Publication II

G. Camarillo, Adding Consent-based Communications to SIP. Adding Consent-based Communications to SIP. *IEEE Vehicular Technology Magazine*, Vol. 2, No. 2, Pages 30-37, June 2007.

Adding Consent-Based Communications to SIP

Gonzalo Camarillo, Ericsson Research

Abstract: *This paper describes a framework for consent-based communications in SIP. This framework allows users to grant or deny SIP entities permission to communicate with those users. Consent-based communications prevent some types of amplification-based denial of service attacks and can be used to alleviate the effects of spam in SIP. This paper also presents the results of our experiments implementing this framework.*

1. Introduction

The Session Initiation Protocol (SIP) [1] is the main signalling protocol of the IP Multimedia Subsystem (IMS). SIP is a rendezvous protocol that allows user agents to locate each other in order to establish sessions between them.

SIP also provides user mobility. Users are reachable under the same identifier (e.g., a SIP URI) regardless of their location. This is achieved by having users register their user agents' current location with their registrars, which are then able to route incoming requests for those users to their user agents.

The identifier under which a particular user is reachable is typically referred to as the user's AoR (Address of Record). In the IMS, an AoR corresponds to a public user identity. The user's registrar stores the bindings between the user's AoRs and the current location of the user agents where the user is currently reachable.

The rendezvous function in SIP was designed to operate between users with multiple user agents. When a

Digital Object Identifier 10.1109/MVT.2007.913286

proxy server receives an incoming request for a user, the proxy server can try, sequentially or in parallel, the locations where the user may be reachable at that point. A proxy server that attempts to reach users by sending several requests to different locations is referred to as a forking proxy server.

Therefore a forking proxy server by design is a traffic amplifier that receives an incoming request and can generate a potentially large number of outgoing requests. An attacker able to choose the locations a forking proxy server will try can use that server as an amplifier to launch denial of service (DoS) attacks. Attackers may also attempt to launch this amplification-based DoS attack using back-to-back user agents (B2BUAs) instead of forking proxies. This paper proposes a framework to avoid this type of attack.

The remainder of this paper is organized as follows: Section 2 introduces the concept of translations and discusses problems associated with the way they are usually set up; Section 3 describes a framework to solve these problems, while Section 4 analyzes the applicability of this framework to two-party communication scenarios; Section 5 describes the results of our experiments when implementing this framework; Section 6 outlines our future work on this area, and finally, Section 7 contains the conclusions of this paper.

2. SIP Routing and Translations

SIP routing is based on translations, which are typically performed by proxy servers or B2BUAs. A SIP element that performs translations is referred to as a relay. A relay, which can be seen as a SIP-level router, receives an incoming request addressed to a particular URI (e.g., an AoR) and based on information available to the relay, it chooses a new next-hop URI and relays the request to that next-hop URI.

The original URI the incoming request was addressed to is referred to as its target URI. The next-hop URI of the outgoing request is referred to as its recipient URI. The process performed by the relay is referred to as a translation. This process is illustrated by Figure 1.

Note that, as shown in Figure 1, a translation can result in several recipient URIs. In that case, the relay performing the translation may be configured to send, in parallel, a separate request to each of those recipient URIs.

The information a relay uses to perform a translation is referred to as the relay's translation logic.

2.1 URI-List Services

As stated earlier, relays can be proxy servers or B2BUAs. Section 1 described how forking proxy servers perform translations that involve multiple recipient URIs. URI-list services are an example of B2BUAs performing translations towards multiple recipient URIs. While a forking proxy server delivers a request to a given user at several URIs, a URI-list service delivers a request to multiple users, each of them identified by a different recipient URI.

SESSION INITIATION PROTOCOL IS THE MAIN SIGNALLING PROTOCOL OF THE IP MULTIMEDIA SUBSYSTEM AND PROVIDES USER MOBILITY. USERS ARE REACHABLE UNDER THE SAME IDENTIFIER REGARDLESS OF THEIR LOCATION.

The translation logic of a URI-list service consists of a target URI, which identifies the URI-list service, and a number of recipient URIs. There are URI-list services specified for several SIP methods, such as INVITE, MESSAGE, and SUBSCRIBE.

The push-to-talk over cellular (PoC) service specified by the Open Mobile Alliance (OMA) [2] uses URI-list services to establish multi-party conferences. The OMA PoC service is typically implemented on top of the IMS.

2.2 Translation Logic Manipulation

Translations are set up and managed by manipulating the relay's translation logic. For example, if a new recipient URI needs to be added to a translation, this recipient URI is added to the translation logic of the relay performing the translation.

There exist different mechanisms to manipulate translation logic. Different types of relays may implement different translation logic manipulation mechanisms. Two existing mechanisms are REGISTER transactions and the Extensible Markup Language (XML) Configuration Access Protocol (XCAP) [3].

REGISTER transactions bind an AoR to the recipient URI where the AoR's owner is currently reachable. This way, REGISTER transactions can be used to manipulate the translation logic of a translation whose target URI is that AoR.

XCAP is an HTTP-based protocol that can be used to manipulate URI lists. XCAP manipulates a relay's translation logic by modifying the URI list that represents the translation's recipient URIs.

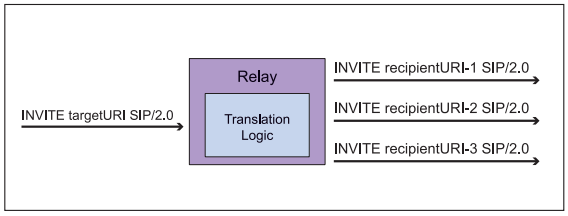

FIGURE 1 A translation at a relay.

TRADITIONALLY, ATTACKERS CAN USE RELAYS TO SEND UNSOLICITED TRAFFIC TO A POTENTIALLY LARGE NUMBER OF RECIPIENT URIS, FORMING A DENIAL OF SERVICE ATTACK.

2.3 Issues Related to How Translations Are Set Up

Mechanisms to manipulate translation logic have traditionally focused on authenticating and authorizing the user performing the manipulation. For example, registrars can use digest-based authentication to ensure that only the owner of a given AoR binds it to one or more recipient URIs. Relays using XCAP can also use digest-based authentication to ensure that only authorized parties add new recipient URIs to a given translation.

Nevertheless, relays do not typically obtain permissions from the owner of a recipient URI before adding it to their translation logic. Moreover, once a recipient URI is added to a translation, the owner of that recipient URI does not typically have any means to remove it from the translation.

Consequently, attackers can use relays to send unsolicited traffic to a potentially large number of recipient URIs. The attacker adds the recipient URIs of the victims to a relay's translation logic so that the relay acts as an amplifier. The relay will receive requests sent to its target URI and deliver a copy of them to all its recipient URIs. The owner of those recipient URIs will have no means to let the relay know that they do not want to receive traffic from it anymore.

There is one added problem: the attacker can add recipient URIs that do not correspond to a SIP node. In this way, relays can be used to flood with traffic any node in the network; not only SIP nodes. Attackers can even add several recipient URIs that route to the same subnetwork in order to attack that subnetwork's access router.

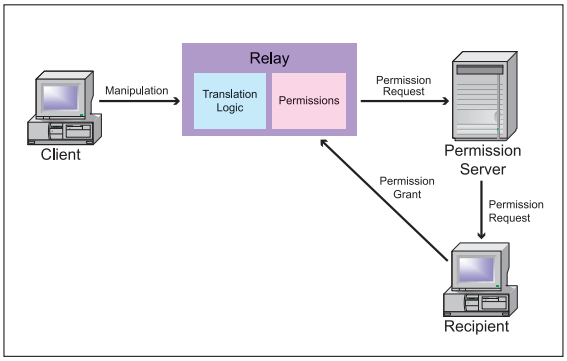

FIGURE 2 Framework architecture.

The goal of the framework described by this paper is to allow the owner of a given recipient URI to grant or deny a relay permission to generate traffic towards that recipient URI. This way, relays will only amplify traffic towards destinations that gave their consent to receive traffic beforehand.

3. A Framework for Consent-Based Communications in SIP

The consent framework architecture is shown in Figure 2. In addition to translation logic, relays store information about the permissions received from recipient URIs' owners. When a client attempts to add a new recipient URI to the relay's translation logic, the relay does not add it directly. Instead, the relay asks the recipient URI's owner for permission to add that recipient URI to its translation logic.

Since users are not on-line all the time, the architecture contains a new architectural element: the permission server. When a recipient URI's owner is not on-line, permission requests addressed to that URI are received and stored by the owner's permission server. When the owner becomes available again, the permission server delivers the permission requests to the owner.

On receiving a permission request, the recipient URI's owner grants or denies the relay permission to add that recipient URI to the relay's translation logic. A relay only adds a new recipient URI to its translation logic if it first receives permission to do so from the recipient's URI owner.

The following sections discuss the different steps involved in the process just described, the challenges faced at each step, and the design choices made when designing this framework.

3.1 Information About Permission Status

When a user attempts to add a recipient URI to a relay's translation logic, that URI is not added until the recipient URI's owner gives the relay permission to actually add it.

The process of obtaining this type of permission by the relay can take a long time because it typically requires user interaction.

During this time, the user that attempted to add the recipient URI may wish to be informed about the status of the URI. The user may want to know when the recipient URI is actually added to the relay's translation logic.

The framework uses of an event package to keep users informed about the permission-related status of the recipient URIs to be added to a relay's translation logic. This event package is independent of the mechanism used to manipulate the relay's translation logic. Consequently, this event package can be used, for example, between users and relays that support XCAP, and between users and registrars.

This event package, which is called the 'pending additions' event package, informs users if the permission to add a given recipient URI has been requested but no answer has been received, has already been granted, or has been denied. Additionally, the event package can report error conditions such as when a relay cannot contact a recipient URI in order to request permission to add it.

3.2 Amplification Avoidance

When a relay receives a request to add a set of new recipient URIs to its translation logic, the relay needs to ask the recipient URIs' owners permission to actually add them. Relays ask recipient URI owners for permission by sending them a SIP request.

Nevertheless, if the relay went ahead and sent a SIP request to each recipient URI being added, it could be used as an amplifier to launch DoS attacks. An attacker could use, for example, XCAP to add a large number of recipient URIs. The relay would then generate a large number of SIP requests asking for permission to add those URIs to its translation logic.

This framework uses a credit-based authorization approach to avoid this type of amplification. This type of authorization is also used by lower-layer mobility mechanisms in order to avoid amplifications attacks to their address-binding update processes [4].

The idea behind credit-based authorization is that a user adding a set of recipient URIs needs to generate a similar amount of traffic towards the relay as the relay will generate towards the recipient URIs in order to request permissions. This property has to be met by any translation logic manipulation mechanism implemented at a relay.

However, different manipulation mechanisms can implement this property in different ways. For XCAP, this framework recommends that additions of recipient URIs are limited to one per HTTP transaction. For REGISTER transactions, the limit is one recipient URI per REGISTER transaction.

As the results in Section 5 show, even implementing the previous limitations in XCAP and REGISTER, the amount of traffic generated by the user towards the relay is less than the traffic generated by the relay towards the recipient URIs. However, this framework recommends those limitations and not others that may be more accurate measuring bandwidth for simplicity reasons.

3.3 State Information Stored by Relays

This framework requires relays to store permission-related information about recipient URIs. The amount of information that relays need to store has been minimized because of the nature of the relationship between the entities generating the information and the relays storing it.

Relays implementing this framework need to store permission-related information about each recipient URI they handle. This permission-related information is generated by each of the owners of those recipient URIs. However, the relay may not have any previous relation with the recipient URI's owner. For example, when a user adds a recipient URI to a group hosted by the PoC server of the user's domain, the user has a relation with the PoC server; they are in the same administrative domain. However, the recipient URI added by the user may belong to a complete different domain.

Relays at a domain have little incentive to store complex information generated by users at other domains. Consequently, this framework only requires relays to store very simple permission-related information about recipient URIs. This information consists of whether or not the recipient URI's owner gave the relay permission. Recipient URI's owners cannot use finer granularity, such as which types of SIP requests are allowed, in their permissions. If a recipient URI's owner is interested in such a fine granularity, it will need to be provided by the owner's proxy server handling incoming requests or by the owner's user agent itself.

THE IDEA BEHIND CREDIT-BASED AUTHORIZATION IS THAT A USER ADDING A SET OF RECIPIENT URIS NEEDS TO GENERATE A SIMILAR AMOUNT OF TRAFFIC TOWARDS THE RELAY AS THE RELAY WILL GENERATE TOWARDS THE RECIPIENT URIS.

3.4 Backwards Compatibility

As stated in Section 3.2, relays ask recipient URI owners for permission by sending them a SIP request. This framework proposes that the SIP method used to request permissions is a MESSAGE method [5].

A MESSAGE request is more appropriate than a new request for backwards compatibility reasons. Old clients simply reject unknown requests but display the contents of any incoming MESSAGE request to the user.

A MESSAGE request asking for permission to add a recipient URI to a relay's translation logic carries a permission document. This document, which is written in an XML format, describes the translation and provides URIs for the recipient of the MESSAGE request (i.e., the recipient URI's owner) to grant or deny permission. In addition to this XML document, the MESSAGE request carries a human readable part with the same information so than old clients that do not understand the XML format can still display it to the user.

The XML format for permission documents used by this framework is based on the common policy format [6]. See Section 3.7 for an example of a permission document written in this format.

THE GOAL OF THE FRAMEWORK IS TO ALLOW THE OWNER OF A GIVEN RECIPIENT URI TO GRANT OR DENY A RELAY PERMISSION TO GENERATE TRAFFIC TOWARDS THAT RECIPIENT URI.

3.5 Permission Servers

As discussed in Section 3, permission servers store incoming MESSAGE requests carrying permission documents when users are not available. Once a user becomes available, the user's permission server delivers the permission documents received so far to the user.

Another advantage derived from using MESSAGE requests to carry permission documents is that permission servers are just store-and-forward servers. As a result, the same store-and-forward server that receives instant messages for a user when this is off-line and delivers them when the user returns on-line can be used to store incoming permission documents when the user is off-line. An example of a store-and-forward server is an instant message to email gateway.

3.6 Permission Grant

When a user receives a permission document, the user can use the URIs received in the document to grant or deny the relay permission. This framework recommends that the URIs used to grant and deny permissions be SIP or HTTP URIs. Users send PUBLISH requests to SIP URIs and GET requests to HTTP URIs.

It is important that relays make sure that the entity using a URI to grant permission is the recipient URI's owner, and not any other entity (e.g., an attacker). In order to perform this authorization, this framework recommends to use a return-routability check, which is implemented as follows.

Relays include URIs to grant and deny permissions in the permission documents they generate. These URIs need to be unguessable (e.g., long and random). If the MESSAGE requests carrying those permission documents are sent using SIPS URIs, they will be integrity protected and encrypted until they reach their recipient URIs, since a SIPS URI requires a secure transport connection. This ensures that the only entity able to use a given (unguessable) URIs is the recipient URIs' owner.

URIs to deny permission can be used by recipients at any time to revoke permissions that were granted in the past.

3.7 Message Flow

Figure 3 presents this framework's typical message flow. This flow is the result of all the considerations discussed in the previous sections.

User A adds user B to the relay's translation logic using XCAP (1). User A subscribes to the 'pending additions' event package at the relay (5) in order to be informed of the permission-related status of B's addition.

In order to ask B for permission to be added to the relay's translation logic, the relay sends a MESSAGE

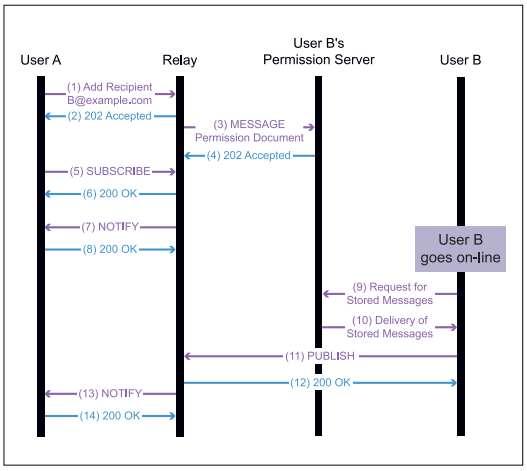

FIGURE 3 Framework's message flow.

```
<cp:ruleset>
 <cp:rule id="1">
  <cp:conditions>
   <cp:identity>
    <cp:id entity="B@example.com"
           scheme="sip"/>
   </cp:identity>
   <target>
    <cp:id entity="alices-friends@relay.com"
           scheme="sip"/>
   </target>
   <sender>
    <cp:any/>
   </sender>
  </cp:conditions>
  <cp:actions>
   <trans-handling
     perm-uri="sips:foo@relay.com">grant
   </trans-handling>
   <trans-handling
     perm-uri="sips:bar@relay.com">deny
   </trans-handling>
  </cp:actions>
  <cp:transformations/>
 </cp:rule>
</cp:ruleset>
```

FIGURE 4 Permission Document.

request (3) to B. This message request is sent to a SIPS URI and, consequently, is encrypted and integrity protected. The MESSAGE request contains two body parts. The first body part contains the XML document in Figure III-G (the initial XML headings related to namespaces have been omitted for clarity; 'cp:' refers to elements of the common policy namespace). The other body part contains the following human-readable text: "If you want to receive traffic from <sip:alices-friends@example.com>, please use the following URI <sips:foo@example.com>. Otherwise, use the following URI <sips:bar@example.com>".

Since user B is off-line when the MESSAGE request arrives to user B's permission server, the permission server stores it until user B goes on-line again. At this point, user B retrieves the contents of the MESSAGE request from the permission server (9). User B grants the relay permission to perform the translation described in the permission document by sending a PUBLISH request (11) to <sips:foo@relay.com>.

Finally, the relay informs user A that user B's URI has been added to the relay's translation logic using a NOTIFY (13). The subscription to the 'pending additions' event can be cancelled once the NOTIFY is sent or it can continue informing of 'pending additions' on the list until the sender decides to unsubscribe.

4. Consent in Two-party Scenarios

Section 2.3 stated that this framework aims to prevent amplification attacks by introducing consent-based communications into SIP. However, this framework can also be useful in scenarios that do not involve amplification.

Consent can be added to two-party communications in order to create closed communication groups. This type of communication is fairly common in many instant messaging programs where a given user is only allowed to communicate with other users that accept to communicate with that user beforehand. Some programs assume an implicit reciprocal consent schema where any user in a contact list is allowed to initiate communications with the contact list's owner. This way, users accept to communicate with a given user implicitly by placing that user in their contact lists.

In a scenario where this type of consent is used, a user that wishes to communicate with another one needs to first send an invitation to start a relationship that will allow them to communicate. That is, the invited user's consent is needed before the actual communication can take place.

Consent-based communications can transform the way users handle unsolicited communications or spam. Without consent, users can be flooded with unsolicited communication attempts and instant messages. Incoming communication attempts typically disrupt users by triggering an alert signal (e.g., their SIP phone ring). Instant messages are typically displayed

WHERE CONSENT IS USED, A USER THAT WISHES TO COMMUNICATE WITH ANOTHER ONE NEEDS TO FIRST SEND AN INVITATION TO START A RELATIONSHIP THAT WILL ALLOW THEM TO COMMUNICATE.

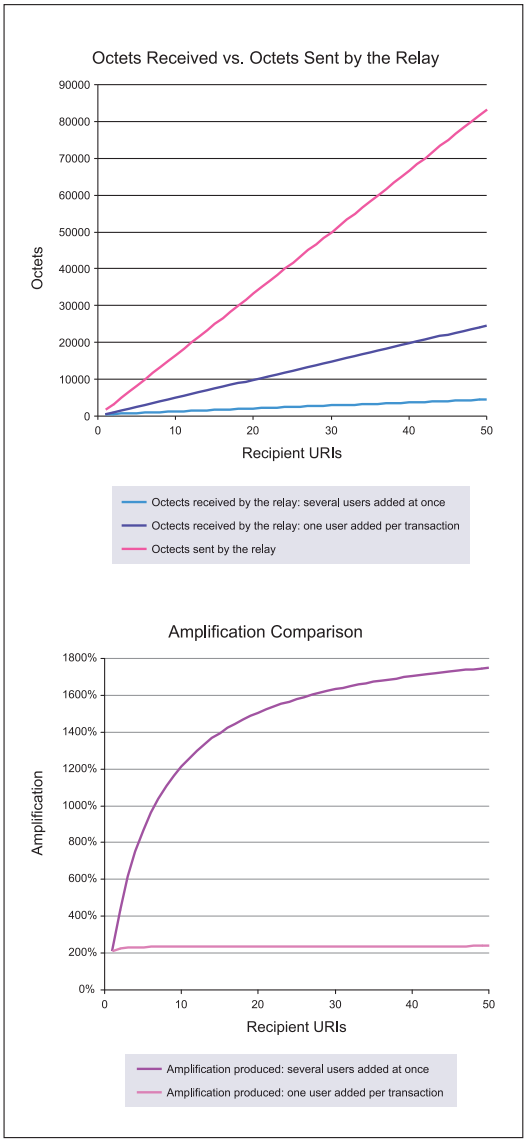

FIGURE 5 Transaction set up using XCAP.

OUR EXPERIMENTS SHOW THAT THE FRAMEWORK PROPOSED BY THIS PAPER HELPS REDUCE UNDESIRED AMPLIFICATIONS AT RELAYS WITHOUT ADDING EXCESSIVE COMPLEXITY.

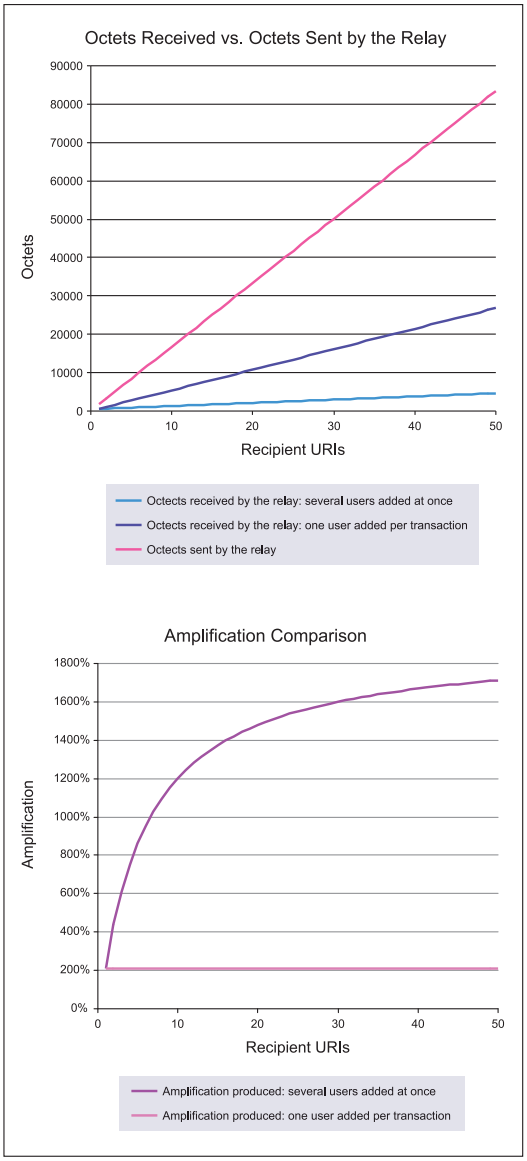

FIGURE 6 Transaction set up using REGISTER.

to the user by instant messaging applications without even asking for confirmation. These properties make unsolicited communication attempts and instant messages annoying for most users.

When consent-based communications are used, a given user agent simply rejects all communication attempts and instant messages from unauthorized users until those users ask for permission to send them. Of course, spammers can still flood a user with consent requests. However, those requests can be handled in a less-disruptive way for the user. For example, the user can check once a day the consent requests received during that day.

Note that, even though instant messaging users are used to closed-group communications, phone users are used to be able to call directly any other user. When being able to contact unknown users is desirable, consent-based communications are not a good solution to handle two-party communications that do not involve relays.

5. Implementation Experience

Our test bed includes a PoC server that supports XCAP-based list management, a registrar, and several clients. This test bed allows us to analyze the amplification produced by a relay (the PoC server or the registrar) sending MESSAGE requests with permission documents to the URIs being added to its translation logic. Figures 5 and 6 show the results of our experiments.

In both experiments, a client sets up a translation at a relay. In the experiment of Figure 5, the relay is our PoC server and the translation is set up using XCAP. In the experiment of Figure 6, the relay is our registrar and the translation is set up using REGISTER transactions.

The client setting up the translation at the relay first uses a single (XCAP or REGISTER) transaction to set up a translation with a number of recipient URIs. Then, the client sets up the same translation following the recommendation in Section 3.2. That is, the client uses a separate transaction per recipient URI in order to minimize the traffic amplification at the relay.

When a translation is set up, the relay sends a MESSAGE request with a permission document to each of the recipient URIs in the translation.

Figure 5 shows the results of our experiment with our PoC server. The PoC server always sent more octets (in the form of MESSAGE requests) than it received (in the form of XCAP requests). Therefore, the relay always acted as an amplifier. This is not surprising, because while an XCAP requests to add a recipient URI (including the HTTP header) consisted of around 500 octets, the MESSAGE requests were larger.

However, when a separate XCAP transaction per recipient URI was used, the amplification was sensibly lower than when a single XCAP transaction created the whole translation. The former method produced an

amplification of around 240%, which was effectively independent of the number of recipient URIs of the translation (the transaction to create the initial translation with the first recipient URI was slightly larger than subsequent transactions to add new recipient URIs to that translation). The latter method produced an amplification that increased with the number of recipient URIs. A translation with 50 recipient URIs set up with this method produced an amplification of around 1750%.

Figure 6 shows the results of our experiment with our registrar. The results are similar to those of Figure 5. Setting up the translation with a REGISTER transaction per recipient URI produced an amplification of around 210%, which was independent of the number of recipient URIs of the translation. Setting up the whole translation with a single REGISTER request produced an amplification that increased with the number of recipient URIs. A translation with 50 recipient URIs set up with this method produced an amplification of around 1710%. The REGISTER requests used in this experiment consisted of around 500 octets.

6. Future Work

Our future work includes standardizing the elements of this framework in the IETF (Internet Engineering Task Force). A proposal for the 'pending addition' event package discussed in Section 3.1 can be found at [7]. A proposal for the XML format for permission documents discussed in Section 3.4 can be found at [8]. A general description of this framework discussing details of interest for implementers can be found at [9].

Our future work also includes proposing the addition of consent-based communications into IMS services, such as the OMA PoC service, that use relays extensively.

7. Conclusions

This paper describes a framework to add consent-based communications to SIP. This framework reuses existing SIP components at maximum in order to minimize the implementation efforts to include it in existing relays.

Our experiments show that the framework proposed by this paper helps reduce undesired amplifications at relays without adding excessive complexity to the whole SIP architecture. While existing relays can create undesired amplifications that increase with the number of recipient URIs, relays implementing this framework create substantially smaller amplifications that are independent of the number of recipient URIs.

The use of this framework introduces delays to the process of adding new URIs to a relay's translation logic. However, once a relay obtains permissions from all its translation's recipients, session establishments do not experience any delay related to the use of this framework.

It would be possible to completely remove the amplifications addressed by this framework, as opposed to minimize them, but that would add extra complexity to the architecture. One of the initial goals of this framework was to keep the solution simple so that implementers can add it to their existing SIP entities easily.

8. Acknowledgments

Jonathan Rosenberg, Henning Schulzrinne, and Ignacio Más provided useful ideas and discussions on the topics tackled by this paper.

Author Information

Gonzalo Camarillo received M.Sc. degrees in electrical engineering from the Stockholm (Sweden) Royal Institute of Technology and from Universidad Politecnica de Madrid (Spain). He has co-authored, among other standards, the SIP specification (RFC 3261). He is the IETF liaison manager to 3GPP and currently co-chairs the SIPPING and HIP working groups in the IETF. He is the author of the books "SIP Demystified" and "The 3G IP Multimedia Subsystem (IMS)". He heads the Advanced Signalling Research Laboratory in Ericsson Finland. His research interests include signalling, multimedia applications, transport protocols, and network security.

References

1. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," Internet Engineering Task Force, RFC 3261, June 2002.
2. Open Mobile Alliance, "Push to Talk Over Cellular Version 1.0," Open Mobile Alliance," Candidate Enabler Release, May 2005.
3. J. Rosenberg, "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)," Internet Engineering Task Force, Internet-Draft draft-ietf-simple-xcap-11, May 2006, work in progress.
4. C. Vogt and J. Arkko, "Credit-Based Authorization for Concurrent Reachability Verification," Internet Engineering Task Force, Internet-Draft draft-vogt-mobopts-simple-cba-00, Feb. 2006, work in progress.
5. B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging," Internet Engineering Task Force, RFC 3428, Dec. 2002.
6. H. Schulzrinne, "Common Policy: An XML Document Format for Expressing Privacy Preferences," Internet Engineering Task Force, Internet-Draft draft-ietf-geopriv-common-policy-10, May 2006, work in progress.
7. G. Camarillo, "The Session Initiation Protocol (SIP) Pending Additions Event Package," Internet Engineering Task Force, Internet-Draft draft-camarillo-sipping-pending-additions-00, June 2006, work in progress.
8. G. Camarillo, "A Document Format for Requesting Consent," Internet Engineering Task Force, Internet-Draft draft-camarillo-sipping-consent-format-01, June 2006, work in progress.
9. J. Rosenberg, G. Camarillo, and D. Willis, "A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP)," Internet Engineering Task Force, Internet-Draft draft-ietf-sipping-consent-framework-05, June 2006, work in progress.

VT