that he wants to receive advice of charge information for a given communication (e.g., session, subscription, instant message, etc.). A mechanism to invoke the service based on a SIP-event base subscription and notification has been analyzed. However, this solution will introduce a synchronization problem, due to indicating the request for the service out-of-band. The basic problem is how to identify the SIP request for which the advice should be provided prior to sending the request, when such request has not even been created.

We propose, though, that the SIP request for which the Advice of Service information is requested is complemented with a new header that invokes the service. Once the service is properly invoked, there are a number of available mechanisms to deliver the information to the user, including but not restricted to, audio visual announcements, instant message, etc.

A detailed proposal for a new P-AoC header field is described in draft-garcia-sipping-etsi-ngn-p-headers-00 [7].

2.5 Communication Completion on Busy Subscriber (CCBS) [REQ-CCBS-1]to [REQ-CCBS4]

Discussion in ETSI TISPAN are leaning towards a segmented implementation of the CCBS service, where there is an application server which is serving the caller, and another application server which serving the callee. The main role of application servers is to

provide queue management. This is based on the modelling in the ISDN stage 2 and operates in this manner to allow interworking with the related ISDN service through a SIP/PSTN gateway.

We model the actual implementation of the CCBS service according to the following:

- When the AS of the callee detects that the callee is busy and that the callee supports the dialog event package [9], it forwards the busy response to the AS of the caller with an indication that the CCBS service is possible (i.e. queuing capabilities, and dialog event are supported, REQ-CCBS-1 and REQ-CCBS-2).
- Upon receipt of this indication, the caller is offered to activate the CCBS service (e.g. the AS of the caller plays an announcement with DTMF collection, etc).
- If the caller accepts the service activation, the AS of the caller subscribes to the CCBS service (e.g. subscribes to the CCBS queue of the callee to the dialog status of this callee, REQ-CCBS-2).
- Upon receipt of the CCBS subscription request, the AS of the caller confirms that CCBS monitoring has started,
- the AS of the callee then subscribes to the dialog event [9] of the callee,
- Upon receipt of a notification that the callee is free, the AS of the callee, notifies the AS of the caller.

- The AS of the caller sets-up the CCBS call using either a 3rd party control mechanism of a Refer with an indication that this is a CCBS call (REQ-CCBS-4).

In order to provide compatibility with terminals that implement the dialog event package, subscriptions that may originate or terminate in a terminal will be implemented according to the dialog event package [9].

Subscriptions not involving terminals (i.e., such as the subscriptions from one application server to another one) need to be complemented with additional information (REQ-CCBS-3). It is proposed to define a new "CCBS" event package for backwards compatibility purposes. The main difference between the "CCBS" event package and the ôdialogö event package lies in the way subscriptions and notifications are handled, there is no need to change the contents of the XML documents exchanged therein. Consequently, the CCBS event package notifications should contain a dialog information document as described in [9]. This allows to "tunnel" dialog information documents contained in dialog package notifications originated by endpoints into CCBS package notifications sent by application servers. The additional information to the dialog event package for providing queues management and concurs to build a subscription target is as follows:

queue: this information is needed for the application server to know whether to insert this new subscription in the queue or not. We suggest to add a new "queue" parameter to the "dialog" Event header field value. Possible values are "true","false", "suspend" (to suspend its place in queue), "resume" to resume its place in queue.

Hence, the CCBS Event: header will look like, for example

To start CCBS subscription:
Event: ccbs;queue=true;caller=sip:+390112285111@example.com

To suspend CCBS service (for instance, if caller becomes busy):
Event: ccbs;queue=suspend;caller= sip:+390112285111@example.com

To resume CCBS service:
Event: ccbs;queue=resume;caller= sip:+390112285111@example.com

A value of "ccbs" or "dialog" in the 486 Busy Here response will indicate the URI where the subscription could be made. This URI could, e.g., a GRUU. Additionally, the presence in a 486 Busy Here response of an Allow-Events header field with the value "CCBS" would help to determine the support for the service. However, RFC 3265 [11] seems to indicate that the Allow-Evens header is only meaningful in requests and 200 or 489 responses.

Implementation of REQ-CCBS-3 requires that the second time that the caller sends the INVITE request to the callee, as a result of an indication that the callee is not busy any longer, the INVITE request is marked with an flag indicating that the INVITE is the result of CCBS service. It is proposed that a Call-Info header field is extended with a new value of the "purpose" parameter. Hence, at the time of the CCBS call, the AS can insert the Call-Info: header into the INVITE message directed to user B when triggering the 3rd party call, or instruct the terminal to do so in the Refer-To: header inserting the æCall-Info:Æ header as a URL header (after the æ?Æ character).

Note: With regard to CCBS the discussion is ongoing what kind of solution could be taken. If a new Event Package is needed or if the extension of the existing dialog event package is good enough for CCBS.

2.6 Communication Completion on no Reply (CCNR) [REQ-CCNR-1] to [REQ-CCNR-4]

The implementation of the CCNR service does not require any extra implementation beyond the solutions proposed for implementing the CCBS service.

2.7 Malicious Communication IDentification (MCID) [REQ-MCID-1] and [REQ-MCID-2]

The implementation of the MCID service suggests to split the solution into the mechanism whereby a callee can indicate to an application server that he suspects a call is malicious [REQ-MCID-1], from the mechanism whereby an application server acquires the caller's identity if not present in the SIP request [REQ-MCID-2].

A possible solution for implementing REQ-MCID-1 consists of the user subscribing to a new event package. The application server will act as a notifier. Since the user does not need to receive any information, other than a message indicating whether the service has been successfully activated or not, we propose to do a fetch operation (as per RFC 3265 [11]) with the new event package.

We propose, therefore, a new event "mcid" event package. The value is accompanied by package parameters indicating the call to be identified (e.g., by its from-tag, call-id, and to-tag (if available)). The event package itself should contain a simple indication of the acceptance or not of the service.

To implement REQ-MCID-2 we envision that all SIP requests addressed to the user will be routed through the MCID application server. The application server will analyze all SIP requests. Two possbilities might take place: the SIP request contains trusted identity information (e.g., a trusted P-Asserted-Identity [8] header field, or an Identity [12] header field); or such identity is not present in the request, in which case, it might be still available upon request. This is the sometimes the case when a call is originated in the PSTN, because sometimes the calling party number is only available upon request.

To meet this requirement and be able to request the identity of the originator, we propose a SIP event package for requesting identity information. In most cases this will not be used, but in cases of interworking with the PSTN, the PSTN gateway will receive a SUBSCRIBE request for the new event package, will do the PSTN operations to retrive the calling party number, and will provide the appropriate notification to the subscriber (the application server).

2.8 Communication Waiting (CW) [REQ-CW-1] and [REQ-CW-2]

A solution that meets REQ-CW-1 suggests to use send an instant message indicating the ser the relevant parameters of the waiting call.

To implement REQ-CW-2 we suggest to use a 182 Queue response until the callee accepts the incoming session.

2.9 Communication Diversion (CDIV) [REQ-CDIV-1] to [REQ-CDIV-4]

For supporting CDIV service we envision the usage of draft-ietf-sip-history-info-06 [6] and draft-elwell-sipping-redirection-reason-01 [2].We therefore request the adoption of draft-elwell-sipping-redirection-reason as a working group charter item.

2.10 [REQ-CAT-1] and [REQ-CAT-2]

To support REQ-CAT-1 and REQ-CAT-2 we propose that, a PSTN Gateway (UA) or SIP proxy that has knowledge of the user's category inserts a P-Caller-Category header field categorizing the caller.

Sometimes the category of the caller is determined to be "operator". In such case, the presence of the Accept-Language header field can be combined to determine the language of the operator.

Use of this mechanism in any other context has serious security shortcomings, namely that there is absolutely no guarantee that the information has not been modified, or was even correct in the first place.

The proposed syntax for this header:

```
P-Caller-Category   = "P-Caller-Category " HCOLON p-cat-spec
                          *(COMMA p-cat-spec)
p-cat-spec     = "operator" / "subscriber" / "data" /
                  "test" / "payphone" / "mobile" / token
```

Example: P-Caller-Category = "payphone"

An other possibility is to use the solution described within draft-mahy-iptel-cpc-00 [14]. This solution was dicussed in the past within IPTEL but not follwed on

Question is which would be the appropriate way to follow.

4. Security Considerations

The requirements in this document are intended to result in a mechanism with applicability for ETSI NGN and NOT for the general Internet.

Use of this mechanism in any other context has serious security shortcomings, namely that there is absolutely no guarantee that the information has not been modified, or was even correct in the first place.

5. IANA Considerations

This document discusses implementation possibilities and does not pretend to be a firm proposal for the implementation of any of the solutions. Therefore, this document does not provide any action to IANA.

References

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

[2] Elwell, J., "SIP Reason header extension for indicating redirection reasons", draft-elwell-sipping-redirection-reason-02.txt (work in progress), June 2005.

[3] Jesske, R., Alexeitsev, D., Garcia-Martin, M. "Input Requirements for the Session Initiation Protocol (SIP) in support for the European Telecommunications Standards Institute (ETSI) Next Generation Network (NGN) simulation services", draft-jesske-sipping-tispan-requirements-01.txt (work in progress), June 2005.

[4] 3GPP TS 24.229: "IP Multimedia Call Control Protocol based on SIP and SDP".

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

[6] Barnes, M., "An Extension to the Session Initiation Protocol for Request History Information", draft-ietf-sip-history-info-06.txt (work in progress), January 2005.

[7] Garcia-Martin, M. "Private Header (P-Header) Extensions to the Session Initiation Protocol(SIP) in support of the European Telecommunications Standards Institute (ETSI) Next Generation Networks (NGN)", draft-garcia-sipping-etsi-ngn-p-headers-00.txt (work in progress), June 2005.

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

[9] Rosenberg, J., Schulzrinne, H., Mahy, R"An INVITE Initiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-06.txt (work in progress), April 2005.

[10] Rosenberg, J. "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-03 (work in progress), February 2005.

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

[12] Peterson, J., Jennings, C. "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05.txt (work in progress), March 2005.

[13] ETSI EN 300 485: "Integrated Services Digital Network (ISDN); Definition and usage of cause and location in Digital Subscriber Signalling System No. one (DSS1) and Signalling System No. 7 (SS7) ISDN User Part (ISUP) [ITU-T Recommendation Q.850 (1998) with addendum modified]".

[14] R. Mahy "The Calling Party's Category tel URI Parameter" draft-mahy-iptel-cpc-00, June 2003

Contributors

Keith Drage
Lucent Technologies
SN5 6PP SWINDON
United Kingdom
Phone: +44 1793 897312
Email: drage@lucent.com

Sebastien Garcin
France Telecom
38-40, Rue du General Leclerc
92130 Issy Les Moulineaux
France

Acknowledgments

The authors would like to thank the participants of the ETSI TISPAN WG3 for the comments and initial review of this document.

Author's Addresses

Roland Jesske
Deutsche Telekom
Am Kavalleriesand 3
64307 Darmstadt
Germany
Phone: +496151835940
Email: r.jesske@t-com.net

Denis Alexeitsev
Deutsche Telekom
Am Kavalleriesand 3
64307 Darmstadt
Germany
Phone: +496151832130
Email: d.alexeitsev@t-com.net

Miguel A. Garcia-Martin
Nokia
P.O. Box 407
NOKIA GROUP, FIN 00045
Finland
EMail: miguel.an.garcia@nokia.com

Full Copyright Statement

Acknowledgement

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

Sipping Working Group
Internet Draft
Expires: December 13, 2005

Roland Jesske
Deutsche Telekom
Denis Alexeitsev
Deutsche Telekom
Miguel Garcia-Martin
Nokia

June 14, 2005

Input Requirements for the Session Initiation Protocol (SIP) in support for the European Telecommunications Standards Institute (ETSI) Next Generation Network (NGN) simulation services

draft-jesske-sipping-tispan-requirements-01

Status of this Memo

Abstract

This document describes a set of requirements to the Session Initiation Protocol (SIP) [1] in support for simulation services provided in the context of ETSI Next Generation Networks (NGN). These requirements should help to find SIP solutions to provide the services described within this document.

Table of Contents

1. Overview

The European Telecommunications Standards Institute (ETSI) Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN) is defining the release 1 of the TISPAN Next Generation Network (NGN). Generally NGN is largely based on the 3rd Generation mobile Partnership Project (3GPP) IP Multimedia Subsystem (IMS) Release 7.

The TISPAN NGN project has selected SIP profiled by 3GPP TS 24.229 [26] for the IMS as the protocol used to establish and tear down multimedia sessions in the context of NGN. The goal for TISPAN is that only one IMS core specification is defined for both wire-line and wire-less multimedia applications.

While defining multimedia applications it is also needed to support existing Integrated Services Digital Network and Public Switched Telephone Network (ISDN/PSTN) supplementary services based on IMS. We refer to supplementary services provided with SIP in the context of NGN as 'simulation services'. They are referred to as simulation services because they need to be adapted to be provided with SIP, so small variations are expected when compared with the equivalent ISDN/PSTN supplementary service. The 3GPP TS 24.229 [26] is used to simulate the regarding services but to fulfill the requirements defined within TISPAN NGN Release 1 some further SIP support is needed.

This document defines some input requirements to support the implementation of simulation services. It is generally understood that not every requirement listed in this memo will require a SIP

extension. A companion Internet Draft analyses possible implementations of these requirements and explores different extensions when those are needed.

All mentioned 3GPP and ETSI Standards are free available under http://pda.etsi.org/pda/queryform.asp and http://www.3gpp.org/ftp/Specs/html-info/

The resulting work of this collaboration will eventually be contributed to International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) as part of their NGN work to have an alignment between the work of the standardization organizations.

2. Requirements for supporting simulation services within SIP

2.1 Simulation Services supported by TISPAN in Release 1

The following simulation services are supported by ETSI NGN Release 1:

-Communications DIVersion (CDIV). This simulation service allows the diversion of communications and the regarding service interworking with the PSTN/ISDN. The service comprises the equivalent PSTN/ISDN supplementary service for Call Forwarding Unconditional (CFU), Call Forwarding Busy (CFB), Call Forwarding on No Reply (CFNR), and Call Deflection (CD). The CFU supplementary service is described in ETSI EN 300 200 [7]. The CFB supplementary service is described in ETSI EN 300 199 [8]. The CFNR supplementary service is described in ETSI EN 300 201 [9]. The CD supplementary service is described in ETSI EN 300 202 [10].

- Incoming Communication Barring (ICB). This simulation service allows a user to block incoming communications based on the identity of the caller. The Call Barring supplementary service is described in ETSI ETS 300 520 [22].

- CONFerence (CONF). This simulation service provides the possibility to hold centralized conferences with 3 or more users. The CONF supplementary service is described in ETSI ETS 300 183 [14].

- Message Waiting Indication (MWI). This simulation service provides the user with information about the status of a voice/video/multimedia mailbox. The MWI supplementary service is described in ETSI EN 300 650 [19].

- Originating Indication Presentation (OIP)/Originating Indication Restriction (OIR). These simulation services support the presentation or restriction of the caller’s identity to the callee. They are the

simulation of the ISDN/PSTN Calling Line Identification Presentation/Restriction (CLIP/CLIR) supplementary services. The CLIP supplementary service is described in ETSI EN 300 089 [5]. The CLIR supplementary service is described in ETSI EN 300 090 [6].

- Terminating Indication Presentation (TIP)/Terminating Indication Restriction (TIR). These simulation services support the presentation or restriction of callee's identity to the caller. They are the simulation of the ISDN/PSTN Connected Line Identification Presentation/Restriction (COLP/COLR) supplementary services. The COLP supplementary service is described in ETSI EN 300 094 [23]. The COLR supplementary service is described in ETSI EN 300 095 [24].

- Communication Waiting (CW). This simulation service provides the ability of the callee to be informed at the time a communication is coming in, and that no resources are available for that incoming communication. The callee has then the choice of accepting, rejecting or ignoring the incoming communication. The caller will be informed that his communication is waiting. The CW supplementary service is described in ETSI ETS 300 056 [11].

- Communication HOLD (HOLD). This simulation service supports the possibility of suspending the communication (on hold) while for example another communication with another user is to be done. The CW supplementary service is described in ETSI ETS 300 139 [12].

- Anonymous Communication Rejection (ACR). This simulation service allows a user to reject incoming communications when the caller is anonymous. The ACR supplementary service is described in ETSI EN 300 798 [21].

- Advice of Charge (AoC). This simulation service allows the caller to request the displaying of tariff information related to the communication. The service can operate at setup time (AoC-S), during a session (AoC-D), or at the end of it (AoC-E). The AoC-S supplementary service is described in ETSI ETS 300 178 [16]. The AoC-D supplementary service is described in ETSI ETS 300 179 [17]. The AoC-E supplementary service is described in ETSI ETS 300 180 [18].

- Communication Completion on Busy Subscriber (CCBS). This simulation service supports the ability of a caller to complete a requested communication to a busy callee without having to make a new communication attempt when the callee becomes not busy anymore. It is possible for the caller to request several communications to be under the CCBS requested status. Also the callee can be subject to several CCBS communications from different callers. Additionally, the service provides queue management to arbitrate several CCBS requests to the

same callee. The CCBS supplementary service is described in ETSI EN 300 357 [15].

- Communication Completion on no Reply (CCNR). This simulation service supports the ability of the caller to complete a requested communication to a callee without having to make a new communication attempt when the callee showed activity (a communication attempt was done). The CCNR supplementary service is described in ETSI EN 301 134 [25].

- Malicious Communication IDentification (MCID). This simulation service enables the callee to indicate that an incoming communication is considered to be malicious and it should be identified and registered. The MCID supplementary service is described in ETSI ETS 300 128 [13].

- Explicit Communication Transfer (ECT). This simulation service allows the user having two separate sessions to connect the other two users together into a single session. The ECT supplementary service is described in ETSI EN 300 367 [20].

2.2 Requirements to support the TISPAN Simulation Services.

[REQ-GEN-1]
For all simulation services interoperability with the PSTN/ISDN is needed. Solutions that support the following requirements must interwork with the corresponding PSTN/ISDN supplementary service.

[REQ-ACR-1]
The ACR simulation service requires the caller to be informed that the communication was rejected due to this service. This is needed to inform the upstream elements (UAC, e.g., a PSTN/ISDN gateway) about the reason for the rejection.

[REQ-ACR-2]
It must be possible that authorized callers are not subject to the ACR service, thus, allowing the callee to receive anonymous requests from authorized callers. This effectively requires a mechanism to override the ACR service depending on the identity and authorization of the caller.

This is needed, e.g., for when a police officer or any other authority is anonymously calling to a user having the ACR simulation service enabled.

[REQ-TIP-1]
For supporting the TIP/TIR service when a communication is interworking with SIP-based Private Branch Exchanges (PBX) a

mechanism is required whereby the private extension of a PBX user is conveyed to the caller. This allows the caller to call back the PBX user directly.

For example a user calling a PBX desk (e.g., +49 6151 83 0000 for Deutsche Telekom in Darmstadt) will be forwarded to an Extension (e.g., +49 6151 83 5940 for Roland Jesske). It is required that the caller gets the latter identity, which is allocated to the callee.

[REQ-TIP-2]
The identity mentioned in REQ-TIP-1 has to be backwards compatible with the SIP identity described within RFC 3325 [4].

[REQ-AoC-1]
In order to support the AoC simulation service, a mechanism is needed whereby the caller can invoke the AoC simulation service in a given communication. This invocation is sent when initializing the communication.

[REQ-AoC-2]
As part of the AoC simulation service a mechanism is needed to asynchronously transport the charging information. This information will be displayed to the requesting user.

Asynchronously transport means that the information shall be transported at any time during and after (e.g., within a certain period of time) the communication, but within the session context, when it is needed.

[REQ-CCBS-1]
In order to assure that end to end functionality of the CCBS service is possible, it is required that the UAC gets knowledge of the availability of the CCBS service at the UAS or the PSTN/ISDN terminal on a communication by communication basis.

[REQ-CCBS-2]
The entity providing the CCBS service needs to know the change of the status of a UAS (e.g., a transition from busy to idle) and it should have the ability to recognize if the UAS is able to provide such indication.

[REQ-CCBS-3]
The CCBS simulation service should be able to handle queues and arbitrate multiple simultaneous CCBS requests according to a locally defined policy (e.g., first in first out).

[REQ-CCBS-4]
It should be also possible for CCBS request initiators to suspend, resume and cancel their pending CCBS requests.

[REQ-CCNR-1]
In order to assure that end to end functionality of the CCNR service is possible, it is required that the UAC gets knowledge of the availability of the CCNR service at the UAS or the PSTN/ISDN terminal on a communication by communication basis.

[REQ-CCNR-2]
The entity providing the CCNR service needs to know the change of the status of a UAS (e.g., a transition from idle to another state) and it should have the ability to recognize if the UAS is able to provide such indication.

[REQ-CCNR-3]
The CCNR simulation service should be able to handle queues and arbitrate multiple simultaneous CCNR requests according to a locally defined policy (e.g., first in first out).

[REQ-CCNR-4]
It should be also possible for CCNR request initiators to suspend, resume and cancel their pending CCNR requests.

[REQ-MCID-1]
In order to support the MCID simulation service there must be a mechanism whereby a user can provide an indication that an incoming request or session is considered to be malicious. The user can provide this indication at the start, during or within a certain time after a session or request.

Note: The requirement reads about the ability of the callee to provide an indication of malicious call, but there is no requirement to supply the caller’s identity to the called.

[REQ-MCID-2]
For interoperability reasons, the MCID simulation service logic needs to get the knowledge that, even if the originator identity is missing in the signalling, it can available upon request. This is due to, e.g., interworking with the PSTN network, where, in some cases, the originator’s identity is only available upon explicit request. The information can be received asynchronously in a timeframe of 1-30 seconds even after the session has been closed.

[REQ-CW-1]
To implement the CW simulation service, ETSI envisages the usage of an application server that detects some busy conditions on behalf of the user. To support this scenario a mechanism is required to inform the callee that a communication is waiting.

[REQ-CW-2]
It must be possible for CW to inform the caller that an application server is holding the communication until the callee is available.

[REQ-CDIV-1]
To support the CDIV simulation service it is required that the caller is informed if a communication diversion takes place.

[REQ-CDIV-2]
To support the CDIV simulation service a mechanism that shows or restrict the indication of the diverting user is required.

[REQ-CDIV-3]
To support the CDIV simulation service it is required that the reason of the redirection is delivered to the caller and callee.

[REQ-CDIV-4]
For interoperability reasons and service compatibility with the CDIV simulation service, it is required that the history of the communication is sent in forward and backward directions to the caller and callee, respectively.

[REQ-CAT-1]
For service applicability to special groups and interoperability with the PSTN/ISDN an indication of the originating party category is needed. This is needed due to the fact that some services apply a special behavior to special user groups (e.g., like Pay-Phones).

[REQ-CAT-2]
The originating party category referred to in REQ-CAT-1 must be inserted by a trusted entity.

4. Security Considerations

The requirements in this document are intended to result in a mechanism with general applicability for ETSI NGN and are generally not applicable to the public Internet.

Use of these mechanisms in any other context has serious security shortcomings, namely there is absolutely no guarantee that the information has not been modified, or was even correct in the first place.

5. IANA Considerations

This document does not have any implications for IANA.

References

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

[2] Barnes, M. "An Extension to the Session Initiation Protocol for Request History Information", draft-ietf-sip-history-info-06.txt, January 17, 2005.

[3] Elwell, J.; "SIP Reason header extension for indicating redirection reasons", draft-elwell-sipping-redirection-reason-02.txt, June 2005.

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

[5] ETSI EN 300 089 (V3.1.1): "Integrated Services Digital Network (ISDN); Calling Line Identification Presentation (CLIP) supplementary service; Service description".

[6] ETSI EN 300 090 (V1.2.1): "Integrated Services Digital Network (ISDN); Calling Line Identification Restriction (CLIR) supplementary service; Service description".

[7] ETSI EN 300 200 (Edition 1): "Integrated Services Digital Network (ISDN); Call Forwarding Unconditional (CFU) supplementary service; Service description".

[8] ETSI EN 300 199 (V1.2.1): "Integrated Services Digital Network (ISDN); Call Forwarding Busy (CFB) supplementary service; Service description".

[9] ETIS EN 300 201 (V1.2.1): "Integrated Services Digital Network (ISDN); Call Forwarding No Reply (CFNR) supplementary service; Service description".

[10] ETSI ETS 300 202 (Edition 1): "Integrated Services Digital Network (ISDN); Call Deflection (CD) supplementary service; Service description".

[11] ETSI ETS 300 056 (Edition 1): "Integrated Services Digital Network (ISDN); Call Waiting (CW) supplementary service; Service description".

[12] ETSI ETS 300 139 (Edition 1): "Integrated Services Digital Network (ISDN); Call Hold (HOLD) supplementary service; Service description".

[13] ETSI ETS 300 128 (Edition 1): "Integrated Services Digital Network (ISDN); Malicious Call Identification (MCID) supplementary service; Service description".

[14] ETSI ETS 300 183 (Edition 1): "Integrated Services Digital Network (ISDN); Conference call, add-on (CONF) supplementary service; Service description".

[15] ETSI EN 300 357 (V1.2.1): "Integrated Services Digital Network (ISDN); Completion of Calls to Busy Subscriber (CCBS) supplementary service; Service description".

[16] ETSI ETS 300 178 (Edition 1): "Integrated Services Digital Network (ISDN); Advice of Charge: charging information at call set-up time (AOC-S) supplementary service; Service description".

[17] ETSI ETS 300 179 (Edition 1): "Integrated Services Digital Network (ISDN); Advice of Charge: charging information during the call (AOC-D) supplementary service; Service description".

[18] ETSI ETS 300 180 (Edition 1): "Integrated Services Digital Network (ISDN); Advice of Charge: charging information at the end of the call (AOC-E) supplementary service; Service description".

[19] ETSI EN 300 650 (V1.2.1): "Integrated Services Digital Network (ISDN); Message Waiting Indication (MWI) supplementary service; Service description".

[20] ETSI EN 300 367 (V1.2.1): "Integrated Services Digital Network (ISDN); Explicit Call Transfer (ECT) supplementary service; Service description".

[21] ETSI EN 301 798 (V1.1.1): "Services and Protocols for Advanced Networks (SPAN); Anonymous Call Rejection (ACR) Supplementary Service; Service description".

[22] ETSI ETS 300 520: "European digital cellular telecommunications system (Phase 2); Call Barring (CB) supplementary services; Stage 1 (GSM 02.88)".

[23] ETSI EN 300 094: "Integrated Services Digital Network (ISDN); Connected Line Identification Presentation (COLP) supplementary service; Service description".

[24] ETSI ETS 300 095: "Integrated Services Digital Network (ISDN); Connected Line Identification Restriction (COLR) supplementary service; Service description".

[25] ETSI EN 301 134: "Integrated Services Digital Network (ISDN); Completion of Calls on No Reply (CCNR) supplementary service; Service description".

[26] 3GPP TS 24.229: "IP Multimedia Call Control Protocol based on SIP and SDP".

Contributors

Keith Drage
GSM OPTIMUS HOUSE
SN5 6PP SWINDON
United Kingdom
Phone: +44 1793 897312
Email: drage@lucent.com

Sebastien Garcin
France Telecom
38-40, Rue du General Leclerc
92130 Issy Les Moulineaux
France

Acknowledgments

The authors would like to thank Anna Martinez people of ETSI TISPAN WG3 for their comments.

Author's Addresses

Roland Jesske
Deutsche Telekom
Am Kavalleriesand 3
64307 Darmstadt
Germany
Phone: +496151835940
Email: r.jesske@t-com.net

Denis Alexeitsev
Deutsche Telekom
Am Kavalleriesand 3
64307 Darmstadt
Germany

Phone: +496151832130
Email: d.alexeitsev@t-com.net

Miguel A. Garcia-Martin
Nokia
P.O. Box 407
NOKIA GROUP, FIN 00045
Finland

EMail: miguel.an.garcia@nokia.com

Full Copyright Statement

this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

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

Sipping
Internet-Draft
Expires: January 16, 2006

P. Kyzivat
Cisco Systems, Inc.

July 15, 2005

Reg Event Package Extension for GRUUs
draft-kyzivat-sipping-gruu-reg-event-03

Status of this Memo

Copyright Notice

Abstract

This draft defines an extension to RFC 3680 [1] for representing the GRUU associated with a Contact.

Table of Contents

1. Introduction

The addition of GRUU (Globally Routable Unique URI) support to the REGISTER message, defined in [2], introduces another element of state to the registrar. Subscribers to the registration event package [1] will sometimes have need for the new state.

For example, the Welcome Notices example in [1] will only operate correctly if the contact address in the reg event notification is reachable by the sender of the welcome notice. When the registering device is using the gruu extension, it is likely that the registered contact address will not be globally addressable, and the gruu should be used as the target address for the MESSAGE.

Another case where this feature may be helpful is within the 3GPP IP Multimedia Subsystem (IMS). IMS employs a technique where a REGISTER of a contact address to one Address of Record (AOR) causes the implicit registration of the same contact to other associated AORs. If a GRUU is requested and obtained as part of the registration request, then additional GRUUs will also be needed for the implicit registrations. While assigning the additional GRUUs is straightforward, informing the registering UA of them is not. In IMS, UAs typically subscribe to the 'reg' event, and subscriptions to the 'reg' event for an AOR result in notifications containing registration state for all the associated AORs. The proposed extension provides a way to easily deliver the GRUUs for the associated AORs.

The reg event package has provision for including extension elements within the <contact> element. This document defines a new element that may be used in that context to deliver the GRUU corresponding to the contact.

2. Description

A new element (<gruu>) is defined which contains a GRUU.

This optional element is included within the body of a NOTIFY for the "reg" event package when a GRUU is associated with the contact. The contact URI and the GRUU are then both available to the watcher.

3. Notifier Processing of SUBSCRIBE Requests

Unchanged from RFC 3680 [1].

4. Notifier Generation of NOTIFY Requests

A notifier for the "reg" event package [1] SHOULD include the <gruu>

element when a contact has an Instance ID and a GRUU is associated with the combination of the AOR and the Instance ID. When present, the <gruu> element MUST be be positioned as an instance of the <any> element within the <contact> element.

5. Subscriber Processing of NOTIFY Requests

When a subscriber receives a "reg" event notification [1] with a <contact> containing a <gruu>, it SHOULD use the gruu in preference to the corresponding <uri> when sending SIP requests to the contact.

Subscribers that are unaware of this extension will, as required by [1], ignore the <gruu> element.

6. Sample reginfo Document

The following is an example registration information document including the new element:

```
<?xml version="1.0"?>
    <reginfo xmlns="urn:ietf:params:xml:ns:reginfo"
        xmlns:gr="urn:ietf:params:xml:ns:gruu"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        version="0" state="full">
      <registration aor="sip:user@example.com" id="as9"
          state="active">
        <contact id="76" state="active" event="registered"
          duration-registered="7322"
          q="0.8">
            <uri>sip:user@192.0.2.1</uri>
            <unknown-param name="+sip.instance">
              "<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
            </unknown-param>
            <gr:gruu>
              sip:user@example.com;opaque=hha9s8d-999a
            </gruu>
        </contact>
      </registration>
    </reginfo>
```

7. Examples

Note: In the following examples the SIP messages have been simplified, removing headers that are not pertinent to the example. The conventions of [7] are used to describe representation of long message lines.

7.1 Example: Welcome Notice

Consider the Welcome Notices example in [1]. When the application server receives a notification of a new registration containing the reginfo shown in Section 6 it should address messages using the contained GRUU as follows:

```
MESSAGE sip:user@example.com;opaque=hha9s8d-999a SIP/2.0
To: <sip:user@example.com>
From: "SIPland Notifier" <sip:notifier@example.com>
Content-Type: text/plain
Content-Length: ...

Welcome to SIPland!
Blah, blah, blah.
```

7.2 Example: Implicit Registration

In an 3GPP IMS setting, a UA may send a single register message, requesting assignment of a gruu, as follows:

```
REGISTER sip:example.net SIP/2.0
From: <sip:user_aor_1@example.net>;tag=5ab4
To: <sip:user_aor_1@example.net>
Contact: <sip:ua.example.com>
  ;expires=3600
  ;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
Supported: path, gruu
Content-Length: 0
```

The response reports success of the registration and returns the GRUU assigned for the combination of AOR, Instance ID, and Contact. It also indicates (via the P-Associated-URI header [5]) that there are two other associated AORs that may have been implicitly registered using the same contact. But each of those implicitly registered AORs will have had a unique GRUU assigned, and there is no way defined to report that assignment in the response.

```
SIP/2.0 200 OK
From: <sip:user_aor_1@example.net>;tag=5ab4
To: <sip:user_aor_1@example.net>;tag=373392
Path: <sip:proxy.example.net;lr>
Service-Route: <sip:proxy.example.net;lr>
Contact: <sip:ua.example.com>
  ;expires=3600
  ;+sip.instance="<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
  ;gruu="<sip:user_aor_1@example.net;opaque=hha9s8d-999a>"
P-Associated-URI: <sip:user_aor_2@example.net>,
  <sip:+358504821437@example.net;user=phone>
Content-Length: 0
```

The UA then subscribes to the 'reg' event package as follows:

```
SUBSCRIBE sip:user_aor_1@example.net SIP/2.0
From: <sip:user_aor_1@example.net>;tag=27182
To: <sip:user_aor_1@example.net>
Route: <sip:proxy.example.net;lr>
Event: reg
Expires: 3600
Accept: application/reginfo+xml
Contact: <sip:user_aor_1@example.net;opaque=hha9s8d-999a>
Content-Length: 0
```

(The successful response to the subscription is not shown.) Once the subscription is established an initial notification is sent giving registration status. In IMS deployments the response includes, in addition to the status for the requested URI, the status for the other associated URIs.

```
NOTIFY sip:user_aor_1@example.net;opaque=hha9s8d-999a SIP/2.0
From: <sip:user_aor_1@example.net>;tag=27182
To: <sip:user_aor_1@example.net>;tag=262281
Subscription-State: active;expires=3600
Event: reg
Content-Type: application/reginfo+xml
Contact: <sip:registrar.example.net>
Content-Length: (...)

<?xml version="1.0"?>
    <reginfo xmlns="urn:ietf:params:xml:ns:reginfo"
        xmlns:gr="urn:ietf:params:xml:ns:gruu"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        version="1" state="full">
      <registration aor="sip:user_aor_1@example.net" id="a7"
          state="active">
        <contact id="92" state="active" event="registered"
```

```
                duration-registered="1" expires="3599">
                  <uri>
                     sip:ua.example.com
                  </uri>
                  <unknown-param name="+sip.instance">
                     "<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
                  </unknown-param>
      <allOneLine>
                  <gr:gruu>sip:user_aor_1@example.net
      ;opaque=hha9s8d-999a</gruu>
      </allOneLine>
              </contact>
            </registration>
            <registration aor="sip:user_aor_2@example.net" id="a8"
                 state="active">
              <contact id="93" state="active" event="created"
                 duration-registered="1" expires="3599">
                  <uri>
                     sip:ua.example.com
                  </uri>
                  <unknown-param name="+sip.instance">
                     "<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
                  </unknown-param>
      <allOneLine>
                  <gr:gruu>sip:user_aor_2@example.net
      ;opaque=hha9s8d-999b</gruu>
      </allOneLine>
              </contact>
            </registration>
            <registration
                 aor="sip:+358504821437@example.net;user=phone"
                 id="a9"
                 state="active">
              <contact id="94" state="active" event="created"
                 duration-registered="1" expires="3599">
                  <uri>
                     sip:ua.example.com
                  </uri>
                  <unknown-param name="+sip.instance">
                     "<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>"
                  </unknown-param>
      <allOneLine>
                  <gr:gruu>sip:+358504821437@example.net;user=phone
      ;opaque=hha9s8d-999c</gruu>
      </allOneLine>
              </contact>
            </registration>
          </reginfo>
```

The status indicates that the associated URIs all have the same contact registered. It also includes the unique GRUU that has been assigned to each. The UA may then retain those GRUUs for use when establishing dialogs using the corresponding AORs.

8. XML Schema Definition

A gruu document is an XML document that MUST be well-formed and SHOULD be valid. Gruu documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying gruu documents. The namespace URI for elements defined for this purpose is a URN, using the namespace identifier 'ietf'. This URN is:

urn:ietf:params:xml:ns:gruu

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:gruu"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="urn:ietf:params:xml:ns:gruu">
  <xs:element name="gruu" type="xs:anyURI"/>
</xs:schema>
```

9. IANA Considerations

This document calls for IANA to register a new XML namespace URN and schema per [3].

URI: urn:ietf:params:xml:ns:gruu

Description: TBD

Registrant Contact: TBD

XML: TBD

10. Security Considerations

Security considerations for the registration event package is discussed in RFC 3680 [1], and those considerations apply here.

The addition of gruu information does not impact security negatively because the gruu is less sensitive than the contact URI itself.

11. Acknowledgements

The author would like to thank Jonathan Rosenberg for encouraging this draft.

12. References

12.1 Normative References

[1] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

[2] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-04 (work in progress), July 2005.

[3] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

12.2 Informative References

[4] 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.

[5] Garcia-Martin, M., Henrikson, E., and D. Mills, "Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3rd-Generation Partnership Project (3GPP)", RFC 3455, January 2003.

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

[7] Sparks, R., "Session Initiation Protocol Torture Test Messages", draft-ietf-sipping-torture-tests-07 (work in progress), May 2005.

Author's Address

Paul H. Kyzivat
Cisco Systems, Inc.
1414 Massachusetts Avenue
Boxborough, MA 01719
USA

Email: pkyzivat@cisco.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING WG
Internet-Draft
Expires: January 9, 2006

R. Mahy
SIP Edge

July 8, 2005

A Solution to the Heterogeneous Error Response Forking Problem (HERFP) in the Session Initiation Protocol (SIP)

draft-mahy-sipping-herfp-fix-00.txt

Status of this Memo

Copyright Notice

Abstract

The SIP protocol defines a role for proxy servers which can forward requests to multiple contacts associated with a specific resource or person. While each of these contacts is expected to send a response of some kind, responses for each branch are not necessarily sent back to the original requester. The proxy server forwards only the "best" final response back to the original request. This behavior causes a situation known as the Herterogeneous Error Response Forking Problem (HERFP) in which the original requester has no opportunity to see or

fix a variety of potentially repairable errors. This document describes a backwards compatible solution to the HERFP problem for INVITE transactions.

Table of Contents

1. Conventions

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

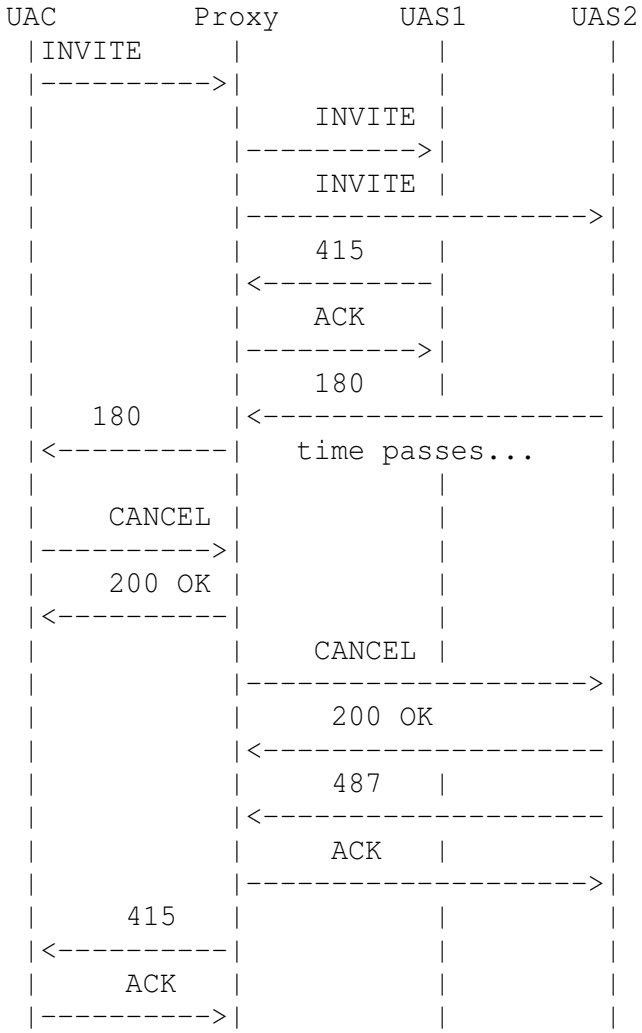
2. Background

The Session Initiation Protocol (SIP) [1] defines several logical roles, including proxy servers (which forward requests toward their destination), and User Agents (which originate and respond to requests). In addition to transparently forwarding requests, SIP proxy servers can also "fork" requests to multiple User Agent Servers (UAS) if the proxy is authoritative for the domain portion of the Request URI. When forking, proxies forward the same request to multiple contacts which typically have registered as instances of a particular user or service. A proxy can forward requests simultaneously (parallel forking), in series (serial forking), or in combination. As a request is forwarded to a set of contacts, each UAS that receives the request is expected to send a response.

When a proxy forks, it first builds a "target set", a list of User Agent Servers to whom requests will be forwarded. Once forwarding a request, the proxy collects responses from each UAS in a "response context". For INVITE requests, proxies immediately forward all provisional responses and 200-class (success) final responses back to the UAC. For other final responses (regardless of the method of the request), only a single "best" response is sent back to the UAC. The proxy has to delay sending the final response until all branches have completed. This is especially problematic for INVITE transactions, since they can theoretically pend for several minutes, after which most humans have given up attempting communication. In addition, many common SIP error responses are automatically repairable and are used extensively to allow User Agents to negotiate capabilities. These repairable errors are often completely lost if another User Agent finds the request acceptable or returns a "better" error response.

For non-INVITE requests (for example, a SUBSCRIBE request) provisional responses are practically non-existent and only one final response is sent, even if multiple branches returned a 200 response. The SIP events framework (RFC 3265 [6]) effectively deals with HERFP by using NOTIFY requests to convey the success or failure of a SUBSCRIBE request. The single response to a SUBSCRIBE might even arrive after the corresponding NOTIFY request making it effectively redundant. Consequently, this document only addresses HERFP for INVITE transactions. Sending requests other than INVITE and SUBSCRIBE in a manner which causes them to fork is contraindicated.

To illustrate a simple case of HERFP, the UAC below sends a request which includes a body format which is understood by UAS2, but not by UAS1. For example, the UAC might have used a multipart/mixed with a session description and an optional image or sound. UAC1 does not support multipart/mixed, so it returns a 415 response. The UAC can trivially repair this 415 response by resending the request with just the session description. Unfortunately, the proxy has to wait until all branches generate a final response before forwarding the best response. Since the request was acceptable to UAS2, the proxy waits for that branch to finish before it can repair the error. In many cases, the proxy will wait for a long enough amount of time that the human operating the UAC gives up and abandons the call.



3. Overview of Solution

HERFP was first described in late 2001. It has remained one of the most challenging problems remaining for the SIP protocol. To effectively address the problem, it is useful to examine the overall goals for a solution to HERFP.

- o Convey the semantics of repairable error responses directly to the sender of a (dialog-forming) INVITE request.
- o Provide an opportunity for a UAC to retry an INVITE to one branch without canceling other pending branches.
- o Do not require modification of the SIP transaction state machine.
- o Work through existing RFC 3261 compliant proxy servers.
- o Allow the forking proxy to still add or cancel branches.
- o Work consistently with unmodified User Agent Servers.

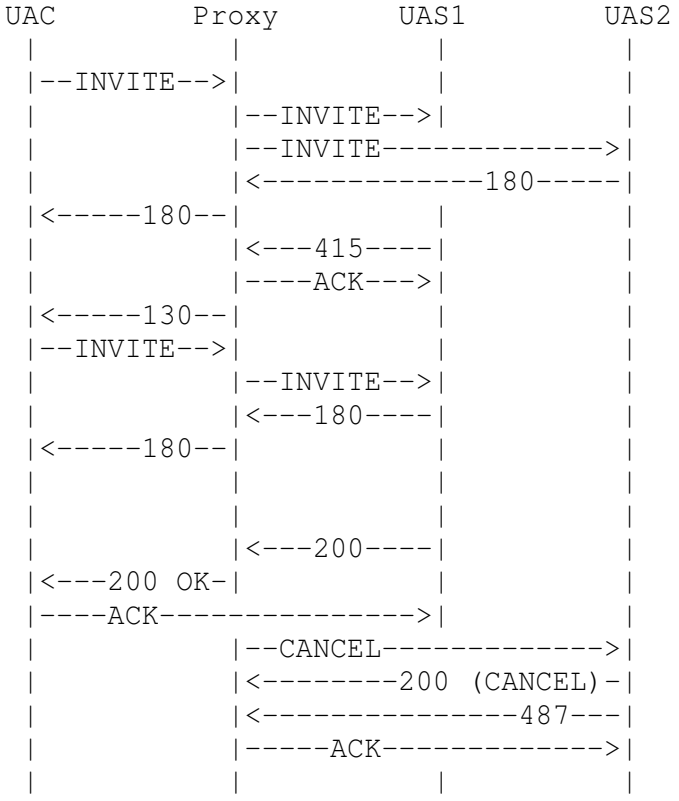
A previous attempt [7] to solve HERFP required each UAS to generate a new provisional response encapsulating the actual final response. However the entire HERFP problem stems from the fact that different UAS implementations will behave differently and frequently implement different sets of extensions. The last goal reflects that a satisfactory solution should work with unmodified User Agent Servers.

Instead of requiring new UAS behavior, this solution enlists the services of the proxy to generate a provisional response of its own (a 130 Repairable Error response) for each branch. Each 130 response encapsulates the repairable final response from one branch. The proxy acts temporarily as a UAS to send these provisional responses. The proxy generates and provides a new URI that the UAC will contact after repairing the error. This URI is similar in spirit to a Globally Unique UA URI (GRUU) [5], except that the URI refers to a specific branch of a specific target set only. Each new URI refers only to one specific failed branch, but is still associated with the list of candidate recipients of the original transaction (the target set).

A UAC which supports this extension reacts to a 130 response by sending a new INVITE request (with the same Call-ID) to the URI in the Contact header of the 130 response. This new request is generated in the same context as the original INVITE request, which is unaffected by the new request. The proxy can still try new branches in the candidate set or cancel old ones. Using this technique, the original requester can immediately fix repairable error responses.

Now consider the same example described above but employing the solution described in this document. The UAC sends a request with a multipart/mixed body. The Proxy forwards this request to UAS1 and UAS2. UAS1 sends the proxy a 415 response. The proxy generates a

URI with the appropriate properties, and generates a 130 Repairable Error response with the 415 response embedded as a message/sip body. The UAC sends a new INVITE to the URI that the proxy generated with only a session description in the body. The proxy forwards the INVITE to UAS1, but manages the forking logic as if the new request was in the original target set. When UAS1 sends a 200 OK, the proxy cancels the branch with UAS2.



4. Proxy Behavior

4.1 Handling repairable errors

A proxy which supports this extension performs the following steps when receiving a repairable error:

- Determine if the UAC supports this extension
- Determine if the proxy is awaiting pending responses to complete the response context.
- Generate a URI which identifies this specific branch
- Encapsulate the original response in a message/sip body

- o Generate a 130 Repairable Error provisional response
- o Add an Identity header or its equivalent for responses
- o Send the 130 response appropriately

To determine if the UAC supports this extension, the proxy needs to check for the presence of the "herf" option-tag in the original INVITE request (typically in a Supported header). If the UAC does not advertise support for this option, response processing continues normally. The proxy also checks the response context of the request. If there are no more branches pending in the response context of this transaction, processing continues normally. The rest of this section assumes that the UAC supports this extension, and that there are pending branches remaining in the response context.

When a proxy receives a 400-class or 500-class response other than a 503, 487, or 408, the proxy SHOULD generate a 130 Repairable Error response as a User Agent Server. If the proxy receives a 300-class response, the proxy can decide based on local policy whether to recurse, or generate a 130 Repairable Error response.

To generate a 130 response, the proxy first creates a message/sip body containing the original (3xx, 4xx, or 5xx) response. The Content-Disposition header for this for this body MUST be "signal" [or we could define a new disposition called "error"]. The proxy does not add the response to the response context for the purpose of returning the best response. The proxy generates a unique To tag for the response. The response context continues to pend until the proxy has positive knowledge that the 130 response was successfully received by the UAC (either the corresponding 130 response is acknowledged or the single-branch URI is contacted).

Next, the proxy generates a "single-branch" URI which corresponds to this branch of this target set. The hostport production of the single-branch URI MUST be identical to the hostport production from the Request URI of the original request. If the Request URI of the original request was a SIPS URI, the single-branch URI MUST be a SIPS URI as well unless the error response was a 416 Unsupported URI Scheme, in which case the proxy SHOULD generate a single-branch URI using the SIP scheme. Otherwise the construction of a single-branch URI is local policy of the proxy and is not subject to standardization.

The proxy SHOULD embed a To header in the single-branch URI that corresponds to the Identity of the branch. Typically, this identity is the same identity which was in the original request. The scheme of the embedded To URI MUST match the scheme of the single-branch URI. The hostport of the embedded To URI MUST be domain for which the proxy can provide the Identity service.

The proxy now generates a 130 Repairable Error provisional response and adds a Contact header field containing the single-branch URI (including the embedded To header), and the message/sip body containing the original response.

The proxy SHOULD add an Identity header or its equivalent used for response identity. This insures the integrity and authenticity of the 130 response and protects from tampering the linkage between the URI provided in the Contact header of the 130 response and the original request.

At this point, the proxy is ready to send the provisional response. If the original INVITE included the 100rel option-tag, the proxy sends the 130 response reliably according to the rules in RFC 3262 [2]. Whether the 130 was sent reliably or unreliably, the proxy MUST retransmit the 130 response every 60 seconds until the proxy has positive knowledge that the 130 response was successfully received by the UAC (either the corresponding 130 response is acknowledged or the single-branch URI is contacted).

Note that provisional responses in SIP can be sent reliably or unreliably. This mechanism can be used in either case. The proxy MUST support the ability to send provisional reliable responses (RFC 3262). Whether the proxy sends 130 Repairable Error responses reliably or unreliably is up to the UAC. If the UAC indicates that it supports reliable provisional responses, the proxy server sends them reliably. Otherwise the proxy sends them unreliably. In most networks the unreliable provisionals will arrive and provide the desired behavior. This represents a significant improvement over current behavior. If the unreliable provisionals do not arrive, we have not solved HERFP, but the situation is no worse than with existing implementations.

If the proxy responds reliably it MUST include an answer (if the INVITE contained an offer) or an offer (otherwise) in the 130 response. The proxy can satisfy this requirement by generating a minimal offer or answer. A minimally appropriate answer declines all media lines in the offer. A minimally appropriate offer includes no media lines. When a 130 is sent reliably, the message/sip body containing the error and the session description are placed into a multipart/mixed body in the 130 response. UACs which support this extension and provisional reliability MUST support the multipart/mixed MIME type.

4.2 Receiving subsequent requests with the single-branch property

As soon as the proxy is contacted at a single-branch URI for the first time, the proxy tries to find the appropriate branch. If the proxy cannot find the appropriate branch it MUST return a 481

response. If the proxy finds the branch, it marks the original response context for that branch as if the branch returned a 487 response. If the request is a PRACK, the proxy returns a 200 OK response to the PRACK with an appropriate RAck header. If the request is a CANCEL, the proxy returns a 200 OK response to the CANCEL and notes that this branch has been cancelled. If the request is an INVITE, the proxy generates a response context for the new request consisting of one target and forwards the INVITE to the UAS for that target.

The proxy forwards provisional response for the new response context normally. When a final response to the new request is received it is forwarded immediately since the new response context consists of only one branch. If the final response to the new INVITE request is a 200-class or 600-class response, the proxy MUST CANCEL all other pending branches which were created from or related to the original INVITE request. In other words, the proxy must find all pending branches of both the "parent" transaction and all pending "sibling" transactions. In addition, the proxy MUST invalidate all the single-branch URIs associated with the original request.

Note that for a particular branch, the proxy might receive a new INVITE request which repairs one error, but for which there are other unresolved, but repairable error responses. While this situation is currently rare, proxy server MUST NOT invalidate single-branch URIs until Timer C expires for that branch, the branch is cancelled by the UAC, or a 200-class or 600-class response has been received on a parent or sibling transaction.

5. User Agent Client Behavior

A User Agent Client which supports this extension SHOULD advertise for this extension by including the "herf" option-tag in a Supported header field value in dialog-forming INVITE requests. The UAC needs the ability to send multiple invitations in the same user interface context, for example as if the UAC tried multiple contacts from a 300-class response simultaneously.

When a User Agent which supports this extension receives a 130 Repairable Error response to an INVITE request, it performs the following steps.

- Verify the validity of the Identity headers (if present)
- Send a PRACK request if reliability was requested
- Determine if the error is repairable
- Either generate a new INVITE to repair the error, or generate a CANCEL request to acknowledge receipt of the 130 response.

The UAC SHOULD first verify that the 130 response was sent by a host which is authoritative for the domain of the original request and

that the 130 response was not tampered with en route. The UAC checks that the Identity hash verifies and that the signer of the Identity header corresponds to the hostport production from the Request URI of the original request.

If the 130 response was sent reliably, the UAC MUST send a PRACK request to the URI in the Contact header field of the 130 response.

Next the UAC determines if it can and is willing to repair the error by examining the message/sip body (which may be a MIME part inside a multipart/mixed body). UACs which support this extension and provisional reliability MUST support the multipart/mixed MIME type. The UAC MAY decide based on local policy not to repair the error or it may be unable to do so. In that case, the UAC MUST send a CANCEL request to the URI in the Contact header field of the 130 response. Note that this CANCEL only cancels a single branch.

If the UAC is willing and able to repair the error, it generates a new INVITE request using the same Call-ID, but a different from-tag. It then sends this new INVITE to the URI in the Contact header field of the 130 response. If an embedded To header is present in the Contact URI, the UAC MUST override the To header of the new INVITE to use the value provided in the Contact header.

6. User Agent Server Behavior

This document requires no new behavior by User Agent Servers. It was designed to work only if the User Agent Client and the Proxy support this extension. There is an opportunity to improve the current situation when only the UAC and one UAS cooperate. Such behavior is potentially complimentary, but out of scope of this document.

7. Security Considerations

An attacker that maliciously injects 130 responses could theoretically direct a large number of new requests towards a specific proxy. To prevent this attack, the UAC SHOULD verify that a 130 response has a valid Identity header (or its response equivalent) signed using a key from a certificate whose subjectAltName is equivalent to the hostport production from the Request URI, and that the certificate is rooted in a trusted certificate chain. The security considerations of a 130 response in this context are identical to injecting a malicious 300-class response.

A UAS that maliciously injects a 130 could theoretically downgrade the security of a dialog from SIPS to SIP. The UAC SHOULD include configurable policy to automatically repair or ignore 416 responses or to prompt the user.

A UAS that maliciously injects a 130 could selectively disable capabilities or extensions. The security considerations of such an attack are similar to injecting the corresponding 400-class response.

8. IANA Considerations

The following entries should be added to the registries for SIP option-tags and response-codes, respectively.

8.1 The "herf" option-tag

Name of option:	herf
Description:	Support for safe forking in the face of heterogeneous error responses
SIP headers defined:	none
Normative description:	This document

8.2 The "130 Repairable Error" response-code

Response Code Number: 130
Default Reason Phrase: Repairable Error

9. Acknowledgments

This idea was the result of 1) participating in discussions with Jonathan Rosenberg, Paul Kyzivat, Jon Peterson, and Cullen Jennings on the properties of URIs in conjunction with the GRUU extension; 2) thoughts I had while implementing best response matching in the repro open-source SIP proxy, 3) numerous discussions about response Identity with Jon Peterson and Cullen Jennings, 4) and a discussion with Mark Eastman about a solution to the Early Attended Transfer problem.

10. References

10.1 Normative References

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

[2] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262,

June 2002.

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

[4] Peterson, J., "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-04 (work in progress), February 2005.

10.2 Informational References

[5] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-03 (work in progress), February 2005.

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

URIs

[7] <http://scm.sipfoundry.org/rep/ietf-drafts/rohan/herf-fix/draft-rosenberg-sip-unify-00.txt>

[8] <http://scm.sipfoundry.org/rep/ietf-drafts/rohan/herfp/draft-rosenberg-sip-unify-00.txt>

Author's Address

Rohan Mahy
SIP Edge

Email: rohan@ekabal.com

Appendix A. Historical Context

The Heterogeneous Error Response Forking Problem (HERFP) was described in various SIP working group mailing list threads in late 2001 and then described more formally in a long expired Internet Draft (draft-rosenberg-sip-unify-00.txt [8]) in January of 2002. The problem description from the draft is copied here.

A.1 HERFP Problem Description

HERFP, as it is called, is, in our opinion, the most complex remaining problem with the SIP specification.

It relates to the rules for response processing at a forking proxy.

A proxy never forwards more than one error response back to the [User Agent Client (UAC)]. This is needed to prevent response implosion, but more importantly, to support services at proxies. A forking proxy only returns an error response upstream if all forked requests generate an error response. However, a 200 OK [to an INVITE] is always forwarded upstream immediately.

The problem is that if a request forks, and one UAS generates an error because the INVITE is not acceptable for some reason (no credentials, bad , bad body type, unsupported extension, etc.), that response is held at the forking proxy until the other forks respond. Of course, another branch may find the request acceptable, and therefore never generate an error response. The effect is to cancel out the benefits of forking.

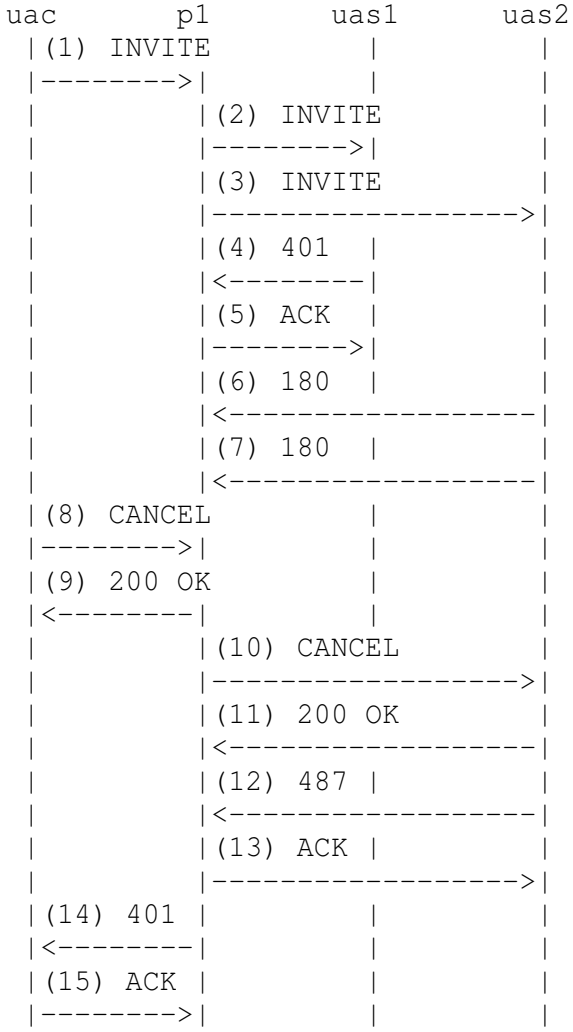
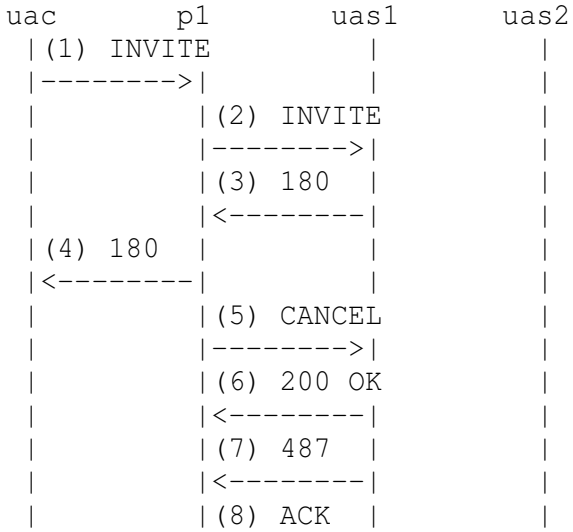


Figure 2: Basic HERFP Case

Figure 2 shows the simplest form of the problem. In this flow, the UAC sends an INVITE to proxy P1, which forks to UAS1 and UAS2. UAS1 might be a cell phone, and UAS2 a business phone. UAS1 rejects with a 401, and so never rings. However, UAS2 does not require credentials (or the request already had them), and therefore it rings. However, the user is not at their business phone, although they are available at the cell phone. After ringing for 20s, the caller gives up, and therefore sends CANCEL. This stops UAS2 from ringing, and results in the proxy forwarding the now-old 401 to the UAC. The UAC is not likely to retry, since the user just hung up. Thus, no call is

Another HERFP case is shown in Figure 3. This is a case of sequential forking for a call forwarding service. The UAC calls a user, and the proxy first forks the call to UAS1. The user is not there, so the phone rings for 5s, and is then cancelled by the proxy, which forks to UAS2. UAS2 challenges, resulting in a 401 being returned to the UAC. The UAC tries again, which causes re-invocation of the call forwarding service! UAC1 rings once more for another 5s, and then finally the call is connected to UAS2. Interestingly, if the first UAC doesn't challenge but the others do, and there are N phones tried before completion, the first phone will ring N times! A user standing by UAS1 but electing to not answer will probably view it as a prank or malicious call.

The problem is that information needs to be propagated back to the UAC immediately, and the UAC needs to resubmit it, but the resubmission should not affect services somehow, e.g., should not re-invoke them as above.



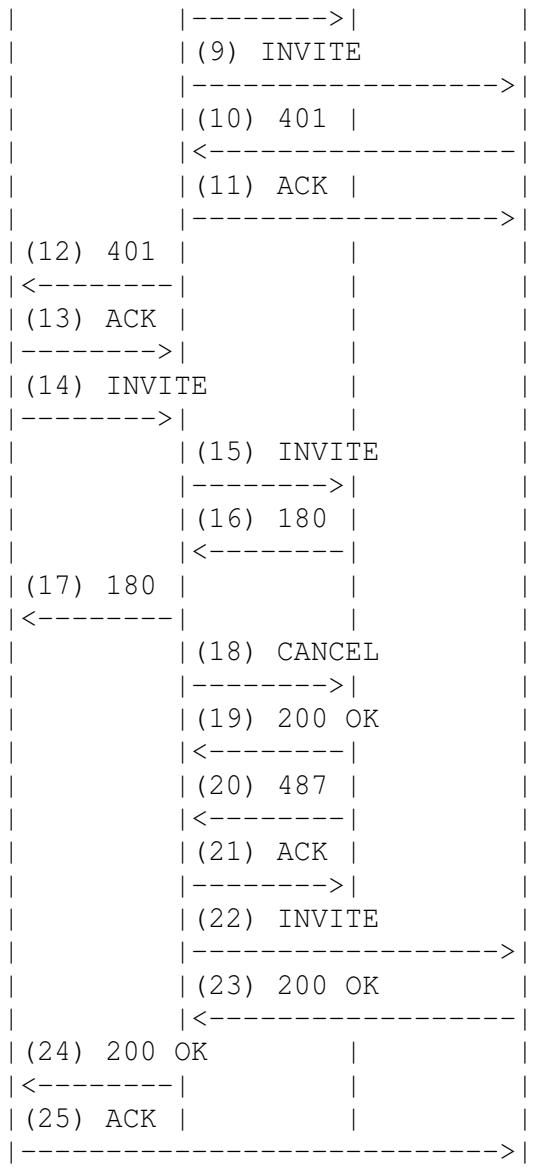


Figure 3: Forking HERFP Case

A.2 The 155 Response

One aspect of our proposal is that a UAS can send a 155 response, instead of a final response, when supported by the UAC, to support services that are complicated by HERFP. The COMET can then be used to provide whatever information is requested by the error response. The COMET would operate just like the a re-INVITE would operate if the actual final response had been sent.

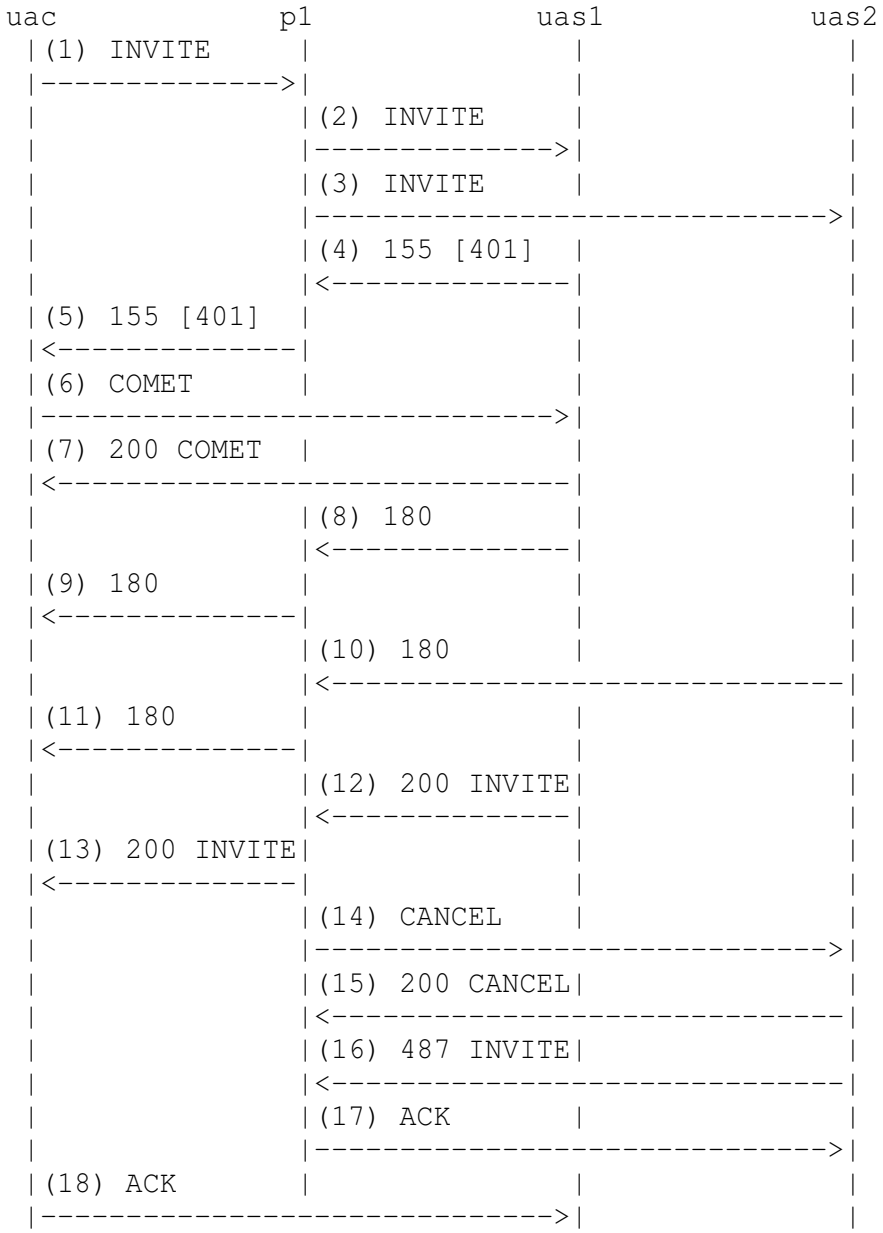


Figure 13: HERFP fix, scenario 1

To show the effectiveness of our proposal, we once again consider the two scenarios of Section 2.6. The call flow for the first case is now shown in Figure 13, except now this time with our proposed solution. For brevity, we ignore the PRACKs for the provisional responses. As before, the caller sends an INVITE, and the proxy forks it. This time, the proxy inserts a Require header in the

request that indicates services are being offered based on dialog state, and so the UAS should send provisionals instead of finals. UAS1 challenges for credentials, but this time, it sends a 155 response that contains the challenge in the WWW-Authenticate header (message 4). The proxy passes this upstream to the UAC. The UAC formulates the response, and places it in an Authorization header in the COMET (message 6). This goes directly to the UAS (the proxy did not record-route). Since the credentials are valid, the UAS proceeds with the session and rings (message 8), which is passed to the UAC. UAS2 does not challenge, and generates an immediate 180, which is passed to the UAC as well. In this example, as discussed in Section 2.6, the user is at UAS1, the call is answered there, resulting in a 200 OK (message 12). The proxy cancels the branch towards UAS2, and the call completes successfully this time!

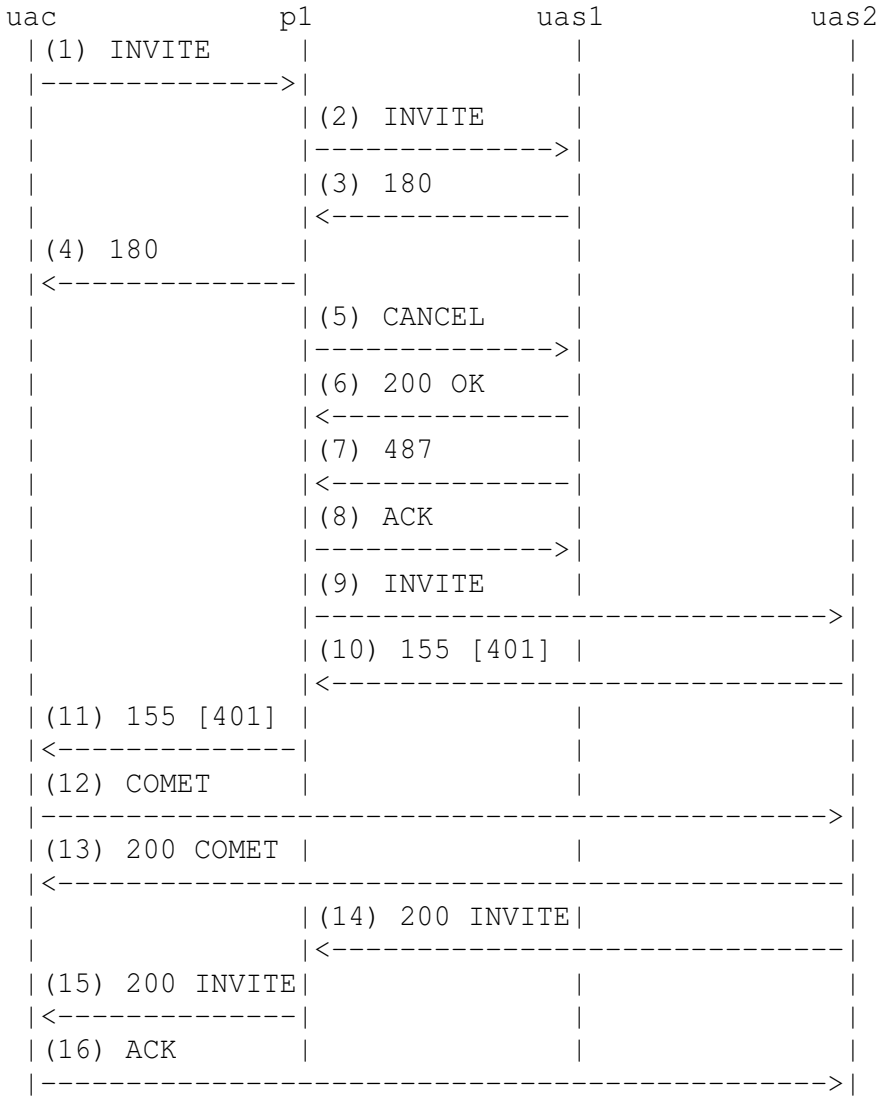


Figure 14: HERFP fix, scenario 2

Consider the second example of Section 2.6. The flow for this example, this time with our proposed solution, is shown in Figure 14. The initial flow proceeds as in Figure 3. UAS1 is rung, and there is no answer, resulting in a cancellation and an attempt to ring UAS2. UAS2 wishes to challenge. However, this time, it issues a 155 that otherwise looks like a 401, which contains a WWW-Authenticate header with the challenge. This response is passed to the proxy and forwarded to the UAC (once again, PRACK requests are not shown). The UAC generates credentials for the challenge, and sends a COMET with the response to the challenge. This is sent directly to UAS2, since the proxy did not record-route. The credentials are accepted,

causing the phone to ring. The user is there, so they pick up, generating a 200 OK, which is passed to the UAC, which sends an ACK to complete the call. Once again, a successful call setup!

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

Network Working Group
Internet-Draft
Expires: January 13, 2006

A. Niemi
Nokia Research Center

July 12, 2005

Problems with the Session Initiation Protocol (SIP) Events Framework
draft-niemi-sipping-subnot-issues-00

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) events framework enables receiving asynchronous notification of events related to SIP systems. This framework defines the procedures for creating, refreshing and terminating subscriptions, as well as fetching and periodic polling of resource state. These procedures have a serious deficiency in that they do not allow state to persist over a subscription refresh, or between two consecutive polls. Another related but different problem relates to the relative intolerance of the framework to interferences in networking connectivity of subscribers in long-

lasting subscriptions. This document explains the problems in more detail and discusses possible solutions.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) events framework provides an extensible facility for requesting notification of certain events from other SIP nodes. This framework includes procedures for creating, refreshing and terminating of subscriptions, as well as the possibility to periodically fetch or poll the event resource.

Several instantiations of this framework, called event packages have been defined, e.g., for presence [4], message waiting indications [5] and registrations [6].

In certain conditions, the overhead induced by having to maintain subscriptions becomes prohibitively high for subscribers. Polling of resource state behaves in a similarly suboptimal way in cases where the state has not changed since the previous poll occurred. In general, the problem lies in the inability to persist state across a subscription refresh, or two consecutive fetches.

Another related but different problem lies in with the inability of the notifier to fail soft in case a temporary network outage that leads to a NOTIFY request timing out, causing the subscription to terminate. Subscribers may be unaware of this until they refresh, which might be even days later.

This memo discusses these problems in more detail, and ventures into solution space by providing a possible ways to reduce the impact of these problems.

1.1 Document Conventions

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

2. Motivations and Background

2.1 Overview

A SUBSCRIBE request creates a subscription with a finite lifetime. This lifetime is negotiated using the Expires header field, and unless the subscription is refreshed by the subscriber before the expiration is met, the soft state is cleared. The frequency of these subscription refreshes depends on the event package, and can range from minutes to hours to months in some cases.

Changes in connectivity represent another impetus for a subscriber

re-subscribing. If the subscriber's point of attachment to the Internet changes, e.g., due to dynamic address allocation, the subscriber needs to re-subscribe in order to update the dialog endpoint, which is carried in the Contact header field.

2.2 Problem: High Subscription Maintenance Costs

The SIP events framework does not include different methods for initial sibscriptions, subscription refreshes and fetches inside and outside of the SIP dialog. Instead, the SUBSCRIBE method is overloaded to perform all of these actions, and the notifier behavior is identical in each of them; each SUBSCRIBE request generates a NOTIFY request containing the latest resource state. This inability to persist state across a SUBSCRIBE request results in substantial overhead in maintaining subscriptions. This materializes in the form of increased network traffic and unnecessary processing overhead for both the subscriber and the notifier.

There are certain conditions that aggravate the problem. Such conditions usually entail such things as:

- Large entity bodies in the payloads of notifications
- High rate of subscription refreshes
- Relatively low rate of actual notifications triggered by state changes

Some of the same problems affect fetching and polling of event state as well. Regarding polling, if we look at the performance of Hypertext Transfer Protocol (HTTP) [7] in similar scenarios, it performs substantially better when resources are tagged with an entity-tag, and each GET is a conditional one using the "If-None-Match" header field. If the resource has not changed between successive polls, an error response is returned indicating this fact, and the resource is not transmitted again.

The SIP PUBLISH [2] method also contains a similar feature, where a refresh of a publication is done by reference to its assigned entity-tag, instead of retransmitting the event state each time the publication expiration is extended.

2.3 Problem: Low Tolerance to Connectivity Interferences in Long-lasting Subscriptions

Another related but separate problem arises from long-lasting subscriptions where during the subscription lifetime, the subscriber experiences intermittent connectivity. The problem is that if a

NOTIFY happens to time-out because of such temporary problems in connectivity, the subscription is terminated, but the subscriber has really no way of finding this out. The subscriber will only find out when its time to refresh the subscription upon which it will receive a 481 error response, and have to re-subscribe. In other words, subscriptions have a very low tolerance for networking interference, i.e., the notifier does not fail soft.

This problem manifests itself as a temporary zombie subscription, which can result in poor user experience. The longer the subscription expiration, the longer time it takes for the subscriber to notice this zombie state, and the poorer the user experience becomes. The problem is aggravated with event packages that recommend long subscription expirations, e.g., the certificate event package [8]

2.4 Requirements

As a summary, here is a short list of required functionality to solve the presented issues:

REQ1: It must be possible to suppress the NOTIFY request (and the event body therein) triggered by a subscription refresh, if the subscriber already has possession of the latest event state of the resource

REQ2: It must be possible to suppress the NOTIFY request (and the event body therein) triggered by a fetch, if the subscriber already has possession of the latest event state

REQ3: It must be possible for the notifier to fail soft in case temporary interferences in the subscriber's connectivity. In other words, the notifier must tolerate notification time outs without severing the subscription, especially in long-lasting subscriptions.

3. Description of Potential Solutions

This section lists some possible solutions to the problem. This text is only meant as a high-level overview.

3.1 Entity-tags and Conditional Requests

3.1.1 Overview

This potential solution entails replicating similar features from HTTP, namely entity-tags and conditional requests. Some existing

header field and response code definitions can be reused from the PUBLISH [2] specification.

3.1.2 Detailed Description

Each initial SUBSCRIBE request would be exactly as currently defined. However, each NOTIFY request would contain an entity-tag in a SIP-ETag header field. Each subsequent SUBSCRIBE request would include a SIP-If-None-Match header field containing the entity-tag received in the previous NOTIFY request. This header makes the SUBSCRIBE request conditional -- the request will only progress if the condition is met. In case the entity-tag has not changed, the condition is not met, and the notifier responds with a 412 (Conditional Request Failed) response.

The fact that the condition fails, also means that the NOTIFY request is suppressed and the subscription continues as before.

OPEN ISSUE: To make this work, the SUBSCRIBE has to partially succeed, i.e., the subscription expiry needs to be refreshed, even though the NOTIFY is suppressed. It isn't entirely clear if this is allowed with a 4xx response. Do we need a new 2xx response code?

In case the entity-tag has changed, the notifier behaves normally, and the SUBSCRIBE triggers a NOTIFY request carrying the latest resource state.

The advantages of this solution are clear:

- It allows resource state to persist over a subscription refresh. I.e., a subscription refresh due to a changed IP address, or extension of the expiry time no longer triggers a notification carrying full event state.

- It allows resource state to persist accross two consecutive fetches. A fetch would not trigger a NOTIFY if the resource state had not changed (i.e., its entity tag had not changed) since the previous fetch.

  OPEN ISSUE: Another option to maintaining subscriptions with little or no overhead is to define an alternative to SUBSCRIBE that installs a hard-state subscription at the notifier.

3.1.3 Backwards Compatibility

The proposed solution is backwards compatible with SIP events [3] in

that a notifier supporting this mechanism will insert a SIP entity-tag in its NOTIFY requests, and a subscriber that understands this mechanism will know how to use them in creating a conditional request.

Unaware subscribers will simply ignore the entity-tag, make unconditional requests and get the usual defined behavior from the notifier.

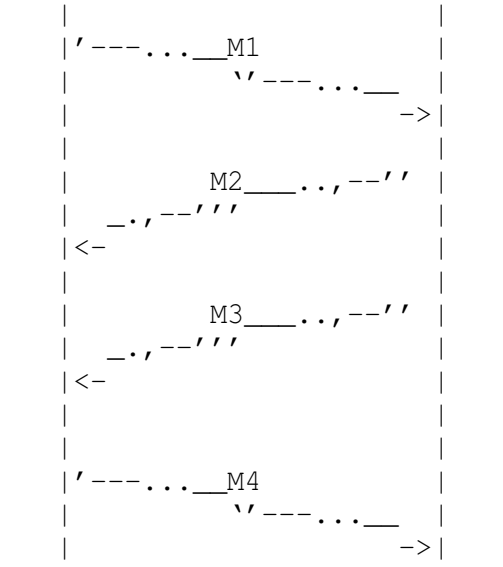
As a hint to the notifier, the subscriber could also use the Supported header field to advertize support for this feature, for example, like this:

```
Supported: etags
```

3.1.4 Examples

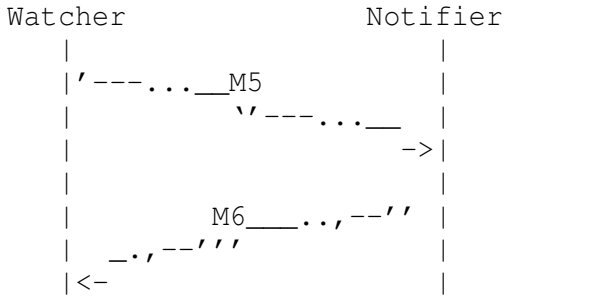
Below is an example message flow that utilizes conditional SUBSCRIBE requests and entity-tags.

Initial subscription, at t=0:



M1: SUBSCRIBE, no entity-tag, Expires: 3600. M2: 200 OK. M3: NOTIFY, SIP-ETag: 0001. M4: 200 OK, Expires: 3600

Subscription refresh, at t=3000:



M5: SUBSCRIBE, If-None-Match: 0001, Expires:3600. M6: 412 Conditional Request Failed, Expires: 3600.

3.2 Rules for Terminating a Subscription

To allow a notifier to fail soft requires changes to the notifier behavior defined in the SIP events framework [3].

Currently, the notifier is instructed to terminate the subscription ("MUST" strength) in case a NOTIFY request times out. Instead, the notifier should be allowed to keep the subscription alive.

> OPEN ISSUE: Perhaps the notifier could install such subscriptions into "probation" state, keep sending the notifications. For example, it could be defined such that those NOTIFYs that are in Subscription-State: "probation", only NULL bodies are sent, and the subscriber needs to refresh in order to lift the state back to "active" and get the actual event state delivered to it.

4. Conclusions

In this memo, we describe the problem of high costs in maintaining SIP event subscriptions, and specifically the inability to persist state accross subscription refreshes or consequtive fetches in the SIP events framework. A related problem that deals with the inability to tolerate temporary connectivity problems in long-lasting subscriptions is also presented.

The proposal is to acknowledge the problems exist and take the proposed solutions as the baseline towards fixing the problems.

5. IANA Considerations

This document includes no actions for IANA at this time.

6. Security Considerations

This document includes no security considerations at this time.

7. References

7.1 Normative References

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

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

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

7.2 Informative References

[4] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[5] Mahy, R., "A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP)", RFC 3842, August 2004.

[6] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

[7] 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.

[8] Jennings, C. and J. Peterson, "Certificate Management Service for The Session Initiation Protocol (SIP)", draft-ietf-sipping-certs-01 (work in progress), February 2005.

Author's Address

Aki Niemi
Nokia Research Center
P.O. Box 407
NOKIA GROUP, FIN 00045
Finland

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

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING
Internet-Draft
Expires: October 21, 2005

D. Petrie
SIPez LLC
S. Lawrence
Pingtel Corp.
M. Dolly
AT&T Labs
V. Hilt
Bell Labs/Lucent Technologies

April 19, 2005

A Schema and Guidelines for Defining Session Initiation Protocol User Agent Profile Data Sets

draft-petrie-sipping-profile-datasets-02.txt

Status of this Memo

Copyright Notice

Abstract

This document defines the requirements and a format for SIP user agent profile data. An overall schema is specified for the

definition of profile data sets. The schema also provides for expressing constraints for how multiple sources of profile data are to be combined. This document provides a guide to considerations, policies and syntax for defining data sets to be included in profile data. It also explores some specific data sets to test the requirements, assumptions and syntax.

Table of Contents

1. Motivation

Today all SIP user agent implementers use proprietary means of expressing and delivering user, device, and local network profile information to the user agent. The SIP User Agent Profile Delivery Framework [I-D.ietf-sipping-config-framework] specifies how SIP user agents locate and retrieve profile data specific to the user, the device, and the local network. It is important for SIP User Agents to be able to obtain and use these multiple sources of profile data in order to support a wide range of applications without undue complexity.

The SIP User Agent Profile Delivery Framework does not define a format for the actual profile data. This document proposes the requirements, a high level schema for, and guide to how these data sets can be defined. The goal is to enable any SIP user agent to obtain profile data and be functional in a new environment independent of the implementation or model of user agent. The nature of having profile data from four potential sources requires the definition of policies on how to apply the data in an interoperable way across implementations which may have widely varying capabilities.

The ultimate objective of the framework described in the SIP User Agent Profile Delivery Framework and this document is to provide a start up experience similar to that of users of an analog telephone. From the point of view of a user, you just plug in an analog telephone and it works (assuming that you have made the right arrangements with your local phone company). There is no end user setup required to make an analog phone work, at least in a basic sense. So the objective here is to be able to take a new SIP user agent out of the box, plug it in or install the software and have it get its profiles without human intervention other than security measures. This is necessary for cost effective deployment of large numbers of user agents. All user agents do not provide telephone capabilities, but the user set up experience goal is applicable to most of the range of user agent capabilities.

2. Introduction

2.1 Requirements Terminology

Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in RFC 2119[RFC2119].

2.2 Profile Data Terminology

property - a named configurable characteristic of a user agent. A given property has a well-defined range of possible values. A given property may be defined to have a range of values, allow for simultaneous use of many values (as in a list of allowed possibilities), or a set of related values that collectively form a single profile information item.

setting - the binding of a specific value or set of values to a given property.

profile - a collection of settings to be applied for a specific user, device, or local network.

device - SIP user agent, either software or hardware appliance. This is a logical concept, as there may be no physical dedicated device or it may be part of an assembly of devices. In this document, the terms "user agent" and "device" are interchangeable.

user profile - the profile that applies to a specific user. This is best illustrated by the "hotelling" use case - a user has an association for some period of time with a particular device. The user profile is that set of profile data the user wants to associate with that device (e.g. ringtones used when someone calls them, the user's shortcuts).

device profile - data profile that applies to a specific device. In the "hotelling" use case, this is the data that is bound to the device itself independent of the user. It relates to specific capabilities of the device and/or preferences of the owner of the device.

local network profile - data that applies to the user agent in the context of the local network. This is best illustrated by roaming applications; a new device appears in the local network (or a device appears in a new network, depending on the point of view). The local network profile includes settings and perhaps policies that allow the user agent to function in the local network (e.g. how to traverse NAT or firewall, bandwidth constraints).

data set - a collection of properties.

working profile - the set of property values actually set in a SIP User Agent as a result of merging the profiles from all sources; the actual effective profile for the user agent .

merging - the operation of resolving overlapping settings from multiple profiles. Overlap occurs when the same property occurs in multiple profiles (e.g. device, user, application, local network).

2.3 Overview

In this document requirements are specified for containing and expressing profile data for use on SIP user agents. Though much of this can be considered independent of SIP there is one primary

requirement that is not well satisfied through more generic profile data mechanisms. SIP User Agent set up requires the agent to merge settings, which may overlap, from potentially four different sources (see [I-D.ietf-sipping-config-framework]); each source must not only be able to provide profile information, but also express policies regarding how the profile settings may be combined with that from other sources.

A schema and syntax is defined in this document to specify properties that may be aggregated to construct profiles. The general design philosophy is that many small data sets provide flexibility to the implementer to support the aggregated set that best matches the capability of the user agent. The actual properties are not defined in this document (see [I-D.ietf-sipping-session-indep-policy] and [reference: Core SIP Dataset]). However, some examples are explored here to illustrate the proposed mechanisms and to validate the requirements.

This document defines a set of considerations, syntax and policies that must be specified when defining data sets. These are to help authors of data set specifications to define data sets that will work in the overall schema defined in this document. The actual specification of these data sets is outside the scope of this document.

3. Design Considerations

The following section defines some of the design considerations that were taken into account when defining the schema, syntax and policies for generating and applying profile data. Section 3.2.6 describes need for merging of the four data set soruces provided in [I-D.ietf-sipping-config-framework].

3.1 Use Cases

In the following use case scenarios the device profile is provided by the device owner/manager. The owner/manager may be a service provider, an enterprise or a user administering the device setup. The user is assumed to be the end user operating the user agent to perform SIP functions such as telephony, IM etc. In the scenarios that the user profile is provided, the user profile contains user specific properties that the end user has set directly or indirectly through an administration process. The local network profiles represent the suggested policy behavior that the local network operator would like user agents to adhere to. From a security perspective, the local network operator cannot trust the user agent to follow the local network profile policy. The local network operator must use a means external to the user agent to enforce these

policies. The local network profile is intended to express to the user agent, the policies that the user agent should follow if the user agent wants to function properly in the local network.

3.1.1 Outbound Proxy Setting

First consider the use cases for a simple user agent property: the outbound proxy. It is not likely that the user would want to influence the outbound proxy for SIP signaling. Conceptually an application might wish to use a specific outbound proxy for signaling related to that application. For this use case, assume that the only the device owner/manager or the local network operator are likely to want to set the outbound proxy property. The device profile defines an outbound proxy perhaps so that the device owner/manager can monitor all signaling. The local network operator also defines an outbound proxy because the proxy allows the SIP signaling to get through a NAT or firewall.

It seems there are few possible solutions to this conflict resolution problem:

- The simple solution is to define a policy where the local network profile overrides the device profile. In this approach the local network profile wins.
- A more flexible solution allows the profiles a means to express a strength to the property (e.g. mandatory use or allow use). In this scenario the device profile could express a default outbound proxy by expressing a "allow" use strength to the property. The local network profile could then override the default outbound property (set in the device profile) by putting a "mandatory" use strength on the property.
- One more possibility is to allow the aggregation of the outbound proxies. In this scenario SIP messages would be sent with a pre-populated route set that had two hops. First the outbound proxy set in the local network profile, then the outbound proxy set in the device profile.

The aggregation approach is closest to solving the requirements to the use case above. By aggregating the two outbound proxies, the local network provided outbound proxy allows the signaling to get out of the local network and the device profile provided outbound proxy is able to monitor all SIP signaling from the user agent.

3.1.2 Codec Settings

Use cases for the codec properties are illustrated here as they are likely one of the more complicated sets of properties with respect to merging and constraining across more than one profile. There are reasonable scenarios where requirements can be rationalized that the

device, user and local network profiles may each wish to express preferences and constrains of the codec properties. Without getting into details or syntax of the codec properties, it is assumed that codec properties will need to express a codec definition and a preference order. This is the order that these codecs will be put in SDP for codec negotiation purposes.

The following scenarios illustrate some possible combinations of sources of codec properties from the device, user and local network profiles. The scenarios identify rationale for providing codec properties in each of the profiles.

3.1.2.1 Codec Setting Not Set

In the scenario where a device has no profiles or the profiles contain no codec properties, the device will enable a default set and preference order of codecs. The default set and preference order of codecs is a implementer specific choice. In some implementations it is s subset of the codecs supported by the device.

3.1.2.2 Codec Set in Device Profile

Let us assume a scenario where user agents providing telephony capabilities are deployed. The deployment has very simple requirements such that the user agents have fixed locations and are always associated with the same user. This scenario does not need the separation between the user, device and local network profiles. A single profile would suffice. Another scenario having similar requirements is one where the user and local network profiles do not provide any codec related properties. This might be because the user does not care what codecs are used and the local network does not wish to impose any constraints on the codes used in the network. In the following use case, the device profile is the only source of codec properties.

The codecs in the device profile may differ from the set of codecs supported by the device, due to the administrator of the device profile wanting:

- To have a uniform set of codecs used across all device types
- To exclude the use of a specific codec due to performance issues/ concerns

The resulting device profile data further will constrain the list of codecs that get applied. In addition, the administrator may want to list the order of which the codecs are to be applied. In this scenario the device profile data will dictate the ordered list of codecs to be applied. The use agent will ignore codec types included in the profile that the user agent does not support.

3.1.2.3 Set in Device and User Profiles

In the following scenario users are allowed to express a preference over codecs. Users are probably not likely to express specific codes in the form of G.7XX, etc. They are likely to want to express a preference in the form of wideband, normal and low bandwidth. In the following use case, device and user profiles contain codec properties.

The user may prefer a higher quality codec to be used, if available. Thereby the user profile data may provide an ordered list of codecs to be applied. The device profile also specifies a list of codecs and a default preference order.

The merging of the data sources is as follows:

- The ordering of the codecs will be determined from the user profile data, which overrides the codec preference ordering from the device profile data.
- The set of codecs that may be applied, are the codecs listed in the user data constrained by the list of codecs from the device profile data.

The case in which none of the codecs in the resulting merged profile data sets are supported by the device, the profile data constitutes a misconfiguration between device and user profiles. It may not be possible to successfully establish a session in this case. It is suggested that the user agent provide feedback to the user indicating the misconfiguration. The user agent may also attempt to function in the network by ignoring one or more of the profile constraints.

3.1.2.4 Set in Device and Local Profiles

In another scenario the user is not allowed or does not care to express codec preferences. The owner/manager of the device defines the set of codecs which may be used on the device along with a preference ordering of codecs. There is no user profile or the user profile contains no codec properties. The local network wishes to define a policy over codec usage in the network. It is not clear there is a requirement that the local network be able to express a preference order. However the network operator is very likely to want to express a set of codecs that can or should not be used. The constraints that the local network operator wishes suggest may relate to the goal of controlling bandwidth or conveying what will work over the available WAN connection. In the following use case, device and local network profiles provide codec properties. The local network may limit the type of codecs that can be applied due to resources available.

The merging of the data sources is as follows:

- The set of codecs that may be used is the ordered list of codecs from the device profile data, further constrained by the local network profile data.

The case in which none of the codecs in the resulting merged profile data sets are supported by the device, the profile data constitutes a misconfiguration between local network and device. It may not be possible to successfully establish a session in this case. It is suggested that the user agent provide feedback to the user indicating the misconfiguration. The user agent may also attempt to function in the network by ignoring one or more of the profile constraints.

3.1.2.5 Set in Device, User and Local Profiles

In this scenario everyone has an opinion on the codecs to be used. The device owner/manager wishes to define a set of codes based upon best interoperability of known end points in the environment. The user wishes to express preferences in the codecs (e.g. prefers wideband audio). The local network wishes to constrain the codecs based upon bandwidth (e.g. a wireless network with limited local network bandwidth, a SOHO network with dialup connectivity, a small office with shared 256kbps WAN connectivity). In the following scenario, device, user and local network profiles provide codec properties.

The merging of the data sources is as follows:

- The ordering of the codecs will be determined from the user profile data, which overrides the ordering from the device profile data.
- The set of codecs that may be used are the codecs listed in the device profile data, constrained by the list of codecs from the user profile data and further constrained by the list of codecs from the local network profile data.

The case in which none of the codecs in the resulting merged profile data sets are supported by the device, the profile data constitutes a misconfiguration between device, user and local network profiles. It may not be possible to successfully establish a session in this case. It is suggested that the user agent provide feedback to the user indicating the misconfiguration. The user agent may also attempt to function in the network by ignoring one or more of the profile constraints.

3.1.2.6 Derived Requirements

1. A device will have a set of codecs supported, that may be offered. The list of codecs supported by a device may differ from the list of codecs in the device profile data. The list of codecs in the device profile data that get applied is the subset of the codecs supported by the device. Codecs listed in profiles that are not supported by the device are ignored.
2. The device profile data will have a default ordered list of codecs, which implies a preference order that may be offered.
3. The user profile data may provide an ordered list of user preferred codecs. The ordering of the codecs in the user profile data will override the ordering of the codecs in the device profile data. The user list of codecs may further constrain the list of codecs to be used.
4. The local network profile data may provide a list of codecs supported. This list will further constrain the list of codecs that may be offered.
5. The application profile data containing codec data will be ignored.
6. The profiles need the ability to express codecs that may be used and codecs that should not be used.

3.1.3 Transport Protocol Setting

This section describes use cases related to the use of the SIP transport protocol settings for a user agent. It is assumed that user agents are configurable to define what transport protocols (e.g. UDP, TCP, TLS) are to be used for the SIP signaling as well as the default order in which to attempt each of the protocol.

3.1.3.1 Setting Not Set

When none of the profiles are available or the profiles do not specify the SIP transport protocol setting, the device's default signaling transport(s) will be used.

3.1.3.2 Set in Device Profile

In the following scenario, the device profile is the only source of profile data. The signaling transports contained in the device profile may differ from the set of signaling transports supported by the device. This may be due to the administrator of the device profile wanting:

- To have a uniform use of signaling transports used across all device types.
- To mandate TLS for security reasons.
- To exclude the use of a specific signaling transport due to performance issues/concerns.

- To indicate the prefered, default order in which to attempt using each of the transport protocols.

This will result in the device profile data further constraining the list of signaling transports that could be used. The highest preference ordered signaling transport from the device profile data set will be used first.

3.1.3.3 Set in Device and User Profiles

The following scenario extends the prior case described above. SIP transport protocol properties are provided in both the device and user profiles. Consider that SIP user agents, like email agents, may want to provide the user with options to:

- Mandate that secure transport must be used. If secure transport is not possible the user does not want to use the user agent.
- Prefer secure transport. Attempt to use secure transport. If secure transport will not work, use which ever transport protocol will make communication work.

When the user mandates the use of secure signaling transports only, the user wishes to constrain the available signaling transports to TLS. When indicating a preference to secure transport, the use is specifying a preference order for the use of transport protocols where TLS is the highest priority.

Now consider the merging strategy required to accomplish the goals of this use case scenario where the device and user profiles both contain SIP transport protocol properties. The merging of the data sources is as follows:

- The set of signaling transports that are allowed to be used is constrained by the device profile data. This is further constrained by the user profile data.
- The signalling transports attempted will be those from the merged, constrained list in order of highest to lowest priority.

3.1.3.4 Set in Device and Local Profiles

In the following scenario, device and local network profile data is available. The local network may have a limited set of signaling transports that it supports due to NAT or firewall constraints.

The merging of the data sources is as follows:

- The set of signaling transports that may be used is the ordered list of signaling transports from the device profile data, further constrained by the local network profile data.

The case in which none of the local network data signaling transports

are supported by the device profile data constitutes a misconfiguration between local network and device. The device might not be able to successfully establish a session in this case.

3.1.3.5 Derived Requirements

1. A device will have a set of signaling transports that it supports (note: one can be a set), with a default signaling transport.
2. The set of signaling transports supported by a device may differ from the set of signaling transports in the device profile data. The set of signaling transports in the device profile data is an ordered list, that is a subset of the set of signaling transports supported by the device. This may be due to performance issues associated with one of the signaling transport(s).
3. The user profile data may provide a list of preferred signaling transports to be used (e.g., TLS for securing the signaling).
4. The local network profile data provides a list of signaling transports supported, and will constrain the set of signaling transports that could be used.

3.2 Requirement Descriptions

3.2.1 Implementer Extensibility

Implementers must be able to differentiate each implementation. In addition, it does not serve user agent owners and administrators well to require an orchestrated upgrade for all user agent implementations and profile delivery servers before a new capability or feature can be supported with the required profile data. Hence one of the most important requirements is to support the ability of implementers to extend specified standard data sets to include additional related features and flexibility. It MUST be possible to extend a data set without breaking user agents that support that data set. This may require that user agents ignore parts of a data set that it does not implement or extensions that it does support.

3.2.2 Flexible Capabilities

User agents vary quite widely in their capabilities. Some user agents function like traditional telephones. Some user agents support only text messaging. Some user agents support many media types such as video. Some user agents that function like a telephone have a single line, some have large numbers of lines. There is no such thing as one size fits all. It MUST be possible for an implementer to choose which data sets to support based upon the capabilities that are supported by the user agent. The schema for containing the profile data MUST support a profile that contains only the data sets that a user agent supports. This allows the profile

delivery server to create small profiles for specific devices. However a user agent SHOULD ignore properties for capabilities that it does not support. This allows the profile delivery server to be ignorant of the capabilities of the device. The degree to which the profile delivery server has intelligence of the user agent capabilities is an implementation choice.

3.2.3 XML

XML is perhaps not really a requirement, but a solution base upon requirements. However it is hard to ignore the desire to utilize readily available tools to manage and manipulate profile data such as XSLT, XPATH and XCAP. The requirement that should be considered when defining the schema and syntax is that many user agents have limited resources for supporting advanced XML operation. The simplest XML construct possible should be used, that support the required functionality. Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols [RFC3470] provides useful information in this regard.

3.2.4 Access Control

Many user agents (e.g. appliances and softphones running on PCs) provide user interfaces that permit the user to edit properties that are logically part of user, application, device or local network profiles. Operators and administrators would like to be able to specify what an end user can change in those profiles and what an end user is not allowed to change. There may also be sensitive data the user agent requires to function, but that the operator of the system does not want the end user to see. For some properties the system operator may allow the user a fixed set of choices among the supported set of possible values. It MUST be possible to express whether an end user may change a data set property. It MUST be possible to express that a property should not be made visible to the end user. It MUST be possible to express allowable values or ranges that the end user may change a property to. The access control information SHOULD be optional to the data set. It might be useful if it was possible to express the access control independent of the properties themselves. The access control specification by itself might be useful to express a general policy that the device owner or local network operator wish to impose.

3.2.5 Data Constraints and Range Definition

There is a need for property value types such as free form text, token/enumerations, integers, real numbers, etc. Many of these properties will have constrained values as opposed to the range of all possible values. These constrains may be due to protocol

definitions, implementation limitations, and/or the desire (e.g. by the user, device owner, local network operator) to impose policy on the user agent. The ability to express the property constraints is useful from the perspective of access control as described in the above section. It is also useful to parameterize a user interface (e.g. on the user agent itself or on the profile delivery server) which provides a facility to modify profile data. It MUST be possible for the schema to specify property constraints as ranges or discrete sets of possible values. These constrains SHOULD be optional to the data set. It might be useful if it was possible to express the constraints independent of the properties themselves. The constraints without the property values might be used to specify the capabilities of a particular user agent implementation.

3.2.6 Support of User, Application, Device, Local Network Sources

[I-D.ietf-sipping-config-framework] specifies a mechanism where the user agent retrieves profile data from as many as four different sources. The separation of the user profile facilitates a hotelling capability and the ability to easily re-assign a user to a different device. The separation of the local network profile facilitates properties specific to operating in the local network in a roaming scenario (e.g. outbound proxy or NAT traversal properties). The local network profile may also impose policy as describe in the next section. The device profile facilitates device capability based properties as well as a means for the device owner or manager (e.g. enterprise or service provider) to impose policy.

The multiple potential sources of profile data add some complexity to the user agent that must consolidate these separate profiles into a single working profile. It would be simpler if we could define each property as only allowed in one of the profiles. However it overly constrains the profiles and takes away desired functionality such as hotelling, roaming and shared profile management. It would also be simpler if we could define one rule for all profile data sets and properties by which we merge the profile (e.g. local network profile overwrites user profile which overwrites device profile for all data). However this too is overly restrictive and eliminates some very useful functionality.

The rules to merge profile data sets needs to be defined for each data set. In some cases an entire data set must be considered atomic when merging one profile source with another. In other cases it makes sense to merge profile data sets, aggregating properties from the data set provided in each of the profiles. It may also be desirable to have the effect of filtering of data set properties. The desired effect might be for the owner of the device or the local network operator to constrain what values are allowed for properties

in the profiles. This may also be the mechanism to facilitate imposing of policy as described in the next section. The operation of resolving overlapping data sets from multiple profiles, regardless of the means or net result, will be referred to as "merging" in this document.

A profile must have the means to constrain the merging algorithm. Due to the differences in the desired outcome for each data setting, the merging algorithm is specific to the setting. When defining a property setting, the merging alorithm must also be defined. A few of the more commonly used merging alorithms are defined in this document. Most settings are likely to use the common set defined in this document. However authors of profile datasets may define new alorithms for merging dataset properties (see Section 4.6 and Section 5.3).

3.2.7 The Ability to Specify Policy

Local network operators would like to impose policy on users and devices operating in their network. There is a need to constrain the operation and require specific behavior in the network. This might be as simple as to get access to the Internet, user agents must use a specified outbound proxy and NAT traversal mechanism. The network might have limited bandwidth such that the operator would like to constrain codecs or media streams to keep the network functional. The local network may provide emergency service behavior or functionality properties that are more specific than those provided by the device or user profile. The examples here focus on constraints to impose policy from the local network. However the facility to impose policy may be equally useful to the user and device profiles.

It MUST be possible to impose policy in any of the profile sources that constrains, overwrites or modifies properties provided in data sets from other sources.

4. Overall Data Set Schema

This document defines an XML Schema, for SIP Profile Data Sets that provides:

- a base element type "setting" from which all settings in other schema definitions inherit (this allows other definitions to specify the content models for ways of combining settings; it is analogous to a C++ virtual base class).
- Attributes to the "setting" element that define constraints and other properties used to impose policy that apply to the element value. These attributes are inherited by elements that are derived from the abstract settings element.

- o A root element for all property sets (the outermost container).

The full text of the schema is in Appendix A; the following describes the usage of the schema in defining properties and combining them to construct the working profile of a User Agent.

4.1 Data Primitives

Each property in a profile data set is defined using XML Schema Datatypes [W3C.REC-xmlschema-2] and XML Schema Structures [W3C.REC-xmlschema-1]. A property is modeled by an XML element derived from the "setting" element in the SIP Profile Data Set Schema. The element content is the setting value.

Properties consisting of one single value can be expressed using a single XML element. Properties that may consist of multiple values require the use of container elements. A container element is defined for such a property. This container can contain multiple XML elements, which each defines a possible value for that property (see examples in Section 4.5.2).

When constructing a property set, the creator of a profile may not be able to know all of the capabilities of the User Agent that will receive that property set. The creator of profile constraints or policies should be aware that a user agent may ignore properties that are unsupported or do not apply to its capabilities.

> OPEN ISSUE: Can a user agent generally ignore unsupported elements, even if they are marked as mandatory? This seems to be ok (e.g. a IM client can most likely safely ignore an element that defines a mandatory audio codec). Are there cases where this would cause problems?

4.2 Use of Namespaces

XML namespaces [W3C.REC-xml-names-19990114] provide a means to uniquely identify the elements and datatypes defined in a data set. It is therefore RECOMMENDED that each data set specifies its own namespace. The namespace URIs SHOULD be URNs [RFC2141], using the namespace identifier 'ietf' defined by [RFC2648] and extended by [I-D.mealling-iana-xmlns-registry].

4.3 The 'property_set' Element

The root element of a property set is "property_set"; it is the container that is provided to the user agent. The elements contained within a property_set contain the specific properties which are to be applied to the user agent. The properties may be simple types with a

single value, complex types or container elements with a list of properties.

4.4 The Abstract 'setting_container' Element

The "setting_container" element is the abstract element in which a list of properties which allow mutliple values may be contained. Elements derived from the "setting_container" element may contain zero or more elements derived from the "setting" element. The "setting_container" element has an "excluded_policy" attribute.

4.5 The Abstract 'setting' Element

The setting element is the abstract element from which all profile properties or settings shall inherit.

The setting element has a number of attributes that provide functionalities, which are generally useful for many properties. These attributes are inherited by properties that are derived from the settings element. This enables the re-use of common functionalities and ensures a common syntax for these elements across different data sets. The following functionalities are provided by attributes of the settings element:

- Property Access Control: 'visibility' attribute
- Policies: 'policy' attribute

Additional attributes are defined in the schema that may used in elements derived from "setting". By default these attributes cannot be set. These attribute must be explicitly added to elements derived from the "setting" element:

- Unidirectional Properties: 'direction' attribute
- Preferences: 'q' attribute

4.5.1 The 'visibility' Attribute

The attribute "visibility" is defined on the "setting" element to specify whether or not the user agent is permitted to display the property value to the user. This is used to hide setting values that the profile administrator may not want the user to see or know. The "visibility" attribute has two possible values:

- visible: specifies that display of the property value is not restricted. This is the default value of the attribute if it is not specified.

- hidden: Specifies that the user agent SHOULD NOT display the property value. Display of the property value may be allowed using special administrative interfaces, but is not appropriate to the ordinary user.

4.5.2 The 'policy' Attributes

The setting element has an optional 'policy' attribute. The policy attribute is used to define the constraining properties of an element. It defines how the element value is used by an endpoint (e.g. whether it can or can not be used in a session). The following values are defined for the 'policy' attribute:

- mandatory: the value contained in the element is mandatory and MUST be used in sessions. This is the default value that is used if the 'policy' attribute is omitted.
- allow: the value contained in the element is allowed and MAY be used in sessions.
- disallow: the value contained in the element is forbidden and MUST NOT be used in sessions.

The policy attribute can be omitted if the default policy 'mandatory' applies (i.e. the property must be considered when setting up a session). The following is an example of a policy defining an upper limit for media bandwidth:

```
<max-bandwidth>80</max-bandwidth>
```

4.5.3 The 'excluded_policy' Attributes

The "setting_container" element has an optional 'excluded_policy' attribute. This attribute specifies the default policy for all values that are not in the container. Elements that are present in the container have their own 'policy' attribute, which defines the policy for that element. The following values are defined for the 'excluded_policy' attribute:

- allow: values not listed in the container are allowed and MAY be used in sessions. This is the default value that is used if the 'excluded_policy' attribute is omitted.
- disallow: values not listed in the container are forbidden and MUST NOT be used in sessions.

The excluded_policy attribute can be omitted if the default policy 'allow' applies. The following example shows a policy that requires the media type audio and allows video and disallows all other media

types in sessions:

```
<media-types excluded_policy="disallow">
  <media-type policy="mandatory">audio</media-type>
  <media-type policy="allow">video</media-type>
</media-types>
```

4.5.4 The 'direction' Attribute

Some properties are unidirectional and only apply to messages or data streams transmitted into one direction. For example, a property for media streams can be restricted to outgoing media streams only. Unidirectional properties can be expressed by adding a 'direction' attribute to the respective element.

The 'direction' attribute can have the following values:

- recvonly: the property only applies to incoming messages/streams.
- sendonly: the property only applies to outgoing messages/streams.
- sendrecv: the property applies to messages/streams in both directions. This is the default value that is used if the 'direction' attribute is omitted.

4.5.5 The 'q' Attribute

It should be possible to express a preference for a certain value, if multiple values are allowed within a property. For example, it should be possible to express that the codecs G.711 and G.729 are allowed, but G.711 is preferred. Preferences can be expressed by adding a 'q' attribute to a property element. Elements derived from the "setting" element for which multiple occurances and values are allowed SHOULD have a "q" attribute if the order is signficant. Typically these elements are contained in an element derived from the "setting_container" element. The 'q' attribute is only meaningful if the 'policy' attribute set to 'allowed' or "mandatory". It must be ignored in all other cases.

An element with a higher 'q' value is preferred over one with a lower 'q' value. 'q' attribute values range from 0 to 1. The default value is 0.5.

4.6 Merging Property Sets

A UA may receive property sets from multiple sources, which need to be merged into a single combined document the UA can work with.

Properties that have a single value (e.g. the maximum bandwidth allowed) require that a common value is determined for this property during the merging process. The merging rules for determining this value need to be defined individually for each element in the schema definition. Properties that allow multiple values (i.e. property containers) need to be merged by combining the values from the different data sets. The following sections describe common merging algorithms. A data set definition may refer to these algorithms.

4.6.1 Single Numeric Value Merging Algorithm

A general merging rule for elements with numeric values is to select the largest or the smallest value. For example, a merging rule for a <max-bandwidth> element would be to select the smallest value from the values that are in the competing data sets.

4.6.2 Multiple Enumerated Value Merging Algorithm

Multiple values in property containers are merged by combining the values from each of the competing data sets. This is accomplished by copying the elements from each property container into the merged container. Elements with identical values are only copied once. The 'policy' attribute of two elements with the same value is adjusted during the merging process according to Table 1. If an element exists only in one property container, then the default policy of the other container (i.e. the excluded_policy) is used when accessing Table 1. For example, if an element is mandatory in one data set and allowed in the other data set, it will be mandatory in the merged data set. Finally, the excluded_policy attributes of the containers are also merged using Table 1. In addition to these merging rules, each schema may define specific merging rules for each property container.

set 1 \ set 2	mandatory	allow	disallow
mandatory	mandatory	mandatory	conflict!
allow	mandatory	allow	disallow
disallow	conflict!	disallow	disallow

Table 1: merging policies.

The following example illustrates the merging process for two data sets. All elements are merged into one container and the policy attributes are adjusted according to Table 1. The merged container has the default policy disallow, which is determined using Table 1. The entry for PCMA in the merged data set is redundant since it has the default policy.

```
Data set 1:
<codecs excluded_policy='allow'>
  <codec policy='disallow'>PCMA</codec>
</codecs>

Data set 2:
<codecs excluded_policy='disallow'>
  <codec policy='allow'>PCMA</codec>
  <codec policy='allow'>G729</codec>
</codecs>

Merged data set:
<codecs excluded_policy='disallow'>
  <codec policy='disallow'>PCMA</codec>
  <codec policy='allow'>G729</codec>
</codecs>
```

Some constellations of policy attributes can not be merged. They constitute a conflict that can not be resolved automatically. For example, two data sets may define two non-overlapping sets of allowed audio codecs. If the use of these properties is enforced by the network, the UA may experience difficulties or may not be able to set up a session at all.

The combined property set MUST again be valid and well-formed according to the schema definitions. A conflict occurs if the combined property set is not a well-formed document after the merging process is completed.

4.6.3 Closest Value First Merging Algorithm

Some properties require that the values from different data sets are ordered based on the origin of the data set during the merging process. Property values that come from a domain close to the user agent take precedence over values that were in a data set delivered by a remote domain. This order can be used, for example, to select the property value from the closest domain. In many cases, this is the local domain of the user agent. For example, the URI of an outbound proxy could be merged this way. This order can also be used to generate an ordered list of property values during the merging process. For example, multiple values for media intermediaries can be ordered so that the closest media intermediary is traversed before the second closest intermediary and so on.

This merging algorithm requires that the source of a data set is considered.

If property sets are delivered through the configuration framework

[I-D.ietf-sipping-config-framework], the value received through a subscription using the "local-network" profile-type takes precedence over values received through other profile-type subscriptions.

OPEN ISSUE: Can we define an order for 'device', 'user', and 'application' profiles?

The session-specific policy mechanism [I-D.hilt-sipping-session-spec-policy] provides an order among policy servers. This order is based on the order, in which a SIP message traverses the network, starting with the closest domain. This order can directly be used to order property values as described above.

4.7 Common Types

[The schema will also define a set of common types that are used in defining data sets (e.g. name-addr) in a future version of this draft.]

5. Defining Data Sets

This section covers several issues that should be take into consideration when specifying new data set schemas. This is intended to be a guide to authors writing specifications defining a new data set schema or extensions to existing ones.

5.1 Namespace

It is RECOMMENDED that a data set defines a new XML namespace [W3C.REC-xml-names-19990114] to scope all of the properties that are defined in the name space.

5.2 Property Definitions

The properties defined in a data set schema may be simple (i.e. having a single value) or they may be complex (i.e. a container with multiple values). Each property in the data set SHOULD inherit from the "setting" element. Complex properties and all of their child elements each should inherit from "setting" as well.

A data set specification should contain a section which defines the meaning of all of the properties contained in the data set. The objective is to define the property such that implementers have a clear definition and semantics to interpret properties in a consistent way. User agents not only need to use the same profile content, they need to apply the properties in a consistent way to achieve true interoperability.

The following information should be defined for each property in a data set:

- description: describe the meaning and application of the property.
- cardinality: define how many instances of this property element may occur in a data set (e.g. zero, one or many) as well as its relationship to any other properties in this or other data sets.
- default value: define the default value of this property if it is not set. Describe if the default is different if the property is present and not set vs. completely absent from the data set. Define if the default varies in relation to another property.

5.3 Merging Data Sets

User agents may receive data sets from multiple sources. They need to merge these data sets in order to create an overall data set they can work with. Collisions on data sets may occur if multiple sources provide different values for the same properties. These collisions need to be resolved during the merging process.

A data set schema MUST define rules for merging data sets from different sources for each property that is defined. Considerations for merging data sets are discussed in Section 4.6. A data set schema must define if and how these consideration apply and MAY define alternative merging rules for specific settings. A data set schema must also identify combinations of properties that constitute a conflict that can't resolved. It may provide additional guidelines for the behavior of a user agent in these cases.

6. Candidate Data Sets

The following sections name some of the candidate data sets that are or may be defined. These data sets can be aggregated to form profiles appropriate to the capabilities of a user agent implementation.

- SIP Protocol Data Set: the lowest common denominator set of properties common to all SIP user agents of any capability. A schema covering the elements of this data set can be found in XXX.
- Media Data Set: this data set contains media related policies. A schema covering the elements of this data set can be found in [I-D.ietf-sipping-session-indep-policy].
- Identity Data Set: AORs and lines.
- HTTP Protocol Data Set: server settings. Proxy for clients.
- NAT Traversal Data Set: settings for STUN, TURN etc.
- Address Book:
- Buddy List:

- o SIP Digit Maps Data Set:

7. Security Considerations

Security is mostly a delivery problem. The delivery framework SHOULD provide a secure means of delivering the profile data as it may contain sensitive data that would be undesirable if it were stolen or sniffed. Storage of the profile on the profile delivery server and user agent is an implementation problem. The profile delivery server and the user agent SHOULD provide protection that prevents unauthorized access of the profile data. The profile delivery server and the user agent SHOULD enforce the access control policies defined in the profile data sets if present.

[The point of the access control construct on the data set is to provide some security policy on the visibility and ability to change sensitive properties. Does the access control mechanism also create a security problem where the local network can set or hide properties from the user?]

Some transport mechanisms for delivery of the profile data do not provide a secure means of delivery. In addition some user agents may not have the resources to support the secure mechanism used for delivery (e.g. TLS).

8. IANA Considerations

[TBD] XML Schema name space registration

9. Change History

9.1 Changes from draft-petrie-sipping-profile-datasets-01

Split out the core SIP Protocol dataset into a separate draft.

Schema changes: created setting_container, added q and direction attributes along with other tweeks to the schema.

Better integration and coordination with [I-D.ietf-sipping-session-indep-policy]. The media/codec dataset is now completely contained in the policy draft.

9.2 Changes from draft-petrie-sipping-profile-datasets-00

Added use case scenarios for codecs, SIP transport protocol and outbound proxy to better illustrate requirements. Some of the derived requirements are listed with the use cases.

Added settings element attributes "policy" and "visibility" to

provide merging constraints and access control capability. Removed the element based merging constraints using the: forbid, set_any, set_all and set_one elements. This greatly simplifies the degree of XML operations required to perform the request merging.

Defined default merging policy and profile source precedence along with the option for different policies to be describe in specific settings definition documents.

Added example merging with XML profiles from device and user for the SIP transport protocol.

10. References

[I-D.hilt-sipping-session-spec-policy]
Hilt, V., Camarillo, G., and J. Rosenberg, "A Delivery Mechanism for Session-Specific Session Initiation Protocol (SIP) Session Policies", draft-hilt-sipping-session-spec-policy-03 (work in progress), July 2005.

[I-D.ietf-sipping-config-framework]
Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-06 (work in progress), February 2005.

[I-D.ietf-sipping-session-indep-policy]
Hilt, V., "Session Initiation Protocol (SIP) Session Policies - Document Format and Session-Independent Delivery Mechanism", draft-ietf-sipping-session-indep-policy-02 (work in progress), February 2005.

[I-D.mealling-iana-xmlns-registry]
Mealling, M., "The IETF XML Registry", draft-mealling-iana-xmlns-registry-05 (work in progress), June 2003.

[I-D.sinnreich-sipdev-req]
Sinnreich, H., "SIP Telephony Device Requirements and Configuration", draft-sinnreich-sipdev-req-07 (work in progress), June 2005.

[RFC0822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC2648] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

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

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

[RFC3470] Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols", BCP 70, RFC 3470, January 2003.

[W3C.REC-xml-names-19990114] Hollander, D., Bray, T., and A. Layman, "Namespaces in XML", W3C REC REC-xml-names-19990114, January 1999.

[W3C.REC-xmlschema-1] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC-xmlschema-1, May 2001, <http://www.w3.org/TR/xmlschema-1/>.

[W3C.REC-xmlschema-2] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes", W3C REC-xmlschema-2, May 2001, <http://www.w3.org/TR/xmlschema-2/>.

Authors' Addresses

Daniel Petrie
SIPez LLC
34 Robbins Rd.
Arlington, MA 02476
US

Phone: +1 617 273 4000
Email: dan.ietf AT SIPez DOT com
URI: http://www.SIPez.com/

Scott Lawrence
Pingtel Corp.
400 W. Cummings Park
Suite 2200
Woburn, MA 01801
US

Phone: "Scott Lawrence (+1 781 938 5306)" <sip:slawrence@pingtel.com>
Email: slawrence AT pingtel DOT com
URI: http://skrb.org/scott/

Martin Dolly
AT&T Labs
200 Laurel Avenue
Middletowm, NJ 07748
US

Phone:
Email: mdolly AT att DOT com
URI:

Volker Hilt
Bell Labs/Lucent Technologies
101 Crawfords Corner Rd
Holmdel, NJ 07733
USA

Email: volkerh@bell-labs.com

Appendix A. SIP UA Profile Schema

```
<?xml version='1.0' encoding='iso-8859-1' standalone='yes'?>
<!DOCTYPE schema [
<!ENTITY % doc_src
"http://scm.sipfoundry.org/rep/ietf-draft/petrie/profile-data-sets">
]>
<!--
    XML Schema for SIP Profile Data Sets
  -->
<schema
xmlns:spds='http://sipfoundry.org/schema/profile-data-sets-02'
targetNamespace='http://sipfoundry.org/schema/profile-data-sets-02'
xmlns='http://www.w3.org/2001/XMLSchema'
    >
 <annotation>
   <documentation>
```

```
        Proposed XML metalanguage for the description of
        SIP User Agent Profile Data Sets.
      </documentation>
      <documentation source='%doc_src;'/>
    </annotation>

   <!-- Types
     Later versions of the Internet-Draft of which this is a part may
     include additional data type definitions and entities useful
     in defining SIP data.
    -->

    <simpleType name="port_num">
     <restriction base="integer">
      <minExclusive value='0' />
      <maxInclusive value='65535' />
     </restriction>
    </simpleType>

    <simpleType name="q_val">
     <restriction base="float">
      <minInclusive value='0' />
      <maxInclusive value='1' />
     </restriction>
    </simpleType>

    <simpleType name="transport_protocol">
      <restriction base="string">
        <enumeration value="TCP"/>
        <enumeration value="UDP"/>
        <enumeration value="TLS"/>
      </restriction>
    </simpleType>

   <!-- Attrubutes that may be optionally used
    -->

   <attributeGroup name="multi_setting_attributes" >
     <annotation>
       <documentation>
         The multi_setting_attributes attribute group is
         for attributes that are applicable to settings that
         may have multiple values in a container.
       </documentation>
       <documentation source='%doc_src;'/>
     </annotation>
     <attribute name="q" type="q_val" default="0.5" >
       <annotation>
```

```
        <documentation>
          The q attribute is used to define a preference for a
          setting.  It can be used to define that one value
          of a setting is preferred over another value.
        </documentation>
        <documentation source='%doc_src;'/>
      </annotation>
    </attribute>
  </attributeGroup>

  <attributeGroup name="directional_setting_attributes" >
    <annotation>
      <documentation>
        The multi_setting_attributes attribute group is
        for attributes that are applicable to settings that
        have a directional implication (e.g. incoming or
        outgoing).
      </documentation>
      <documentation source='%doc_src;'/>
    </annotation>
    <attribute name="direction" default="sendrecv" >
      <annotation>
        <documentation>
          The direction attribute is used to define
          unidirectional settings.
        </documentation>
        <documentation source='%doc_src;'/>
      </annotation>
      <simpleType>
        <restriction>
          <enumeration value="sendrecv"/>
          <enumeration value="sendonly"/>
          <enumeration value="recvonly"/>
        </restriction>
      </simpleType>
    </attribute>
  </attributeGroup>

  <!-- Elements
    Later versions of the Internet-Draft of which this is a part may
    include additional data type definitions and entities useful
    in defining SIP data.
   -->

   <element name="property_set">
     <annotation>
       <documentation>
       The property_set element is the root element returned in
```

```
        response to a request for a profile data set.
        </documentation>
      </annotation>
      <complexType>
        <sequence minOccurs="0" maxOccurs="unbounded">
          <choice>
            <element ref="spds:setting" />
            <element ref="spds:setting_container" />
          </choice>
        </sequence>
      </complexType>
    </element>

    <element name="setting_container" abstract="true">
    <!-- TBD -->
      <complexType>
        <complexContent>
          <restriction base="anyType">
            <sequence minOccurs="0" maxOccurs="unbounded">
              <element ref="spds:setting" />
            </sequence>
            <attribute name="excluded_policy"
                       default="allowed">
              <annotation>
                <documentation>
                   The container_policy attibute is used to define the
                   policy for settings not explcitly contained in the
                   container.  disallowed means that setting
                   values not included in the container are considered
                   to be diallowed.   The value of allowed
                   indicates that values not included in the container
                   are allowed.
                </documentation>
                <documentation source='%doc_src;'/>
              </annotation>
              <simpleType>
                <restriction>
                  <enumeration value="disallowed"/>
                  <enumeration value="allowed"/>
                </restriction>
              </simpleType>
            </attribute>
          </restriction>
        </complexContent>
      </complexType>
    </element>

    <element name="setting" abstract="true">
```

```
    <annotation>
     <documentation>
      The 'setting' element is an abstract used as the basis for the
      definition of the setting elements in property schemas derived
      from this one.

      It serves here as a placeholder in constructing the content
      models for the container elements used to group settings into
      sets.
     </documentation>
     <documentation source='%doc_src;'/>
    </annotation>
    <complexType>
     <complexContent>
      <restriction base="anyType">
       <attribute name="policy" default="mandatory" >
         <annotation>
           <documentation>
             The policy attribute is used to define the strength to
             which a setting should be used.  It can also be viewed
             as the finality to which a setting may be overrided.
           </documentation>
           <documentation source='%doc_src;'/>
         </annotation>
         <simpleType>
          <restriction>
           <enumeration value="allow"/>
           <enumeration value="disallow"/>
           <enumeration value="mandatory"/>
          </restriction>
         </simpleType>
       </attribute>
       <attribute name="visibility" default="visible" >
         <annotation>
           <documentation>
             The visibility attribute indicates whether the user
             agent should show the setting value(s) to the user.
           </documentation>
           <documentation source='%doc_src;'/>
         </annotation>
         <simpleType>
          <restriction>
           <enumeration value="visible"/>
           <enumeration value="hidden"/>
          </restriction>
         </simpleType>
       </attribute>
      </restriction>
```

```
      </complexContent>
     </complexType>
    </element>

   </schema>
```

Appendix B. Acknowledgments

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING
Internet-Draft
Expires: January 12, 2006

D. Petrie
SIPez LLC.
M. Dolly
AT&T Labs
V. Hilt
Bell Labs/Lucent Technologies
July 11, 2005

The Core Session Initiation Protocol User Agent Profile Data Set
draft-petrie-sipping-sip-dataset-00.txt

Status of this Memo

Copyright Notice

Abstract

This document defines the properties and format for the core SIP user agent profile data set. The properties defined in this document are expected to be common to most SIP user agents regardless of whether the user agent support audio, video, text or any combination of media. These core SIP properties are considered to to be a data set.

Several datasets may be combined into documents or profiles that are provided to SIP user agents so that they can operate with the desired behavior.

Table of Contents

1. Motivation

The SIP Profile Data Sets defined in this document support the principle to enable SIP User Agents to obtain and use profile data sets from multiple sources in order to support a wide range of applications without undue complexity.

The SIP Protocol Data Set is intended the be the lowest common denominator among all user agent types regardless of capability. This data set contains properties that all user agents require. That does not mean that all of these properties are mandatory.

2. Introduction

This document defines the properties and format for the core SIP user agent profile data set. The following properties are defined in this document:

- transport_protocols
- outbound_proxies
- sip_methods
- sip_option_tags

and, are expected to be common to most SIP user agents regardless of whether the user agent support audio, video, text or any combination of media. These core SIP properties are considered to to be a data set.

2.1 Requirements Terminology

Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in RFC 2119[RFC2119].

2.2 Profile Data Terminology

property - a named configurable characteristic of a user agent. A given property has a well-defined range of possible values. A given property may be defined to have range of values, allow for simultaneous use of many values (as in a list of allowed possibilities), or be a set of related values that collectively form a single profile information item.

setting - the binding of a specific value or set of values to a given property.

profile - a collection of settings to be applied for a specific user, device, or local network.

device - SIP user agent, either software or hardware appliance. This is a logical concept, as there may be no physical dedicated device or it may be part of an assembly of devices. In this document, the terms "user agent" and "device" are interchangeable.
user profile - the profile that applies to a specific user. This is best illustrated by the "hotelling" use case - a user has an association for some period of time with a particular device. The user profile is that set of profile data the user wants to associate with that device (e.g. ringtones used when someone calls them, the user's shortcuts).
device profile - data profile that applies to a specific device. In the "hotelling" use case, this is the data that is bound to the device itself independent of the user. It relates to specific capabilities of the device and/or preferences of the owner of the device.
local network profile - data that applies to the user agent in the context of the local network. This is best illustrated by roaming applications; a new device appears in the local network (or a device appears in a new network, depending on the point of view). The local network profile includes settings and perhaps policies that allow the user agent to function in the local network (e.g. how to traverse NAT or firewall, bandwidth constraints).
data set - a collection of properties.
working profile - the set of property values actually set in a SIP User Agent as a result of merging the profiles from all sources; the actual effective profile for the user agent .
merging - the operation of resolving overlapping settings from multiple profiles. Overlap occurs when the same property occurs in multiple profiles (e.g. user, device, local network).

2.3 Overview

The Core SIP UA profile data set is defined in Section 3 and complies with the guidelines provided in Section 5 of [I-D.petrie-sipping-profile-datasets]

Section 4 provides illustrative example profiles and use cases for merging. Security considerations are addresed in Section 5.

The following is an example instance of the SIP protocol data set. Note the use of the policy attribute.

```
<property_set>
  <transport_protocols>
    <transport_protocol policy="allow">
      <name>UDP</name>
      <port>5060</port>
    </transport_protocol>
    <transport_protocol policy="allow">
      <name>TCP</name>
      <port>5060</port>
    </transport_protocol>
    <transport_protocol policy="allow">
      <name>TLS</name>
      <port>5061</port>
    </transport_protocol>
  </transport_protocols>
  <outbound_proxies>
    <outbound_proxy policy="mandatory">
      sip:outproxy.example.com
    </outbound_proxy>
  </outbound_proxies>
  <sip_methods>
    <sip_method policy="disallow">INFO</sip_method>
  </sip_methods>
  <sip_option_tags>
    <sip_option_tag policy="disallow">join</sip_option_tag>
  </sip_option_tags>
</property_set>
```

3. Core SIP Data Set

The XML schema defined in this document extends the root element "property_set" schema defined in I-D.petrie-sipping-profile-datasets.

3.1 Transport Protocol Data Set

3.1.1 transport_protocols Data Set Properties Definitions

transport_protocols - This property contains properties related to SIP transport protocols, and is an XML element that extends on the XML "setting_container" element contained in the root "property_set" element. It serves as a container for a list of SIP transport protcols to allow or disallow. There may be zero or one elements.

3.1.2 transport_protocols Element Definition

transport_protocol - The "transport_protocol" is an XML element that extends the "setting" element contained in the "transport_protocols" element. The "transport_protocol" element contains properties related to a SIP transport protocol. It names the transport protocol, defines whether the protocol is enabled or not and defines the port to which that protocol is bound. If the protocol is named it defaults to enabled if not explicitly set. If the port property is not set, it defaults to the default specified by the specification which binds the protocol to SIP. The user agent should enable all the set transport protocols that are supported by the user agent. The user agent ignores protocol bindings that it does not support. The user agent may default transport protocols that it supports to enabled, if a protocol property for that transport protocol is not present in the data set. The order of the list of transport_protocol setting values indicated by the "q" attribute indicates the order of preference. There may be zero or more "transport_protocol" elements in the "transport_protocols" element.

name - This XML element identifies the specific transport protocol, and extends the "setting" element contained in transport_protocols. There must be exactly one "name" element in a "transport_protocol".

port - This element identifies the port for binding the transport protocol, and extends the "setting" element contained in transport_protocols. There must be exactly one "port" element in a "transport_protocol".

3.1.3 Merging Different Sources of a transport_protocol Data Set

The "transport_protocol" property uses the "policy" attrribute to identify whether the transport protocol is mandatory, allowed or disallowed. The "q" attribute is used for ordering of the list. In addition, a visibility attribute may be present.

If there are matches on multiple "name" element values, the "policy" attribute will determine which is mandatory. As defined in Merging Datasets [I-D.petrie-sipping-profile-datasets] properties with a "policy" attribute value of "manditory" are used over those with other "policy" attribute values. If there are multiple "transport_protocol" elements from different profiles with the same "name" element value and "policy" attribute values of "allows", then the resulting merged "transport_protocols" element will contain one "transport_protocol" element having a "name" element of that value. The "port" element value will be determined in the following order of the source profile, when there are multiple "transport_protocol" elements from different profiles with the same "name" element value

and "policy" attribute value of "allow":
Local
Device
User
Application

3.2 outbound_proxy

3.2.1 outbound_proxy Data Set Properties Definitions

outbound_proxies - The "outbound_proxies" property is an XML element that extends on the XML "setting" element contained in the root "property_set" element. It serves as a container for a list of outbound proxies. There may be zero or one element. The default outbound proxy, through which all SIP requests, not explicitly routed, should be sent. The format of this parameter is of name-addr as specified in [RFC3261]. This property is optional. If absent or not set, SIP requests are sent to directly to the URI of the request. If set the effect of this property is to add a loose route as defined in [RFC3261] for the next hop destination.

3.2.2 outbound_proxies Element Definition

outbound_proxies - The "outbound_proxy" is an XML element that extends the XML "setting" element contained in "outbound_proxies". There may be zero, one or many "outbound_proxy" elements. It provides default value for an outbound proxy, through which all SIP requests, not explicitly routed, should be sent. The format of this parameter is of name-addr as specified in [RFC3261]. This property is optional. If absent or not set, SIP requests are sent to directly to the URI of the request. If set the effect of this property is to add a loose route as defined in [RFC3261] for the next hop destination. Multiple "outbound_proxy" elements may be contained in the "outbound_proxies" element to form a route set."

3.2.3 outbound_proxies Merging Different Sources of a Data Set

The aggregation approach is used to resolve conflicts. By aggregating the multiple outbound proxies, the local network profile provided outbound proxy allows the signaling to get out of the local network and the device profile provided outbound proxy is able to monitor all SIP signaling from the user agent. The order of the resulting merged, route set is determined by the "q" attibute.

3.3 sip_methods

3.3.1 sip_methods Data Set Properties Definitions

sip_methods - This property contains properties related to SIP Methods, and is an XML element that extends on the XML "setting" element contained in the root "property_set" element. It serves as a container for a list of SIP request methods to allow or disallow. Typically, only provide by the device dataset. The "sip_methods" element is intended to provide a means of enabling or diabling features in the SIP user agent based upon the SIP request method.

3.3.2 sip_methods Element Definition

sip_method - An element to specify a SIP method, and extends the "setting" element contained in the "sip_methods" element. There may be zero or more elements. For user agents that support the method indicated, this element serves as a switch to enable or disable the named SIP method as indicated by the "policy" attribute.

3.3.3 sip_methods Merging Different Sources of a Data Set

The "sip_methods" Data Set uses the aggregation merging policy defined in [I-D.petrie-sipping-profile-datasets]. When multiple "sip_method" elements with the same value are provided, the "policy" attribute is used to determine precidence.

3.4 sip_option_tags

3.4.1 sip_option_tags Data Set Properties Definitions

sip_option_tags - This property specifies a container for a list of SIP option tags that are allowed or disallowed, and is an XML element that extends on the XML "setting" element contained in the root "property_set" element. For user agents that suppport features indicated by option tags, this element serves as a list of features to turn on or off as indicated by the "policy" attribute in the "sip_option_tag" element.

3.4.2 sip_option_tags Element Definition

sip_option_tag - An element to specify a SIP option tag, and extends the "setting" element and is contained in "sip_option_tags" element. There may be zero or more elements "sip_option_tag". For user agents that suppport features indicated by option tags, this element serves as a switch to enable or disable the named SIP option as indicated by the policy attribute in the

"sip_option_tag" element.

3.4.3 sip_option_tags Merging Different Sources of a Data Set

The sip_option_tags Data Set uses the default aggregation merging policy defined in [I-D.petrie-sipping-profile-datasets]. When multiple "sip_method" elements with the same value are provided, the "policy" attribute is used to determine precidence.

4. Example Profiles and Use

4.1 Merge Two Data Sets

Consider the use case described in [I-D.petrie-sipping-profile-datasets] where the user wishes to indicate that only secure SIP transport should be used. The device profile may contain SIP Protocol Data Set (see Section 3.1) settings that look like the following:

```
<property_set>
  <transport_protocols>
    <transport_protocol policy="allow">
      <name>UDP</name>
      <port>5060</port>
    </transport_protocol>
    <transport_protocol policy="allow">
      <name>TCP</name>
      <port>5060</port>
    </transport_protocol>
    <transport_protocol policy="allow">
      <name>TLS</name>
      <port>5061</port>
    </transport_protocol>
  </transport_protocols>
  <outbound_proxies>
    <outbound_proxy policy="mandatory">
      sip:outproxy.example.com
    </outbound_proxy>
  </outbound_proxies>
  <sip_methods>
    <sip_method policy="disallow">INFO</sip_method>
  </sip_methods>
  <sip_option_tags>
    <sip_option_tag policy="disallow">join</sip_option_tag>
  </sip_option_tags>
</property_set>
```

The user profile which indicates that only TLS should be used would look like (Note: this example also indicates that port 5061 should be used with a mandatory policy as well. This may be more constrained than the user really wants.):

```
<property_set>
  <transport_protocols>
    <transport_protocol policy="mandatory">
      <name>TLS</name>
      <port>5061</port>
    </transport_protocol>
    <transport_protocol policy="disallow">
      <name>UDP</name>
    </transport_protocol>
    <transport_protocol policy="disallow">
      <name>TCP</name>
    </transport_protocol>
  </transport_protocols>
</property_set>
```

The merged result of the device and user profile would look like:

```
<property_set>
  <transport_protocols>
    <transport_protocol policy="mandatory">
      <name>TLS</name>
      <port>5061</port>
    </transport_protocol>
    <transport_protocol policy="disallow">
      <name>UDP</name>
    </transport_protocol>
    <transport_protocol policy="disallow">
      <name>TCP</name>
    </transport_protocol>
  </transport_protocols>
  <outbound_proxies>
    <outbound_proxy policy="mandatory">
      sip:outproxy.example.com
    </outbound_proxy>
  </outbound_proxies>
  <sip_methods>
    <sip_method policy="disallow">INFO</sip_method>
  </sip_methods>
  <sip_option_tags>
    <sip_option_tag policy="disallow">join</sip_option_tag>
  </sip_option_tags>
</property_set>
```

4.2 Policy Filtering

(allowed and disallowed protocols)

4.3 Override

(device prefers default ports 5060, local net requires port 11000)

5. Security Considerations

Security is mostly a profile delivery problem. The delivery framework MUST provide a secure means of delivering the profile data as it may contain sensitive data that would be undesirable if it were stolen or sniffed. Storage of the profile on the profile delivery server and user agent is an implementation problem. The profile delivery server and the user agent MUST provide protection that prevents unauthorized access of the profile data. The profile delivery server and the user agent MUST enforce the access control policies defined in the profile data sets if present.

6. Changes from draft-petrie-sipping-profile-datasets-01

The core SIP profile data set was split out from the examples in draft-petrie-sipping-profile-datasets-01 to create a stand alone data set definition.

7. References

[I-D.ietf-sipping-config-framework]
Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-06 (work in progress), February 2005.

[I-D.petrie-sipping-profile-datasets]
Petrie, D., "A Schema for Session Initiation Protocol User Agent Profile Data Sets", draft-petrie-sipping-profile-datasets-00 (work in progress), July 2004.

[I-D.sinnreich-sipdev-req]
Sinnreich, H., "SIP Telephony Device Requirements and Configuration", draft-sinnreich-sipdev-req-07 (work in progress), June 2005.

[RFC0822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[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.

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

[RFC3470] Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols", BCP 70, RFC 3470, January 2003.

[W3C.REC-xml-names] Bray, T., Hollander, D., and A. Layman, "Namespaces in XML", W3C REC-xml-names, January 1999, <http://www.w3.org/TR/REC-xml-names>.

[W3C.REC-xmlschema-1] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC-xmlschema-1, May 2001, <http://www.w3.org/TR/xmlschema-1/>.

[W3C.REC-xmlschema-2] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes", W3C REC-xmlschema-2, May 2001, <http://www.w3.org/TR/xmlschema-2/>.

Authors' Addresses

Daniel Petrie
SIPez LLC.
34 Robbins Rd.
Arlington, MA 02476
US

Phone: +1 617 273 4000
Email: dan.ietf AT SIPez DOT com
URI: http://www.sipez.com/

Martin Dolly
AT&T Labs
200 Laurel Avenue
Middletowm, NJ 07748
US

Phone:
Email: mdolly AT att DOT com
URI:

Volker Hilt
Bell Labs/Lucent Technologies
101 Crawfords Corner Rd
Holmdel, NJ 07733
US

Phone:
Email: volkerh@bell-labs.com
URI:

Appendix A. SIP Protocol Dataset Schema

The following is the schema for the SIP protocol data set.

```
<?xml version='1.0' encoding='iso-8859-1' standalone='yes'?>
<!--
    XML Schema for SIP Protocol core Data Sets
  -->
<schema
xmlns:spds='http://sipfoundry.org/schema/sip-ua-profile-07'
targetNamespace='http://sipfoundry.org/schema/sip-protocol-00'
xmlns='http://www.w3.org/2001/XMLSchema'
    >
 <annotation>
   <documentation>
     SIP Protocol Properties.
   </documentation>
 </annotation>

 <element name="transport_protocols"
          substitutionGroup="spds::setting_container">
  <annotation>
   <documentation>
    Container for a set of transport protocol
    bindings for SIP.
   </documentation>
  </annotation>
```

```
      <complexType>
       <sequence minOccurs="0" maxOccurs="unbounded">
        <element ref="transport_protocol" />
       </sequence>
      </complexType>
    </element>

    <element name="transport_protocol"
             substitutionGroup="spds::setting">
     <annotation>
      <documentation>
       Container for the properties for a single transport protocol
       binding for SIP.
      </documentation>
     </annotation>
      <complexType>
       <sequence>
        <element ref="name" />
        <element ref="port" />
       </sequence>
       <attributeGroup ref="spds:multi_setting_attributes" />
      </complexType>
    </element>

    <element name="name" type="spds:transport">
     <annotation>
      <documentation>
       Name of the specific transport protocol
      </documentation>
     </annotation>
    </element>

    <element name="port" type="spds:port_num">
     <annotation>
      <documentation>
       Port binding for the transport protocol
      </documentation>
     </annotation>
    </element>

    <element name="outbound_proxies"
             substitutionGroup="spds::setting_container">
     <annotation>
      <documentation>
       Container for outbound_proxy elements which define a preset
       route set.  The q attribute determines the order of the
       routes in the route set.
      </documentation>
```

```
     </annotation>
      <complexType>
       <sequence minOccurs="0" maxOccurs="unbounded">
        <element ref="outbound_proxy" />
       </sequence>
      </complexType>
    </element>

    <element name="outbound_proxy" substitutionGroup="spds::setting">
     <annotation>
      <documentation>
       The next hop proxy for SIP requests without a defined
       route set.  Value is of name-addr format.  There should
       probably be a type defined for name-addr that outbound_proxy
       inherits from.
      </documentation>
     </annotation>
     <complexType>
       <attributeGroup ref="spds:multi_setting_attributes" />
     </complexType>
    </element>

    <element name="sip_methods"
             substitutionGroup="spds::setting_container">
     <annotation>
      <documentation>
       Container for list of SIP request methods to allow or
       disallow.
      </documentation>
     </annotation>
      <complexType>
       <sequence minOccurs="0" maxOccurs="unbounded">
        <element ref="sip_method" />
       </sequence>
      </complexType>
    </element>

    <element name="sip_method" substitutionGroup="spds::setting">
     <annotation>
      <documentation>
       An element to specify a SIP method.  For user agents
       that support the method indicated, this element serves
       as a switch to enable or disable the named SIP method
       as indicated by the policy attribute.
      </documentation>
     </annotation>
     <complexType>
       <attributeGroup ref="spds:directional_setting_attributes" />
```

```
     </complexType>
    </element>

    <element name="sip_option_tags"
             substitutionGroup="spds::setting_container">
     <annotation>
      <documentation>
       Container for list of SIP option tags to allow or
       disallow.  For user agents that support features
       indicted by option tags, this element serves as a
       list of features to turn on or off as indicated by
       the policy attribute in the sip_option_tag element.
      </documentation>
     </annotation>
      <complexType>
       <sequence minOccurs="0" maxOccurs="unbounded">
        <element ref="sip_option_tag" />
       </sequence>
      </complexType>
    </element>

    <element name="sip_option_tag" substitutionGroup="spds::setting">
     <annotation>
      <documentation>
       An element to specify a SIP option tag.  For user agents
       that support the options indicated, this element serves
       as a switch to enable or disable the named SIP option
       as indicated by the policy attribute.
      </documentation>
     </annotation>
     <complexType>
       <attributeGroup ref="spds:directional_setting_attributes" />
     </complexType>
    </element>

   </schema>
```

Appendix B. Acknowledgments

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING
Internet-Draft
Expires: January 14, 2006

J. Rosenberg
Cisco Systems

July 13, 2005

Registration Coupled Subscriptions in the Session Initiation Protocol (SIP)
draft-rosenberg-sipping-reg-sub-00

Status of this Memo

Copyright Notice

Abstract

When a Session Initiation Protocol (SIP) user agent starts up, it registers to the network and initiates numerous subscriptions in order to learn about various network events. This results in a chatty startup procedure which substantially impacts recovery times under avalanche restart. This specification proposes a mechanism whereby the subscriptions can be established as a side effect of the registration, alleviating this problem.

Table of Contents

1. Introduction

When a Session Initiation Protocol (SIP) [1] user agent starts up, it typically follows a series of message exchanges with servers in the network. At a minimum, this startup procedure involves a SIP registration that allows the user agent to receive incoming requests. However, over time, numerous event packages [2] have been defined that provide a user agent with useful information through the duration of its connection to the network. These packages include:

Message Waiting: RFC 3842 [11] provides a message waiting indication event package. Typically, a user agent would subscribe to its own Address-of-Record (AOR) for this event package, in order to find out about messages that have been left for that user. This provides the familiar "message waiting lamp" on many business telephones. It is valuable for a user agent to subscribe to this package through the duration of its registration, in the event that messages are explicitly directed to a user's voicemail and do not ring their phone (this can happen, for example, if the caller utilizes the caller preferences specification [12] to direct a call to voicemail).

Registration Event: RFC 3680 [13] allows a user agent to learn about the status of its registration. Typically, a user agent would subscribe to its own AOR for this event package, in order to find out if the network has removed its registration. Such removals happen in cases of graceful network shutdown, or when a user needs to re-register and re-authenticate due to concerns on validity of credentials.

Presence List: A user may have a "buddy list", which contains a list of users whose presence is desired. A user will subscribe to their buddy list using an event list subscription [14] to the presence event package [15]. This is done by subscribing to a resource that is synonomous with the user's own buddy list.

Watcher Info: In order to find out about attempts that have been made to subscribe to a users presence, that user makes use of the watcher info event template package [16]. They would do this by subscribing to their own AOR with the presence.winfo event package. Subscription attempts that are unauthorized will result in a notification, informing the user of this fact and allowing them to approve or deny the subscription.

Dialog Events: Certain features, such as single line extension, require a user agent to find out about calls in progress on other user agents associated with the same AOR. This is done through subscriptions to the dialog event package [17]. The user agent

would typically subscribe to their own AOR, and learn about calls in progress to other user agents.

Configuration Events: The configuration event package [18] allows a UA to learn about changes in its configuration. This is done by having the UA subscribe to its own identity (which may be the AOR) for the config event package.

As a consequence of this, each time a user agent starts up, they will generate a REGISTER transaction, plus a SUBSCRIBE and a NOTIFY transaction for each event package the user agent is interested in. Based on the above discussion, this could be upwards of six event packages, resulting in a total of fourteen transactions that take place on startup. Furthermore, each of these subscriptions needs to be periodically refreshed (as does the registration), resulting in ongoing messaging.

This overhead is particularly problematic during an avalanche restart. This occurs when a failure event of some sort causes all user agents to simultaneously re-register. This is most common when recovering after a power outage. When the power returns, all the user agents will start booting simultaneously, and at the same time, each will execute their startup sequence. The more complex this sequence, the longer it takes for the system to return to service, and the more robust the network has to be. Another cause of avalanche restart is recovery after a catastrophic network failure, such as a network partition. If a network partition should last longer than the subscription lifetime, once the partition heals, each client will discover this and attempt to re-register and re-subscribe to each event package.

The overhead is also problematic on wireless links and other interfaces where bandwidth is at a premium.

2. Requirements

A solution to this problem should meet the following requirements:

1. The solution must substantially reduce the amount of SIP messaging traffic that takes place when a user agent starts up.

2. The solution must substantially reduce the amount of network processing that needs to take place when a user agent starts up.

3. The solution must not fundamentally alter the event model of RFC3265.

3. Proposed Solution

This document proposes a solution to this problem, based on the following observations:

1. In all of the above cases, the subscription is desired for the duration of the registration of the UA.

2. In all of the above cases, the user agent is subscribing to a resource which it owns; either its AOR or a related resource, like a buddy list. As a consequence, the authorization policies for the subscriptions always allow that user to subscribe. A policy in which a user can subscribe to their own events are called "self authorization".

3.1 Overview of Operation

Based on these observations, the approach proposed here is to strongly couple subscriptions with registrations, and to actually use the registration to create the subscriptions. A subscription that is created as a result of a successful registration is called a registration-coupled subscription. The basic approach is shown in

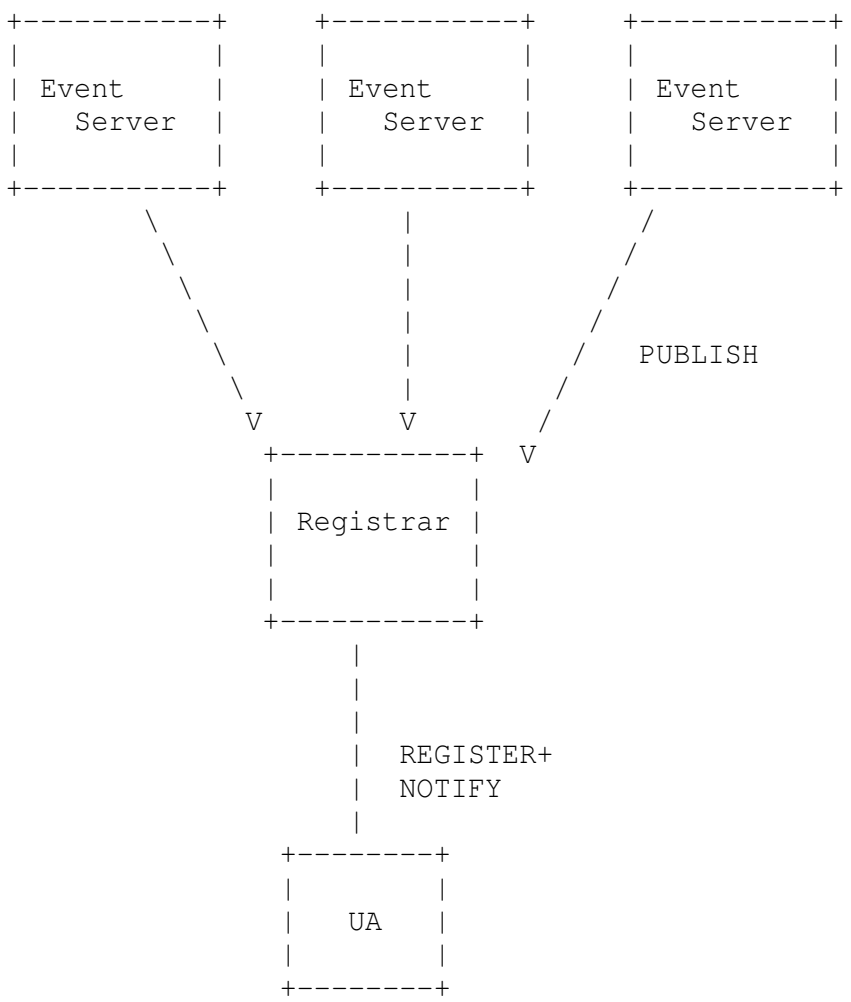


Figure 1

To create a registration-coupled subscription, a UA includes a Subscription header field in its REGISTER message. This header field includes a list of the desired event packages, and for each, the resource to which a subscription is desired and any event header field parameters. There is no need for a Require header field. The registrar looks for the Subscription header field. For each value, it examines the event package and target resource. If the resource is in the domain of the registrar, and the resource has an authorization policy of "self", and the registrar allows registration coupled subscriptions for that event package, the registrar creates the dialog and a subscription. The 200 OK to the REGISTER contains an indication of whether the subscription was created, and if so, the remote tag needed to complete the dialog identifier.

The UAC will create a dialog and a subscription for each value of the

Subscription header field in the response. As there will be one of these per event package, the end result is a single dialog for each event package that the client wants to subscribe to. Dialogs are not shared across event packages. The dialog identifiers are obtained by copying the Call-ID and local tag from the REGISTER, with the remote tag from the Subscription header field value. Similarly, the registrar will create a subscription. The dialog identifiers and local sequence number are set in the same way. Its route set is taken from the Path header field from the registration [4].

At this point, a proper subscription is established at the UA and the registrar. The registrar can send a NOTIFY at any time. The initial NOTIFY normally sent upon receipt of a SUBSCRIBE is not required, as the REGISTER response serves that purpose. The subscriptions are all refreshed through registration refreshes. If the UAC omits an event and resource from a Subscription header field in its REGISTER, it means that the client wishes to unsubscribe. Similarly, if the 200 OK to the REGISTER omits that event package and resource, it means that the subscription was terminated. However, the client cannot ever send a SUBSCRIBE to refresh the subscription. Any such request is rejected with a 403.

It is important to note that there is a dialog properly established as part of this mechanism. The dialog is established by providing the dialog parameters through the registration, and then to make the dialog state part of the registration state. The dialog is then refreshed and maintained just like registration state. If a user has multiple user agents registered to the same AOR, multiple dialogs would be created. This means that the dialogs terminate on the registrar as well. In order for events to be delivered to the clients in NOTIFY messages, an event server generates a PUBLISH message when it wants to send an event to a user agent. The PUBLISH is routed to the registrar, where it examines the URI in the request URI. If the user is registered, it goes through each registered contact. If the registration of that contact had created a coupled subscription, the registrar checks if the registration-coupled subscriptions include the event package in the PUBLISH. If they do, the registrar copies the event data in the body of the PUBLISH into a NOTIFY, and sends it to the user agent.

As an additional mechanism, the event servers themselves can subscribe to the registration event package for all subscribers. WHenever a user registers, a notification would get delivered to the event server. It can then check which users are registered or not, and use this information to determine whether or not it wishes to send a PUBLISH. Alternatively, the reg-event notifications can contian all of the information on the registration-coupled subscriptions - their dialog identifiers, event packages, and so on.

This would allow the event server itself to "take over" the subscription, and take ownership of the dialog. In that case, it can send the NOTIFY directly, instead of sending a PUBLISH to the registrar. Indeed, the event server can make a decision on a subscriber-by-subscriber basis as to whether it wishes to own the dialogs or not.

3.2 User Agent Behavior

A user agent SHOULD be configured with a set of event packages that it wishes to couple with its registrations. For each such package, when the client performs its initial registration, it includes a Subscription header field value into its request. That value contains the address-of-record for the target of the subscription. This AOR MUST be one within the same domain as the domain of registration. Typically, it will be the same as the AOR for the user themselves. The UA includes any parameters it would otherwise include in the Event header field into the Subscription header field. The UA SHOULD include an Accept header field in the request, and include the content types the client supports for that event package. Otherwise, the registration is generated identically to a normal registration.

If the response to the REGISTER is a 200 OK, the client looks for the Subscription header field. If the header field is not present, the user agent knows that either this mechanism is not supported in the registar, or is supported, but not in use for any of the event packages requested by the client. In that case, the user agent SHOULD proceed with a normal subscription according to the specifics of the event packages the client is interested in.

If the 200 OK response to the REGISTER did contain a Subscription header field, the user agent goes through each value. It constructs a dialog by setting the Call-ID to the value in the REGISTER response, the local tag to the From tag the client placed in the REGISTER request, and the remote tag from the value of the Subscription header field. The local URI is set to the value in the From header field of the REGISTER request, and the remote URI to the value in the To header field of the REGISTER request. The local and remote CSeq are initially empty. Since the client never sends a request within the dialog, the local CSeq never needs to be populated. Similarly, the route set is empty. If the REGISTER request was sent over TLS, and the Request-URI was a sips URI, the "secure" flag for the dialog is set.

The dialog state persists for the duration of the registration of that contact. When the UA determines that the contact expires, the dialog state is destroyed. A UA can determine that a contact has

expired because it times out and is not refreshed, or because the client receives a registration event notification informing it that the contact has been terminated.

If the client had included a Subscription header field in the request for a particular event package, and the REGISTER response contained a Subscrption header field, but that package was not listed, it means that the registrar is either refusing a subscription-coupled registration for that event package, or that subscription failed for some reason. To determine the exact problem, the client SHOULD perform a regular, separate subscription to that event package.

At any point during the lifetime of the registration, the client may receive a NOTIFY on the dialog created by the registration. Processing of that NOTIFY happens as described in the relevant event package and according to the details of RFC 3265.

A registration refresh occurs identically to an initial registration. A client MUST include a Subscription header field value for each dialog it wishes to retain. If a client omits a Subscription header field value for a particular event package, the dialog associated with that event package is terminated upon receipt of a 200 OK to the REGISTER request.

If a client wishes to perform a subscription with event filters that need to be placed in the body of a request, the mechanism here cannot be used. Rather, the client should perform a normal subscription using SUBSCRIBE. An alternative would be to include the event filters as a body of the REGISTER request. Header field parameters could associated each MIME body with a particular event package. However, this introduces a lot of complexity for a corner case. As such, this document recommends just performing a regular subscription to handle these cases.

3.3 Registrar Behavior

3.3.1 REGISTER Processing

When a registrar receives a REGISTER request, it processes the registration normally per RFC 3261. If the result would otherwise have been a successful registration resulting in a 200 OK, the procedures defined here are followed.

The registrar checks for the presence of the Subscription header field in the REGISTER request. The processing that follows is performed for each value of this header field. Firstly, the registrar checks to see if it supports registration-coupled subscriptions for that particular event package. Performing them for

any particular event package is a matter of local policy. Typically, it would be allowed when an event server is present in the network which supports the capabilities defined here. If the registrar doesn't support registration-coupled subscriptions for that event package, it goes on to the next value of the Subscription header field. Otherwise, processing continues.

Next, the registrar validates that the resource in the header field value is a valid resource within the domain of the registrar. If it is, processing continues. Otherwise, the registrar goes on to the next value of the Subscription header field. Next, it checks whether or not the UAC is authorized to subscribe to the resource. The means by which authorization occurs is outside the scope of this specification. Typically, registration-coupled subscriptions are performed with subscriptions where the authorization policy is such that a user is allowed to subscribe to themselves, and no others. This authorization policy, called "self", is readily provisioned on the registrar, and would not require complex interactions with other event servers. If the registrar cannot determine authorization, or if the subscription is not authorized, the registrar goes on to the next value of the Subscription header field. Otherwise, processing continues.

At this point, the subscription has been authorized. The registrar stores the event header field parameters in the Subscription header field value as part of the state associated with the registered contact. These parameters are carried as a quoted string in the Subscription header field, so that they are readily separable from the Subscription header field parameters. It also stores the event package. The registrar chooses a tag that will serve as the remote tag of the dialog, according to the procedures of RFC 3261. This tag is also stored as part of the state associated with the registered contact. The Call-ID and From tag from the REGISTER request would have already been stored as part of normal registration processing, as would the Path header field value. The registrar also stores the From header field of the REGISTER message.

In the 200 OK to the REGISTER request, the registrar includes the Subscription header field. Each value contains the event package name for each registration-coupled subscription that was created, along with the tag that completes the dialog. The AOR SHOULD NOT be included.

3.3.2 PUBLISH Processing

This specification allows a registrar to act as an event server for registration-coupled subscriptions. When the registrar receives a PUBLISH message for a particular address-of-record, it checks that

the PUBLISH has arrived from an event server that is authorized to publish events for the subscriber. Typically, this is done based on the maintentance of a TLS connection between the registrar and the event server, used to identify the source of the messages to the registrar. The registrar would typically authorize PUBLISH messages for a specific event package only if they came from a specific event server.

Once the sender of the PUBLISH is authorized, the registrar performs a registration query for the AOR in the Request-URI of the PUBLISH message. It checks to see if there are any contacts registered for that AOR that have registration-coupled subscriptions for that event package. For each contact it finds, the registrar constructs a NOTIFY message. The Call-ID of this NOTIFY is taken from the stored state associated with the registration. The From header field URI is set to the AOR of the user. The To header field URI is set to the value in the From header field of the most recent REGISTER message. The tag in the From header field is populated with the tag associated with the registration. The tag in the To header field is populated with the tag stored with the Contact. The Event header field of the NOTIFY is set to the event header field stored with the Contact. The body of the NOTIFY is taken from the body of the PUBLISH. The remainder of the NOTIFY is consructed as per RFC 3261, and then sent as a mid-dialog request.

The registrar then generates a 200 OK to the PUBLISH request. If the registrar found no matching registration-coupled subscriptions for the PUBLISH, it generates a 403 response to the PUBLISH request. This informs the event server that its event was not delivered.

3.4 Event Server Behavior

It is assumed that event servers learn about events for a particular package for a particular subscriber through any number of means. These can include non-SIP mechanisms, SIP subscriptions to a resource, and so on. However, they cannot include a SIP PUBLISH message sent to the AOR of the subscriber; those PUBLISH messages are routed to the registrar according to this specification.

An event server MAY act as the dialog owner, or MAY leave that responsibility to the registrar. However, it MUST NOT do both for the same subscriber within the duration of a registration from that subscriber. To act as a dialog owner, the event server subscribes to the registration event package. It MAY subscribe to this event package for each subcscriber individually, or it MAY subscribe to a resource that represents all subscribers or a group of users at the registrar (for example, sip:all-users@example.com). The latter is preferable since it avoids the need for per-user subscription

maintenance at the event server.

The notifications of the dialog event package will contain information on each registration-coupled subscripton for a subscriber. If the event server is acting as a dialog owner, it MUST store this information. Effectively, the reg-event notification creates the dialog state and the event subscription at the event server. When the event server wishes to send an event, it creates a NOTIFY using the dialog state and sends it, per RFC 3265 and RFC 3261 procedures. These NOTIFY messages won't even traverse the registrar.

If the event server is not acting as a dialog owner, when it wishes to send a notification, it sends a PUBLISH message. The request-URI of the PUBLISH is set to the AOR of the subscriber for whom a notification is to be delivered. The content of the PUBLISH contains the event state that is to be delivered to the watcher. The Event header field is populated with the value of the event package for which the notifications are intended. This PUBLISH message is sent, and will be routed to the registrar. The processing above will result in a NOTIFY being sent to each registered contact for that AOR.

The choice of whether to act as dialog owner or not depends on several factors. When the event server leaves dialog ownership to the registrar, it alleviates the need for the event server to maintain any kind of per-subscriber state. However, it imposes additional work on the registrar to perform the registration queries and construction of NOTIFY messages. Thus, this mode is useful for very infrequent events, such as a request to update a configuration profile in the configuration event package. Dialog ownership makes more sense for more frequent events. Also, since the registrar doesnt know the actual event state, it cannot send an initial NOTIFY with the current state when the dialog is first created. It relies on the event server to do that. As a result, if an event package requires state to be delivered as part of a NOTIFY generated when the subscription is created, the event server needs to maintain ownership of the dialog, or the hybrid model below needs to be used.

A hybrid model is also possible. An event server can receive reg-event notifications, but not store dialog state. When it sees that the user has registerd or unregistered, it can send a PUBLISH message. This is useful for infrequent notifications that need to be triggered on registration. The hybrid model also allows the event server to generate a PUBLISH when a client first registers, that contains the current value of the event state. This will cause the registrar to send a NOTIFY message with the current state. This is useful for event packages where it is desireable to send event state as part of the initial NOTIFY.

The hybrid model is particularly attractive, since it alleviates the need for the event server to maintain any kind of dialog state or per-subscriber subscription state, and yet it allows for the full features of a traditional event subscription.

3.5 Subscription Header Field

The grammar for the Subscription header field is:

```
Subscription    =  "Subscription" HCOLON (sub-param *(COMMA
                   sub-param))
sub-param       =  event-type *(SEMI sub-param)
sub-event-param =  sub-aor / sub-event-param / tag-param / generic-param
sub-aor         =  "aor" EQUAL (SIP-URI / SIPS-URI)
sub-event-param =  "e-param" EQUAL quoted-string
```

Figure 3 and Figure 4 are an extension of Tables 2 and 3 in RFC 3261 [1] for the Subscription header field. The column "INF" is for the INFO method [5], "PRA" is for the PRACK method [6], "UPD" is for the UPDATE method [7], "SUB" is for the SUBSCRIBE method [2], "NOT" is for the NOTIFY method [2], "MSG" is for the MESSAGE method [8], "PUB" is for the PUBLISH method [9], and "REF" is for the REFER method [10].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	REF
Subscription	R	-	-	-	-	-	-	o	-
Subscription	2xx	-	-	-	-	-	-	o	-

Figure 3: Subscription header field

Header field	where	proxy	PRA	UPD	SUB	NOT	INF	MSG	PUB
Subscription	R	-	-	-	-	-	-	-	-
Subscription	2xx	-	-	-	-	-	-	-	-

Figure 4: Subscription header field

3.6 Examples

3.6.1 Registrar has Dialog Ownership

In this example, the registrar holds ownership of the dialog. The event server is a message waiting indicator server that publishes MWI events.

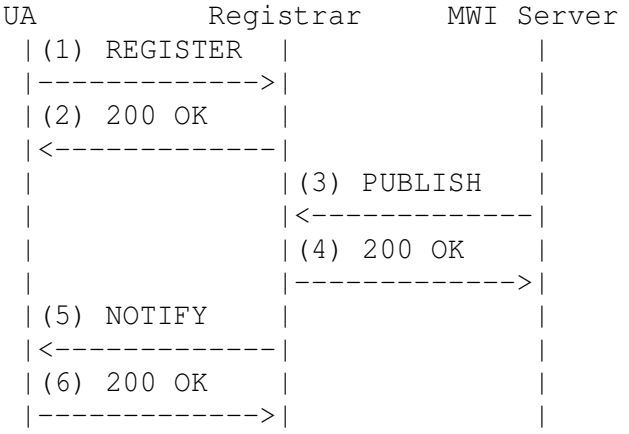


Figure 5: Registrar Owned Dialogs

The REGISTER message (1) would look like:

```
REGISTER sip:example.com SIP/2.0
To: sip:joe@example.com
From: sip:joe@example.com;tag=asd9887g
Subscription: message-summary;aor=sip:joe@example.com
Expires: 3600
Via: SIP/2.0/UDP client.biloxi.example.com;branch=z9hG4bKnashds7
Max-Forwards: 70
Call-ID: 1j9FpLxk3uxtm8tn@biloxi.example.com
CSeq: 1 REGISTER
Content-Length: 0
Contact: sip:client.biloxi.example.com
```

The 200 OK to the REGISTER indicates successful creation of the dialog:

```
SIP/2.0 200 OK
To: sip:joe@example.com;tag=99j9jj
From: sip:joe@example.com;tag=asd9887g
Subscription: message-summary;tag=ghghghg
Expires: 3600
Via: SIP/2.0/UDP client.biloxi.example.com;branch=z9hG4bKnashds7
Max-Forwards: 70
Call-ID: 1j9FpLxk3uxtm8tn@biloxi.example.com
CSeq: 1 REGISTER
Content-Length: 0
```

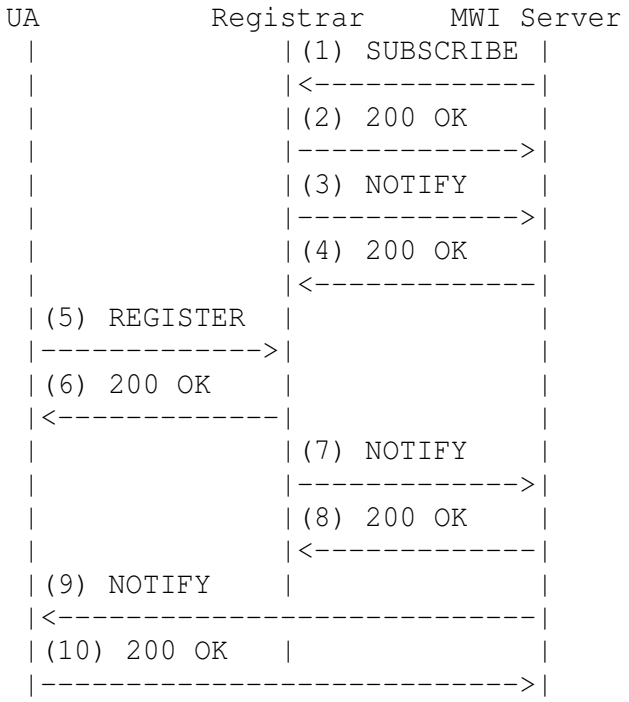
The PUBLISH from the event server comes when a new message arrives:

```
PUBLISH sip:joe@example.com SIP/2.0
To: sip:joe@example.com
From: sip:mwi-server@example.com
Event: message-summary
Via: SIP/2.0/UDP mwi.example.com;branch=z9hG4bKnashas--d9
Call-ID: 3k9FpLxhg88asd7m8tn@mwi.example.com
CSeq: 1 PUBLISH
Content-Type: application/simple-message-summary
Content-Length: ---

Messages-Waiting: yes
Message-Account: sip:joe@mwi.example.com
Voice-Message: 2/8 (0/2)
```

This results in a notification from the registrar:

```
NOTIFY sip:client.biloxi.example.com SIP/2.0
To: sip:joe@example.com;tag=asd9887g
From: sip:joe@example.com;tag=ghghghg
Event: message-summary
Via: SIP/2.0/UDP reg.example.com;branch=z9hG4bKnashas--d10
Call-ID: 1j9FpLxk3uxtm8tn@biloxi.example.com
CSeq: 1 NOTIFY
Content-Type: application/simple-message-summary
Content-Length: ---

Messages-Waiting: yes
Message-Account: sip:joe@mwi.example.com
Voice-Message: 2/8 (0/2)
```

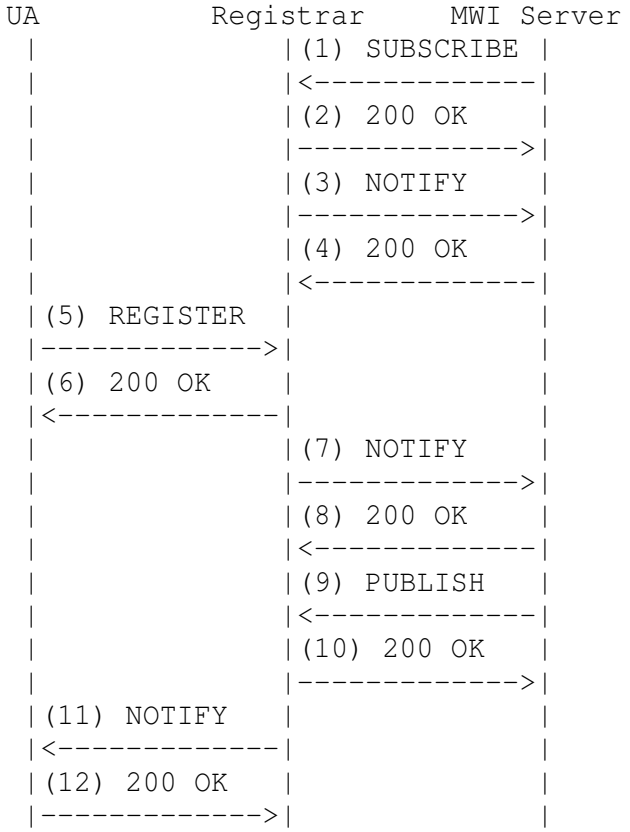
3.6.2 Event Server Owned Dialog



When the message waiting server starts up, it subscribes to the registration event package at the registrar (message 1). The request URI identifies all users in the domain. This generates a 200 OK (message 2), followed by a NOTIFY (message 3). This NOTIFY doesn't contain any event state (there is too much), but it confirms the subscription.

At some point later, the UA in question registers. The registration sequence (messages 5/6) are as above. This causes a reg-event NOTIFY to be sent to the mwi server (message 7). This tells the server about the creation of a new contact, and also tells it that a MWI registration-coupled subscription was created. It provides the dialog identifiers to the MWI server. Next, the MWI server generates a NOTIFY to tell the client about the event state (9).

3.6.3 Hybrid Model



When the message waiting server starts up, it subscribes to the registration event package at the registrar (message 1). The request URI identifies all users in the domain. This generates a 200 OK (message 2), followed by a NOTIFY (message 3). This NOTIFY doesn't contain any event state (there is too much), but it confirms the subscription.

At some point later, the UA in question registers. The registration sequence (messages 5/6) are as above. This causes a reg-event NOTIFY to be sent to the mwi server (message 7). This tells the server about the creation of a new contact, and also tells it that a MWI registration-coupled subscription was created. It provides the dialog identifiers to the MWI server. However, instead of sending the NOTIFY, the MWI server discards the dialog information. It sends a PUBLISH request (message 9) identically to the case where the registrar owns the dialog. This causes the registrar to send the notification (message 11).

4. References

4.1 Normative References

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

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

[3] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration", RFC 3608, October 2003.

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

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

[6] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

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

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

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

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

4.2 Informative References

[11] Mahy, R., "A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP)", RFC 3842, August 2004.

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

[13] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

[14] Roach, A., Rosenberg, J., and B. Campbell, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists", draft-ietf-simple-event-list-07 (work in progress), January 2005.

[15] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[16] Rosenberg, J., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)", RFC 3857, August 2004.

[17] Rosenberg, J., "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-06 (work in progress), April 2005.

[18] Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-06 (work in progress), February 2005.

Author's Address

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

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

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

SIPPING Working Group
Internet-Draft
Expires: January 10, 2006

H. Schulzrinne
Columbia University
E. Shim
Panasonic
July 9, 2005

Recommended Relationships between Different Types of Identifiers. draft-schulzrinne-sipping-id-relationships-00.txt

Status of this Memo

Copyright Notice

Abstract

Evolution of communications technologies brought us various types of identifiers or addresses that identify persons such as SIP URIs, email addresses, and telephone numbers. Proper relationships among different type of identifiers can enable various services, which, otherwise, are difficult to realize. This memo provides guidelines for managing relationships between SIP URIs other personal identifiers.

1. Introduction

In the absence of widely-deployed public directories, users often have only partial information about the various communication URI schemes for people they are trying to reach. They might have an old business card or RFC, typically containing a phone number or email address, but may need to contact the individual by some other means, such as via SIP or XMPP. Usage of newer protocols is facilitated if a communicating party is likely to be able to obtain such addresses.

A number of communications-related URIs, such as for email [4] [5], SIP [1] and XMPP [6] use the basic 'user@host' form. Particularly since implementations often allow usage of such identifiers without prefixing it with the URI scheme, non-technical users expect these identifiers to work across different means of communication and, in particular, expect that they reach the same person if they do work.

In some cases, if a SIP or other presence-related address such as an xmpp URI is known, one can try to subscribe to that address, with the presence object possibly returning the email address. However, this is not likely to work consistently, particularly since revealing presence information requires more trust than simply revealing one's email address.

Thus, given the limitations of electronic means of relating different communications-related URI schemes for individuals and services, users are likely to guess. Communication is facilitated and communication failures are prevented if identifiers are constructed in a predictable and consistent manner.

This document makes two core recommendations:

(1) Individuals should be able to choose user identifiers across URI schemes that are the same.

(2) Assignment policies within a domain should not assign the same user part in different URI schemes to different individuals.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [2].

SIP URI: Uniform Resource Indicators identifyng communication resources for SIP as defined in Section 19, RFC 3261 [1].

Its general form is:

sip:user:password@host:port;uri-parameters?headers.

SIPS URI: Same as SIP URI except that the SIP protocol runs on top of the Transport Layer Security (TLS) protocol [3]. It is also defined in Section 19, RFC 3261 [1]. Its general form is the same as the SIP URI except that it starts with 'sips:' rather than 'sip:'.

SIP address: A SIP URI or SIPS URI.

telephone number: A string of decimal digits that uniquely indicates the network termination point. The string contains the information necessary to route the call to this point. There are two categories of telephone numbers: public telephone numbers and private telephone numbers. This definition is from RFC 3966 [7] which derived the definition from [11].

tel URI: A resource identifier from a telephone number as defined in RFC 3966[7].

email address: A character string that identifies a user to whom mail will be sent or a location into which mail will be deposited. The standard email address naming convention is defined to be "user@host". A more rigorous definition can be found in RFC2821 [4] and RFC2822 [5].

3. Recommended Practices

3.1 A SIP address and an email address with the same user and domain parts

A SIP address MUST NOT have the same user and domain parts as an email address unless both refer to the same person or service.

Therefore, a SIP address and an email address with the same user and domain parts MUST refer to the same person or service. For example, the following SIP address and email address

sip:bob@example.com:5060;transport=udp

(mailto:)bob@example.com

MUST refer to the same person.

3.2 Two SIP addresses with the same user and domain parts

Any two SIP addresses MUST NOT have the same user and domain parts unless both refer to the same person or service.

For example, the following SIP addresses

sip:bob@example.com:5060;transport=udp

sips:bob@example.com;transport=tcp

MUST refer to the same person or service even though they are not equivalent according to the SIP specification [1] .

3.3 A SIP address and its email-equivalent

All SIP addresses SHOULD have a working email-equivalent as long as the SIP addresses are referring to people.

For example, for the following SIP address

sip:bob@example.com:6000;transport=tcp

a working email-equivalent SHOULD exist, such as

(mailto:)bob_the_builder@example.net.

The above example illustrates that the working email-equivalent does not have to have the same user and domain parts as the SIP address. How to find the email-equivalent for a given SIP address is out of scope.

If a SIP address refers to a telephone (number), it MAY not have an email-equivalent.

3.4 A tel URI and its email-equivalent

A tel URI MAY not have an email-equivalent.

3.5 An email address and its SIP-equivalent

Some email addresses may not have SIP equivalents, e.g., because the domains don’t support SIP services.

3.6 Email user names and SIP user parts

Providers of SIP services SHOULD allow all valid email user names as SIP address user parts.

3.7 A telephone number and its SIP-equivalent

Telephone numbers are mapped to SIP URIs without visual separators (hyphen, etc.), as partially described in the tel URI RFC [7] and the SIP RFC [1]. The parameter 'user' with its value 'phone' SHOULD be included in the SIP URIs.

For example, the following public telephone number

+1-212-555-1234

is mapped to the following SIP URI

sip:+12125551234;user=phone.

4. Use Cases

Below are just two example use cases showing the benefits that can be accrued by the recommended relationships in the above.

4.1 Leaving voicemails using emails in P2P IP telephony systems

Let say there are two identifiers, a SIP URI sip:bob@example.com and an email address bob@example.com. Imagine that there is no voicemail server associated with sip:bob@example.com and the human user owning the SIP URI could not take a call when another user called at the URI. It would be very useful if the caller can leave a voicemail by email. This scenario is particularly useful when SIP UAs are operating in a peer-to-peer fashion. Peer-to-peer networks for SIP-based communications were discussed recently in several drafts [8][9][10] . If the email address bob@example.com is assigned to the same person owning sip:bob@example.com by a global rule, it is straightforward to which email address the voicemail should be emailed. Instead, if the email address bob@example.com happens to be assigned to a different person, the caller will end up leaving the voicemail to a wrong person.

4.2 Common authentication for IP telephony and email systems

An organization, in their deployment of a SIP-based IP telephony system, set a policy that the SIP URI and the email address with the same user information and the host information components, i.e.

identical except the protocol component should be assigned only to the same user and let the users use their email system usernames and passwords for authentication with the IP telephony system, reducing the administration overhead and increasing the convenience of users at the same time.

5. Security Considerations

One could argue that making identifiers the same across communication means is likely to increase undesirable communication, such as spam. However, communication identifiers are often short and easily guessable, so that those intent on sending spam can exhaustively search the namespace until a working address has been found. Similarly, a single instance of an address "leaked" on a web page is often sufficient to introduce the address into the pool of spam-receiving addresses. Thus, the protection of address hiding appears to be limited, but the negative impact on desirable communication is clear. It is not the role of this document to force users to make such a trade-off between the possible benefits of address hiding and easier reachability, but rather to facilitate such choice.

This document therefore does not require that users choose the same 'user' part, but suggests that providers of such services make it easy for users to choose such a convention.

Preventing two users to share the same identifiers across URIs increases security, as it makes it less likely that a user sends confidential information to the wrong destination, in the mistaken belief that they are owned by the same person.

6. Acknowledgments

Arjun Roychowdhury's comments are appreciated.

7. References

7.1 Normative References

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

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

[3] Alen, C. and T. Dierks, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[4] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[5] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[6] Saint-Andre, Ed., P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

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

7.2 Informative References

[8] Johnston, A., "SIP, P2P, and Internet Communications", draft-johnston-sipping-p2p-ipcom-01 (work in progress), March 2005.

[9] Bryan, D. and C. Jennings, "A P2P Approach to SIP Registration", draft-bryan-sipping-p2p-00 (work in progress), January 2005.

[10] Philip, P. and B. Poustchi, "Industrial-Strength P2P SIP", draft-matthews-sipping-p2p-industrial-strength-00 (work in progress), February 2005.

[11] ITU-T, "The International Public Telecommunication Number Plan", Recommendation E.164, May 1997.

Authors' Addresses

Henning Schulzrinne
Department of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA

Email: schulzrinne@cs.columbia.edu

Eunsoo Shim
Panasonic Digital Networking Laboratory
Two Research Way, 3rd Floor
Princeton, NJ 08540
USA

Email: eunsoo@research.panasonic.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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

Session Initiation Protocol
Internet-Draft
Expires: December 29, 2005

R. Shacham
H. Schulzrinne
Columbia University
W. Kellerer
S. Thakolsri
DoCoMo Eurolabs

June 27, 2005

Specifying Media Privacy Requirements in the Session Initiation Protocol (SIP)
draft-shacham-sip-media-privacy-00

Status of this Memo

Copyright Notice

Abstract

Participants in a normal phone conversation can assume that, given the appropriate measures are taken against network eavesdropping, what they say is only heard by the other participant. The use of speakerphones or visual output devices displaying video or messaging

removes this assumption. In the Session Initiation Protocol (SIP), a call may also be transfered to another device, suddenly compromising the other participant's privacy. Therefore, this document proposes SDP and SIP protocol extensions that allow participants to specify their privacy requirements for the other party's device, and discusses how they may be used in different session scenarios. It also defines an SDP extension for allowing or disallowing the recording of the session.

1. Overview

Participants in a normal phone conversation can assume that what they say is only heard by the other participant. The use of speakerphones or visual output devices displaying video or messaging removes this assumption. In the Session Initiation Protocol (SIP) [1], a call may also be transfered to another device, as specified in [4], suddenly compromising the other participant's privacy. This document proposes two protocol extensions to be used in SIP sessions that allow participants to specify their privacy requirements: an extension to Caller Preferences [2] and two new attributes in the Session Description Protocol [3].

The two methods, together, aim to support privacy in a number of ways during a session. These ways apply either during call setup or in the middle of a call. During call setup, the call will only be set up on devices that satisfy the privacy requirements of each party. Although a device may support a certain level of privacy, a specific use of the device, such as a mobile phone's speakerphone capability, may compromise this privacy. Therefore, the device that processes the request should disallow such use, or at least warn its user that the other party has requested a private conversation.

Once a session has been established, a user may try to alter it in ways that compromise the intended privacy. For instance, he may choose to turn on the speakerphone or transfer the audio to a speaker system in the room. The information ascertained in the call setup must govern the entire session, so that such changes are disallowed. If the device does not exercise such control on the user, the remote device may still block undesirable changes in the session. While it will have no recourse if the device allows the speakerphone to be activated, it can block attempted session transfer.

If the content of the session becomes more private, a participant may wish to update the privacy restrictions. For instance, he may need to give private information such as a credit card number, and would like to ensure that only the other user can hear. If the other device currently provides sufficient privacy, the update serves to notify the other device of the change, so that future changes will be

disallowed. If the other device is currently not private enough, the remote participant is taking an active role in ensuring that the other participant is using an appropriate device. It may remotely force the other user's device to retrieve a session currently on another device, or output audio to the user's earpiece which is currently being heard on the phone's speaker.

In addition to the level of privacy of a session, a participant may also be concerned about its recording. This document proposes a way for a user to allow or disallow recording of the session.

2. Device privacy levels

This document proposes a three-level system for characterizing the privacy of a device based on who can see or hear its output. The levels, in descending order of privacy are "user", "organization", "public". The privacy level "user" indicates that the communication between the participants cannot be heard or seen by any other person. A loudspeaker may still be on if the user is in the room alone. The level "organization" means that the communication may only be perceived by those with whom the device user shares an affiliation, such as a company, an institution or a group of friends. The other participant need not have any affiliation with the organization. The level "public" indicates that there are no restrictions on privacy. The device may change its level based on circumstances in its environment. For instance, a speakerphone in a company conference room may have privacy level "user" as long as nobody besides the user is present. Once other users are detected through a mechanism such as an identification card reader (which detects specific users) or a sensor at the door (which simply detects traffic), the phone would update its privacy level to "organization". If, after the change, the device no longer provides a level of privacy sufficient for the session, based on previously conveyed information as described in Section 1, it should either take the proper action to make itself more private, notify its user, or notify the other participant.

3. Caller preferences for privacy

Caller preferences [2] are a set of extensions to SIP which allow a caller to specify how his request should be handled by a server. The extensions consist of three headers, "Request-Disposition", "Reject-Contact", "Accept-Contact". "Request-Disposition" is used to specify the process by which the server should choose the contact of the recipient to which to route the call, while the other two headers are used to require specific attributes in the chosen contact or give preference to contacts with certain attributes. This document proposes extending the framework to include a new feature preference called "privacy" which may take any of the three values mentioned

above in Section 2. The caller may include this parameter in the "Reject-Contact" header to disallow the call being routed to any device with a privacy level lower than or equal to that specified. For example, including the following header would keep the request from being routed to any device which would allow others to see or hear the output:

Reject-Contact: *;privacy="organization"

Alternatively, the use of the "Accept-Contact" header can be used to give preference to a device higher than or equal to a given privacy level. Using the "require" parameter will ensure that the call will only be proxied to such a device. The same selection criteria conveyed in the above header can be conveyed as follows:

The registration of a user's contact must include the privacy level of the device using the specifications in [5] in order to allow the proxy to match the correct contact with the caller's request. A device which has multiple modes with different privacy levels, such as a phone with a speakerphone capability, should specify the highest privacy level that it provides. Once it has received the request, it should limit the use of the device in accordance with the information contained therein. For example, the registration sent for a user on an IP phone that has a speakerphone capability would include the following header:

Contact: <sip:bob@phone1.example.com>;privacy="user"

4. SDP attributes for privacy

We specify an extension to SDP to allow a session to be negotiated based on privacy requirements. Two attributes are defined, "provided-privacy" and "required-privacy", each a value attribute which may take any of the values specified in Section 2. An SDP description may include either or both of the attributes. We currently define this attribute as being only session-level, for the sake of simplicity and since that granularity is likely to suffice for most uses. An example fragment of an SDP body follows:

```
c=IN IP4 212.78.32.6
a=provided-privacy:user
a=required-privacy:user
```

Here, the sender notifies the other party that he is able to provide user-level privacy, and requires the same level from the other device in order to establish a session. This attribute is treated as any other in the offer/answer exchange, in that the recipient must be

able to provide privacy at least at the specified level in order to establish the session.

5. Providing privacy

The two extensions described may be used concurrently to provide the privacy services in Section 1. Including the "Reject-Contact" or "Accept-Contact" headers in a request will route the call to an appropriate device. Since caller preferences are only defined for the initiator of a session, the callee must use, in its response, the SDP extensions described in order to require that the caller use a device that is sufficiently private.

Once the two user devices have established a session between them, the information exchanged during call setup is also used to limit the use of the device. If a device has received a request with either the "Reject-Contact" header, the "Accept-Contact" header or a response that includes a "a=required-privacy" line, it must not, for the duration of the session, allow the user to make any adjustments to the session that violate whatever privacy requirements are contained therein. If the other participant has required "user" level privacy, the device must not allow the user to turn on the speakerphone at any time unless it has a way of knowing that there is no other user in the area. Likewise, it must not allow the user to transfer the call to a device where the media may be seen or heard by others.

If a device still allows the user to attempt a transfer, the remote device may stop it from taking place. There are two session transfer modes mentioned in [4], Mobile Node Control (MNC) mode and Session Handoff (SH) mode. In both, the flows involve a request/response interaction with the remote user's device. The remote device may specify in the body of its response that it will only allow sessions with devices that have a specific level of privacy, thereby not allowing the session transfer to take place otherwise.

If either party wishes to update the privacy of the call, the SDP attribute must be used, since caller preferences are not defined for mid-call messages. As described in Section 1, this may simply make the other device aware of the increased privacy of the session or may actually remotely force a change on the other device. For example, in order to update the session privacy to "user" level, a participant sends an INVITE request with the SDP "required-privacy" attribute set to "user". If the call is currently on the user's own device, the device responds with its own SDP parameters, as it normally would. If the user is currently using the speakerphone, the device redirects the output to the earpiece. If the user currently has part of the call on an external device which may be perceived by other users, his

own device must retrieve the session in order to comply with the session update. It responds with an acceptable SDP body, namely its own parameters, and subsequently terminates its own session with the local audio device in order to remove the remote participant's media from there.

6. SDP attribures for recording

Since the recording of a session is not an intrinsic attribute of a device and does not effect call routing, it is not useful to express it as a caller preference. Rather, this document defines two SDP attributes that may be used to express information about session recording. As in Section 4, we only define these attributes at the session level for the sake of simplicity. An SDP description containing the "record" attribute conveys to the other participant that he would like to allow recording of the session, which is likely to mean that he will, in fact, record it. The SDP may also contain the "norecord" attribute to convey that the user is requesting to disallow recording of the session. These attributes may be used during call setup or in mid-call.

7. Security Considerations

This document, itself, is concerned with providing SIP session participants with their desired level of privacy. It is only concerned with conveying to the other participant requirements for handling media streams once they are received. It does not provide for the confidentiality or integrity of media streams, which are provided by the Secure Real-time Transport Protocol (SRTP) [6].

8. IANA Considerations

This document defines a new feature parameter called "privacy" whose use is governed by [2]. It must take one of the following values: "user", "organization", "public".

It also defines four SDP attributes. Two of these attributes serve to allow for negotiation based on the privacy level of the devices. The attributes, "required-privacy" and "provided-privacy", are session-level value attributes which must take one of the values listed above. The other two attributes, "record" and "norecord" serve to allow and disallow the recording of the session. They are session-level property attributes.

9. References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Sparks, R., Handley, A., and E. Schooler, "SIP: Session

Initiation Protocol", RFC 3261, June 2002.

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

[3] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[4] Shacham, R., Schulzrinne, H., Thakolsri, S., and W. Kellerer, "Session Initiation Protocol (SIP) Session Mobility, IETF Internet Draft (Work in Progress)", February 2005.

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

[6] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

Authors' Addresses

Ron Shacham
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA

Email: rs2194@cs.columbia.edu

Henning Schulzrinne
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA

Email: hgs@cs.columbia.edu

Wolfgang Kellerer
DoCoMo Eurolabs
Landsberger Str. 312
Munich 80687
Germany

Email: kellerer@docomolab-euro.com

Srisakul Thakolsri
DoCoMo Eurolabs
Landsberger Str. 312
Munich 80687
Germany

Email: thakolsri@docomolab-euro.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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