SIPPING Working Group
Internet-Draft
Expires: October 6, 2006

V. Hilt
Bell Labs/Lucent Technologies
G. Camarillo
Ericsson
J. Rosenberg
Cisco Systems
April 4, 2006

A Framework for Session Initiation Protocol (SIP) Session Policies
draft-ietf-sipping-session-policy-framework-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on October 6, 2006.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

Proxy servers play a central role as an intermediary in the Session Initiation Protocol (SIP) as they define and impact policies on call routing, rendezvous, and other call features. However, there is currently no standard mechanism by which a proxy can influence session policies such as the codecs or media types to be used. This

document specifies a framework for SIP session policies. It defines two types of session policies, session-specific and session-independent policies and introduces a model, an overall architecture and the protocol components needed for session policies.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [6] is a signaling protocol for creating, modifying and terminating multimedia sessions. A central element in SIP is the proxy server. Proxy servers are intermediaries that are responsible for request routing, rendezvous, authentication and authorization, mobility, and other signaling services. However, proxies are divorced from the actual sessions - audio, video, and messaging - that SIP establishes. Details of the sessions are carried in the payload of SIP messages, and are usually described with the Session Description Protocol (SDP) [7]. Indeed, SIP provides end-to-end encryption features using S/MIME, so that all information about the sessions can be hidden from eavesdroppers and proxies alike.

However, experience has shown that there is a need for SIP intermediaries to impact aspects of a session. For example, SIP may be used in a wireless network, which has limited resources for media traffic. During periods of high activity, the wireless network provider wants to restrict the amount of bandwidth available to each individual user. With session policies, an intermediary in the wireless network can inform the user agent about the bandwidth it currently has available. This information enables the user agent to make an informed decision about the number of streams, the media types, and the codecs it can successfully use in a session.

In another example, a SIP user agent is using a network which is connected to the public Internet through a firewall or a network border device. The network provider would like to tell the user agent that it needs to send its media streams to a specific IP address and port on the firewall or border device to reach the public Internet. Knowing this policy enables the user agent to set up sessions across the firewall or the network border. In contrast to other methods for inserting a media intermediary, the use of session policies does not require the inspection or modification of SIP message bodies. Other user cases for session policies are described 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.

Two types of session policies exist: session-specific policies and session-independent policies. Session-specific policies are policies that are created for one particular session, in response to the session description for this session. For example, a session-specific policy may modify the IP addresses and ports of media streams in a specific session. Since session-specific policies are tailored to a session, they only apply to the session they are created for. Session-specific policies are created on a session-by-session basis during the establishment of the session.

Session-independent policies on the other hand are policies that are created independent of a session and generally apply to the SIP sessions set up by a user agent. Since these policies are not based on a specific session description, they can be created and conveyed to the user agent at any time, independent of an attempt to set up a session.

This specification defines a framework for SIP session policies. It specifies a model, the overall architecture, and the protocol components that are needed for session-independent and 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. Session-Independent Policies

This section defines a model and the protocol components for session-independent policies.

3.1. Overview

Setting up session-independent policies involves the following steps:

1. A user agent requests session-independent policies from the policy servers in the local network and home domain. These two domains most likely have session-independent policies for a user agent. A user agent typically request these policies when it

starts up or connects to a new network domain.
2. The policy server selects the policies that apply to this user agent. The policy server may have general policies that apply to all users or maintain separate policies for each individual user. The selected policies are returned to the user agent.
3. The policy server may update the policies, for example, when network conditions change.

3.2. Protocol

A UA subscribes to session-independent policies using the "ua profile" event package defined in the Framework for SIP User Agent Profile Delivery [4]. This event package has been designed to support subscriptions to user agent configuration information as well as to session-specific policies. A server can provide session-independent policies and configuration information through the same subscription. However, it is expected that session-independent policies and configuration information will often be provided by different servers, which may even be in different domains. In addition, session-independent policies may change more frequently than configuration information since they may consider external information, such as the network status.

3.2.1. Subscription

Session-independent policies are usually provided by the network domain the UA is physically connected to (i.e. the local network domain). This domain may, for example, have policies that limit the bandwidth currently available to each user. Frequently, the domain a user registers at (i.e., the domain in the address-of-record (AoR)) will also provides session-independent policies. This domain may, for example, provide policies needed for a service the user has subscribed to.

The "ua profile" event package [4] provides a mechanism to discover policy servers in these two domains. The "localnetwork" profile-type enables a UA to discover a servers in the local network domain; the "user" profile type enables the discovery of a server in the AoR domain. A UA SHOULD attempt to discover and subscribe to the policy servers in these two domains.

A UA SHOULD create a SUBSCRIBE request when the following events occur:

- The UA registers a new AoR or it removes a AoR from the set of AoRs it has registered. In these cases, the UA SHOULD establish subscriptions for each new AoR using the "user" and the "localnetwork" profile-types. The UA SHOULD terminate all

subscriptions for AoRs it has removed.

- The UA changes the domain it is connected to. The UA SHOULD terminate all existing subscriptions for the "localnetwork" profile-type. It SHOULD then create a new subscription for each AoR using the "localnetwork" profile-type. This way, the UA stops receiving policies from the previous local domain and starts receives policies from the new local domain instead. The UA 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-independent policies.

3.2.2. Content

The "ua profile" event package is an abstract event package that does not define a default content type for subscriptions. A user agent subscribing to session-independent policies SHOULD include the MIME type for the Schema for SIP User Agent Profile Data Sets [9] and the "application/session-policy+xml" format [3] in the Accept header of a SUBSCRIBE request. The Schema for SIP User Agent Profile Data Sets is an abstract format for configuration data that is extended by the "application/session-policy+xml" format for media-related policies. These are the default formats for subscriptions to session-independent policies and MUST be supported by a user agent compliant to this specification.

A policy server MAY send a notification to the subscriber every time the session-independent policy covered by the subscription changes. The definition of what causes a policy to change is at the discretion of the administrator. A change in the policy may be triggered, for example, by a change in the network status or simply by an update of the service level agreement with the customer. The session-independent policy contained in notification MUST represent a complete session-independent policy. Deltas to previous policies or partial policies are not supported.

4. Session-Specific Policies

This section defines a model, an architecture and the protocol components for session-specific policies.

4.1. Architecture

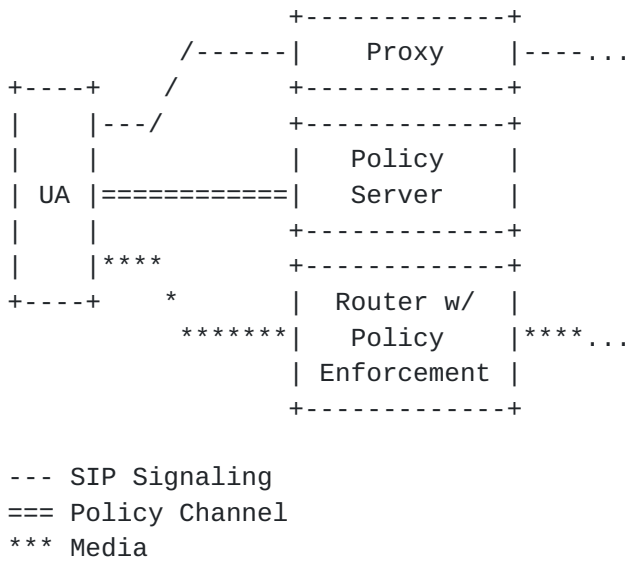


Figure 1

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 role of the proxy 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 each UA knows 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. The role of the policy server is to deliver session policies to UAs. The policy server receives session information, uses this information to determine the policies that apply to the session and returns the corresponding session policy to the UA. The mechanism for generating policies (i.e. making policy decisions) is outside the scope of this specification. A policy server may, for example, query an external entity to get the policies that apply to a session or it may directly incorporate a policy decision point and generate policies locally.

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 mechanism to convey session information to a policy server and to return the policies that apply to a session to the UA.

4.2. Overall Operation

The protocol defined in this specification clearly separates SIP signaling and the exchange of policy information. 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 policy information between endpoint and a policy server in the network is piggybacked onto the SIP signaling messages that are exchanged between endpoints.

The main advantage 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 a policy server. This decoupling provides a number of desirable properties. It enables the use of separate encryption mechanisms on the signaling path (to secure the communication between endpoints) and on the policy channel (to secure the communication between endpoint and policy server). Policies can be submitted directly from the policy server to the endpoint and never travel along the signaling path, possibly crossing many domains. Endpoints set up a separate policy channel to each policy server and can specifically decide which information they want to disclose to which policy server. Finally, policy servers do not need to rely on a SIP signaling message flowing by to send policies or policy updates to the endpoint. A policy server can use the policy channel at any time to update session policies as needed. A disadvantage of the separate channel model is that it requires additional messages for the exchange of policy information.

Following this model, the signaling for session-specific policies involves the following two fundamental tasks:

1. UA/policy server rendezvous: a UA setting up a session needs to be able to discover the policy servers that are relevant to this session. In principle, each domain that is traversed by the signaling messages of a session can have session policies in place and therefore run a policy server. However, session-specific policies are usually only provided by the local domain of the user agent.

2. Policy channel: once the UA has discovered the relevant policy servers for a session, it needs to retrieve the policies that apply to the current session from these servers.

The exchange of policy information on the policy channel follows the model described below:

1. A user agent submits a session description to the policy server and asks whether a session using this session description is permissible.
2. The policy server generates a policy decision for this session and returns the decision to the user agent. Possible policy decisions are to (1) deny the session, (2) propose changes to the session description with which the session is acceptable, or (3) accept the session as it was proposed.
3. The policy server possibly updates the policy decision at a later time.

The protocol mechanisms for UA/policy server rendezvous and the mechanism used on the policy channel are discussed in the following sections.

4.3. Examples

This section provides two examples to illustrate the overall operation of session-specific policies. The call flows depict the rendezvous mechanism between UA and policy server in detail and indicate the points at which the UA exchanges policy information with the policy server.

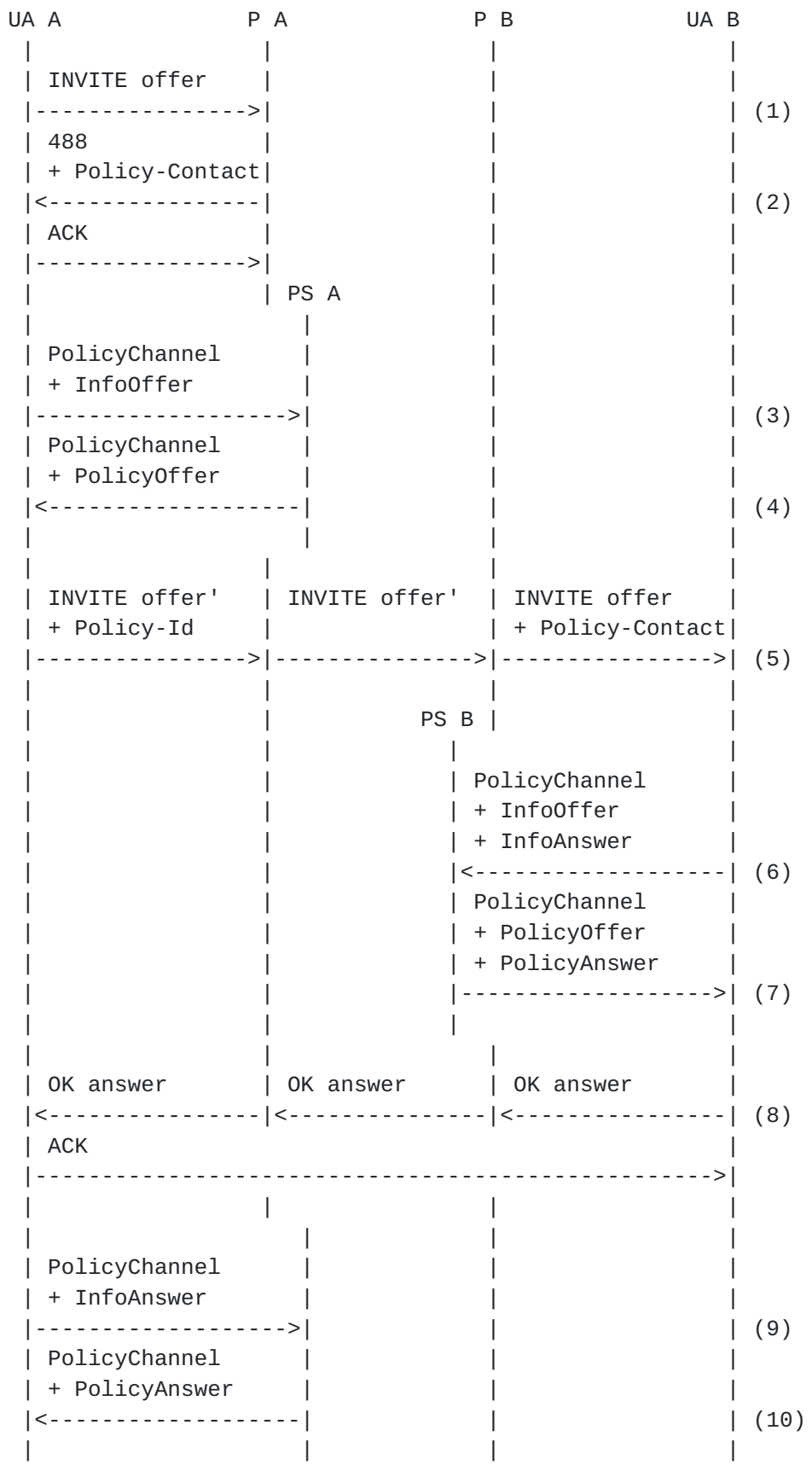
The example is based on the following scenario: there are two domains (domain A and domain B), which both have session-specific policies for the UAs in their domain. Both domains do not provide policies to the UAs outside of their domain. The two domains have a proxy (P A and P B) and a policy server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

4.3.1. 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

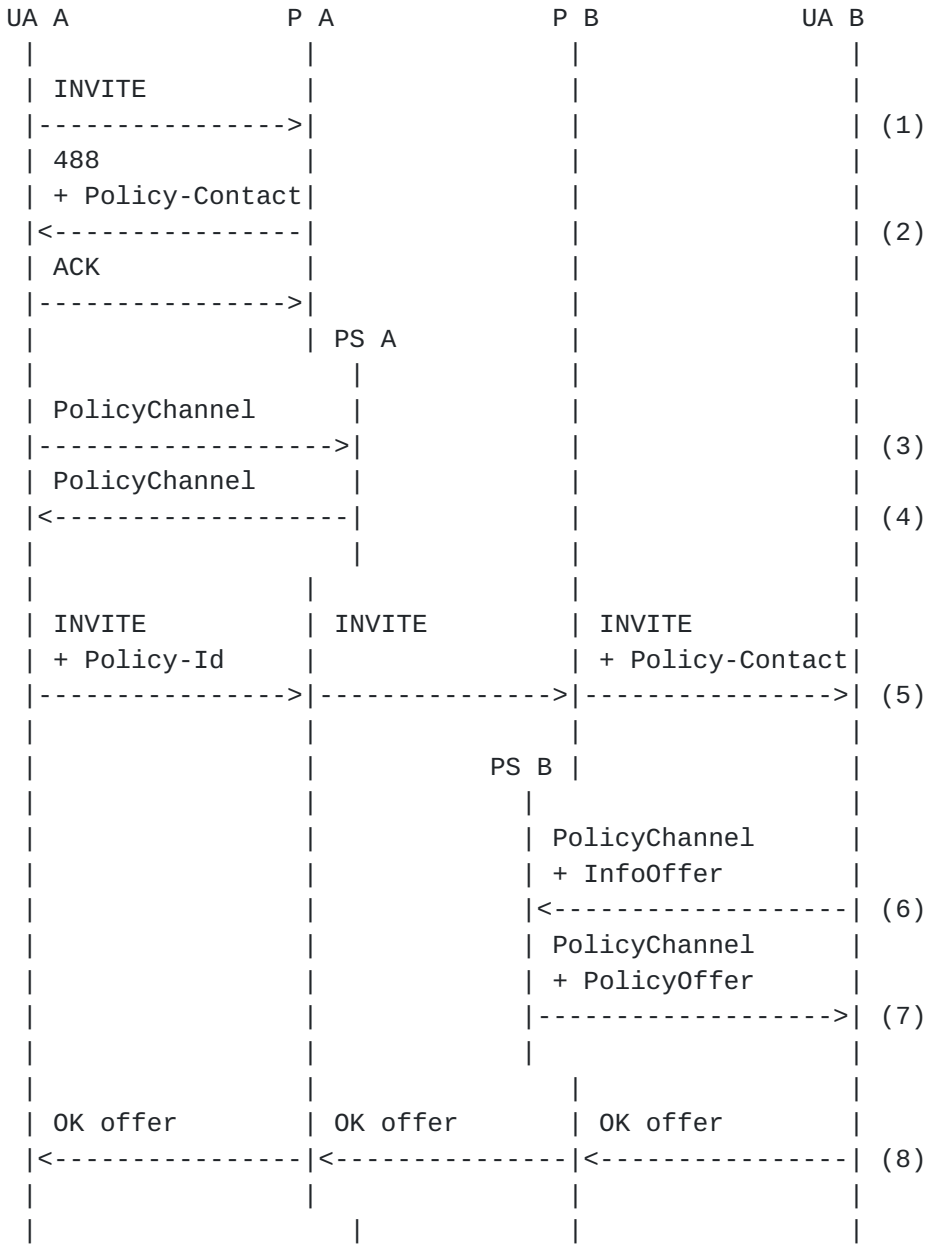
of the received policies and includes a Policy-Id header to indicate that it has already contacted PS A. P A does not reject the INVITE this time and removes the Policy-Id header when forwarding the INVITE. P B adds a Policy-Contact header containing the URI of PS B. (6) UA B uses this URI to contact 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.

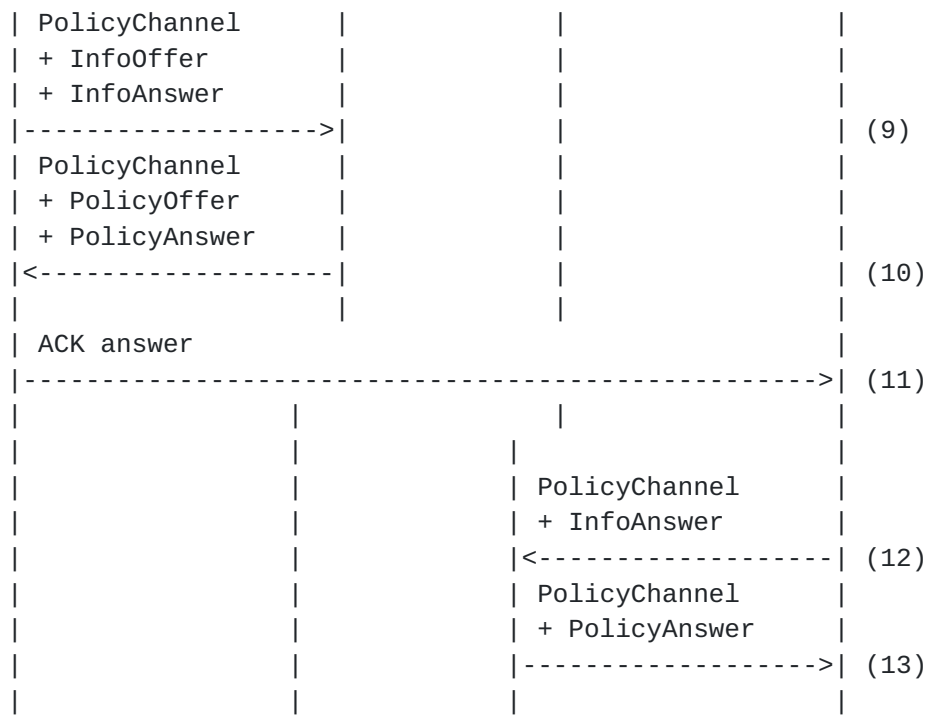

UA A
P A
P B
UA B
INVITE offer
(1)
488
+ Policy-Contact
(2)
ACK
PS A
PolicyChannel
+ InfoOffer
(3)
PolicyChannel
+ PolicyOffer
(4)
INVITE offer'
+ Policy-Id
INVITE offer'
INVITE offer
+ Policy-Contact
(5)
PS B
PolicyChannel
+ InfoOffer
+ InfoAnswer
(6)
PolicyChannel
+ PolicyOffer
+ PolicyAnswer
(7)
OK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

4.3.2. 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 [10].





4.4. 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.

4.4.1. 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 defined in this specification 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 use this URI to contact the policy server and ask for policies for current session. The UAC SHOULD apply the policies received and resend the updated request.

The UAC MUST insert a Policy-Id header into the updated request. The Policy-Id header MUST include the URIs of all policy servers the UAC has contacted during the processing of the 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. The UAC SHOULD NOT change this implicit order among policy servers.

Session policies may apply to the offer, the answer or both session descriptions. Depending on the type of session policies, a UAC may need to submit the offer and/or the answer to the policy server.

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 [10]).

4.4.2. Caching of Policy Server URIs

A UAC SHOULD cache the URI of the policy server in the local domain. It may receive this URI in a 488 from the local proxy after sending an INVITE message. The UA may also receive an initial local policy server URI through configuration or other means. This way, it can be avoided that the local proxy needs to reject the first INVITE in order to initialize the UAs policy server URI cache. A policy server URI received through configuration can, of course, be overwritten by a policy server URI in a 488 response.

The UAC SHOULD use the cached policy server 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. Domains can prevent the UAC from caching the local policy server URI. This is useful, 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. It SHOULD remove the current URI from the cache when receiving a "non-cacheable" 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, traversing different domains.

The UAC SHOULD store the list of policy server URIs is has contacted for a session. The UAC should keep this list until the session is terminated. The UAC SHOULD contact the policy server URIs in this list for each mid-dialog INVITE or UPDATE request. This avoids the retransmission of policy server URIs for each mid-dialog requests.

4.4.3. 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 ask for 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 type. In this case, it MUST contact the same policy servers it has contacted for the offer.

4.4.4. 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 all existing values at the beginning of the list. A proxy MUST NOT change the order of existing Policy-Contact header values.

4.4.5. 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 [6].

Table 1 is an extension of Tables 2 and 3 in [6]. The column 'UPD' is for the UPDATE method [5].

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

4.5. Policy Channel

The main task of the policy channel is to enable the transmission of session descriptions (i.e. the offer and the answer) of a session to the policy server and to transmit the resulting session policies back to the UA.

A UA uses the "session-spec-policy" event package [2] to subscribe to session-specific policies on a policy server. The UA receives the policies that apply to a session through this subscription. The policy server returns the initial policies for a session when the subscription is established and may notify the UA when there are updates to these policies.

When a UA receives a policy update, it SHOULD apply the update to the current session. Typically, this will require the UA to generate a re-INVITE or UPDATE message and re-negotiate the session description. For example, if a policy update disallows the use of video and video is part of the current session description, then the UA will need to create an new session description offer without video. After receiving this offer, the peer UA knows that video can't be used any more and responds with the corresponding answer.

Before a policy server can generate a session-specific policy, it needs to receive the session descriptions of a session.

The session-spec-policy event package enables a UA to include the session descriptions in the body of the SUBSCRIBE request (see [2]). These session descriptions are used by the policy server to generate the session policies the UA is subscribing to. The policy server returns these policies in NOTIFY messages. Detailed call flows can be found in Appendix B.

An alternative approach is using the SUBSCRIBE and PUBLISH methods.

A UA inserts the session descriptions into the body of a PUBLISH request and sends it to the policy server. After publishing the session descriptions, the UA uses the session-spec-policy event package to subscribe to the resulting session policies.

The PUBLISH requests use a new event package for session descriptions

[needs to be defined]. The published session descriptions establish a state on the policy server. The policy server uses this state as input to generate session policies for the described session. These policies form a separate state on the policy server, to which the UA can subscribe to using the session-spec-policy event package. This effectively decouples the transmission of session descriptions (via PUBLISH requests) from the transmission of session policies (through the subscription). PUBLISH and SUBSCRIBE requests for the same session use identical Request-URIs and event parameters so that a policy server can correlate both. Detailed call flows can be found in Appendix B.

OPEN ISSUE: Need to select one approach for conveying session descriptions to the policy server!! The following provides an short analysis of both approaches. Call flows can be found in Appendix B.

PUBLISH and SUBSCRIBE:

* Session descriptions can in some cases be submitted to the policy server by an entity that is different from the entity that subscribes to session policies (e.g. by a proxy). However, the separation of session description sender and policy receiver may lead to deadlocks in some scenarios. This may occur if the subscriber to session policies (e.g. the UA) is waiting for a policy but the entity responsible for submitting session descriptions (e.g. the proxy) can't submit the necessary session descriptions because it has not yet received them.
* PUBLISH and SUBSCRIBE both establish their own soft states in the policy server. Thus, there are two states that need to be maintained by the policy server and both need to be refreshed individually.
* This approach requires the implementation of two event packages by UA and policy server.

SUBSCRIBE:

* Using SUBSCRIBE simplifies the message flow between UA and policy server. In a simple session (offer in INVITE, no state refreshes) there are four messages less to be transmitted.
* No need to establish, maintain and refresh two different states on the policy server. This simplifies UA and policy server implementation.
* Session descriptions are directly coupled to a subscription to policies. There is no need to correlate two states on the policy server. Also, no need to cover cases in which session descriptions are published without a policy subscription and vice versa.
* Only one event package needs to be implemented by UA and the policy server.

* SUBSCRIBE bodies usually have the semantic of a filter criteria. I.e. they are used to select the resource the subscription is for out of a set of existing objects. Here, SUBSCRIBE bodies are used to as input to generate the resource the subscription is for. This is a change in the use of SUBSCRIBE bodies.

5. Security Considerations

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

[TBD.]

6. IANA Considerations

[TBD.]

Appendix A. Acknowledgements

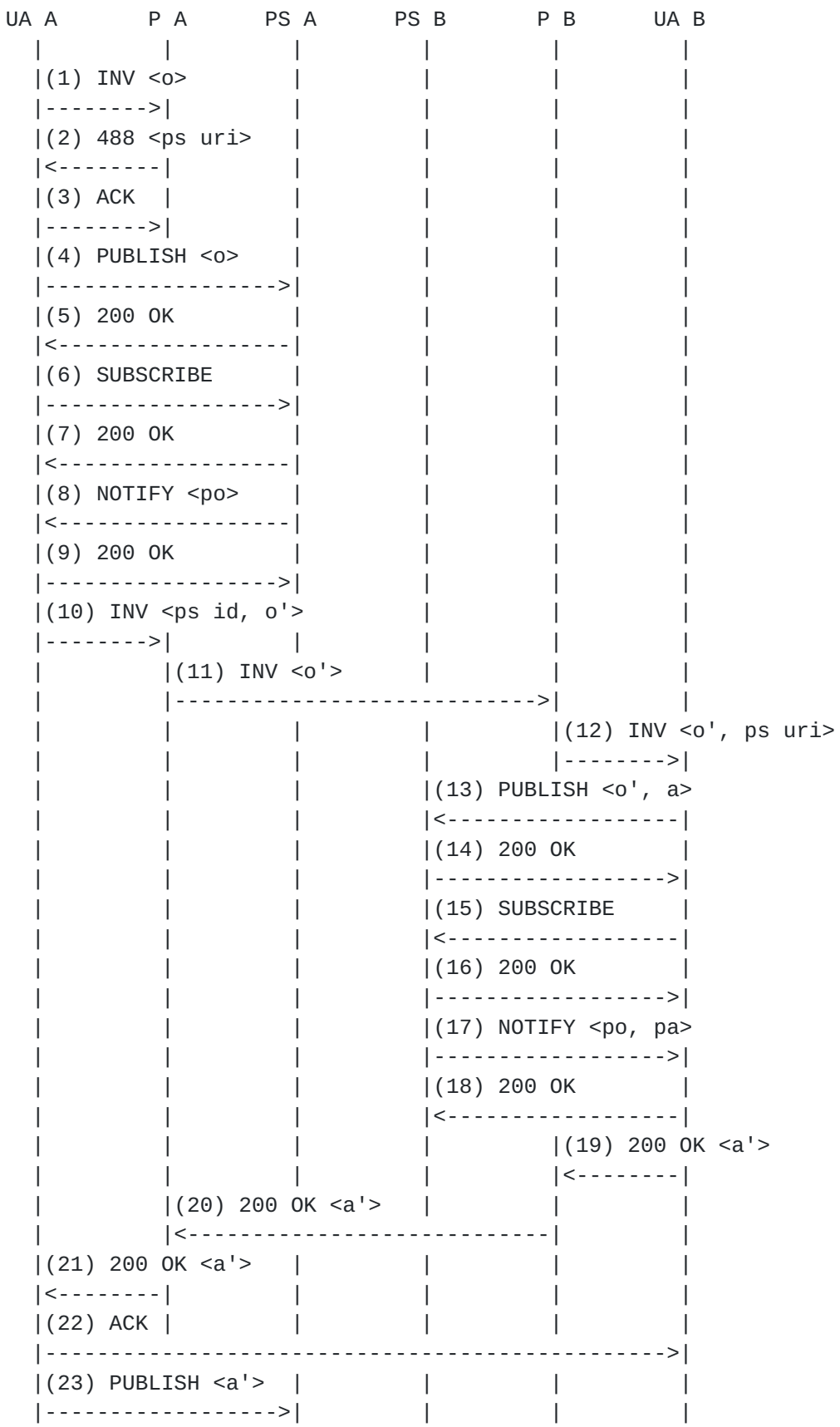
Many thanks to Allison Mankin for the discussions and the suggestions for this draft. Many thanks also to everyone who contributed by providing feedback on the mailing list and in IETF meetings.

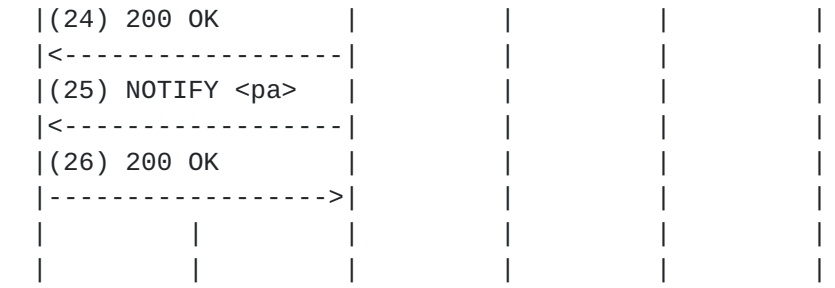
Appendix B. Call Flows

The following call flows illustrate the overall operation of session-specific policies using PUBLISH/SUBSCRIBE and SUBSCRIBE only. The call flows contain all messages needed for UA/policy server rendezvous and for the policy channel. The following abbreviations are used:

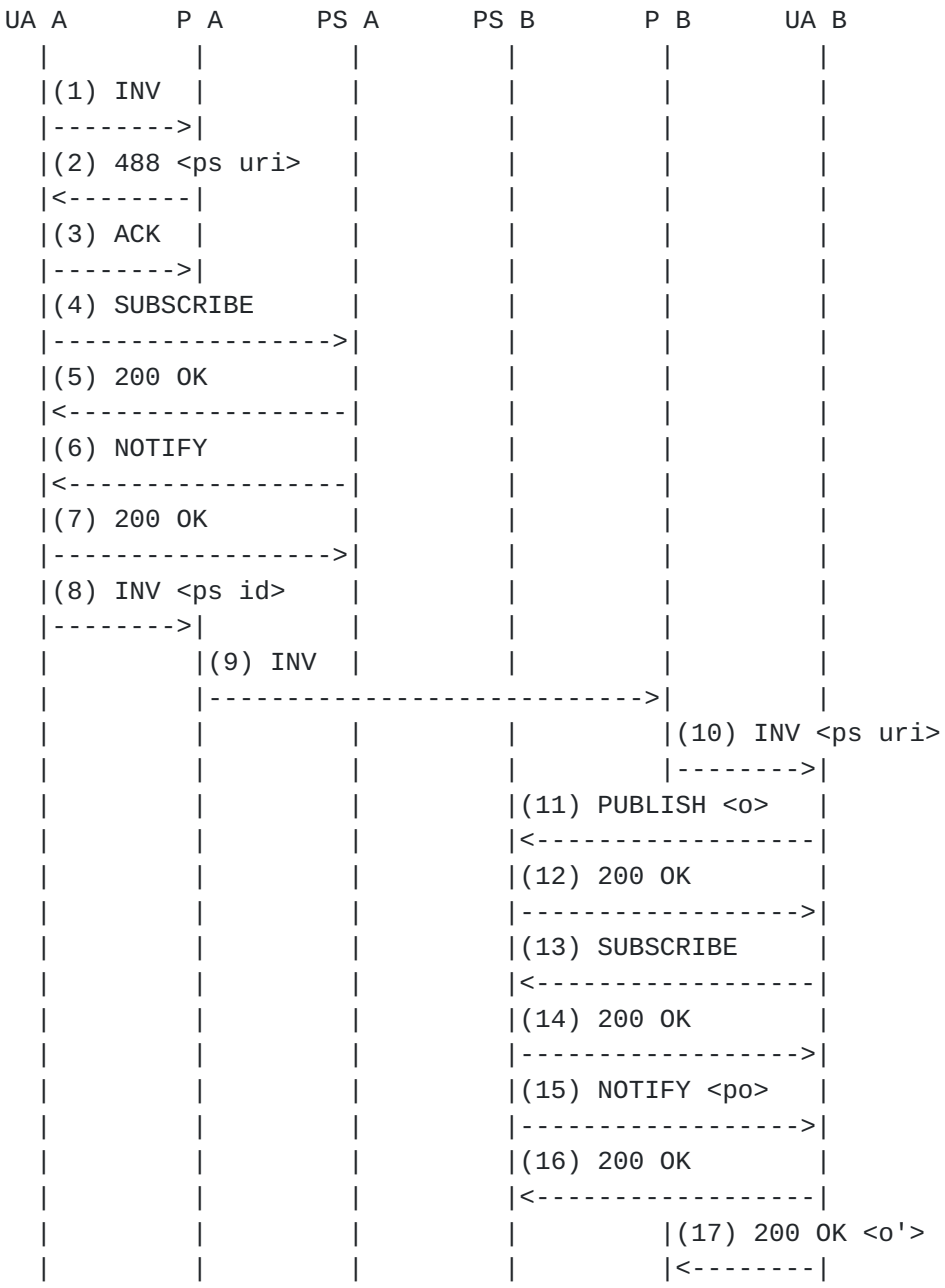
o: offer
o': offer modified by a policy
po: offer policy
a: answer
a': answer modified by a policy
pa: answer policy
ps uri: policy server URI (in Policy-Contact header)
ps id: policy server id (in Policy-Id header)

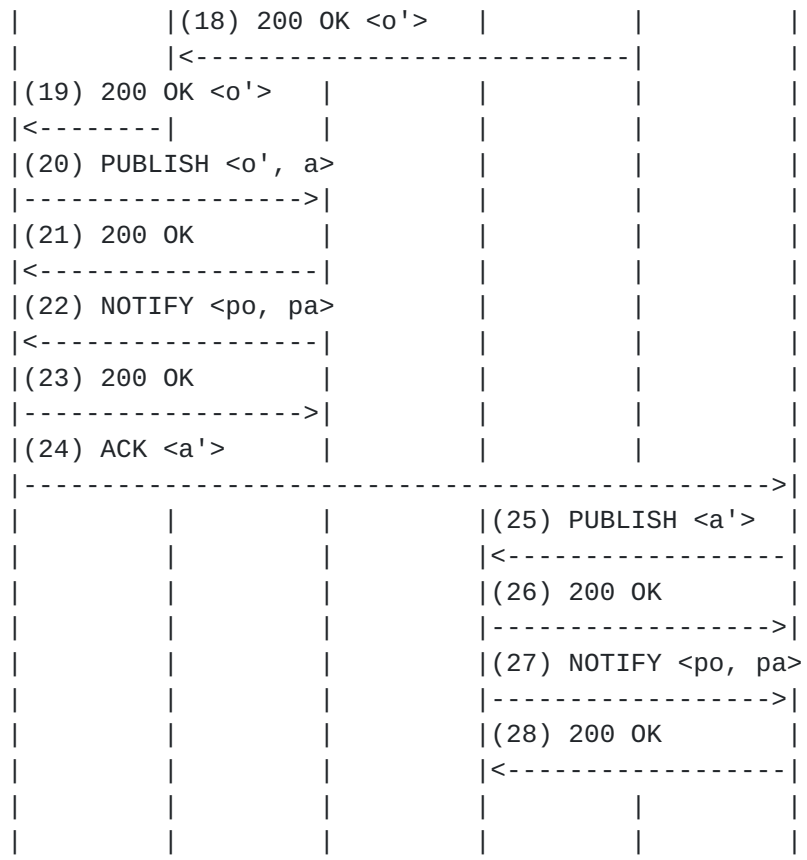
B.1. PUBLISH/SUBSCRIBE - Offer in Invite


UA A
P A
PS A
PS B
P B
UA B
(1) INV <o>
(2) 488 <ps uri>
(3) ACK
(4) PUBLISH <o>
(5) 200 OK
(6) SUBSCRIBE
(7) 200 OK
(8) NOTIFY <po>
(9) 200 OK
(10) INV <ps id, o'>
(11) INV <o'>
(12) INV <o', ps uri>
(13) PUBLISH <o', a>
(14) 200 OK
(15) SUBSCRIBE
(16) 200 OK
(17) NOTIFY <po, pa>
(18) 200 OK
(19) 200 OK <a'>
(20) 200 OK <a'>
(21) 200 OK <a'>
(22) ACK
(23) PUBLISH <a'>

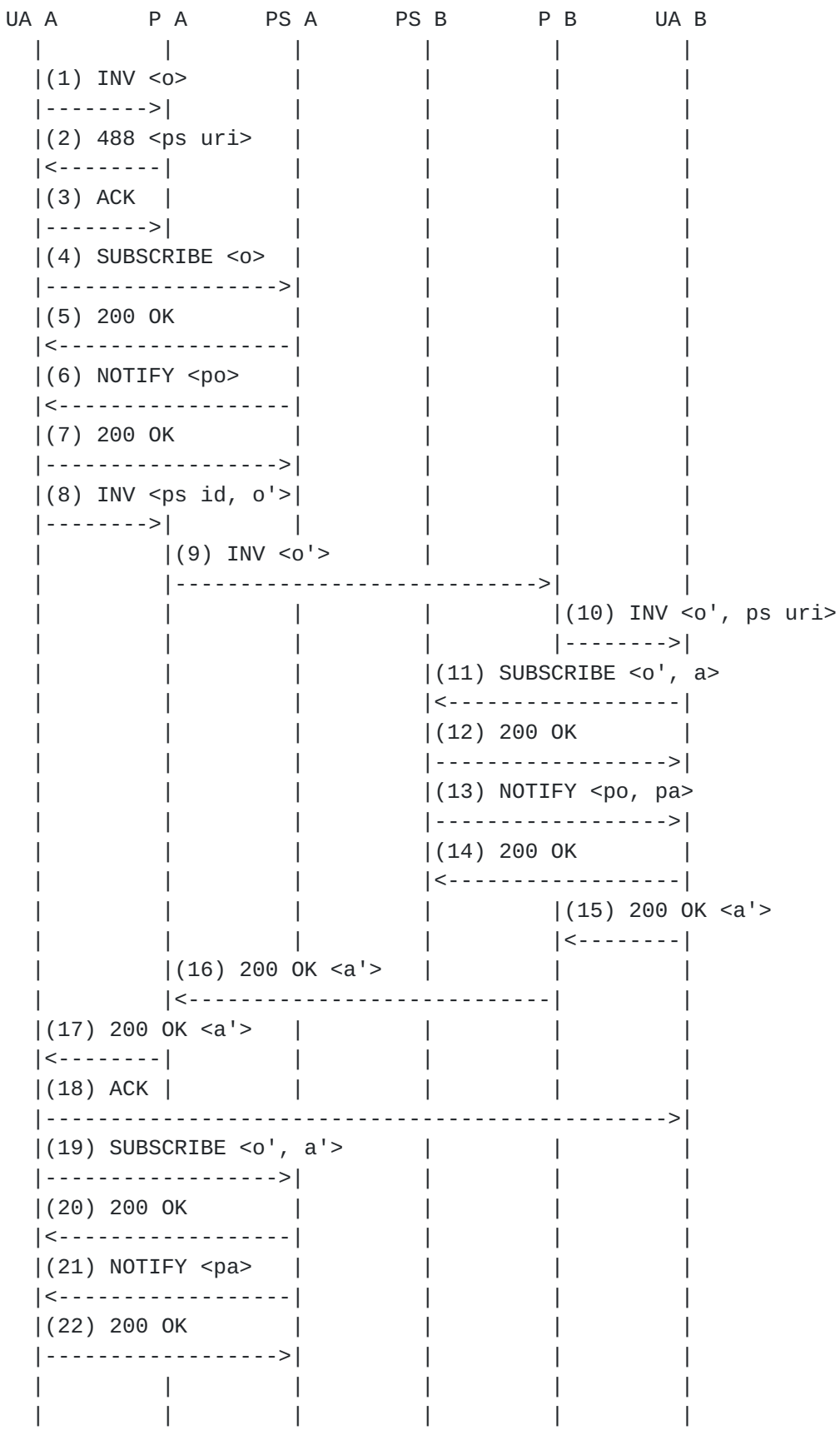


B.2. PUBLISH/SUBSCRIBE - Offer in Response

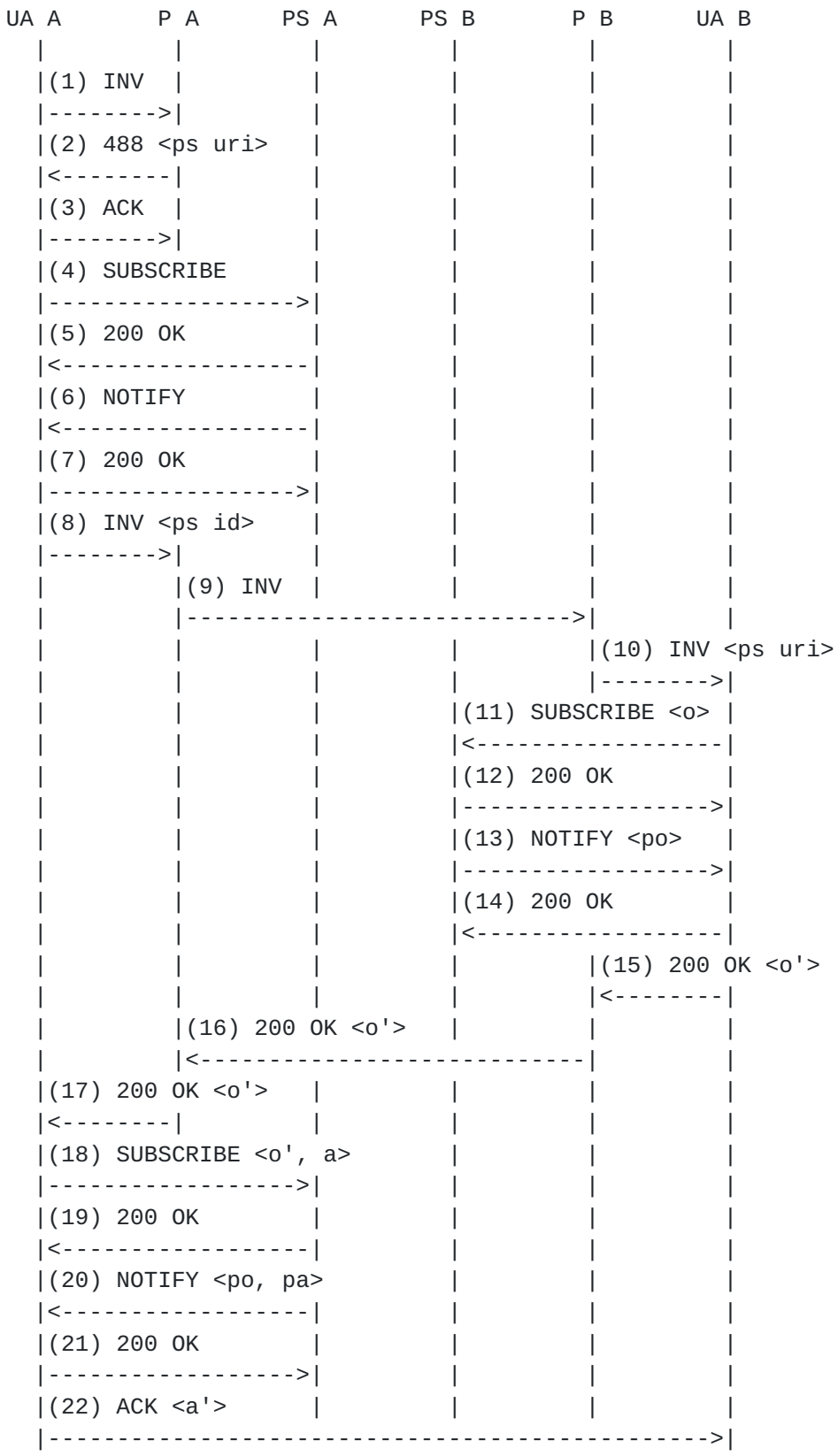


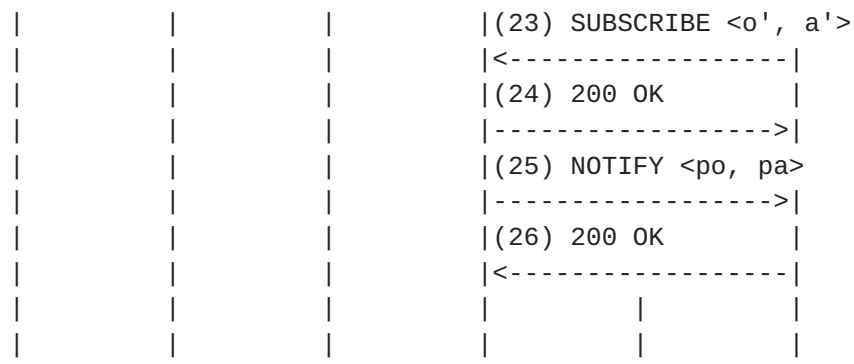


B.3. SUBSCRIBE - Offer in Invite


UA A
P A
PS A
PS B
P B
UA B
(1) INV <o>
(2) 488 <ps uri>
(3) ACK
(4) SUBSCRIBE <o>
(5) 200 OK
(6) NOTIFY <po>
(7) 200 OK
(8) INV <ps id, o'>
(9) INV <o'>
(10) INV <o', ps uri>
(11) SUBSCRIBE <o', a>
(12) 200 OK
(13) NOTIFY <po, pa>
(14) 200 OK
(15) 200 OK <a'>
(16) 200 OK <a'>
(17) 200 OK <a'>
(18) ACK
(19) SUBSCRIBE <o', a'>
(20) 200 OK
(21) NOTIFY <pa>
(22) 200 OK

B.4. SUBSCRIBE - Offer in Response





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] Hilt, V. and G. Camarillo, "A Session Initiation Protocol (SIP) Event Package for Session-Specific Session Policies.", draft-ietf-sipping-session-policy-package-00 (work in progress), April 2006.

[3] Hilt, V., Camarillo, G., and J. Rosenberg, "A User Agent Profile Data Set for Media Policy", draft-ietf-sipping-media-policy-dataset-01 (work in progress), March 2006.

[4] Petrie, D., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-08 (work in progress), March 2006.

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

[6] 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.

7.2. Informative References

[7] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[8] 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.

[9] 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-03 (work in progress), October 2005.

[10] 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.

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

Intellectual Property Statement

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

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

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

Disclaimer of Validity

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

Copyright Statement

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

Acknowledgment

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