Scaling up VoIP: Transport Protocols and Controlling Unwanted Communication Requests

Kumiko Ono

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2012

ABSTRACT

Scaling up VoIP: Transport Protocols and Controlling Unwanted Communication Requests

Kumiko Ono

Millions of people worldwide use voice over IP (VoIP) services not only as cost-effective alternatives to long distance and international calls but also as unified communication tools, such as video conferencing. Owing to the low cost of new user accounts, each person can easily obtain multiple accounts for various purposes. Rich VoIP functions combined with the low cost of new accounts and connections attract many people, resulting in a dramatic increase in the number of active user accounts. Internet telephony service providers (ITSPs), therefore, need to deploy VoIP systems to accommodate this growing demand for VoIP user accounts. Attracted people also include bad actors who make calls that are unwanted to callees. Once ITSPs openly connect with each other, unwanted bulk calls will be at least as serious a problem as email spam. This dissertation studies how we can reduce load both on ITSPs and end users to ensure continuing the success of VoIP services.

From ITSPs' perspective, the scalability of VoIP servers is of importance and concern. Scalability depends on server implementation and the transport protocol for SIP, VoIP signaling. We conduct experiments to understand the impact of connection-oriented transport protocols, namely, TCP and SCTP, because of the additional costs of handling connections. Contradicting the negative perception of connection-oriented transport protocols, our experimental results demonstrate that the TCP implementation in Linux can maintain comparable capacity to UDP, which is a lightweight connection-less transport protocol. The use of SCTP, on the other hand, requires improving the Linux implementation since the not-well-tested implementation makes a server less scalable. We establish the maximum number of concurrent TCP or SCTP connections as baseline data and suggest better server configurations to minimize the negative impact of handling a large number of connections.

Thus, our experimental analysis will also contribute to the design of other servers with a very large number of TCP or SCTP connections.

From the perspective of end users, controlling unwanted calls is vital to preserving the VoIP service utility and value. Prior work on preventing unwanted email or calls has mainly focused on detecting unwanted communication requests, leaving many messages or calls unlabeled since false positives during filtering are unacceptable. Unlike prior work, we explore approaches to identifying a "good" call based on signaling messages rather than content. This is because content-based filtering cannot prevent call spam from disturbing callees since a ringing tone interrupts them before content is sent.

Our first approach uses "cross-media relations." Calls are unlikely to be unwanted if two parties have previously communicated with each other through other communication means. Specifically, we propose two mechanisms using cross-media relations. For the first mechanism, a potential caller offers her contact addresses which might be used in future calls to the callee. For the second mechanism, a callee provides a potential caller with weak secret for future use. When the caller makes a call, she conveys the information to be identified as someone the callee contacted before through other means. Our prototype illustrates how these mechanisms work in Web-then-call and email-then-call scenarios. In addition, our user study of received email messages, calls, SMS messages demonstrates the potential effectiveness of this idea.

Another approach uses caller's attributes, such as organizational affiliation, in the case where two parties have had no prior contact. We introduce a lightweight mechanism for validating user attributes with privacy-awareness and moderate security. Unlike existing mechanisms of asserting user attributes, we design to allow the caller to claim her attributes to callees without needing to prove her identity or her public key. To strike the proper balance between the ease of service deployment and security, our proposed mechanism relies on transitive trust, through an attribute validation server, established over transport layer security. This mechanism uses an attribute reference ID, which limits the lifetime and restricts relying parties. Our prototype demonstrates the simplicity of our concept and the possibility of practical use.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I offer my sincere gratitude to my advisor, Professor Henning Schulzrinne, for his kindness and patience with my years in the Internet Real-Time Laboratory. Henning has guided my path not only with his intellect and knowledge, but also with thoughtfulness about an international student.

Second, I would like to acknowledge my committee members: Professor Steve Bellovin, Professor Salvatore Stolfo, Professor Vishal Misra, and Professor Gil Zussman for their encouragement and valuable comments on my dissertation.

This work would not be possible without the support from NTT and Cisco. I have greatly benefited from interactions and friendships with Koike Arata, Naotaka Morita, Fred Baker, and Cullen Jennings.

I want to extend my thanks to my colleagues. I am grateful to Jan Janak for reviewing my thesis, Jae Woo Lee for encouraging me to finish the program, and Yves Petinot for helping me improve my presentation. I was fortunate to work with the members of the lab: Andrea Forte, Sangho Shin, Kundan Singh, Knarig Arabshian, Charles Shen, Jong Yul Kim, Wonsang Song, Kyung Hwa Kim, Se Gi Hong, Omer Boyaci, Salman Baset, Arezu Moghadam and other members. I also want to express my sincere thanks for their friendship to Shuzo Tarumi, Akira Yanagawa, Hang Zhao, and Yingbo Song. I would also like to thank Aditi Rajoriya and Yifan Zhang for their assistance with implementation.

I am indebted to wonderful members at the computer science department: Rosemary Addarich, Daisy Nguyen, Patricia Hervey, Cindy Walters, Remi Moss, Lily Secora, Jessica Rosa and other members.

Finally, I would like to express my deepest gratitude and appreciation to my family, my mom Machiko, my dad Syohei, my sister Sumiko, for their continuous support and unconditional love, and her children, Yuki and Hina, for their encouragement and smile.

To my parents, sister, Yuki, and Hina

Chapter 1

Introduction

Over the past decade, voice over IP (VoIP) communication has rapidly evolved into an essential application owing to its low cost and service extensibility. VoIP communication initially started as services within an enterprise network as well as low-cost long distance and international calls. It has expanded to accommodate a large number of users over the Internet, to provide interconnection services to the public switched telephone network (PSTN) and cellular networks, and to offer unified communication tools with other services, such as video conferencing, instant messaging, presence, and Web services. It is expected to further grow by mobile phone devices supporting mobile broadband technologies, such as the 3rd Generation Partnership Project (3GPP) Long-Term Evolution (LTE) [3GPP, 2011]. This evolution of VoIP communication has led to a sharp rise in call volume and the number of users. VoIP users have included not only humans but also computers that typically play recorded messages for various purposes, such as notification, marketing, and surveys. This human and computer behavior currently works well within closed networks, where a limited number of VoIP service providers or Internet telephony service providers (ITSPs) connect with each other based on business alliances. However, this behavior will degrade once ITSPs openly connect each other, resulting in certain VoIP users exploiting its low service cost and easy usage to make unwanted bulk calls for commercial gain. With the future deployment of large-scale open VoIP systems, the issue of incoming unwanted calls will worsen than unwanted bulk email or email spam [Andersson *et al.*, 2007], which has caused major problems in the Internet. Therefore, scalability of VoIP communication

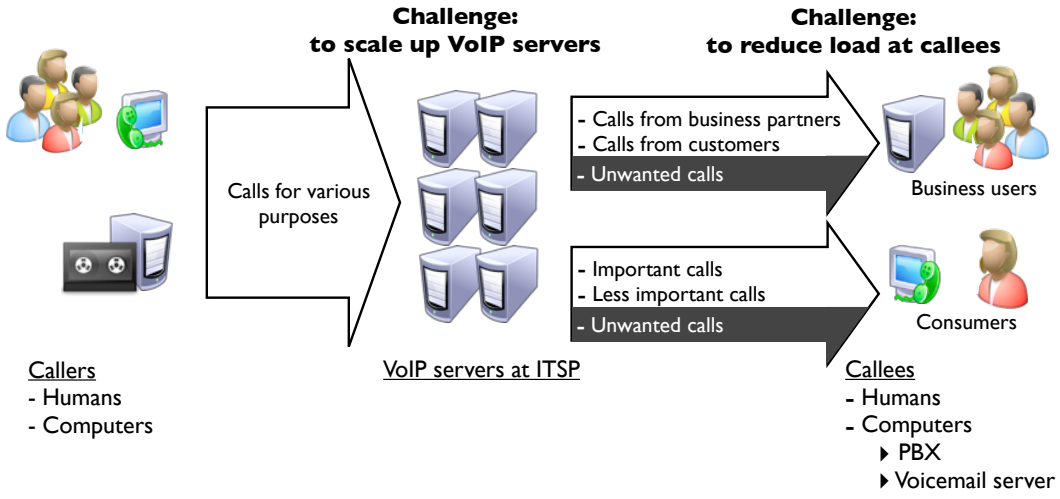


Figure 1.1: An overview of challenges to manage vast traffic

services and its resilience to unwanted calls are of vital importance and concern.

Figure 1.1 shows an overview of challenges to manage a vast amount of VoIP traffic both on VoIP servers operated by ITSPs and on VoIP users, specifically callees. From ITSPs' perspective, the challenge is how to scale up their VoIP system. In general, system scalability is achieved by distributing load across multiple servers [Katz *et al.*, 1994]. To determine how many servers are needed to build a scalable system, it is important to identify capacity and performance of a single server. To provide VoIP services, the Session Initiation Protocol (SIP) [Rosenberg *et al.*, 2002] is widely used as the signaling protocol to set up and tear down a communication session. SIP user agents (UAs) negotiate media information, such as a voice codec, the IP address, and the port number, in the Session Description Protocol (SDP) [Handley *et al.*, 2006], which is attached as a body content in a SIP message. The SIP UAs then establish not only a voice communication session, but more generally, real-time communication sessions between them. SIP can easily extend services by adding a header field or a body content including the SDP to a message, similar to Hypertext Transfer Protocol (HTTP) [Fielding *et al.*, 1999] and email messages [Resnick, 2008]. Thus, it is vital for ITSPs to measure capacity and performance of a large-scale SIP server. On the other hand, from the perspective of VoIP users, the challenge is how to reduce load at callees by controlling unwanted calls. Since most of callees are humans,

it is important to enable callees to receive all important calls without being disturbed by unwanted calls.

1.1 The Need for Reducing Load on SIP Servers: Understanding the Impact of Transport Protocols for SIP

Many ITSPs limit the transport protocol supported by their SIP servers despite that the SIP specification allows users to choose any transport protocol from connection-less lightweight transport protocols such as User Datagram Protocol (UDP) [Postel, 1980] and connection-oriented reliable protocols such as Transmission Control Protocol (TCP) [Postel, 1981] and Stream Control Transmission Protocol (SCTP) [Stewart, 2007], depending on their service requirements or network conditions. Without comprehensive quantitative analysis, it has been widely considered that a large-scale SIP server should offer users only UDP, since the SIP server needs only a single socket to communicate with all the users without maintaining connection state. If a SIP server offers TCP or SCTP connections to users, a large number of connections imposes additional processing costs on the SIP server. As a result, more SIP servers are required to accommodate the same number of users than for UDP and to keep up with the corresponding number of user registration and call requests. However, the number of additional servers is unclear; thus, many ITSPs hesitate to support TCP and SCTP. There are other practical reasons for not supporting SCTP. It is more heavyweight, less capable of network address translator (NAT) and firewall traversal, and less supported by SIP entities. According to the report from SIP interoperability test (SIPit) event [Sparks, 2011], only eight percent of SIP implementations supports SCTP, whereas most of them support both UDP and TCP.

However, there exists a clear demand for supporting connection-oriented protocols from the service requirements of VoIP users, conflicting with the support avoidance by ITSPs. For additional services and security, a SIP message includes SIP extension headers and/or cryptographic signatures, resulting in exceeding the path maximum transmit unit (MTU). Thus, TCP or SCTP segmentation is needed to transmit a feature-rich SIP message. Additionally, to protect messages between a user and the SIP server, a SIP message needs to use

Transport Layer Security (TLS) [Dierks and Rescorla, 2008], which allows a SIP application to invoke security along the signaling path. Thus, in order to fulfill the service and security requirements of VoIP users, it is crucial that ITSPs understand how the transport protocol affects SIP server performance in order to provision their server resources for service deployment.

Our goals are to identify the impact of the use of TCP and SCTP on the performance of a SIP server and to suggest better configurations that minimize any negative effects. Our approach is to experimentally analyze the maximum number of concurrent connections, data transfer latency, and sustainable request rate by identifying bottlenecks in the step-by-step manner using three servers: an echo server, a simplified SIP server, and a full SIP server.

We hypothesize that, despite additional costs of processing a large number of connections, using TCP is not harmful since TCP implementations have been well developed and deployed for scalable HTTP servers [Kegel, 2006]. On the other hand, we suspect the impact of using SCTP is larger than that of using TCP for two reasons. First, the SCTP data structure is more complicated since SCTP was originally designed to carry a telephony signaling protocol, Signaling Systems No. 7 over IP [Coene and Pastor-Balbas, 2006], which requires failover capability. Second, SCTP is a relatively new design, resulting in implementations that have not yet been well tested.

We expect that providing the quantitative analysis of the impact of using TCP or SCTP could help ITSPs re-provision their server resources for their server deployment. Consequently, users would be able to select any transport protocol among UDP, TCP, and SCTP depending on their services or network conditions. Based on these hypotheses, this thesis intends to quantify the impact of using a connection-oriented transport protocol, TCP or SCTP, on the capacity and performance of a SIP server.

1.2 The Need for Reducing Load on Callees: Controlling Unwanted Calls

Receiving unwanted calls is a long-standing problem in the PSTN, but an IP-based infrastructure is more vulnerable to unwanted calls owing to its low cost of user accounts and connections. In a small network where users suffer from few unwanted calls, simple filtering mechanisms based on a caller identifier (ID), such as white-listing and black-listing, work satisfactorily to determine whether or not to accept incoming calls. Additionally, law enforcement, such as the national Do-Not-Call registry [Commission, 2005], is fairly effective in deterring those who place commercial calls without callees' consent.

However, large-scale VoIP communication services diminish the effectiveness of these efforts. There is clear evidence that the problem has become more significant; regulations [FCC Robocalls, 2012] have been tightened against autodialed or pre-recorded messages, so-called robocalls. Caller-ID-based filtering, which is the most common solution, becomes ineffective since callees often receive *good* calls from legitimate callers but carrying unknown or blocked caller IDs. Law enforcement is ineffective for international calls originating from countries beyond jurisdiction. A good call is desirable by callees, as defined in Section 6.4. Examples of these good calls include confirmations of appointments, reservations, or deliveries, and recorded notifications of flight delays or school closing on a snowy day. These good calls, therefore, are often mistakenly labeled as unwanted calls at a call filtering system since their caller IDs are not found in the callee's white list, a list of contact addresses to accept calls, as illustrated in the center of Figure 1.2. In addition, authenticated caller IDs are not always provided in VoIP calls through different networks. For example, international calls or calls through a VoIP–PSTN gateway sometimes have no authenticated caller ID. That is, simple caller-ID-based mechanisms are limited in applicability.

The callee's white list generally contains the addresses from his contact list or address book, which is populated by contact addresses of people connected by strong social ties [Granovetter, 1973], such as family members and friends. The white list is also updated by outgoing communication history. For business use, the white list usually links to a directory service located on an Lightweight Directory Access Protocol (LDAP) [Sermersheim,

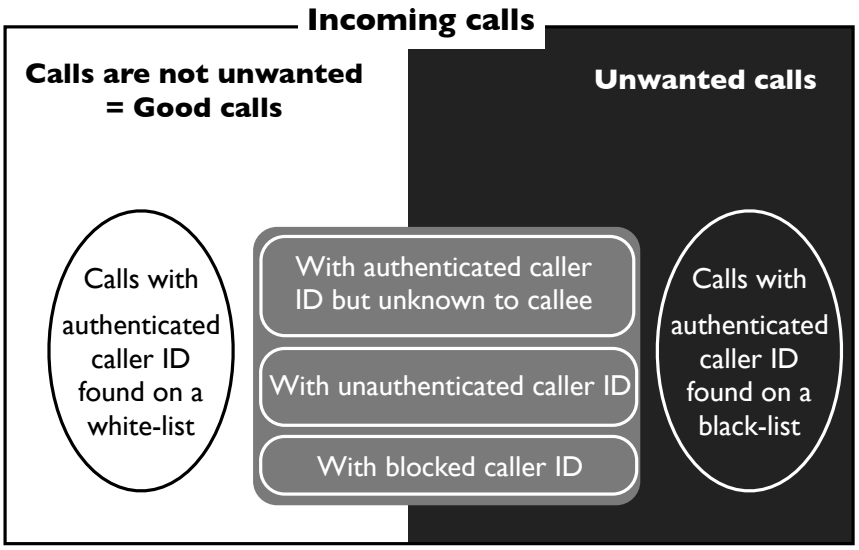


Figure 1.2: An overview of the classification of incoming calls

2006] server. For either use, however, the white list does not usually include the addresses of persons or organizations connected by weak social ties, such as friends of a friend. When people whom are connected by weak social ties place calls for the first time, their calls are often filtered out since their caller IDs are not found in the callee's white list. This is called the *introduction problem*.[1] To mitigate this introduction problem, some systems forward these calls to a voicemail server, rather than rejecting them. However, this is not a desirable solution because it requires callees to check them later and delays notifications.

Our goal, therefore, is to develop more sophisticated mechanisms to help callees identify good calls from persons or organizations connected to the callee even by weak social ties. Callees are assumed to be consumers rather than business users, who include call centers

[1]This term is commonly used but not precisely defined. We define the introduction problem as a problem with assessing a communication with a stranger. A person determines if the stranger is worth establishing a communication with, namely, worth introducing himself or being introduced to. If the stranger can provide the person with a thing indicating that the communication has enough value to be initiated, the problem is solved. This problem occurs in all means of communications: in-person and over networks, such as phone calls, email, or instant messaging (IM). Since IM services typically assess incoming messages simply by looking the origin ID up on a recipient's white list, the messages from good strangers cannot reach the recipient.

needing to accept calls from potential customers.

We hypothesize that a significant fraction of incoming good calls carries a caller ID which is not found in the recipient's address book. We also hypothesize that unwanted calls usually originate from persons or organizations whom the callee has never met before.[2] In contrast, good calls originate from person or organizations who have had contact directly or indirectly with the callee before making a call in order to know the callee's contact address. We focus on this prior contact which has been established through other communication means, such as a Web transaction, or through a common friend whose social graphs overlap. Thus, this thesis proposes a new mechanism using *cross-media relations* as proof of prior contact. As a supplementary mechanism, this thesis also proposes that call acceptance decision is based on a user's attributes without necessarily validating the users's identity. We expect that these new filtering mechanisms could help recipients easily prioritize incoming calls.

1.3 Thesis Outline and Contributions

The goal of this thesis is to reduce load both on SIP servers and on callees. This thesis consists of two parts. Part I (Chapter 2 - 4) presents the measurement to quantify the impact of using connection-oriented transport protocols on the performance of a SIP server. Part II (Chapter 6 - 8) proposes additional filtering mechanisms to facilitate controlling unwanted calls.

The following outlines this thesis and contributions.

Chapter 2 provides a brief overview of a SIP server and transport protocols for SIP. It also reviews related work for a large-scale TCP server and SIP server measurement. It defines the capacity requirements for a large-scale SIP server.

Chapter 3 quantifies the impact of using TCP and SCTP by the measurement of an echo server, eliminating the impact of SIP operations. It details the cost of using TCP from the kernel perspective and identifies the bottleneck on the 32-bit kernel [Ono and Schulzrinne, 2008b]. It also measures the effects of SCTP features related to the capacity and performance of a server, and recommends configurations for minimizing the negative

[2]The exception is unwanted calls originating from compromised machines.

impact of using SCTP, using a one-to-many socket for both clients and server [Ono and Schulzrinne, 2008a]. By pointing out a few issues in the Linux source code, the measurement contributes to the Linux community towards improving the SCTP Linux implementation.

Chapter 4 analyzes the impact of using TCP by the measurement of a SIP server, specifically a SIP registrar. By comparing to the results of the echo server measurement in Chapter 3, it identifies TCP byte-streaming as being responsible for the reduction of the throughput using TCP although the degree of the impact depends on the SIP server implementation. It also suggests accelerating message parsing when discarding SIP messages under high loads in order to slow dropping the success rate. It examines the effect of SCTP being message-oriented using a simplified SIP server, namely, a SIP front-end server which only performs message parsing, but fails in identifying the effect. This failure indicates the message orientation benefits SIP server performance under high loads only.

Chapter 6 defines good calls (or messages), justifying the principle of our two approaches, controlling unwanted calls by identifying good calls is more effective than detecting call spam. It gives an overview of solution space of preventing unwanted email and calls, discussing the limitations of caller-ID-based filtering.

Chapter 7 proposes our first approach using cross-media relations as evidence of prior contact, to help callees identify good calls [Ono and Schulzrinne, 2009a; Ono and Schulzrinne, 2009c]. This approach uses a new SIP header, Sender-References, to convey an additional piece of information, cross-media relations in a SIP message, which is proposed to the IETF [Ono and Schulzrinne, 2009b]. Our prototype demonstrates proof of concept, using Web-then-call and email-then-call. It presents a user study of incoming email messages, calls, and short messaging service (SMS) messages to test the concept and indicates the potential effectiveness of this approach [Ono and Schulzrinne, 2011b].

Chapter 8 presents our second approach using a user's attributes without knowing the user's identity [Ono and Schulzrinne, 2011a]. Our prototype demonstrates proof of concept and the simplicity of the implementation and deployment. It compares the functionality and lines of code with an existing anonymous attribute certificate, U-Prove [Paquin, 2011], and demonstrates the design concept, simplicity.

Part I

The Impact of Transport Protocols on Large-Scale SIP Servers

Chapter 2

Background and Related Work

2.1 Introduction

This chapter provides the background and a summary of related work in the field of SIP server performance measurement that focuses on comparing the impact of transport protocols, such as UDP, TCP, and SCTP. We examine the transport protocols defined in the SIP specification, discussing their current status of implementation and deployment. They are followed by a brief overview of TCP and SCTP, focusing on the features affecting server performance. We then proceed to explain SIP servers, focusing on what the differences between SIP servers and HTTP servers are, and how large our target SIP server is for our measurement. After decomposing factors affecting SIP server performance, we finally provide earlier work on performance improvement of a large-scale server using TCP and on SIP server measurement.

2.2 SIP Specification of Transport Protocols

SIP is the signaling protocol to set up and tear down a real-time communication session such as voice. SIP is a request-response protocol in ASCII text format. By adding a header field or a body content to a SIP message, it can easily extend services, such as a different media session, instant messaging, presence, privacy, and security features. The SIP specification with its extension [Rosenberg *et al.*, 2005] allows SIP entities to choose

a transport protocol from UDP, TCP, and SCTP, per hop. When using UDP, SIP makes use of its timers and retransmission mechanisms for reliability. It mandates UDP and TCP implementation, but implementing TCP for SIP UAs was optional in the earlier version of the specification [Handley *et al.*, 1999]. It has changed to mandatory to implement in response to a demand for handling large SIP messages exceeding the Ethernet MTU.

Although each SIP message used for basic VoIP services is approximately 500 byte long, the size of a SIP message grows as a SIP UA supports many extensions. When a SIP UA supports many media types and their codecs, additional services, and/or security, it generates a large SIP message, especially for the INVITE request, including additional media information, SIP extension headers, and/or cryptographic signatures. As a result, a feature-rich SIP message grows beyond the Ethernet MTU of 1,500 bytes.

To deliver a message exceeding the MTU size at a link layer, using UDP is less desirable since it needs to rely on network layer segmentation, namely, IP fragmentation. IP fragmentation degrades network transmission performance [Kent and Mogul, 1987]. Similar to TCP or SCTP segmentation, IP fragmented packets are not reassembled until they reach the final destination. However, unlike TCP or SCTP segmentation, if one of the fragments is lost, all fragments including successfully transmitted need to be retransmitted. Thus, a SIP UA sending a large message should rely on TCP or SCTP segmentation rather than IP fragmentation.

2.3 SIP Implementation and Deployment of Transport Protocols

Actual implementations, however, according to the reports from the SIP interoperability test (SIPit) events, conform only loosely to the constraint mandating TCP in the SIP specification. In the SIPit19 event held in October 2006 [Sparks, 2006], all 90 implementations that participants brought supported UDP, whereas only 82 percent of them supported TCP. Supporting SCTP was much smaller, six percent. Two years later, a few TCP and SCTP implementations were added [Sparks, 2008]. Yet, a majority of implementations have pre-

ferred UDP.[1]

Many ITSPs have also favored using UDP over TCP, in a user-to-server scenario, to build and operate large-scale VoIP systems. They claimed that a SIP server should only accept UDP messages from users; otherwise, handling a large number of TCP connections imposes additional burdens on the SIP server and would significantly diminish server scalability. Consequently, many ITSPs hesitate to support SIP UAs using TCP despite the demand for handling large SIP messages. Once they are provided with comprehensive quantitative analysis of the cost of using TCP and SCTP for their system, they ought to re-provision server resources for service deployment. This thesis aims to provide provision and deployment guidelines by measuring the impact of using TCP and SCTP.

2.4 TCP and SCTP Features Affecting Server Performance

This section briefly describes TCP and SCTP features that potentially improve SIP server performance. Table 2.4 compares message and connection handling of UDP, TCP, and SCTP. SCTP offers features of both TCP and UDP; similar to TCP, it is connection-oriented and reliable, but at the same time, similar to UDP, it supports one-to-many style sockets, which enable to send and receive from multiple remote endpoints via a single socket, and message-orientation.

2.4.1 Connection Orientation

Using a connection-oriented protocol incurs the cost of handling connections including setup delay. In some cases, however, it helps SIP UAs enable to receive a call request. When a SIP UA is behind a network address translator (NAT) or firewall, using TCP makes it easier to traverse the equipment than using UDP. Once an internal host (i.e., a SIP UA) actively establishes a TCP connection with an external host (i.e., a SIP proxy server), a NAT maintains address and port binding or a firewall keeps the state of the connection in order to allow bidirectional messages over the TCP connection. Similar to TCP, a bidirec-

[1]The recent SIPit summary reported that TCP implementations outnumbered UDP, but the total number of implementations considered, 25, was much smaller. [Sparks, 2011]

	UDP	TCP	SCTP
Support of a message exceeding MTU	No	Yes	Yes
Message-oriented	Yes	No: Byte-stream	Yes
Connection-oriented	No	Yes, established in three-way handshake	Yes, established in four-way handshake
Socket corresponding to remote endpoints	One-to-any	One-to-one	One-to-one or one-to-many
Support of keep-alive to test reachability of the remote endpoints	N/A	Yes, but disabled by default	Yes
NAT binding/firewall state lifetime	short	long	long

Table 2.1: Comparison of message and connection handling among UDP, TCP, and SCTP

tional message flow over UDP is supported by a NAT or firewall, but the binding or state expires more quickly. Based on the lifetime of NAT bindings of the current implementations, a SIP UA using UDP needs to update the NAT binding or the firewall state with a frequency approximately 30 times higher than using TCP, as defined in the SIP Outbound mechanism [Jennings *et al.*, 2009]. Using UDP, therefore, is undesirable not only for SIP UAs behind NATs or firewalls, but also for a SIP proxy server that may be flooded with periodic messages for updating NAT bindings or firewall state.

The update frequency of a NAT binding and firewall state for SCTP is likely to be as low as that for TCP although few NATs and firewalls have supported SCTP so far [Hayes *et al.*, 2008; Hatonen *et al.*, 2010].

2.4.2 One-to-Many Sockets for SCTP

SCTP provides two socket interfaces, *one-to-one* and *one-to-many*, to represent a connection, which is called an *association* in SCTP, between two endpoints. While a one-to-one socket allows existing TCP applications to be easily ported to SCTP, a one-to-many socket

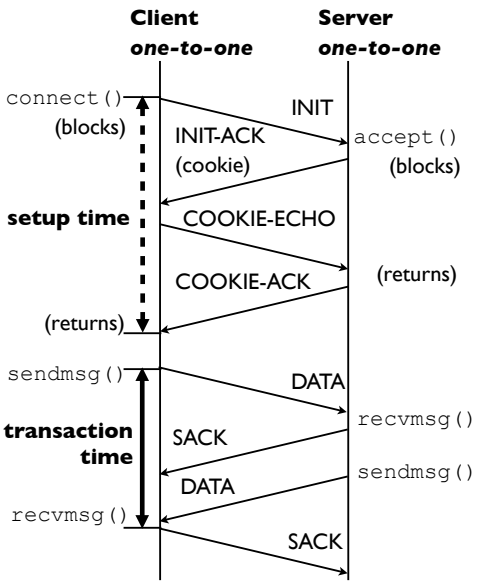


Figure 2.1: Message exchanges using one-to-one sockets for both a server and a client

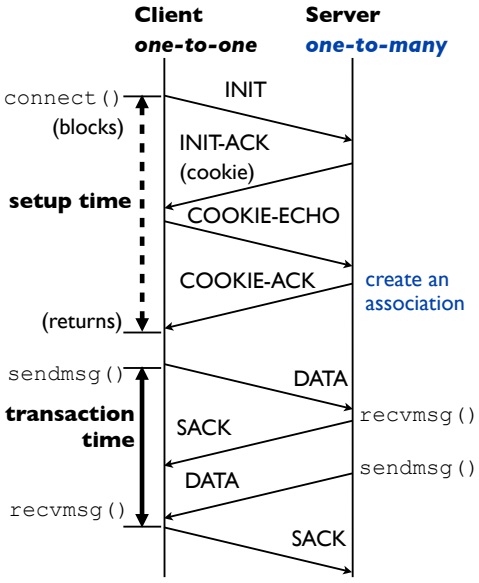


Figure 2.2: Message exchanges using a one-to-many socket for a server

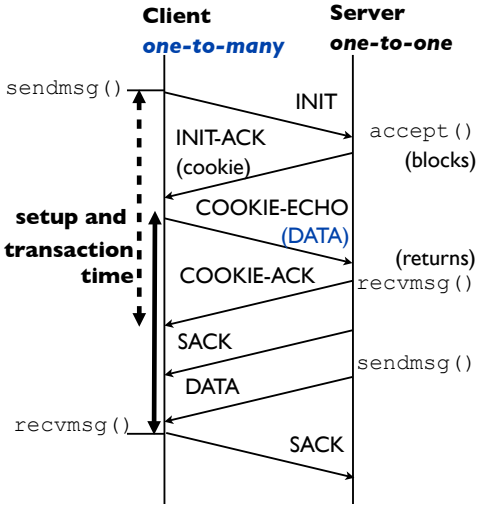


Figure 2.3: Message exchanges using a one-to-many socket for a client: Piggyback setup

enables two SCTP features different from TCP.

First, similar to UDP, using a one-to-many socket enables a single socket to receive messages from multiple associations. As Figure 2.2 shows, a server using a one-to-many socket can create a new association without invoking the `accept()` system call. Consequently, using a one-to-many socket can drastically reduce the number of sockets for a server, although maintaining a large number of associations is still required.

Second, a client using a one-to-many socket, as shown in Figure 2.3, can utilize piggyback setup in order to reduce the setup delay of the SCTP four-way handshake. Figure 2.1 depicts the four-way handshake using a signed cookie to enable the server to accept the SCTP INIT message without maintaining any state. This is a countermeasure to the SCTP INIT or the TCP SYN flooding attack, which is a form of denial of service (DoS) attacks attempting to exhaust memory resources for connections on the server. This four-way handshake requires one more round trip time (RTT) than the three-way handshake in TCP. Thus, the piggyback setup option intends to mitigate the additional RTT of the SCTP handshake by bundling user data into the COOKIE-ECHO message. It is worth noting that piggybacking data on the TCP ACK message does not mitigate setup delay since it can be used over an established connection only.

Therefore, using a one-to-many socket potentially benefits both server and client. At the same time, however, it potentially decreases server request throughput by sharing a single socket with multiple associations. A server using a one-to-many socket sends and receives messages from all associations through a single socket buffer. It demultiplexes received messages by four tuples: source and destination IP addresses and ports. This processing is similar to UDP, but sent messages are kept longer than for UDP since they cannot be removed until the SCTP ACK has been received. As a result, the send buffer may be exhausted at high request rates. This potential impact of sharing a single socket buffer on data transfer latency is investigated by the measurement described in Section 3.5.2. The effect on the number of sustainable associations is described in Section 3.5.1, and the effect of the piggyback setup on mitigating setup delay is in explained Section 3.5.3.

2.4.3 Message Orientation in SCTP

SCTP, similar to UDP, preserves message boundaries so that an application can easily determine whether or not a message has been delivered in full through the socket application interface (API). This message orientation allows to an application to retrieve and parse a single message more efficiently than a TCP byte-stream, which requires to determine the end of a message by parsing the SIP Content-Length header field. Message parsing, however, is necessary for SIP operations regardless of transport protocol. Thus, the benefit of SCTP being message-oriented appears to be negligible, which is examined by the measurement described in Section 4.5.1.

2.5 SIP Servers

SIP is a request-response protocol and has been used in a client-server architecture, similar to HTTP. When SIP servers accept TCP messages, both SIP servers and HTTP servers face similar impact on server performance using TCP. However, major differences exist in service architecture and TCP connection lifetime. It is worth noting that these two types of servers closely resemble each other when HTTP servers are used for real-time communication [Alvestrand, 2011].

2.5.1 Service Architecture

SIP servers consist of three logical functions: a registrar for collecting user location or contact addresses, a proxy for forwarding request and responses, and a redirect server for resolving the address of a callee or diverting a request to another server. A proxy operates in either of three modes: stateless, transaction-stateful, or dialog-stateful. A stateless proxy forwards messages without maintaining any transaction state or generating a 100 Trying response. A transaction-stateful proxy maintains transaction information while a transaction exists, for example, from the instant that it receives a SIP INVITE request to responding with a 200 OK response. A dialog-stateful proxy maintains the dialog or call state machines for the duration of a call. Proxy servers for ITSPs mostly operate in a transaction-stateful or dialog-stateful mode to offer services, such as call admission control

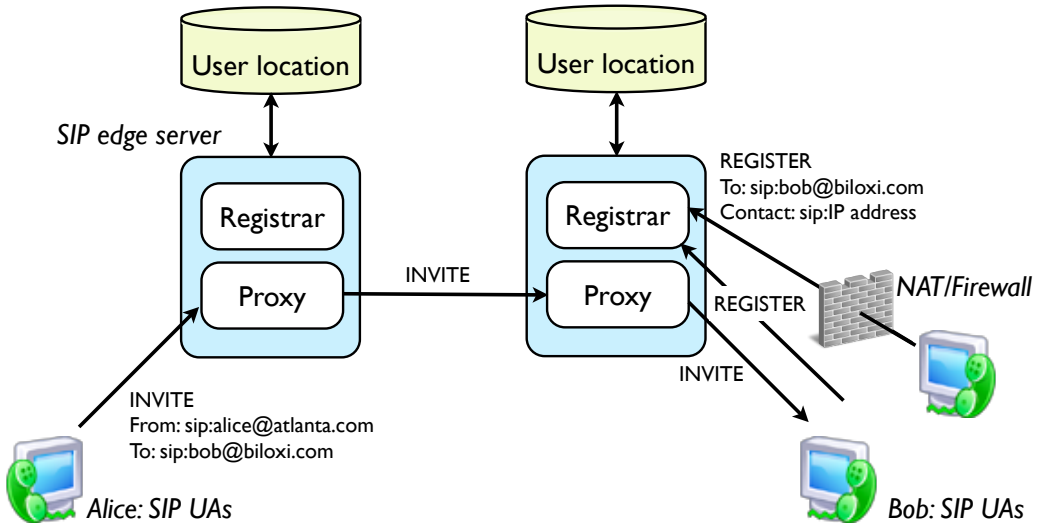


Figure 2.4: SIP edge servers in SIP trapezoid model

based on the codecs in the SDP [Marshall, 2003].

Figure 2.4 illustrates a typical SIP service architecture in SIP trapezoid model where the signaling path between a caller and the callee involves two proxy servers. To be reachable, the callee, Bob, needs to send the registrar a SIP REGISTER request including the IP address he is currently using. When a caller, Alice (sip:alice@atlanta.com), makes a call to Bob (sip:bob@biloxi.com), she sends a SIP INVITE request to a proxy server that is responsible for the atlanta.com domain. The proxy then forwards the request after resolving the service name of the biloxi.com domain to the IP address using DNS or other translation means. The proxy server at the biloxi.com domain finally forwards the request after looking up Bob's contact address that the registrar for the same domain collected. This message direction at the last hop distinguishes SIP from other communication servers like polling-based email[2] or Web servers. A SIP UA receives a request from its inbound SIP proxy server. Since it involves looking up user location to process inbound messages, a proxy server is typically co-located with a registrar for the same domain. Such a server is often referred to as a *SIP edge server*.

This SIP edge server setting is preferred by UAs under two practical scenarios, located

[2]Push email services are similar to SIP. For example, using IMAP IDLE or NOTIFY [Leiba, 1997; Gulbrandsen *et al.*, 2009] requests enables the IMAP server to send a notification message to a client.

behind a NAT or firewall and needing a secure channel. One scenario is the case where a SIP UA resides behind a NAT and needs to receive a SIP request from the proxy, as described in Section 2.4.1. The NAT traversal is made possible if the SIP UA keeps refreshing the NAT binding that was created upon its registration.

Another scenario is the case where a SIP UA needs to receive a SIP request protected with Transport Layer Security (TLS) or Datagram Transport Layer Security (DTLS) without establishing a new security session as the server, which requires its X.509 public key certificate (PKC). If the UA maintains and reuses a TLS connection or a DTLS security association (SA) established upon its registration, the UA can avoid requiring its PKC for a new TLS connection or DTLS SA initiated by the proxy. A similar situation happens when a SIP UA intends to reuse an Internet Protocol Security (IPSec) SA between the UA and the registrar without re-entering its user credential such as a one-time password.

Thus, it is useful for SIP UAs under these practical considerations, regardless of transport protocol, to maintain a connection or binding between a UA and the registrar and reuse it between the UA and the proxy. Consequently, SIP edge servers consisting of registrar and proxy functions have been widespread for both small-scale and large-scale ITSPs. In this thesis, we simply call a SIP edge server a SIP sever.

2.5.2 Long Connection Lifetime

The SIP specification and its SCTP extension recommend a persistent TCP connection and SCTP association, similar to HTTP/1.1 [Fielding *et al.*, 1999]. However, a SIP server, on behalf of users, needs to maintain idle connections longer than HTTP. This requirement comes from the nature of real-time communication and practical considerations. SIP UAs need to wait for arbitrary incoming call requests through the server. If they are behind a NAT or firewall, or intend to use a secure connection with the server, they need to keep the connection open, as explained in Section 2.5.1. In fact, typical HTTP server implementations close idle connections more quickly than SIP severs do. For example, the default idle timeout for Apache 2.4 [Apa, 2012] is five seconds while the default configurations for the Kamailio SIP Server 3.2 (previously called OpenSER) [Kam, 2011] are 120 seconds for a TCP connection and 180 seconds for an SCTP association. Although these configurations

cab be adjusted based on the requirements for server performance and services, these longer timeout values indicate that SIP servers need to maintain longer connections with UAs.

2.6 Capacity Requirements for a SIP Server

A single server does not have to accommodate the whole user population of an ITSP. Each ITSP provisions their server resources based on the baseline capacity data per server. Our measurement goal is to provide the baseline capacity data of a large-scale SIP server, especially the additional cost of using TCP or SCTP. Thus, it is unnecessary to define precise capacity requirements for a large-scale server. However, a rough estimate of the requirements is needed to prepare our measurement environment and evaluate the results.

Traditionally, a call server or central office in the PSTN has been sized by three parameters: the number of users to be connected, maximum call traffic that can be handled, measured in erlang,[3] and maximum processing capacity for call attempts in busy hour call attempts (BHCAs) [Schmidt and Lopez, 1999]. Are these three parameters still useful for sizing a SIP server? The following paragraphs examine each parameter.

The number of users is still the key factor in determining the server capacity since SIP operations involve looking up user information, such as contact addresses for registration and routing. Furthermore, a SIP server accepting connection-oriented transport protocols needs to maintain a persistent connection per user, as described in Section 2.5.2. Thus, the capacity of a SIP server depends on the size of the user population it needs to accommodate.

Traditional call traffic is less important since a SIP server needs to process not only call traffic but also a large volume of messages for other purposes such as registering user location, maintaining connections, and extended services (e.g., instant messaging). Call traffic in the PSTN has been modeled with call duration and inter-arrival times as an exponential distribution [Bellamy, 2000]. Many researchers have attempted to characterize VoIP traffic, but it is still under investigation [Jiang and Schulzrinne, 2000; Dang *et al.*, 2004; Birke *et al.*, 2010]. A traditional call holds network facilities, such as a trunk line, for the duration of a call, whereas a VoIP call only consumes memory to maintain dialog state

[3] One erlang of carried traffic refers to a single resource being occupied continuously.

on a SIP proxy server operating in a dialog-stateful mode, as described in Section 2.5.1. A SIP proxy server in other modes, namely, transaction-stateful or stateless, ignores the dialog state. Instead, a transaction-stateful proxy server maintains transaction state, which usually lasts for a short time period on the millisecond or second time scale. The exception is a transaction that involves human interaction. For example, the INVITE-200 OK transaction in unanswered call scenarios is likely to be long. Yet, the lifetime of the state of both a dialog and transaction is shorter than the TCP connections or user location information maintained on a SIP server. Thus, neither call holding time nor the transaction time considerably affects the SIP server capacity.

Another crucial factor in sizing a SIP server is maximum processing capacity for call attempts and other types of messages. Assuming a user population of 100,000, the average call duration is 3 minutes and 0.1 erlangs based on traditional call traffic model [Bellamy, 2000], the request rate in BHCA is 200,000 BHCA (= 100,000×0.1×(3,600/180)), which corresponds to 55.6 requests/second. On the other hand, observing real VoIP traffic at two ITSPs, one having 2,000,000 users [Birke *et al.*, 2010] and another having 100,000 users, a relatively smaller user population [Baset *et al.*, 2010], their peak call request rates were lower, 21.7 and 15 requests/second, respectively. Higher traffic was reported for registration and updating NAT bindings or firewall state at the smaller ITSP. Periodic messages for registration every 50 minutes and NAT binding update every 15 seconds were observed since most SIP messages were transmitted over UDP.

If SIP UAs use TCP or SCTP instead of UDP, the interval of NAT binding update messages can be longer, every 14 minutes [Jennings *et al.*, 2009]. Even if a SIP UA directly connects to a SIP server, it sometimes needs to keep a TCP connection protected using TLS by sending the TCP or SIP keep-alive messages to the SIP server, as described in Section 2.5.1. The interval of TCP or SIP keep-live messages is two hours or longer [Braden, 1989] or 95 - 120 seconds [Jennings *et al.*, 2009], respectively. However, it is impossible and unnecessary for the SIP server to examine the situation of each SIP UA. For whatever the purpose of periodic messages, NAT binding update or the TCP or SIP keep-alive, SIP UAs send similar messages consisting of zero or four byte payload data to the SIP server. Thus, the SIP server needs to support the most frequent message type, namely, SIP keep-alive

	Required values
User population	300,000
Number of concurrent connections	300,000
Sustainable call request rate	167 requests/second
Sustainable registration request rate	104 requests/second
Sustainable keep-alive request rate	3,158 requests/second

Table 2.2: Requirements for a SIP server

messages, in terms of CPU usage.

In summary, for server sizing purposes, key factors are sustainable request rates for call, registration, and SIP keep-alive messages, in addition to the size of the user population.

Table 2.2 summarizes the capacity requirements for a SIP server in our measurement. The number of users is determined based on our discussion with a telecommunication carrier and our observation of high-capacity call servers in the market such as Lucent's 5E-XC [Alcatel-Lucent, 2002] and Siemens EWSD [Siemens, 2007]. A large-scale SIP server needs to support approximately 250,000 - 300,000 users. These values are more than twice the size of the user population for the ITSP whose traffic was observed, but agree with the operational value that is used by ITSPs. ITSPs usually provision their resources based on the operational upper bound, which is calculated by approximately halving the technical upper bound [WIK, 2000]. Given the user population, 300,000, the following sustainable request rates are calculated. Sustainable call request rate, 167, is calculated based on the traditional call traffic model, assuming 0.1 erlangs and three minute call holding time. For registration, each user is assumed to update its contact addresses every 48 minutes, which is 80 percent of the default registration lifetime, one hour. For maintaining TCP connections, assuming the worst-case scenario, all UAs send keep-alive messages every 95 seconds. It is worth noting that this keep-alive request rate, 3,158 requests/second, is the largest among all request rates, but is still much lower than UDP, which requires a four times higher request rate because of the shorter NAT binding lifetime.

2.7 Components Affecting Server Performance

Our study focuses on the impact of transport protocols on SIP server performance. Conventional SIP server performance has been measured to determine the cost of SIP operations [Cortes *et al.*, 2004]. Several experiments compare sustainable request throughput and other metrics for a SIP proxy server using UDP and TCP, and estimate the penalties of using TCP [Salsano *et al.*, 2002; Nahum *et al.*, 2007]. However, such comparison often fails to determine where the throughput reduction factor exists, either the transport protocol implementation in the operating system or a server application implementation, or both. We decompose server processing which affects performance in order to clarify what a common pitfall is.

1. Processing the transport protocol by the operating system.

   This includes maintaining the state machine of a connection and passing messages to and from applications.

2. Processing sockets or connections by an application using the socket APIs.

3. Processing application-protocol-specific operations by an application including message parsing.

   SIP operations consist of handling SIP request and responses and managing user location addresses into a database. They optionally include user authentication as well as managing the transaction state and the dialog state.

4. Processing security functions by the operating system (e.g., for IPSec) or an application (e.g., for TLS or DTLS), if needed.

The choice of a transport protocol clearly affects the first two components implemented by the operating system and by an application. The choice also affects part of the third component, message parsing, since transport protocols frame messages differently, as described in Section 2.4.3. A common pitfall in performance analysis is to confuse the cost of these first two types of processing. Processing TCP connections by an application depends on which system calls the server implementation invokes, and how. It is difficult to separate the cost

for these two types of processing from the measurement results of SIP server performance. For example, the results obtained by profiling CPU cycles apparently identify the operating system as being responsible for the reduction of the throughput, but that was due to the inefficient handling of TCP connections in a SIP server implementation [Nahum *et al.*, 2007; Ram *et al.*, 2008]. To avoid this pitfall, our measurement proceeds in stages to clarify the real impact of the transport protocol. We next review related work: the improvement of TCP server performance in general, SIP server measurement, and the improvement of SIP server performance.

2.8 Scalable Servers Using TCP

Since TCP has been widely used for popular protocols, such as HTTP, building scalable servers using TCP is a long-standing research topic while using SCTP is relatively new. The performance of the network I/O has been drastically improved by scalable mechanisms such as the `epoll()` system calls [Libenzi, 2002] for Linux systems and the `kqueue()` system call [Lemon, 2001] for BSD systems. Upon being notified of events by the operating system using conventional mechanisms (e.g., the `select()` system call), an application that instructs the operating system to wait for events on a set of file descriptors needs to walk through the entire set in order to find events. In contrast, these new mechanisms enable the application to efficiently retrieve events from an adjusted set of file descriptors on which events occurred. As a result, using the new mechanisms reduces the number of file descriptors and the memory usage for them. The Linux scalability effort project [Nagar *et al.*, 2004] demonstrated that an HTTP server can sustain a high throughput for 100 active connections while maintaining 60,000 idle connections. Kegel [Kegel, 2006] described the configurations related to the network I/O and event delivery to support more than 10,000 clients for a large-scale HTTP server.

We built our measurement servers based on these techniques. We used the `epoll()` system calls to wait efficiently for events on a large number of sockets and lifted the system limit of the number of sockets. Our challenge is to identify the impact of TCP connections and SCTP associations on a server that needs to handle more users by an order of magnitude.

2.9 SIP Server Performance

Measurement

Many measurement efforts have been undertaken to analyze SIP server performance, albeit mostly using UDP or a small number of TCP connections. Several analyses compared the cost of SIP operations on a SIP proxy server using UDP and TCP [Salsano *et al.*, 2002; Nahum *et al.*, 2007; Ram *et al.*, 2008]. The performance penalties of using TCP varied according to the server implementations. The sustainable request throughput ratio of using UDP to using TCP ranged from 1.8 to 2.8 for the basic SIP call setup and teardown and from 1.1 to 1.8 for the basic SIP operations with validating user authentication. Thus, adding procedures on the SIP server reduces the impact of using TCP on the request throughput ratio. Our measurement focuses on the SIP registration without validating user authentication or any other procedures in order to provide baseline performance data that can be used for comparing transport protocols.

A few measurement efforts have been conducted using a large number TCP connections assuming one connection per user. The most relevant experiments were conducted by Shemyak and Vehmanen [Shemyak and Vehmanen, 2007]. They emphasized the advantage of using the `epoll()` system calls on an HTTP server, and claimed that their SIP server could maintain 100,000 TCP connections. They also pointed out the CPU bottleneck by handling SIP REGISTER requests as keep-alive messages for both using TCP and UDP. However, the assumption of using REGISTER requests, which usually involve database access, is not appropriate for keep-alive purposes except mobile phone settings. This thesis estimates the impact of receiving four byte message of the SIP keep-alive mechanism [Jennings *et al.*, 2009] based on the measurement result of the cost of handling TCP FIN requests, which contain zero payload.

Another relevant investigation was performed by Shen and his colleagues [Shen *et al.*, 2010]. They compared sustainable call request rate in different proxy roles (e.g., inbound or outbound) using UDP and multiple TCP connections. The differences for proxy roles imply the costs of opening and closing TCP connections, which reduce the sustainable request throughput by a factor of 1.7 - 4.1. They also found an implementation problem in

their server implementation causing TCP to perform poorly. Their measurements covered all proxy roles, but the reason for the wide range of the cost of using TCP is not clear. Our measurement provides more details of the cost of using TCP, identifying the impact of using TCP each on our echo server and on the SIP server.

Many studies on SCTP server performance have compared using SCTP with TCP server performance. However, most benchmark used services sending a large message like file transfer. A few studies performed using SIP, but mostly demonstrated the effect of using SCTP in a proxy-to-proxy scenario. This is a natural consequence since SCTP was designed for a connection between servers. Camarillo and his colleagues [Camarillo *et al.*, 2003] compared UDP, TCP, and SCTP for a small number of connections between SIP proxy servers to evaluate performance in a congested network, comparing the impact of packet loss and delay, not CPU load. In contrast, our measurement compares these three protocols in a user-to-server scenario since some SCTP features potentially benefit the performance of a SIP sever maintaining a larger number of connections with users.

SIP Server Performance Improvement

Among SIP operations, message parsing and string manipulation are major factors in determining request throughput. To improve performance, Janak [Janak, 2003] suggested limiting message parsing to several selected header fields and selected components with these header fields. Cortes and his colleagues [Cortes *et al.*, 2004] compared four SIP server implementations using UDP, demonstrating the processing times of these two factors dramatically differed among these four implementations. Our SIP server has already implemented limited message parsing. This thesis proposes further accelerating message parsing when only the SIP method type matters, such as for the purpose of prioritizing messages at high loads.

Software architecture for a scalable SIP server has been studied by comparing server performance using UDP. Singh and Schulzrinne [Singh and Schulzrinne, 2005] compared the performance of sipd, a SIP sever our laboratory developed, for different software architectures: event-based, thread pool, and process pool. They suggested that the process pool model has the best performance in terms of response time. Additionally, they proposed a

two stage architecture, where servers at the first stage dispatch messages to multiple servers at the second stage in order to improve concurrency and reliability. Our SIP server measurement uses the same SIP server implementation, sipd, but in the thread pool model. This thesis discusses the impact of the transport protocol on SIP server performance, not the impact of the software architecture.

SIP Server Benchmark

SIPstone [Schulzrinne *et al.*, 2002] is a benchmark tool set which our laboratory developed. This tool measures sustainable request rates of SIP registar and proxy using UDP and TCP. This tool also provide CPU usage, memory usage, and the transaction response time. For our measurement, we added TCP connection configurations to this tool so that each emulated user client can establish a separate TCP connection to the server under test (SUT), and select to close or maintain the connection. While earlier measurement efforts used SIPstone, relatively recent ones used SIPp [Gayraud and Jacques, 2010], an open source benchmark tool, including SIPstone features. This benchmark is included in a benchmark package, SPECsip_Infrastructure2011 [SPEC, 2011]. Although SIPp can generate TCP messages, the benchmark package allows to send only UDP messages to the SUT. Metrics standardization [Davids *et al.*, 2011] has been in progress in the IETF. They do not limit the transport protocol to use. The maximum registration rate in the draft is the same as the sustainable request rate of a registrar in our measurement.

Chapter 3

Understanding the Impact of Using TCP and SCTP on Echo Server Scalability

3.1 Introduction

This chapter describes echo server measurement using a connection-oriented transport protocol, either TCP or SCTP. An echo server is simple and often used as the first example of socket programming. Our echo server responds to an echo client with a copy of a received message without any message parsing. Since the echo server minimizes the impact of the application, the measurement results help us better understand the impact of using TCP or SCTP on server performance. Our goal is to provide enough details from a kernel perspective to understand the cost of handling TCP connections or SCTP associations and the bottleneck. Our measurement results show the upper limit of concurrent TCP connections on a 32-bit commodity server in Section 3.4, indicating its potential capacity to meet our requirements for a SIP server described in Section 2.6. On the other hand, our SCTP measurement (Section 3.5) demonstrates the effect of using a one-to-many socket and points out a few implementation issues in Linux that make it difficult to accommodate a large user population.

Transport protocol	Goals of measurement	Metrics
TCP	To understand - Cost of establishing connections - Cost of maintaining connections	- Maximum number of concurrent connections - Memory usage - CPU utilization
SCTP	To understand - Cost of establishing connections - Cost of maintaining connections - Effect of one-to-many sockets - Effect of piggyback setup	- Setup and transaction response times

Table 3.1: Measurement goals and metrics

3.2 Measurement Goals and Metrics

Table 3.1 shows our goals and metrics for the echo server measurement. Our measurement first focuses on handling TCP connections to determine if the SUT has the potential to meet the capacity requirements for a large-scale SIP server described in Table 2.2, especially the maximum number of concurrent connections. Our measurement then compares the setup time, transaction response time, and CPU utilization to UDP. This comparison intends to identify the impact of a large number of TCP connections: establishing and of maintaining TCP connections. Our measurement finally compares the same metrics in different configurations of the SCTP socket styles for the server and clients to examine the effects of using a one-to-many socket and piggyback setup described in Section 2.4.2.

3.3 Experimental Setup

3.3.1 Server Under Test

The server under test (SUT) ran on a dedicated host equipped with Pentium IV 3 GHz 32-bit dual-core CPU and 4 GB of memory. The SUT ran Linux 2.6.23 configured with the default virtual memory (VM) split of 1G/3G, where the kernel space was 1 GB and the user space 3 GB. The kernel space was temporarily expanded to 2 GB by modifying the VM

split for measuring the maximum number of concurrent connections. The measurement for TCP and SCTP was performed in 2006 and 2007, respectively.

The system parameters were configured to allow a large number of concurrent TCP connections or SCTP associations. The upper limits of the number of file descriptors per system, per process, and per user were raised to 1,000,000. Specifically for SCTP, the following parameters were also modified. To use a one-to-many socket with a large number of associations, the socket buffer size was expanded to approximately 4 MB. To disable the SCTP keep-alive mechanism, the keep-alive interval was increased from the default interval, 30 seconds, to 360 seconds, which was longer than our test duration. By disabling this keep-alive mechanism, we eliminated the unnecessary cost of handling keep-alive messages to focus on our measurement in a user-to-server scenario. The SCTP keep-alive mechanism was originally designed for carrying a telephony signaling protocol, SS7 over IP, between servers in order to achieve robustness using server redundancy.

Our echo server consisted of a single process and single thread. To maximize server scalability, it used the `epoll()` system calls described in Section 2.8. The echo server kept connections open until the echo clients requested to close them or the measurement ended. Upon socket creation, the echo server modified the following socket options. The Nagle algorithm [Nagle, 1984] was disabled to eliminate unnecessary delay when sending multiple small messages. To disable the SCTP keep-alive mechanism for one-to-one sockets, the disable flag was set. For one-to-many sockets, it needs to specify an association ID; thus, we simply expanded the keep-alive interval on the system explained above.

3.3.2 Echo Clients

Our echo clients ran on up to ten hosts with Pentium IV 3 GHz 32-bit CPUs and 1 GB of memory running Redhat Linux 2.6.9. These hosts communicated with the SUT over a 100 Mb/s Ethernet connection at light load. The RTT measured by the `ping` command was roughly 0.1 milliseconds (ms).

For each echo client, the following system parameters were configured to allow a large number of concurrent TCP connections or SCTP associations. The range of ephemeral local port was expanded to 10,000 - 65,535 so that each client could have 55,535 concurrent

connections. The upper limit of the number of file descriptors per process was also raised to 60,000. Similar to the SUT, the SCTP keep-alive mechanism was also disabled for all echo clients.

Each echo client consisted of a single process and a single thread. The echo client sent an approximately 500 byte message over a separate TCP connection to the SUT, and maintained or closed the connection according to the specified configuration. For SCTP measurement, the echo client sent an approximately 1,600 byte message to provide the baseline measurement data for the message orientation test described in Section 4.5. When comparing the setup and transaction response times between TCP and SCTP, the TCP measurement was performed again using approximately 1,600 byte messages.

3.4 Measurement Results Using TCP

This section first provides the measurement results of the echo server using TCP, the maximum number of concurrent connections and memory usage. This section then shows the setup and response times and CPU utilization to estimate the impact of opening, maintaining, and closing a large number of TCP connections.

3.4.1 The Maximum Number of Concurrent Connections

Figure 3.1 shows the memory usage as a function of the number of TCP connections established and maintained for the echo server. This result indicates that overall memory usage increased linearly with the number of concurrent connections until the number of connections reached approximately 419,000 in the default VM split configuration of 1G/3G, where the kernel memory space was 1 GB and the user memory space was 3 GB. Three different request sending rates, 200, 2,500 and 14,800 requests/second, had no significant impact on either of overall or socket buffer memory usage. At any of the three request rates, memory usage for TCP socket buffers was less than 20 MB. By expanding the kernel memory space to 2 GB and shrinking the user memory space to 2 GB, the maximum number of connections rose to 520,000 connections. At or above each maximum number of connections, the server halted. Thus, we can deduce that the bottleneck in our measurement is the amount

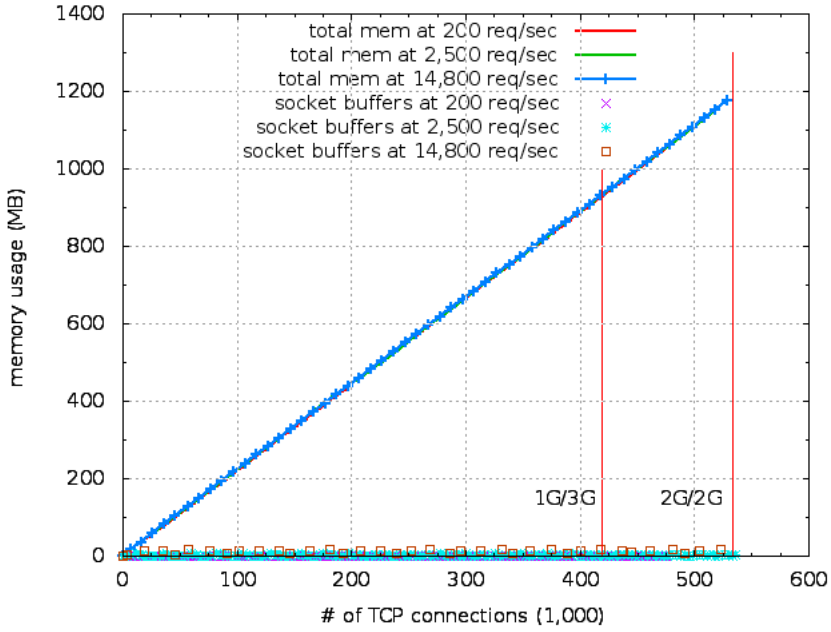


Figure 3.1: Memory usage as a function of the number of TCP connections for echo server

of kernel memory for TCP connections as long as these connections remain, not the amount of socket buffer memory, which is dynamically allocated depending on the request rate and the message size. By dividing the overall memory usage by the number of connections, the memory footprint per TCP connection turns out to be 2.27 KB. This bottleneck has been confirmed by investigating the usage of the slab cache, which is a memory implementation in Linux for frequently allocated and deallocated objects, as described in Appendix A.1.

Therefore, the results of the memory usage measurement confirm that each TCP connection requires 2.27 KB of the slab cache and the bottleneck of sustainable concurrent connections is the amount of allocatable kernel memory for the slab cache. These results demonstrate that the echo server with the default VM split configuration can meet the design requirement of the number of concurrent connections described in Section 2.6. The maximum number of connections established in our measurement is approximately 140 percent of our design target of 300,000 connections. Since SIP and database application processes consume memory in each user space, these measurements also identify the bottleneck of the kernel memory space and demonstrate that a SIP server can fulfill the size requirement of the user population by a large margin.

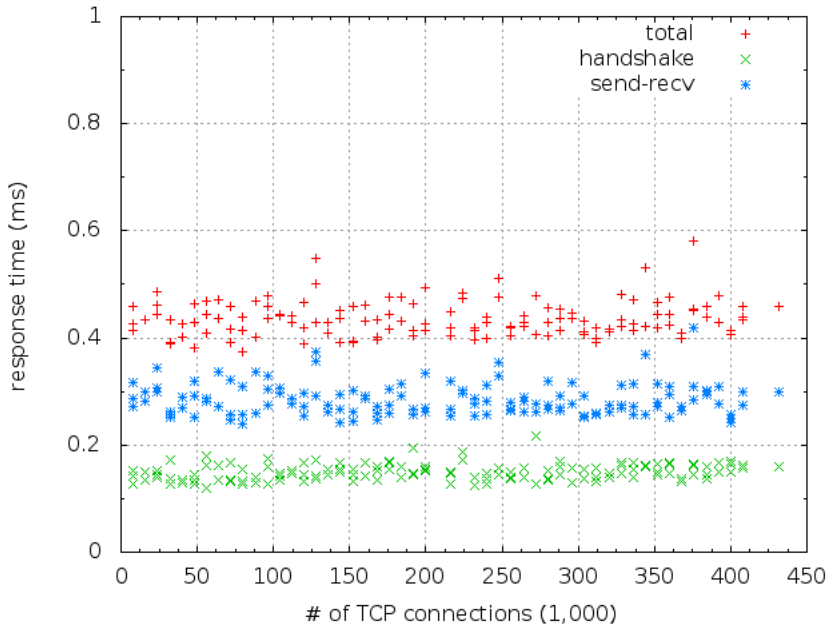


Figure 3.2: Response times for echo server as a function of the number of concurrent TCP connections

3.4.2 The Cost of Establishing and Maintaining TCP Connections

Section 3.4.1 explained that the amount of memory used by establishing or maintaining a TCP connection is 2.27 KB. The request rate or message size does not significantly affect memory usage. This section looks at the other metrics of the cost of establishing and maintaining a TCP connection, namely, setup and transaction response times and CPU utilization.

Figure 3.2 plots the setup and transaction response times as a server establishes new connections at 14,800 connections/second and accumulates them. While the echo server accumulates TCP connections until approximately 419,000, the elapsed times for the TCP three-way handshake remain constant at less than 0.2 ms as the "handshake" data points show and the transaction response time remains constant at around 0.3 ms for the "send-recv" data points show. Thus, opening a TCP connection adds less than 0.2 ms in our measurement environment to the transaction response time, and maintaining connections imposes no cost in terms of the transaction response time.

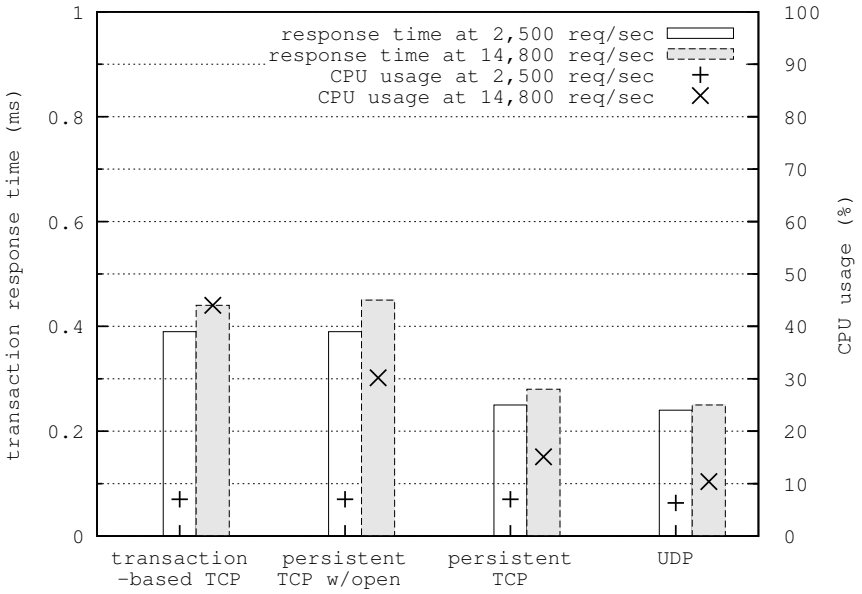


Figure 3.3: Average transaction response times (left axis) and peak CPU utilization (right axis) for echo server

To provide the CPU utilization for opening and closing TCP connections as the baseline measurement data for the SIP server measurement in Section 4.4, Figure 3.3 compares peak CPU utilization and the average transaction response times for the following three TCP configurations and UDP.

Transaction-based TCP Each client creates a new TCP connection before starting a transaction: sending a message to the server and receiving a message copy in return. After the transaction, the client closes the TCP connection.

Persistent TCP with open Each client creates a new TCP connection before starting a transaction. The client keeps the TCP connection open after the transaction.

Persistent TCP Each client sends and receives a message using a pre-established TCP connection with the server. The client keeps the TCP connection open after the transaction.

The differences between the transaction-based TCP and persistent-TCP with open configurations indicate the cost of passively closing TCP connections. The differences between

two types of persistent TCP configurations indicate the cost of passively opening TCP connections, matching the results of setup and transaction response times shown in Figure 3.2.

Therefore, passively opening and closing TCP connections costs a negligible percentage of CPU time at 2,500 requests/second since the CPU times of three TCP configurations are similar. As the request rate increases, the cost of opening and closing TCP connection also increase. They cost approximately 14 percent of CPU time each at 14,800 requests/second, which is significantly above the design requirement described in Section 2.6. Consequently, under the request rates of the design requirement, CPU overhead for opening and closing TCP connections does not limit the scalability of the echo server and is not anticipated to limit the scalability of the SIP server.

It is worth noting that neither TCP nor application-level keep-alive messages were sent in our measurement. We roughly estimate the cost of these keep-alive mechanisms on CPU utilization based on the results in Figure 3.3. If either server or client enables the TCP keep-alive mechanism and maintains a TCP connection more than the shortest keep-alive idle timeout, two hours [Braden, 1989], it sends a TCP keep-alive message. A SIP server may activate the TCP keep-alive mechanism, but SIP UAs do not have to use the TCP keep-alive mechanism since they often make use of the SIP keep-alive mechanism for their purposes, as described in Section 2.4.1. Thus, we assume that only the SIP server activates the TCP keep-alive mechanism with two hour idle timeout. If the server sends all of 300,000 clients keep-alive messages, the sending rate turns out to be 41.7 requests/second and the receiving rate is similar. At this low rate, the additional CPU utilization is negligible since the TCP keep-alive messages in Linux have no data like the TCP FIN messages and the cost of passively closing TCP connections on CPU time at 2,500 request/second was negligible. Even though the server needs to manage TCP keep-alive state and repeat sending the keep-alive message up to nine times in Linux if clients are unreachable or the response packets are lost, the cost of CPU utilization is still low at that low rate. In addition to CPU time, this TCP keep-alive mechanism needs memory to store connection state, but this memory footprint is much smaller than for a TCP socket. On the other hand, the sustainable request rate required for SIP keep-alive messages is much higher, approximately 3,200 requests/second, as shown in Table 2.2. If a SIP server can process SIP keep-alive

messages, each consisting of two sets of carriage return and line feed (CRLF) [Jennings *et al.*, 2009] as efficiently as the TCP FIN messages, the cost of processing SIP keep-alive messages is less than 14 percent, estimated based on the cost of passively closing TCP connections at 14,800 request/second was approximately 14 percent of CPU time, as explained above. Thus, the cost of maintaining connections, which mainly affects CPU time, depends on the request rate of the SIP keep-alive messages.

3.5 Measurement Results Using SCTP

This section describes the measurement results of the echo server using SCTP, compared to those of using TCP shown in Section 3.4. This section also investigates the effect of the one-to-many style socket to find better SCTP configurations that minimize the overhead of SCTP functions for a server.

3.5.1 The Maximum Number of Concurrent Connections and the Effect of One-to-Many Sockets

Table 3.2 compares the maximum numbers of SCTP associations and TCP connections that can be established on a single server and the memory usage per SCTP association or TCP connection.[1] The numbers were measured by increasing associations or connections on the echo server until the system yielded an out-of-memory error for sockets. The maximum number of SCTP associations is only 17 - 21 percent of the TCP limit. Contrary to the expectation that using a one-to-many socket can reduce the memory footprint per SCTP association, the reduction barely makes a difference in comparison to approximately 2 KB required by each TCP connection.

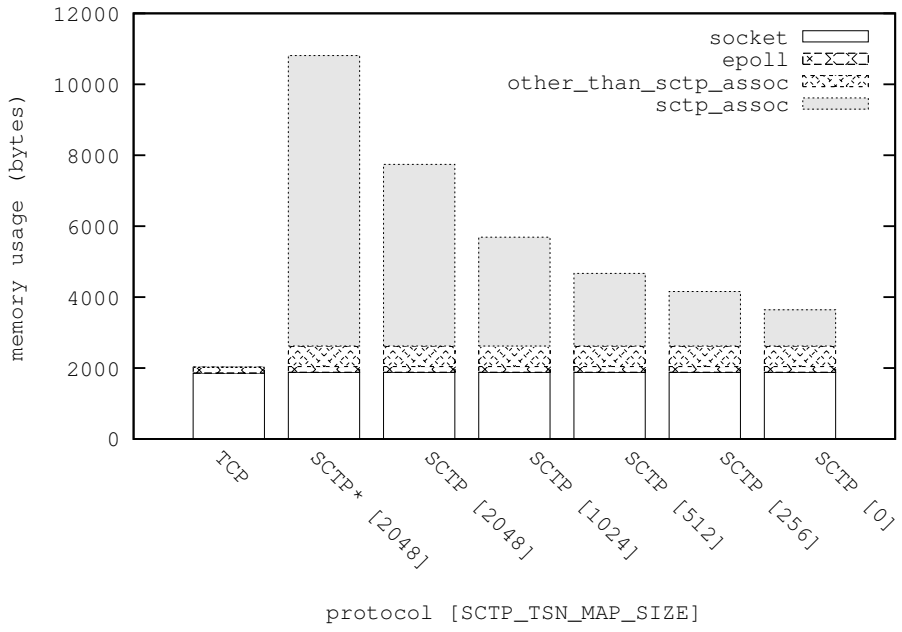
Even though the server using a one-to-many socket only needs to create a single socket, the SCTP association-related data structures consume four times the amount of memory for the socket, as shown in Appendix A.2. Several SCTP-specific data objects suffer from approximately 3 KB of internal fragmentation since they are allocated from general purpose

[1] Although the message size sent from the echo clients was approximately 1,600 bytes, larger than that of using TCP, this difference did not affect the memory usage per association, as explained in Section 3.4.1.

	SCTP		TCP
Socket style on server	One-to-one	One-to-many	
Number of associations/connections	74,680	90,607	419,019
Ratio	0.17	0.21	1.00
Memory usage per association/connection	11.12 KB [3.93 KB]*	8.90 KB [1.83 KB]*	2.05 KB

* After reducing the number of entries in a TSN map to 256

Table 3.2: Maximum numbers of concurrent associations/connections and memory usage



*: Allocated from general purpose slab object

Figure 3.4: Memory usage for a socket as a function of `SCTP_TSN_MAP_SIZE`

slab objects. A general slab object, unlike a data-specific slab object, is allocated only in increments of powers of two. Furthermore, the `sctp_association` structure, which is the largest among SCTP-specific data objects, consumes 8 KB and the dominant sub-member of the `sctp_association` structure accounts for 4 KB. This sub-member stores a transmission sequence number (TSN) map, which traces received TSNs to support unordered data delivery and selective ACK. Thus, to cut down the memory footprint of association-related data, it is crucial to reduce the size of the TSN map.

A simple and easy way to reduce the memory usage is to adjust the TSN map size depending on the requirements of handling unordered data delivery.[2] A large size of the TSN map is unnecessary for SIP signaling, especially between a SIP UA and a SIP server for two reasons. First, a single SIP message is split into at most a few SCTP segments. Second, a SIP UA usually exchanges requests and responses with another SIP UA via SIP servers in an interactive manner; thus, it does not send multiple SIP requests without waiting for their responses. Although SIP signaling between SIP servers, where SIP messages are aggregated and transmitted over a small number of SCTP associations, needs to support unordered data delivery, a small size of the TSN map is sufficient for SIP signaling. Thus, by reducing the TSN map size to 256 entries, we could drastically reduce the memory footprint per SCTP association using a one-to-one socket, as shown in Figure 3.4. As a server maintains more associations, the memory footprint per association using a one-to-many socket falls to a smaller amount. When a server maintains 90,000 associations, for example, the memory footprint per association is 1.83 KB, which is even less than that of TCP value, as shown in Table 3.2. Thus, by removing unnecessary memory allocation, using a one-to-many socket could be very effective in increasing the maximum number of concurrent associations to fullfil the design requirement of a large-scale SIP server, as described in Section 2.6.

[2] The memory usage of the TSN map was radically reduced by replacing a byte map with a bit map that can dynamically grow and shrink. This improvement has been incorporated in the Linux kernel 2.6.27 released in October 2008.

	SCTP			TCP
Socket style on server	One-to-one		One-to-many	
Setup style	Regular	Piggyback		
Setup (ms)	0.34	0.84	0.38-170.91 [0.34][a]	0.17
Transaction (ms)	0.54		0.65-34.14 [0.53][a]	0.48
Total (ms)	0.88	0.84	1.03-205.05 [0.87][a]	0.65

[a] After replacing a linear search with a hash table lookup

Table 3.3: Setup and transaction response times for SCTP and TCP

3.5.2 Data Transfer Latency and the Effect of One-to-Many Sockets

To identify how sharing a single socket buffer in a one-to-many socket slows down data transfer latency, the setup and transaction times were measured and compared among three SCTP configurations and TCP. This measurement also intend to identify how much piggybacking data in the handshake can reduce the overhead of the SCTP four-way handshake.

The setup time of an association is the elapsed time from the instant that the echo client invokes the `connect()` system call to returning from it, as described in Figures 2.1 and 2.2. The transaction time is the elapsed time from the instant that the echo client invokes the `sendmsg()` system call to send an approximately 1,600 byte message to its invoking the `recvmsg()` system call to receive a copy. The echo server received the requests at 2,500 requests/second from our echo clients and accumulated the SCTP associations or TCP connections until the number reached 50,000. Table 3.3 compares the setup and transaction times of the two SCTP socket styles and TCP for the echo server.[3]

While the values using SCTP one-to-one sockets and using TCP remained constant, the setup and transaction times for the echo server using a one-to-many socket grew linearly with the number of associations. The peak CPU utilization was approximately 25 percent for all SCTP configurations while it was less than ten percent for TCP.

[3]These results were slightly larger than the results in Figure 3.3 due to different measurement conditions, the message size and network traffic.

To investigate the reason for this linear increase corresponding to the number of associations using a one-to-many socket, we traced the kernel source code and found that, when receiving the INIT and COOKIE-ECHO messages, an unscalable search algorithm, a linear search, was used to look up a matching association by endpoint. That search always failed when the server received these two messages requesting new associations. Because of using a linear search, using a one-to-many socket increased the setup time as a function of the number of associations linked to a single socket. In addition, when the server sent a message, the `sctp_sendmsg()` function in the kernel called a lookup function which performed a linear search. This linear search increased the transaction time, but not as drastically as that in the setup time. Unlike the setup time, this association search always succeeded and took a variable amount of time depending on where the matching association was stored in the list of associations. Thus, both setup and transaction times could be improved by replacing it with a hash table lookup since the linear search clearly caused the cost of using the one-to-many socket.

After replacing the search algorithm,[4] the setup and transaction times remain constant at 0.34 ms and 0.53 ms, respectively, as shown in brackets in Table 3.3. This replacement, therefore, eliminated the differences in the setup and transaction times between these two SCTP socket styles. Although we suspected using a one-to-many socket would reduce server request throughput in Section 2.4.2, the measurement using the improved implementation shows no negative impact of sharing a single socket buffer with a large number of associations. When using a one-to-many socket, all the server needs to do is to expand the buffer size of send and receive sockets to handle concurrent requests and responses to a large number of associations.

3.5.3 The Effect of Piggyback Setup

As Figure 2.3 shows, the piggyback setup mechanism intends to mitigate the overhead of the SCTP four-way handshake by reducing an additional RTT to half. Table 3.3 compares the setup, transaction, and combined times among the SCTP regular setup, SCTP piggyback

[4]This fix on the association lookup function has been included in a patch for the Linux kernel 2.6.24-rc3 released in November 2007.

	SCTP			TCP
Messages s: sent, r:received	Regular setup (ms)	Piggyback setup (ms)	(ms)	Messages s: sent, r:received
s:INIT	0.00	0.00	0.00	s:SYN
r:INIT-ACK	0.14	0.14	0.09	r:SYN, ACK
s:COOKIE-ECHO	0.01	0.01	0.01	s:ACK
r:COOKIE-ACK	0.13	0.23[b]		
s:DATA[a]	0.00	N/A[c]	0.00	s:DATA
r:DATA[a]	0.37	0.26[d]	0.34	r:DATA
Total	0.65	0.64	0.44	

[a] Since DATA was 1,600 bytes long, it was fragmented into two packets. The elapsed time is between sending the first fragment and receiving the second fragment.

[b] The COOKIE-ACK message was received before the client sends the second segment of DATA.

[c] DATA is piggybacked on the COOKIE-ECHO message. In our measurement, a fragment of DATA is piggybacked.

[d] This indicates the elapsed time between receiving the COOKIE-ACK message and DATA.

Table 3.4: Elapsed times between messages for SCTP and TCP

setup, and TCP. This comparison indicates that using piggyback setup slightly reduced the combined time, but still took longer than TCP by 0.19 ms in our measurement environment.

To investigate the reason that the piggyback setup was not very effective in reducing the overhead incurred by the SCTP fou-way handshake, we measured the elapsed time for each RTT by monitoring the time stamps of sent and received packets from the network using the `tcpdump` program on our echo client.[5] Table 3.4 shows an interesting result that the elapsed time between sending COOKIE-ECHO and receiving COOKIE-ACK grows by 0.1 ms beyond that of the SCTP regular setup. Therefore, in spite of reducing the elapsed time between receiving COOKIE-ACK and receiving a message copy by 0.11 ms, the overall effect of using the piggyback setup is slight in our measurement environment. Since the COOKIE-ECHO message piggybacking a user data exceeded the path MTU, the message was segmented into two packets. The increase in the elapsed time of processing the COOKIE-ECHO message might be caused by its partial delivery, but we failed to identify the reason. Table 3.4 also shows that the SCTP setup using a state cookie, which is meant to protect against the INIT flooding attacks, is more expensive than basic TCP setup, which processes no cookies in the default configuration. Thus, the results indicate that the configuration option of a state cookie for TCP would be expensive. All in all, the effect of using piggyback setup was slight in our measurement so that the SCTP four-way handshake still causes longer setup delay than for the TCP handshake. Yet, using piggyback setup is recommended since the effect of reducing RTTs would be larger in a wider area network where SIP services are typically deployed.

3.6 Conclusion

Our echo server measurement has demonstrated that a large-scale SIP server using TCP can potentially be built with a commodity machine, but not for SCTP. Under our target traffic model, we can conclude that the impact of using TCP on the scalability of the echo server is relatively small since it only includes the setup delay for the TCP three-way handshake,

[5]These elapsed times were smaller than the values in Table 3.3 since they were monitored on a network interface. Thus, they excluded processing times for the echo client application.

690 MB of kernel memory for 300,000 concurrent TCP connections, and CPU time for processing SIP keep-alive messages.

The more connections are needed, the more kernel memory space is needed. However, installing more physical memory for a 32-bit kernel does not help since the kernel process can only handle 4 GB of memory including the user space. The only way to increase the kernel space for a 32-bit kernel is to modify the memory split to 3G/1G, where the kernel space is 3 GB at the expense of the user space. This is, however, not recommended since SIP applications require memory usage corresponding to the number of users. Instead of modifying the memory split, it is better to switch to a 64-bit kernel.[6] Once the kernel can support more than 4 GB of memory for a 64-bit kernel, the bottleneck would move to other factors, such as CPU utilization or the maximum number of file descriptors, which is currently 1,048,576.

Persistent TCP connections are useful for SIP UAs to avoid unnecessary setup delay, to enable NAT or firewall traversal, and to keep security associations. However, while a SIP server keep connections open, CPU utilization grows with the request rate of the SIP keep-alive messages sent from SIP UAs. Compared to UDP, the cost of the keep-alive messages would not be higher since the interval of the messages from SIP UAs using TCP is four times longer. Our rough estimate based on the cost of closing TCP connections indicates that the cost of supporting the SIP keep-alive messages may result in negligible and even if the request rate increased to the UDP rate, approximately 14 percent of the CPU utilization. We leave the measurement and analysis of the impact of the SIP keep-alive messages for future work.

This echo server measurement has shown how using SCTP impacts server scalability and performance by evaluating the effect of SCTP features, namely, one-to-many style sockets and piggyback setup. Using a one-to-many socket was expected to increase the number of sustainable associations by reducing the number of sockets, but the measurement

[6] The 64-bit Linux kernel on the AMD64 CPU supports 256 TB ($= 2^{44}$) of virtual memory where the kernel space is half of that, (i.e., 128 TB). Before a 64-bit system has become prevalent, using a custom Linux kernel such as RedHat Enterprise Linux [RedHat, 2007] was the sole solution. It had supported a 4G/4G VM split taking advantage of a 36-bit address space on the Intel Pentium Pro or later systems [Intel, 1996].

results denied. The echo server capacity decreased to one fifth of that using TCP because of a large amount of association-related data. Furthermore, the echo server performance dropped. The setup and transaction times increased with the number of associations. By reducing the capacity for accepting packets out of order, memory footprint per association using a one-to-many socket could compete with memory footprint per TCP connection. By replacing a linear search with a hash table lookup, the setup and transaction times could remain constant.

Using the piggyback setup slightly decreased the combined time of the setup and transaction in our measurement environment so that the effect was not enough to compete with the time for TCP since handling a signed cookie is expensive. Although the effect was slight in our measurement, the effect of reducing a RTT by piggyback setup would be larger in a wide area network.

Therefore, using a one-to-many socket and piggyback setup is recommended. The SCTP kernel implementation in Linux is far less mature than the TCP implementation; thus, there is still significant room for improvement in the efficiency of handling a large number of associations.

Chapter 4

Understanding the Impact of Using TCP and SCTP on SIP Server Scalability

4.1 Introduction

Chapter 3 has identified the impact of using TCP and SCTP on our echo server. The SUT using TCP can potentially fullfil the capacity requirement for a large-scale SIP server, whereas the SCTP Linux implementation has room for improvement before it is suitable to build a large-scale server. This chapter describes the performance measurement of a SIP server using TCP, which intends to identify the additional impact of using TCP on a SIP server by comparing the measurements to the results of the echo server measurement. Our measurement focuses on SIP registrar performance since the message exchanges are similar to the echo server measurement; thus, the way of using the socket APIs is similar. Based on the results of the registrar test, we discuss the applicability for estimating the impact of using TCP instead of UDP on SIP proxy performance. On the other hand, we perform the measurement using SCTP on a SIP front-end server instead of a full SIP server since the echo server measurement demonstrates that the SCTP Linux implementation is not ready for performance tests. Our measurement focuses on identifying the effect of SCTP being

Transport protocol	Server type	Goal of measurement	Metrics
TCP	SIP server	To understand the impact of using TCP on SIP operations, namely, the REGISTER-200 OK transaction	- Sustainable request rate - Memory usage - CPU utilization - Transaction response time
SCTP	SIP front-end	To understand the effect of message orientation	- Transaction response time

Table 4.1: Measurement goals and metrics

message-orientated instead of byte-streaming.

4.2 Measurement Goals and Metrics

Table 4.1 shows the goals and metrics for our SIP server measurement using TCP and a SIP front-end server measurement using SCTP. For SCTP, our measurement uses our SIP front-end server, which only performs message parsing. For example, when receiving the SIP INVITE request, the SIP front-end server parses the message and responds to a SIP UA with a 200 OK response without involving any other SIP operations, such as database lookups.

To determine the additional cost of using TCP on a SIP server, we measure the response time for the SIP REGISTER-200 OK transaction, memory usage, and CPU utilization for a SIP registrar with three TCP connection configurations by comparing these results with the echo server measurements under the same conditions. We also measure sustainable request rate with the success rate of handling requests.

To identify the effect of SCTP being message-orientated, we compare the setup and transaction response times using the SIP front-end server among UDP, TCP, and SCTP. We also compare them with the results of the echo server measurement to determine the cost of message parsing.

4.3 Experimental Setup

4.3.1 Servers Under Test

The system and configurations were the same as the measurement environment for the echo server described in Section 3.3. The SUT ran Linux 2.6.23 on a dedicated host equipped with Pentium IV 3 GHz 32-bit dual-core CPU and 4 GB of memory. The SIP server under test was sipd [Lennox *et al.*, 2002], which had been developed in our laboratory. Sipd used a single process and multiple threads for high concurrency. 300,000 user accounts were registered in a database using MySQL 4.1.22. Our SIP front-end server which implemented only parts of SIP message parsing consisted of a single process and single thread like the echo server. The measurement for TCP and SCTP was performed in 2006 and 2007, respectively.

4.3.2 User Clients

Similar to the SUT, the systems for user clients and their configurations were the same as the measurement environment for the echo clients described in Section 3.3. The clients ran on up to ten hosts with Pentium IV 3 GHz 32-bit CPUs and 1 GB of memory running Redhat Linux 2.6.9. These hosts communicated with the SUT over a 100 Mb/s Ethernet connection at light load. The RTT measured by the `ping` command was roughly 0.1 ms. Sipstone [Narayanan *et al.*, 2002], a SIP benchmark test suite including a SIP UA emulator was used for the SIP server measurement. We added functionality to sipstone to enable TCP connection configurations.

4.4 Measurement Results Using TCP

4.4.1 The Impact of Using TCP on SIP Operations

To examine the impact of SIP message processing and SIP operations including database lookups, we compare with the previous measurement using the echo server. Figure 4.1 compares the response times for the SIP REGISTER-200 OK transaction including the setup times and CPU utilization at 2,500 requests/second between the SIP server and echo server measurements described in Figure 3.3. The SIP REGISTER-200 OK transaction has the same

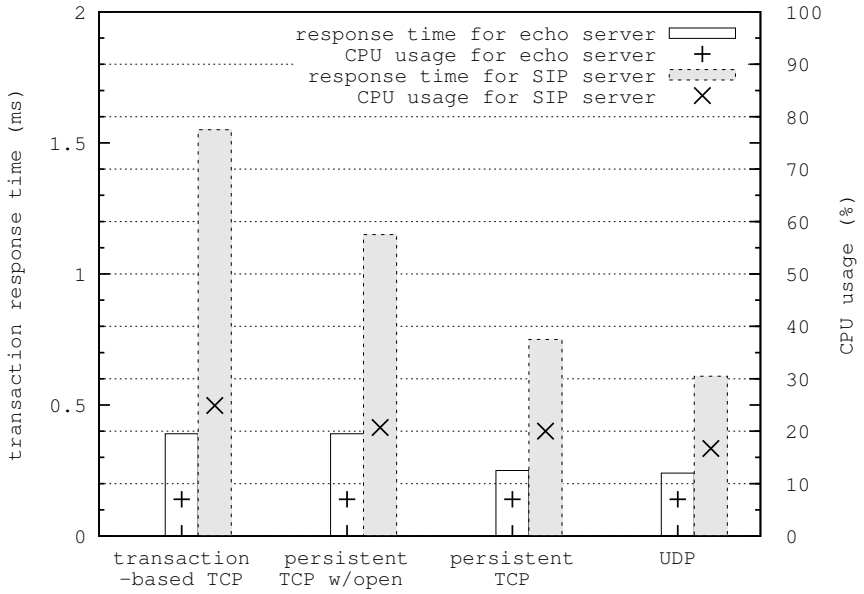


Figure 4.1: Average transaction response times (left axis) and peak CPU utilization (right axis) at 2,500 requests/second for echo server test and REGISTER-200 OK test

number of messages between the server and the clients as the echo server measurement. Both REGISTER request and 200 OK response were approximately 400 byte long while the echo server used messages approximately 500 byte long.

The increases indicate the cost of SIP operations and the difference in the software model of the two servers since the number of messages and transactions are the same. Interestingly, despite the fact that the SIP operations are the same for the four configuration, the increases of the response times differ by 0.4 - 1.2 ms for the four configurations while the increases of the CPU times are comparable at 15 - 18 percent. For example, the cost of establishing a new TCP connection, which is indicated in the difference in the transaction response time between the two persistent TCP configurations, is 0.4 ms on the SIP server, while it is 0.2 ms on the echo server. Thus, we deduce that these increased response times are caused by the difference in the software model of the two servers: the echo server uses a single process and a single thread model, while the SIP server runs a single process and multiple threads in a thread pool model. This difference in the software model is the natural consequence of the difference in server applications between the SIP server and the echo server. SIP

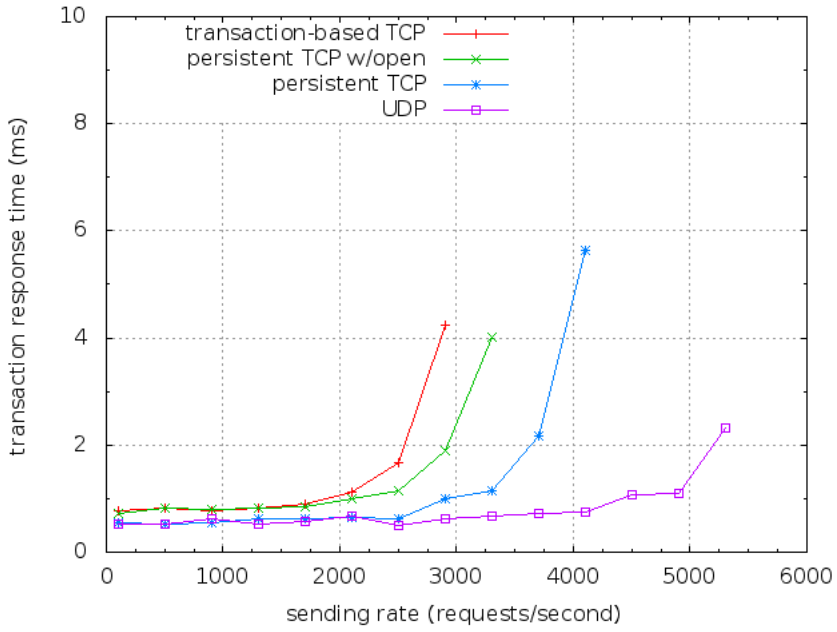


Figure 4.2: Transaction response times as a function of the sending rate for REGISTER-200 OK test for TCP and UDP

servers need to perform SIP operations, which are heavier than creating a message copy on the echo server, and achieve high performance with concurrent threads or processes. The sipd software model uses multi-threads in the thread pool model before improving server performance using the process pool model [Singh and Schulzrinne, 2005]. Sipd assigns available threads from a thread pool to waiting tasks in a queue. All the tasks for handling SIP request or response messages are processed in this manner except a task for reading a socket buffer and parsing a message over TCP. This task is processed by a thread that is generated upon request, not assigned from the thread pool, since the task lasts longer and may exhaust threads in the thread pool.

Similar to the response times, sustainable request rates also differ across the four configurations, namely, the three TCP configurations and the UDP setting. Figure 4.2 compares the transaction response times at various request sending rates at 100 percent success rate. The sustainable request rates are 2,900, 3,300, 4,100, and 5,300 requests/second each for the transaction-based TCP configuration, for the persistent TCP connections with the handshake processing, for the persistent TCP connections, and for UDP. These sustainable

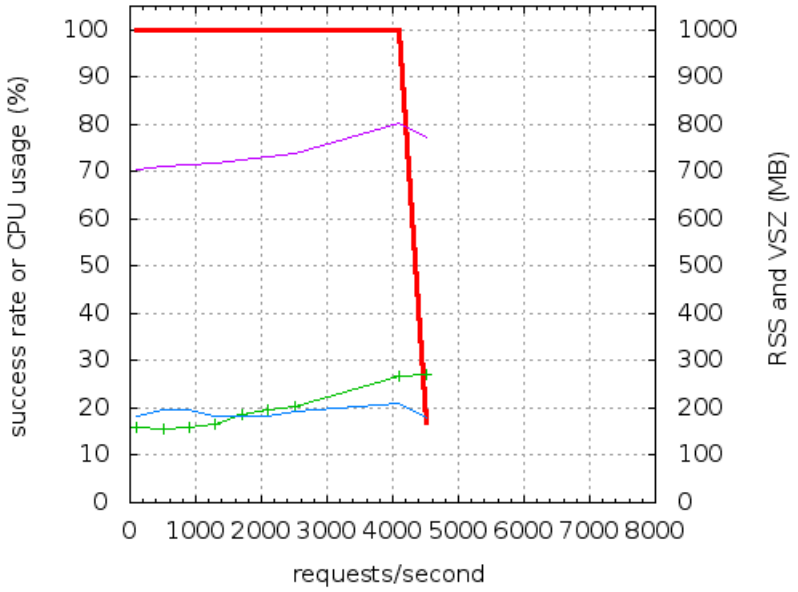


Figure 4.3: Success rate, CPU usage and memory usage for persistent TCP

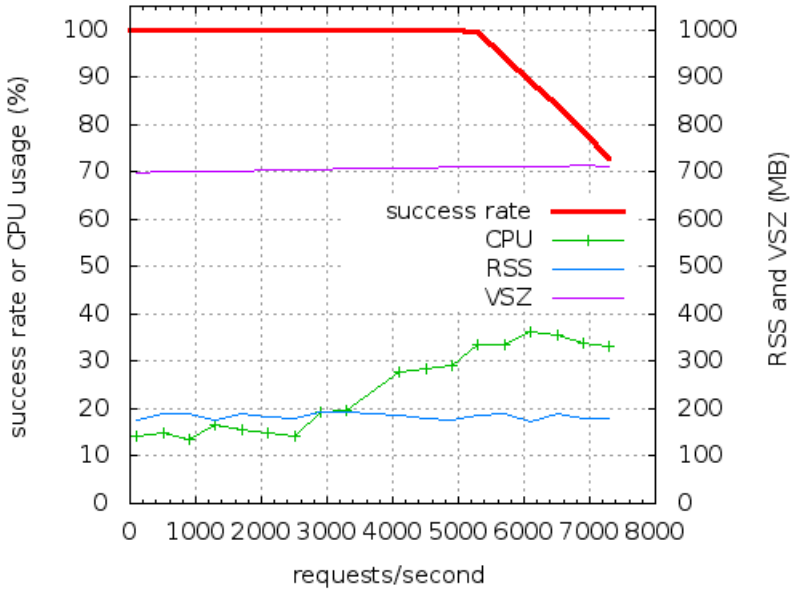


Figure 4.4: Success rate, CPU usage and memory usage for UDP

request rates are at least 27 times higher than the server design requirement described in Table 2.2.

Whereas below 1,600 requests/second, the differences in the transaction response times remain constant, above that, the differences grow substantially. Interestingly, when the SIP server started to fail to handle SIP requests, the system resources had not been exhausted. As shown in Figures 4.3 and 4.4, for the persistent TCP configuration and UDP, CPU utilization was still below 40 percent and the usage of physical memory in resident set size (RSS) and virtual memory in virtual size (VSZ) was below 200 MB and below 800 MB, respectively. The results of the transaction-based TCP and a variant of the persistent TCP including opening TCP connections were similar although no charts are presented. Much earlier than exhausting CPU utilization or memory resources, sipd dropped requests based on the number of tasks waiting in the thread queue. When the number of the waiting tasks exceeds a pre-configured number, sipd invokes its overload control function to drop tasks. The sipd warning messages reported that the overload control function dropped 83 percent of requests for the persistent TCP configuration and 10 - 28 percent of requests for UDP. Thus, we have determined the bottleneck of sustainable request rate is the length limit of the thread queue, rather than memory usage or CPU utilization. More tasks and/or longer tasks required for handling TCP messages decrease the sustainable request rates.

Furthermore, the success rate for persistent TCP dropped precipitously at the load limit, whereas the rate for UDP gradually decreased. The difference between persistent TCP and UDP in dropping the success rate was caused by the difference in the timing of invoking the overload control function between TCP and UDP. Sipd determines if it can handle a UDP message immediately after parsing the first line of the message, enough to sort messages into requests and responses. It favors SIP responses over SIP requests under high loads. On the other hand, sipd first fully parses a TCP message by a thread generated upon request, not assigned from the thread pool, and then detects an overload in the thread queue. This difference is caused by the message orientation of these transport protocols. Since TCP does not preserve message boundaries, a receive buffer may contain only part of a SIP message or multiple SIP messages. Sipd needs to determine the end of a message by parsing the Content-Length header field. This suggests that if sipd can accelerate message

parsing over TCP, especially under high loads, the message parsing can be performed by a thread from the thread pool, resulting in a success rate can drop gradually, similar to UDP. To sort SIP messages, for example, to prioritize SIP responses or BYE requests, which have fewer subsequent messages under high loads, reading the first line of a received TCP stream can help accelerate determining the SIP message type. This message parsing without determining the end of a message does not properly work all the time, but is likely to parse the first line of a message especially at a SIP edge server, where messages are distributed over a large number of connections. Even if sipd fails in sorting messages, such as dropping a fragment of a message or multiple messages at once, it would be permissible under high loads.

It should be noted that this measurement experiment does not identify the cost of the SIP keep-alive messages, but implies that the SIP server needs to process the messages, which contain only four bytes, but arrive at a rate 3,000 requests/second, without overwhelming the thread queue or generating new threads for them. A possible solution is that a dedicated thread peeks at the size of a received message to dispatch tasks by distinguishing the keep-alive messages from regular SIP messages.

4.5 Measurement Results Using SCTP

4.5.1 Data Transfer Latency and the Effect of Message Orientation

Our SIP front-end server measurement focuses on the effect of SCTP offering messages rather than byte-stream. We anticipate that a small effect, as described in Section 2.4.3. Being message-oriented enables an application to easily retrieve a single message from a receive buffer. Upon invoking the `recvmsg()` system call, an application can determine whether or not a message has been delivered in full by checking the message `MSG_EOR` flag. If the message has been delivered in full, the SIP front-end server can read a single message without finding the Content-Length header field, unlike for TCP.

First, to identify the elapsed time for parsing a 1,600 byte message by comparing to the results of the echo server measurement shown in Table 3.3, we measured the setup and transaction times for the three SCTP configurations in addition to TCP transaction-based

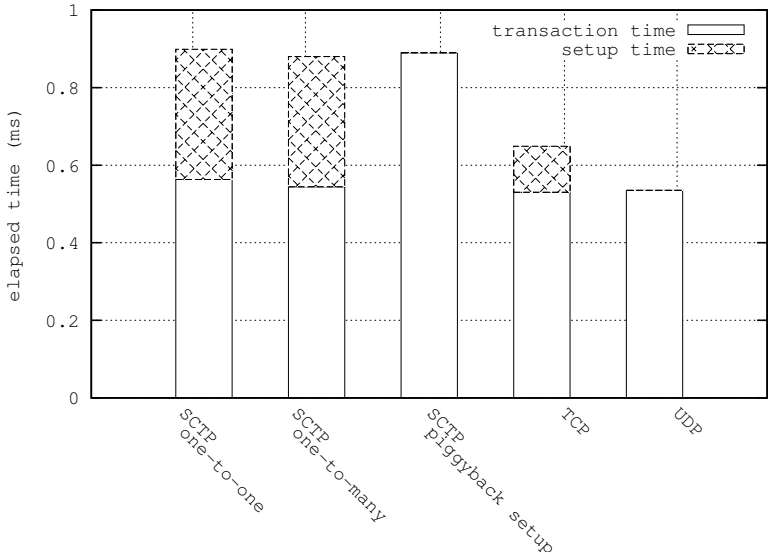


Figure 4.5: Setup and transaction times for SIP front-end server

setting and UDP setting, when the SIP front-end server received the SIP INVITE messages at 2,500 requests/second from the echo clients. Each message was approximately 1,600 byte long, resulting in two packets fragmented by SCTP or TCP segmentation, or IP fragmentation. Figure 4.5 depicts the results of this measurement using the SIP front-end server. Comparing the transaction times between two measurements indicate that the elapsed time for message parsing is only 0.01 - 0.06 ms per message. We then compare transaction times of TCP and the SCTP one-to-many socket configurations shown in Figure 4.5 to see the benefit to efficient message parsing at 2,500 request/rate. No difference between the transaction times of TCP and the SCTP is found. Thus, unfortunately, this measurement was unable to see the benefit of SCTP being message-oriented since the cost of the message parsing is negligible and does not affect server performance at the 2,500 request/second.

However, the benefit of SCTP being message-oriented will be seen under high loads or in a proxy-to-proxy scenario where a TCP connection aggregates messages from multiple users. As Section 4.4.1 describes, SCTP being message-oriented, similar to UDP, allows the SIP server to easily sort messages without determining the message size specified in the SIP header. Thus, being message-oriented will contribute to gradual dropping requests at high loads. Similarly, in a proxy-to-proxy scenario, SCTP being message-orientated allows the

SIP sever to dispatch tasks without message parsing whereas TCP streams requires message parsing to dispatch tasks. Thus, being message-oriented may contribute to promoting concurrent processing. We leave the measurement to identify the sustainable request rate for SCTP for future work.

4.6 Applicability of SIP Registrar Measurements for Estimating the Impact of Using TCP or SCTP on a SIP Proxy Server

We have thus far presented the measurement of the SIP server, specifically, a SIP registrar. It processes the REGISTER-200 OK transaction requiring database access. Unlike the REGISTER request, a SIP proxy server needs not only to receive the INVITE request as a SIP UAS but also to forward it to the adjacent proxy or the destination user as a SIP UAC. Thus, a SIP server plays a double role as UAS and UAC. Additionally, the proxy server needs to handle more subsequent SIP messages and more transactions if the server mode is transaction-stateful. If the server mode is dialog-stateful, handling dialogs is added. However, the choice of the transport protocol is independent of transaction or dialog state on the server.

Given that these additional processes on the SIP server, we discuss how to estimate the impact on the SIP proxy server using our results. Since the number of adjacent proxy servers is relatively small, the connections to other proxy servers does not affect the number of concurrent connections. However, the role as a UAC on the SIP server makes the server unscalable if the SIP server needs to actively establish connections to end users. The maximum number of concurrent connections on the SIP server as a UAC to end users drops to approximately 60,000 because of the range of ephemeral ports, which are assigned only for TCP clients. Thus, it is recommended that the SIP server keeps connections to end users open in order to avoid the limitation of ephemeral ports.

Besides shrinking the upper limit of concurrent connections, the double role of UAs incurs approximately twice the cost of opening and closing TCP connections that have been identified in the echo server measurement in Section 3.4.2. If the SIP server detects the

connection timeout at the application level, the server needs to actively close the connection. For the cost of maintaining connections, we roughly estimated the cost of processing the SIP keep-alive messages based on the cost of the TCP FIN messages because both messages have similar payload sizes and receiving each message is notified the SIP server application. Because of the shorter message and simpler operation, at most 14 percent of CPU time is needed at 14,800 requests/second, which is much lower than approximately 80 percent of CPU time for the SIP keep-alive mechanism using REGISTER requests [Shemyak and Vehmanen, 2007]. However, we do not have any similar measurements to estimate the cost of managing the connection timeout.

The number of messages per call depends on the messages during the call. If four mid-call requests, PRACK, ACK, UPDATE and BYE, are exchanged, the request rate rises to 835 requests/second (= 167×5). Thus, the SIP proxy server requires five times more than the request rate or transaction rate of registration. In a worst-case scenario where each transaction requires a new connection and the double role of UAs doubles the cost of handling the connection, the server needs to handle the SIP REGISTER requests at 1,670 requests/second. As Section 4.4.1 shows, the sustainable request rate for transaction-based TCP configuration is 2,900 requests/second. Therefore, the SIP server will have capacity to handle the required request rate although the sustainable request throughput ratio of using UDP to using TCP is 1.8.

To see whether this estimate is close to the actual performance, we conducted the SIP INVITE-200 OK test where UAs establish and close a dialog exchanging the INVITE, 100 Trying, 180 Ringing, 200 OK, ACK, BYE, 100 Trying, 200 OK requests and responses through the proxy server. This test contains eight messages and three transactions, which is fewer than five transactions used for the estimation. Our measurement demonstrates that the SIP server can handle 700 calls/requests for the transaction-based TCP configuration while it can process 900 calls/requests for UDP. The sustainable request throughput ratio decreases to 1.3 since the additional cost for SIP operations mitigates the impact of the transport protocols.

4.7 Conclusion

Our measurement and analysis have shown that using TCP on the SIP server does not increase the impact of using TCP identified by the echo server measurement under our target traffic model described in Table 2.2. However, at high loads, the impact of using TCP on the SIP server is caused by the difference in message orientation, which is negligible at low loads. The impact of message orientation on server performance depends on the server implementation.

In our SUT, sipd, a longer task of retrieving a SIP message over TCP affects the number of threads and their lifetime per message, resulting in a lower sustainable request rate and a steeper drop of the success rate for the persistent TCP configuration than for UDP. We recommend using persistent TCP connections that can achieved the minimum impact of using TCP. The request throughput ratio of using UDP to using TCP is 1.3 (= 5,300 : 4,100). In real deployment of a SIP server, user authentication and other features are added, resulting in further lowering the impact of using TCP on the sustainable request rate. Consequently, the major impact of using TCP for a large number of connections turns out to be the memory footprint per connection.

To minimize the impact of retrieving a SIP message over TCP, we suggest that a SIP server accelerates message parsing for a limited purpose, namely, message sorting to prioritize certain SIP messages at high loads. Although our measurement does not cover the handling of the SIP keep-alive messages, the SIP serve should handle these messages, which have a distinct message size for each, as a special case to minimize the impact of using TCP. For future work, we leave the measurement to identify the impact of maintaining a large number of connections for SIP, namely, handling the SIP keep-alive messages, short messages at a high rate and managing connection timeout.

For SCTP, our measurement focused on identifying the effect of being message-oriented using the SIP front-end server, but failed. As discussed above, the advantage of being message-oriented is not observable until the request rate increases and 0.01 ms difference in processing time matters to server performance. We also suggest the SCTP measurement to compare the sustainable request rates to identify the impact of using SCTP as an area for future study.

Chapter 5

Guidelines on Transport Protocols

This chapter provides guidelines that our measurements yield for those who implement transport protocols and for those who design the protocols. In our measurements, we found a few issues in the Linux SCTP implementation, but they potentially arise in any implementations or transport protocols. Thus, we generalize our insights and provide the following guidelines.

5.1 Implementation Guidelines

- The implementation of a transport protocol should support possible scenarios more than that are considered by its original design.

  Even if SCTP was originally designed to be used in a server-to-server scenario where a relatively small number of servers connect each other, the implementation should consider a user-to-server scenario where a server connects with a large number of users as well as a server-to-server scenario.

- The implementation of a transport protocol should carefully choose data structures from static and dynamic, examining whether the variables are determined under dynamic conditions, such as network congestion, or relatively static conditions, such as server provisioning.

5.2 Transport Protocol Design Guidelines

- Having both functionalities of UDP and TCP, supporting message segmentation and preserving message boundaries attract large-scale SIP servers.

  There are a few forms of both functionalities of UDP and TCP, such as reliable datagram transport protocol [Partridge and Hinden, 1990]. SIP, which often exchanges messages exceeding the path MTU, is desirable to segment a message by transport layer, avoiding IP layer fragmentation. At the same time, if the transport protocol preserves message boundaries and each message is smaller than a received buffer, a large-scale SIP server can achieve high throughput.

- If a transport protocol supports a smaller set of functionality in the default configuration, it attracts more applications.

  SCTP supports many types of functionality in addition to supporting message segmentation and preserving message boundaries. This adds complexity in the default configuration, such as the keep-alive mechanism. If it allows applications to more easily use a minimal set of functionality, such as supporting message segmentation and preserving message boundaries, SCTP could be easily deployed.

Part II

Controlling Unwanted Communication Requests

Chapter 6

Definitions, Background, and Related Work

6.1 Introduction

Unwanted calls (e.g., calls from telemarketers, charities or pollsters) have not caused as serious a problem as unwanted email, the so-called spam. However, there is clear evidence that the problem becomes more significant; regulations [FCC Robocalls, 2012] have been tightened against autodialed or pre-recorded messages, so-called robocalls. Receiving unwanted calls not only disturbs callees with the ringing, it might also incur a cost to them if they are charged for receiving calls (e.g., minute-based mobile phone services in the U.S.). Thus, people are less tolerant of receiving unwanted calls than unwanted email messages. To control receiving unwanted calls, this thesis explores new ways to identify good calls rather than detecting unwanted ones. We define calls that are important or at least sufficiently important to warrant answering the phone as *good*. The approach to identifying good calls can make it easier for recipients to prioritize incoming calls or messages.

We first present definitions that are used throughout this Part. We then provide an overview of the ways that people trust each other in real life to motivate a discussion of the ways of identifying good calls or messages. We proceed to categorize incoming calls and messages, and define good calls or messages. We give an overview of the solution space for preventing unwanted email message and calls, and review previous work and current prac-

tice, focusing on alternative approaches to detecting unwanted or identifying good messages or calls. We finally analyze caller-ID-based filtering, which is the most common solutions to the problem, pointing out the limited availability of caller-ID authentication and the limitation on use of caller ID as a user attribute.

6.2 Definitions

We define the terminology that is applicable throughout Part II (Chapter 6 - 8).

Communication Endpoint ID (CEID) is an identifier of an endpoint which participates in a communication session through any communication means, such as phone call including VoIP, email, or instant messaging. It is often called a contact address. It is a phone number for phone services and an email address for email. When the signaling protocol for VoIP or instant messaging is SIP, the CEID is called SIP address-of-record (AoR). A CEID is typically issued and assigned by a communication service provider to its user. However it is sometimes assigned to a device, such as a phone number assigned to a landline or a mobile phone device. When the endpoint acts as the originator, the CEID is specifically called an origin ID. For the recipient, it is called a destination ID.

Origin ID is an instance of a CEID and identifier of a person who initiates a communication, such as making a call or sending an email message. The origin ID of a call is called a **caller ID**, and that of an email message is called a **sender ID**, which is typically set in the From header field.

Black-listing is a method of blocking a communication request, such as calls or messages. This method is based typically on a caller ID or sender ID or the network address of a sender in a list, which is called a **black list**.

White-listing is, in contrast to black-listing, a method of accepting a communication request based on a origin ID or the network address of a sender in a list. This list is called **white list**.

Weak social tie is a relationship between two parties who are not regularly communicating, but have previously had a communication or an interaction [Granovetter, 1973].

The interaction may occur through a third party trusted by both parties. For example, acquaintances are connected by weak social ties, while friends are connected by strong social ties.

Good call or message is a call or message that is legal and desirable by recipients. A good call or message is important, or at least sufficiently important to answer or receive.

Cross-media relation is a relation between two parties that has been established through different means of communication when these parties are about to initiate a new communication. This cross-media relation is represented as evidence of the prior communication.

Web-then-call is a cross-media relation between two parties of a call. These parties have had prior contact through a Web transaction. Many variants, including **Email-then-call**, have similar definitions.

Weak secret is a piece of information to prove a prior communication. The information is sufficiently confidential to identify a good call or message.

Principal is an entity whose identity can be authenticated [ITU-T, 1995]. In an attribute validation service, a principal is authenticated by an attribute validation server to issue an attribute credential. In the context of sending an attribute credential in a communication request, the principal acts as the originator, namely, the caller of a call or the sender of a message.

Relying party is an entity who relies on the data in a certificate or a credential in making decisions [ITU-T, 1998]. In an attribute validation service, a relying party uses someone's attributes to assess the risk of accepting a communication request. In the context of receiving a communication request, the relying party acts as the recipient or callee.

Issuer is an entity which issues an attribute credential or assertion for the principal. An issuer also validates the attribute credential or assertion upon a relying party's request.

6.3 Ways People Trust Each Other

To explore possible ways to identify good communication requests, we review how people trust each other in real life. People establish interpersonal relationships based on how they have interacted with each other, physically or electronically. Especially, if a party has "the willingness to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control the other party" [Mayer *et al.*, 1995], a trust relationship is established. Thus, trust relationships between individuals are established based on both interactions between them and the expected risk of the peer's actions.

In most cases, two parties in a trust relationship have previously interacted directly with each other. However, in some cases, they have had no prior direct interaction with each other. Instead, the party may have had indirect interactions that involved an organization or a third-party individual in order to determine the trustworthiness of the other party. Since the risk of accepting communication requests is usually low, a relatively lower level of trust is often sufficient to identify desirable communication requests.

The following sections focus respectively on the case where a trust relationship is the result of a direct interaction, and on the case where the trust relationship is not the result of a direct interaction. In each, we discuss how these relationships can be represented electronically and how they can be used to determine the trustworthiness of communication requests.

6.3.1 Trust Relationships Based on Direct Interactions

In the simplest case, trust relationships are based on direct social interactions, which can be divided into two groups based on how often people communicate with each other.

The first group of individuals, which is typically of limited size, consists of those who regularly communicate with each other. In most cases, these people know each other personally and are connected by strong social ties, which are determined by "a combination of the amount of time, the emotional intensity, the intimacy, and the reciprocal services which characterize the tie" [Granovetter, 1973]. Examples of people connected by strong

social ties include family members, friends, and close colleagues. The second group, which is much larger, consists of people who occasionally interact with each other. This group includes casual acquaintances, colleagues, and business contacts. The social ties for people in this group are weaker and the trustworthiness may be lower than for those in the first group.

People in either group are electronically represented as contact addresses stored in the address book of the person under consideration. While the likelihood of a member of the second group appearing in the address book is lower, they are often listed in social media services as friends within one degree of separation. To identify communication requests from people in these two groups, filtering based on origin ID (i.e., caller ID or sender ID) has been widely used. This filtering requires origin ID authentication. In addition, origin-ID-based filtering is used for detecting and blocking unwanted calls or messages, so-called black-listing.

This thesis analyzes the limitations of caller-ID-based filtering using address books (Section 6.7). This thesis also proposes easier ways of collecting contact addresses from the second group using cross-media relations (Section 7.5).

6.3.2 Trust Relationships without Direct Interactions

It is also possible for trust relationships to be established without the existence of prior direct personal communication (i.e., one-on-one communication). In this case, trust relationships are established using transitive trust through a trusted individual or organization, and/or on their own experience. We divide these trust relationships into the following, often overlapping, categories: using extended social graphs, experience-based, affiliation-based, credential-based, based on evaluation by a trusted third party, behavior-based or appearance-based, and location-based. We also describe how people limit the potential risk of the other's actions. Although our analysis is based on separate categories, it is worth noting that, in practice, people trust each other in one or more ways.

Using extended social graphs

A person often trust persons whom his trusted person trusts, for example, friends of a friend or colleagues of a colleague. These people are listed in social media services as friends with two degree of separation in the extended social graphs. Additionally, they might be found in the destination IDs of email or SMS messages that are received from their friends or from the mailing list they subscribe to.

Section 7.5.1 proposes a way of encouraging users to publish their addresses in social media without privacy concerns. Friends of a friend based on the destination IDs of email messages are examined in our user study of incoming email messages described in Section 7.10.

Based on experience

People determine the trustworthiness of others based on their own experience. For example, customers who have had good shopping experiences with shops (e.g., online shopping sites or local grocery stores) trust the owners or sales persons, even though the customers do not personally know them. This type of trust is the basis of reputation services, such as customer rating system.

This thesis discusses how individuals can use their experience on online activities, such as online shopping via a Web transaction, as cross-media relations described in Chapter 7, in order to determine whether a communication request is sufficiently important to answer. This thesis, however, does not discuss how to use a reputation service since it is difficult to determine the trustworthiness of the provider of the reputation service and the reviewers.

Based on affiliation

People often trust employees or students who belong to a trusted enterprise, school, or organization based on their affiliations. The trust relationship beween a person and an organization is based on the organization's reputation or based on the person's experience. In the latter case, the trust relationship can be regarded as a form of experience-based trust relationship.

A person's affiliation is considered his attribute. Affiliations can be represented as attribute credentials, certificates, or assertions. Although a person's affiliation is sometimes represented as an origin ID, such as the domain part of an email or SIP addresses, it is only useful when the user account management policy of a communication service provider is trusted.

Section 6.8 discusses the limitations on the use of origin ID as a user attribute, emphasizing the difficulty in authenticating caller IDs. Chapter 8 proposes a simple way of validating a caller's attributes without having to authenticate the caller's identity or the caller ID, in order to identify good communication requests.

Based on credentials

Affiliation-based trust relationships can be generalized into credential-based trust relationships. People often trust each other based on credentials that are issued by an organization they trust.

Holding a credential, such as certificates or licenses, or the data on them, such as person's age attested by a driver's license is an attribute of the holder. Thus, similar to affiliation-based trust relationships, attribute credentials represent the concept of trust based on credentials.

Chapter 8 proposes a mechanism of validating a caller's attributes without having to authenticate the caller's identity or CEID.

Based on evaluation by a trusted third party

Similar to using extended social graphs, people often trust each other based on evaluation, such as rating or reputation, by a trusted third-party organization. For example, employers do background checks to confirm that prospect employees are not convicted criminals. Another example is the case where landlords examine tenant applications based on their credit score. Other examples include the cases where people rate enterprises, such as restaurant ratings or the Better Business Bureau (BBB) ratings. These real-life examples indicate that the cost of determining trustworthiness increases with the risk caused by potential actions by the other party. Thus, this evaluation, together with affiliation-based or credential-based

examinations, is often used for determining trust worthiness.

To assess the sender domain of email messages, accreditation services have been provided with the DomainKeys Identified Mail (DKIM) signature [Hansen *et al.*, 2009]. As an extension of assessing the sender domain, vouching services for the content in the email message are being proposed [Hoffman *et al.*, 2009]. Since ratings and the related context can be seen as a variation of a user's attributes, they are represented as attribute certificates or assertions, similar to affiliations and credentials.

Based on behavior or appearance

People sometimes trust each other based on behavior or appearance especially when they do not have any other information about the other party. For example, people may trust neighbors who are well-dressed and greet them with a smile. They may trust those who are in the uniforms of police, safety guards, and even parking attendants. However, this type of trust is fraught with danger of mischaracterization in both directions. People subjectively judge other's behavior or appearance by their values including prejudice, which are the outcome of their experiences. On the other hand, fraudsters and confidence tricksters who are well-dressed attempt to trick people. It is difficult to detect fraud only based on appearance when they pretend to be an official person to gain trust from people. Thus, the behavior or appearance of others is less helpful to trust others; rather, it is often useful for suspecting and distrusting others.

Statistical analysis of content or sender behavior for email messages or calls falls into this category. Similar to real-life examples, statistical analysis is useful for detecting unwanted communication requests rather than good requests.

As Section 6.4 explains, this thesis explores approaches to identifying good communication requests, rather than detecting unwanted requests.

Based on location

People occasionally trust others based on a physical location. For example, people may trust others who live in the same neighborhood, or who work within the same building, by assessing what their social status or professions might be as a consequence. Additionally,

people may trust those who are waiting at a bus stop based on their immediate location since this location implies that they might be knowledgeable about a given bus's schedule. Thus, location can be seen as a translation of the affiliation or the behavior of the other party. Unless physical location is certified by a trusted entity, location-based trust is prone to mischaracterization, similar to the previously explained behavior-based or appearance-based trust relationships. This is because the immediate location is often a transient attribute of a person that only provides indirect evidence for the person's attributes, such as affiliations.

Limiting the risk by physical location or legal division

As explained above, physical location is, in general, insufficient information to take a high risk of another party's actions. Nonetheless, it is useful for limiting the amount of risk taken, especially in the context of online transactions or communications. People may want to ensure that the involved parties or organizations are physically reachable and in their jurisdiction in case they need to be prosecuted.

Limiting the risk by involving a trusted third party

Typically when the predicted risk of other party's action is high, people trust the other party by limiting the risk by involving a trusted third party. For example, people ask to open locks for their cars or homes, if needed, to a bonded locksmith, which is provided binding insurance. Rather than being used by itself, this limitation is usually used together with other trust relationships.

Although the outcome of a communication may expose recipients to financial or other types of loss, the risk of accepting a communication request itself is relatively low. Thus, limiting risks is usually not required for controlling unwanted communication requests.

6.4 Good Calls or Messages

Tables 6.1 and 6.2 categorize incoming calls and messages, respectively, into two groups depending on recipient's preferences: unwanted and desirable. Unwanted calls or messages can be further divided into two groups: harmful and annoying. Harmful calls or messages

Callee's preference	Call spectrum	Examples	Scope of existing mechanisms	Scope of this thesis
Unwanted	Harmful	Threats		
		Fraud	✓	
	Annoying	Telemarketing Originating from callers within jurisdiction Originating from callers beyond jurisdiction	✓	
		Calls from charities Calls from pollsters Prank calls Wrong numbers		
		Calls triggered by past transactions, but not interested any more		
Desirable	Less important	Less important calls, but sufficiently important to answer		✓
	Important	Important calls carrying an unknown caller ID Important calls triggered by past transactions		✓
		Calls from business partners, customers Calls from family, friends, coworkers	✓	✓

Table 6.1: Categories of incoming calls

Recipient's preference	Message spectrum	Examples	Scope of existing mechanisms	Scope of this thesis
Unwanted	Harmful	Threats Viruses		
		Fraud Commercial messages: Abusive content	✓	
	Annoying	Commercial bulk messages Originating from senders within jurisdiction Originating from senders beyond jurisdiction	✓	
		Newsletter triggered by past transactions, but not interested any more		
Desirable	Less important	Less important messages, but sufficiently important to open		✓
	Important	Important messages triggered by past online transactions Important messages carrying an unknown sender ID		✓
		Messages from business partners, customers Messages from family, friends, coworkers	✓	✓

Table 6.2: Categories of incoming email messages

include threats and fraud. Specifically for message, viruses are also included. Annoying calls have a wide range of calls, from telemarketing calls violating regulations, to various types of legal calls. These legal calls include telemarketing with the callee's consent, calls from charities and pollsters, but not limited to them. They also include calls that are triggered by past transactions or relationships, but are unwanted by the callee who finds them to be no longer of interest or value of them. Prank or wrong number calls are also unwanted by callees. Therefore, even if existing spam prevention mechanisms – typically designed to detect fraud and telemarketing calls – can perfectly detect these types of calls, some types of annoying calls reach the callee and require immediate attentions by the ringing. In this case, characterizing fraudsters or telemarketer's behavior does not help callee identify legal calls that the callee is not interested any more. Thus, we focus on identifying desirable calls, rather than detecting unwanted ones.

We call legal and desirable calls or messages *good*. Good calls are important to callees or at least sufficiently important to warrant answering the phone. Filtering calls using an address book has been commonly used, but it cannot identify good calls carrying an unknown caller ID. Thus, this thesis addresses the issues of unwanted calls by identifying good rather than detecting unwanted calls.

It is worth noting that the definition of spam is unclear and legislations against spam vary from country to country [FTC, 2003; CRTC, 2010]. This thesis does not define the term spam, but uses this term when describing existing mechanisms for preventing unwanted email messages or calls.

6.5 Preventing Unwanted Communication Requests: Calls and Email

This section reviews existing techniques for preventing unwanted calls and email messages. We first provide an overview of the solution space. The solution space is rather large, as described in Figure 6.1. There is no panacea for preventing unwanted communication requests; thus, an ensemble of solutions is needed.

To understand the cost balance between the senders and recipients, we divide the so-

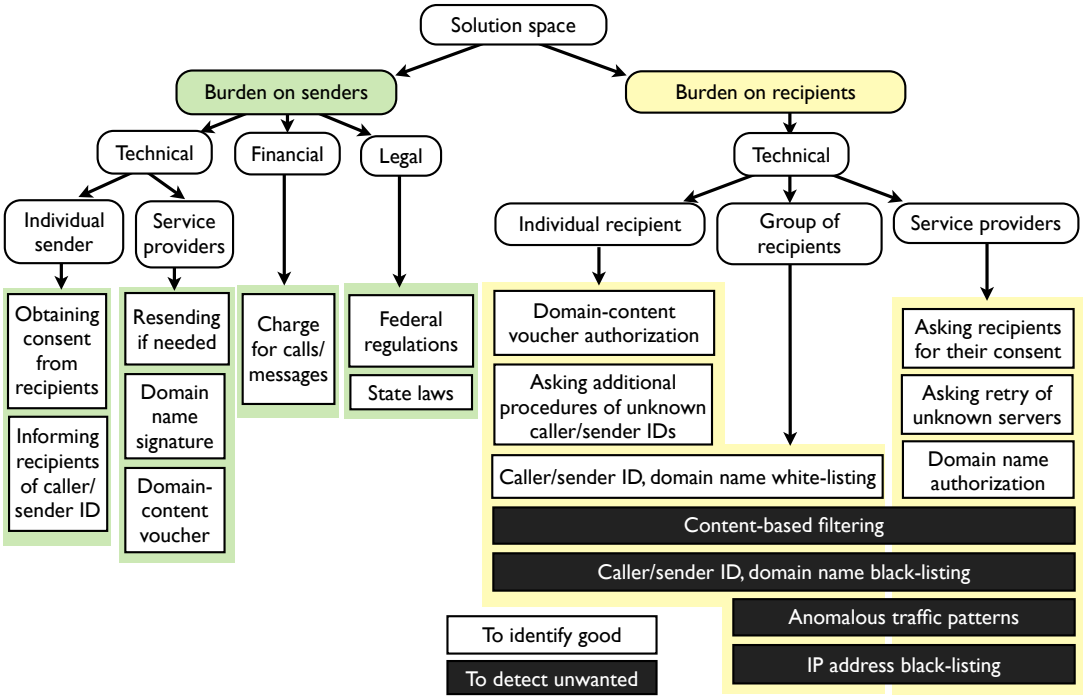


Figure 6.1: Solution space of preventing unwanted calls and email

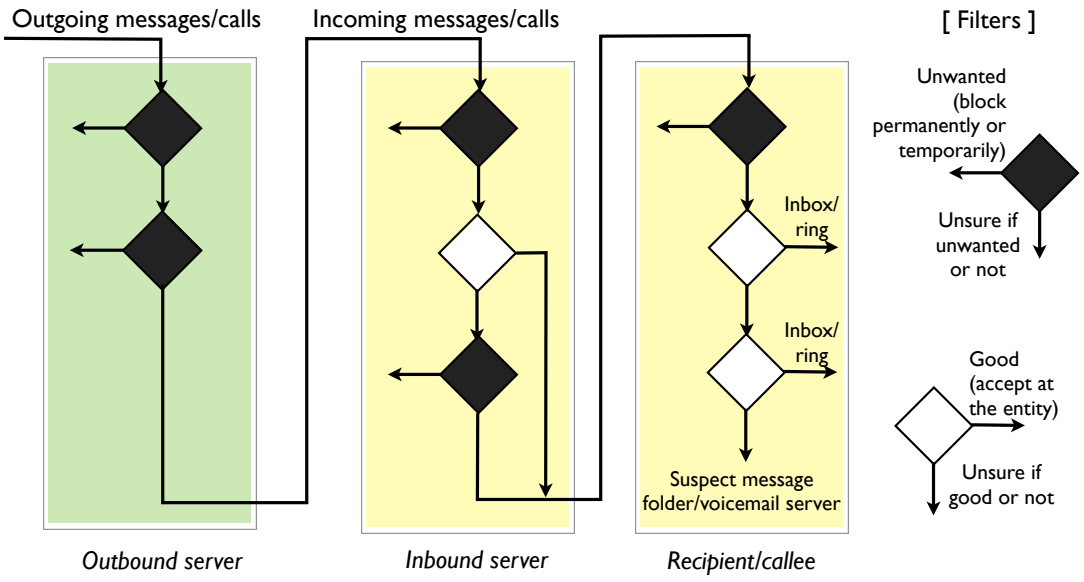


Figure 6.2: Conceptual model of filtering messages or calls on outbound server, inbound server, and recipients

lution space into two categories, based on which entity is burdened. The burden placed on the sender or caller side consists of technical, financial, and/or legal ones. The more techniques for identifying good messages and calls (represented in white boxes) are added on the recipient side, the higher cost for spammers to pretend the senders of good communication requests is added to the sender side. However, adding more techniques, especially based on sender ID or caller ID, for detecting unwanted messages or calls (represented in black boxes) does not increase the cost for spammers to circumvent blocking techniques – spammers can pick a new sender ID or caller ID.

Both on the sender side and the recipient side, the burden on is placed on individual recipients and/or service providers. Specifically for recipients, they sometimes collaborate with each other in preventing unwanted messages. For both sides, the technical burden includes not only the computational cost by machines, and also the cost of procedures by humans.

Burden involves filtering techniques, which are divided into two types of filtering. One filtering type is to determine whether or not to accept a request, for instance, white-listing. Another is to determine whether or not to reject a request, for instance, black-listing. None of filtering mechanisms can categorize all requests perfectly. Thus, a set of filtering mechanisms is typically needed on the outbound servers, the inbound servers and the recipients or callee, as shown in Figure 6.2. The outbound servers, except dedicated senders of unwanted email or calls, need to block them to keep their reputation, not being considered the source of spam.

The goal of this thesis is to propose additional tools while taking into account the cost balance between the sender and recipient sides. Ideally, these solutions should place the cost of sending unwanted messages or calls on the sender side only. However, it is almost impossible to find such solutions since whether or not a message is unwanted often depends on the recipient's preference or decision. Thus, placing some burden, namely, a mechanism for preventing unwanted messages or calls, on the receiver side is inevitable. Furthermore, it is desirable that the mechanism assigned on the recipient should ultimately benefit the recipient, in addition to preventing unwanted messages or calls. This motivated us to study solutions for identifying good messages or calls, making it easier to prioritize them, rather

than detecting and blocking unwanted ones.

The following sections review existing mechanism for detecting unwanted email or calls, and for identifying good ones.

6.5.1 Detecting Unwanted Email or Calls

This section reviews existing techniques for preventing unwanted email messages or calls that have designed for detecting and blocking them, discussing whether the techniques for email are applicable to calls or not.

According to a report from a consortium of service providers [Group, 2011], approximately 90 percent of incoming messages at MTAs were blocked as spam or virus using blacklisting and content-based filtering in 2011. The report indicates that detecting unwanted email messages, especially by servers, works quite effectively. Most existing techniques aim for detecting email or calls that are unwanted by many recipients, not by each recipient. This section uses the term spam to describe unwanted email or calls without considering a recipient's preference.

These techniques use a wide range of inputs, from the network layer address to the message content. We break down these techniques into the following layers: network, transport, and application. Especially for the application layer, we further break them down into message header, content, sender behavior analysis, and adding procedures to be completed by unknown senders.

Each or a set of mechanisms using the inputs is used as a filtering condition installed at the outbound mail servers, the inbound mail servers (both called mail transfer agents (MTAs)) and recipient mail clients, as shown in Figure 6.2. For calls in SIP, filtering systems work at the outbound SIP servers, the inbound SIP servers, and callees.

Based on the network layer address

Blocking spam email messages based on the source IP address, called *IP address blacklisting*, is a simple technique that has been commonly used, typically by inbound MTAs. MTAs look up on the lists of spam source IP addresses and domains, mostly using DNS-based lists [Jung and Sit, 2004; Levine, 2010]. Open relay MTAs, which allows to send

messages without sender authentication, had been included in these lists until open relays became unacceptable and closed [DSBL, 2009].

The benefit of IP address black-listing is that it is lightweight since it can avoid the cost of handling spam messages at the upper layers, namely, TCP and SMTP. The limitation of IP address black-listing is that it is effective in blocking spam only from dedicated spam senders, not from compromised machines, so-called bots, where a large fraction of spam messages originate [Ramachandran and Feamster, 2006]. Furthermore, as is common with any listing mechanisms, the effectiveness of this approach depends on how these lists can be updated, keeping up with the spammers agility. This was observed also by Ramachandran and Feamster, noting that a fraction of spammers had used short-lived routing information to evade being traced.

IP address black-listing is applicable to all communication means over IP networks. However, VoIP servers have been using the opposite approach; accepting traffic only from a limited number of peers.

Based on the transport layer behavior

As a characterization study for spammers' transport layer behavior, Beverly and Sollins [Beverly and Sollins, 2008] observed that spammers send a large number of messages to low-bandwidth networks, leading to network congestion. However, given that the cost of network bandwidth has been significantly reduced, we are uncertain whether this characteristic still holds true.

Originators of call spam do not share this characteristic of email spammers, leading to network congestion because of the following differences in protocols and services. Unlike SMTP, SIP signaling messages and voice packets can be transmitted over UDP. In this case, no indicators of network congestion can be observed at the transport layer. In addition, to deliver their message with acceptable voice quality, originators of call spam have to avoid congestion for themselves. Furthermore, due to the time-consuming nature of phone calls, the call volume cannot be as large as email messages. Even robocalls, which autodial and play pre-recorded or synthesized messages, usually take a few minutes for each call to wait to be answered and play a message, whereas the senders of email spam can instantly send

a large number of message copies to multiple destinations.

Based on the application layer inputs from message header fields

Both the Simple Mail Transfer Protocol (SMTP) envelope [Klensin, 2008] and the email message header fields [Resnick, 2008] include the sender ID of an email message. Based on the observation that many spammers forge a sender ID of email messages, several mechanisms for verifying a sender's domain and a sender ID have been proposed and are currently used [Lyon and Wong, 2006; Wong and Schlitt, 2006; Crocker *et al.*, 2011]. However, these verification mechanisms intend to identify good senders, rather than to detect and block senders of spam messages, since spammers can pick a new domain to evade being blocked.

Blocking email messages based on a sender ID has been widely used especially by recipients. However, spammers can pick a new email address even more easily than a domain to evade being blocked by black-listing. Thus, detecting spam based on a sender's domain or a sender ID has become less effective.

Similar to email, detecting and blocking call spam based on a caller ID have been used as a common solution since it is simple. However, the effectiveness of this solution relies on the assumption that a caller ID can be authenticated by the callee. Without caller ID authentication, originators of call spam can evade being blocked using a forged caller ID. However, caller ID authentication suffers from limitations: for example, a phone number in the tel URI cannot be authenticated by the callee. The limitations of caller-ID-based filtering are further discussed in Section 6.7.

Based on the application layer inputs: Content

Content-based filtering has been commonly used to detect spam, both by MTAs and recipients mail clients. For example, a prominent spam filtering tool, SpamAssassin [Mason, 2011] assigns a score indicating the likelihood of a message being spam based on statistical text analysis [Graham, 2002]. As variants of content-based filtering, the hash of a message that has been received and labeled as spam by someone of collaborators is used to determine whether an email message is spam or not [V. Prakash and J. ODonnell, 2005; Tobin, 2009].

Spammers have recently begun to use image attachments and insert different data in order to have a different message hash as countermeasures to these filtering techniques. Thus, content-based filtering has become more difficult.

For calls, there is no question that content-based filtering is ineffective since the callees are reached and annoyed by the ringing tone before receiving the content, namely voice or video. However, in the case where calls are answered by a machine (e.g., answering machine or voicemail server), content-based filtering might be effective in detecting spam. Relying on the observation that many unwanted calls play pre-recorded messages to decrease the cost of telemarketers, for example, Quittek and his colleagues [Quittek *et al.*, 2007] proposed Turing tests to detect human communication patterns. However, some good calls from government agencies or credit card companies are also automated and pre-recorded messages. Moreover, these tests are likely to annoy human callers.

Based on the application layer inputs: Sender or caller behavior analysis

Statistical analysis of anomalous sender behavior has been explored, which is a relatively new approach, compared to statistical text analysis. By collaborating with a group of individuals, anomalous traffic patterns – based on message volume, frequency, file attachment patterns, and clusters of users who share message copies – are detectable [Stolfo *et al.*, 2006; Boykin and Roychowdhury, 2005].

On the other hand, anomalous caller behavior has been explored less. This is due to the fact that call history data, which are called call detail records (CDRs), contain less data. CDRs typically consist of caller and callee IDs and start and end timestamps, excluding the information conveyed in other SIP header fields. Additionally, since a call request, unlike an email message, has only one destination, it is almost impossible to analyze clusters of users who share a conversation based on call history. Furthermore, the most significant drawback of CDRs is that the content of voice communication, which sometimes is recorded by a voicemail server, is not included. The lack of content in communication history makes difficult offline call classification – spam or not – without the callee's feedback. Therefore, in contrast to email spam where their detection performance can be evaluated using public corpora [spa, 2006; Cormack and Lynam, 2007], statistical analysis of anomalous caller

behavior cannot be performed as readily.

Although they have not been evaluated using real call history data, several frameworks of caller-behavior-based filtering that learn from the callee's feedback have been proposed [Mathieu *et al.*, 2008; Dantu and Kolan, 2005]. Mathieu and his colleagues [Mathieu *et al.*, 2008] proposed analyzing each caller's behavior by measuring the number of concurrent calls, call attempt rate, the callee's address patterns found in subsequent call attempts, and the number of SIP error messages, such as the destination address being not found. However, it is easy for originators of call spam to avoid being detected at the inbound SIP server or a group of callees by distributing calls (e.g., randomizing the order of calls) to multiple inbound SIP servers across service providers. Thus, unless the originating service provider cooperates, it may be difficult to measure the outgoing call volume, similarities of callee's addresses, and error rate. This information may be considered privacy sensitive or a business secret.

Adding procedures to an unknown sender or caller

Based on the observation that spammers' MTAs tend to implement minimal functionality of SMTP, several mechanisms for deterring spammers from sending messages have been proposed [Harris, 2003; Twining *et al.*, 2004]. Greylisting proposed by Harris [Harris, 2003] is representative of the deterring approach. In the greylisting mechanism, inbound MTAs label a sender as unknown if the sender ID of a message is not found in communication log collecting a pair of a sender and the recipient email addresses. Inbound MTAs respond to an unknown sender with a temporary error, and if the sender resend the message, receives it. The deterring approach reasonably places buden on the sender side rather than the recipient side, but it causes good messages from unknown senders to be delayed.

Delaying the signaling message for good calls is unacceptable for initiating real-time communication. Thus, the greylisting approach is difficult to apply to calls.

6.5.2 Identifying Good Email Messages or Calls

This section reviews existing mechanisms for identifying good email messages or calls, comparing them to our work in this thesis. Unlike spam detection mechanisms, this approach has

been less explored; we have only forms of origin-ID-based (i.e., sender-ID-based or caller-ID-based) and consent-based. This section also describes a destination-ID-based mechanism, although it was designed and used mainly for sorting messages.

Based on the application layer inputs: Sender ID or Caller ID

The most common solution for identifying good communication requests is origin-ID-based (i.e., sender-ID-based or caller-ID-based) filtering using the recipient's white list, so-called address book. A white list is populated with the contact addresses of persons or organizations that a recipient has ever communicated with.

A white list is useful for identifying good messages or calls if they meet the following two conditions. One is that the recipient can verify the origin ID authentication. Another is that the origin ID of good messages or calls is included in the white list beforehand. To meet the first condition for email, namely, to allow recipients to authenticate a sender ID from external domains, the DomainKeys Identified Mail (DKIM) signature [Crocker *et al.*, 2011] has been proposed and is currently widely used. DKIM allows the sender domain to sign a set of message header fields including the From header field. Recipients as well as inbound MTAs can verify the sender ID authentication by verifying the signature and assessing the sender domain name. Similarly, a domain-based authentication mechanism for calls using SIP, the SIP identity mechanism [Peterson and Jennings, 2006] has been proposed. However, it is obvious that this domain-based authentication cannot be applied to a phone number in the tel URI. Even for a SIP address-of-record (AoR) in the SIP URI, the authentication service is often unavailable because of the lack of ITSP support. This problem is further described in Section 6.6.

Unless the second condition is met, namely, a sender ID of a good message is not found on a white list, the message cannot be labeled as good. Since a white list of a single user contains a relatively small number of contact addresses, it often encounters the introduction problem.[1] Thus, effective use of a white list depends on how a recipient can collect contact addresses of potential senders beforehand.

To expand white lists leveraging social graphs, Ceglowski and Schachter [Ceglowski and

[1] See the definition in Section 1.2.

Schachter, 2004] introduced the way of sharing privacy-aware address book among friends who exchange email messages. A person sends his friend a message with his hashed address book attached. When a person receives a hashed address book of a friend, he stores the address book as contact addresses of friends of the friend. The hashed address book, only used for filtering purposes, is represented as a Bloom filter [Bloom, 1970], which allows for space efficient strorage. This concept was introduced before social media services had emerged.

Currently, many social media services (e.g., Facebook or LinkedIn) visualize a user's social graphs with their contact addresses. However, the problem is that users of social media services cannot share their contact addresses while preserving their privacy. The users can only select whether or not to publish their contact addresses they can be reached at. One mechanism introduced in Section 7.5.1 proposes a mechanism for filtering calls based on hashed contact addresses from a Web site, typically from a social media Web site.

As another way of collecting contact addresses of potential senders, Shacham and Schulzrinne [Shacham and Schulzrinne, 2007] proposed a new HTTP header that allows a Web client to collect contact addresses from potential callers. We have incorporated this work into our mechanism using cross-media relations for calls preceded by a Web transaction, as described in Section 7.5.1.

Instead of collecting contact addresses of potential callers using social graphs, a mechanism for providing call history in a communication request was proposed by Balasubramaniyan and his colleagues [Balasubramaniyan *et al.*, 2007]. They considered a scenario where a caller (Alice) talked to her friend (Bob) and wants to make a call to a friend of her friend (Bob's friend, Carol). The caller (Alice) provides the callee (Carol) with the call log data (between Alice and Bob) that is certified by Alice's outbound SIP server, called a call credential. The callee (Carol) assesses a call request from an unknown caller (Alice) based on the call credential, which is evidence of a call between her friend (Bob) and the unknown person. People usually trust their own communication history more than their friend's one. If people need to rely on their friend's communication history, they usually verify it by asking their friend, rather than by asking a third-party entity. Our approach using cross-media relations (Section 7.5) relies on evidence of the callee's prior contact, not

an unknown party's communication history. Our approach uses prior contact though a Web transaction or email message exchanges, not limited to a call.

As a variant of sender-ID-based filtering, sender-domain-based filtering is often used based on a list of trusted domains that a recipient configures. This is useful especially for business-to-business communications where recipients authenticate senders by their organization (the domain name), rather than by their name (the user part of an address). This sender-domain-based filtering is effective only in the cases where the callee's trusted domains provide communication services and issue the contact addresses for their users. On the other hand, our proposed mechanism for validating a user's attributes (Chapter 8) allows callees to domain-based filtering, not limited to the domain name of a sender ID, but any domain name the callees trust.

Requiring the consent of a recipient or callee

When the recipient's inbound MTA or inbound SIP server applies white-listing to identifying good messages or calls, the server usually holds messages or calls carrying an unknown origin-ID, rather than immediately blocking them, since no white-lists can perfectly include all origin-IDs of good messages or calls. For example, EarthLink [EarthLink, 2012] server hold such messages as suspected messages in a recipient's message folder. If the recipient checks these messages and gives permission to receive a future message from its sender, the server updates the recipient's white list. A similar example is provided as call screening services by Google voice [voice, 2012]. However, for calls, a consent request is another form of unwanted interception for the callee.

In contrast, our mechanism using cross-media relations (Section 7.6), which can be considered a variant of consent-based framework, does not require an additional call or message for asking the callee for his consent. Rather, focusing on prior contact, the callee grants permission to potential callers through different communication means beforehand. The caller can use a proof of the communication as the callee's consent.

Based on the application layer inputs: Destination ID

The way of setting the destination IDs reveals how the sender wanted to send a message and which address the sender used to reach the recipient. Thus, sorting messages into folders based on the destination IDs has been common practice. With subaddressing [Murchison, 2008], each email address can include an additional value between the user part and the domain part with + (e.g., user+coms101@example.com).

We apply this subaddressing mechanism to a SIP AoR in the SIP URI, to identify good calls preceded by a Web transaction (Section 7.6.1). Because of the syntax similarity between an email address and a SIP URI, applying subaddressing to a SIP URI is straightforward. For a SIP AoR in the tel URI [Schulzrinne, 2004], we borrow the concept of phone number extensions, which is used for call distribution on a PBX, to the tel URI.

6.6 Limited Availability of Caller ID Authentication to Callees

This section discusses technical and practical problems with authenticating a caller ID from the callee's perspective. We start with providing an overview of a caller ID. We then describe anonymous caller ID services, and proceed to discuss technical and practical problems with a caller ID authentication mechanism which is current available, the SIP identity mechanism.

6.6.1 Overview of Caller ID

A caller ID used in a SIP call, which is called SIP address-of-record (AoR), is expressed in one of two schemes: the SIP URI or the tel URI. A SIP URI contains the domain name of the issuer (e.g., sip:username@example.com), similar to an email address. In contrast, a phone number with the tel URI scheme includes country and area codes indicating the geographical location of the originator (e.g., tel:+12121234567).

The geographical location information indicated by the country and area codes has become more uncertain. Although phone numbers have been typically assigned to landlines or mobile phone devices, they have become assigned to the users of VoIP services that offer inter-connection with the PSTN or cellular networks. A VoIP user can pick a phone

number to be displayed as a caller ID when a call reaches a phone device in the PSTN. Without a phone number coupled to a VoIP user, a phone number of the VoIP–PSTN gateway is displayed for the call. Thus, even the country code is unreliable owing to the global availability of VoIP–PSTN services.

6.6.2 Anonymous Caller ID services

Similar to caller ID blocking services provided in the PSTN, ITSPs offer anonymous caller ID services [Peterson, 2002] that allow callers to hide their SIP AoRs for privacy reasons. They sometimes want to avoid getting called back or avoid disclosing the geographical location implied by the country and area codes of a phone number. Anonymous caller ID services obviously conflict with caller ID authentication; thus, caller ID authentication fails to perform.

If an incoming call carries an anonymous caller ID (e.g., `sip:anonymous@anonymous.invalid`), caller-ID-based filtering usually blocks such a call, whether it is a good call or not. Thus, there is a case where a caller wants to provide a piece of information other than the caller ID in order to be identified as a good call and to be answered.

6.6.3 Limitations of Caller ID Authentication Mechanisms

Caller ID authentication falis to perform if the originator's caller ID is replaced with an anonymous caller ID, as Section 6.6.2 described. The following description, thus, discusses the cases where an incoming call carries a caller ID which is not an anonymous caller ID.

If a call originates the domain of the ITSP where the callee connects and the callee trusts its authentication policy, the callee can validate caller ID authentication based on whether the callee can trust the ITSP's authentication policy. If a call comes from a different domain, the callee can authenticate a caller ID using a domain-level authentication mechanism and using transitive trust. A domain-level authentication for SIP, namely, the SIP identity mechanism, has been proposed, but has been suffering from the following technical and practical problems.

Technically, the SIP identity mechanism is available only to a caller ID in the SIP URI, not to a phone number in the tel URI, since it contains no domain name. Thus, there is no

authentication mechanisms for a caller ID in tel URI which is available to the callee. If a call originating the same ITSP as the callee connects and trusts its authentication policy, the callee relies on a caller ID by checking SIP routing-related header fields, such as a SIP Via header filed.

Furthermore, the SIP identity mechanism encounters a practical problem even with a caller ID in the SIP URI. The practical problem throws obstacle to deploy caller ID authentication services because of the following service conflict. The service conflict occurs when a session border controller (SBC) [Hautakorpi *et al.*, 2010] exists along in signaling path. An SBC is an entity that typically manipulates signaling messages based on the ITSP's policies, modifying part of signed information, for example, the Contact header field of a SIP INVITE request in order to hide the ITSP's network topology.

Unlike calls, email services do not suffer from the difficulties in deploying sender ID authentication. A domain-level authentication mechanism for email, such as DKIM, has been supported on many SMTP [Klensin, 2008] servers.

6.7 Limitations of Caller-ID-based Filtering

Filtering call systems based on a caller ID have been commonly used, but they are not always effective for the following reasons. First, caller-ID-based filtering can work properly only in the cases where caller ID authentication is available. Callees cannot authenticate a caller ID in the tel URI or blocked for privacy reasons, as discussed in Section 6.6. Caller-ID based filtering systems, which are usually implemented on a VoIP server, PBX [Hewlett-Packard, 2010], and/or user clients, first need to validate caller ID authentication in order to detect caller ID spoofing.

Second, filtering based on a black list, which consists of contact addresses to reject calls, is less effective for VoIP calls. This is due to the fact that callers can pick a new caller ID easily and cheaply, similar to an email address. Thus, even if the callee's black list contains the contact addresses of unwanted callers and links to a reputation service that gathers IDs of well-known malicious callers, the effectiveness of black-listing is limited. Consequently, rather than black-listing, white-listing, which identifies good calls based on caller IDs in a

white list, plays a large role in preventing unwanted calls.

Third, no list can perfectly include all necessary caller IDs beforehand; thus, the effectiveness of white-listing is also limited. The caller IDs of good calls are not always found in the callee's white list. A white list generally contains the contact addresses of persons or organizations connected to the callee by strong social ties, such as friends or close colleagues, not the contact address of those who are connected by weak social ties, who have had prior contact, but do not regularly communicate with each other. The examples of those who are connected by weak social ties include the callee's friends in the extended social graphs of social media Web sites or the call center of an airline company that the callee booked a flight.

Thus, to determine whether or not to accept incoming calls, simple caller-ID-based mechanisms are limited in applicability and effectiveness. Our challenge is to develop more sophisticated mechanisms in order to conquer the difficulty in labeling the following types of calls:

1. Calls from persons or organizations connected to the callee by weak social ties;
2. Calls from those connected to the callee by strong social ties, but using new, alternative, or unknown caller IDs, for example, from a temporary location like a hotel;
3. Calls with unauthenticated caller IDs, for example, a SIP URI containing an unauthenticated domain or a phone number in the tel URI, which includes no domain name.
4. Calls with blocked caller IDs, (i.e., anonymous caller IDs).

6.8 Limitations on the Use of Caller ID as a Caller's Attribute

This section describes limitations on the use of caller ID as a caller's attribute and on the use of sender ID as a sender's attribute. As discussed in Section 6.3.2, people often assess others based on their attributes, not by their identity. Similarly, recipients may assess communication request based on the originator's attributes, if available. Thus, this thesis

explores a lightweight mechanism for validating the originator's attributes in Chapter 8. We have observed that part of an origin ID (i.e., caller ID or sender ID) is sometimes useful in identifying the originator's attribute, for instance, the domain part in an email address (e.g., columbia.edu). Based on this observation, the use of an authenticated origin ID as the originator's attribute can be considered to be a lightweight mechanism, serving the same purpose as ours. Thus, prior exploring a new mechanism, we evaluate the use of an origin ID as the originator's attribute, focusing on the limitations and problems with that.

Useful attribute in limited cases: Email address or SIP URI Issued by a trusted domain

A domain name of an origin ID is helpful to indicate the originator's attribute only when it satisfies the following two conditions. First, the origin ID must be authenticated by the domain and verifiable by the recipient using a domain-level authentication mechanism, such as DKIM or Sender ID for email, and the SIP identity mechanism for a SIP URI. Second, the domain must be trusted by the recipient from the viewpoint of user account management policy, such as an enterprise and a school issuing CEIDs, strictly limited to their employees and students. Many people greatly benefit from communication services provided by Internet service providers (ISPs) or application service providers (ASPs). Because of their open user account management policy, people can easily obtain and keep using their CEIDs, whether or not their affiliation change. However, at the same time, the domains of their CEIDs cannot be trusted as attributes of the caller or email sender.

A caller ID in the tel URI (i.e., a phone number in the tel URI) can be authenticated by the callee only when the call originates from the same domain where the callee connects and trusts its authentication policy, as described in Section 6.6.3. Under the condition that a phone number in the tel URI is authenticated, country and area codes in the tel URI sometimes help identify the geographical location of the landline caller, but have become less helpful for the following reason. A phone number in the tel URI was originally assigned to a line for landline or a device for wireless telephone services. However, many ISPs and ASPs have recently started to offer connection services to the PSTN and to allow their users to pick their phone number to display on the recipients' device, wherever their geographical

location is. Thus, even the country code in the tel URI has become unreliable information about the caller.

Based on this observation that callees cannot authenticate the caller's phone number in the tel URI, we were motivated to propose a mechanism for validating a user's attributes without authenticating the user's identity described in Chapter 8.

One origin ID per communication request

Assume that an originator has multiple CEIDs issued by multiple domains, which indicate her attributes, affiliations to multiple organizations for instance. If she can include her CEIDs into a single communication request, she might raise the possibility of being accepted by the recipient. However, only a single origin ID can inherently be set in a communication request, namely, the From header field in an email message or in a SIP INVITE request.

Reference integrity problem when using directory service

Even if an authenticated origin ID does not provide sufficient information, recipients might be look up further user attributes through directory services. However, recipients often encounter a reference integrity problem. This reference integrity problem arises when a directory service allows queriers to look up user attributes by the information available to the public, such as by a user's last name, not by the user's email address or SIP AoR. Additionally, even when a directory service allows queries by a user's email address or SIP AoR, but is offered by a third party which is not the issuer of the address, the authenticity of the information is unreliable. For example, the DoctorFinder[2] service provides information about certified medical doctors. When making a query, a querier cannot use the doctor's phone number, but rather needs to the use doctor's last name, street address or specialty, which is available to the public. Thus, if a doctor sends an email message or makes a call that includes such query information and a reference to the DoctorFinder service, the recipient is not convinced of the certainty of the information.

[2]This DoctorFinder service is offered by the American Medical Association at https://extapps.ama-assn.org/doctorfinder/ as of Feb. 1, 2011.

These problems motivated us to a propose mechanism for validating user attributes described in Chapter 8.

Chapter 7

Using Cross-Media Relations to Identify Good Communication Requests

7.1 Introduction

Good calls[1] carrying an unknown caller ID often originate from persons or organizations who have had prior contact through a Web transaction or email exchanges, resulting in the establishment of weak social ties[2] to the callee. According to a 2011 *New York Times* article [Paul, 2011], many callees now prefer receiving calls preceded by email or text message exchanges in order to avoid being disturbed even by calls originating from family members or friends, who are connected by strong social ties. Motivated by these observations, our study focuses on prior contact as an additional indicator for identifying good calls.

This chapter first presents our hypothesis that prior contact is helpful to distinguish between good and unwanted calls. It then defines a piece of information exchanged in prior contact as a *cross-media relation* and provides an overview of the service architecture where both parties establish a cross-media relation. We distinguish two mechanisms

[1] See the definition in Section 6.4.

[2] See the definition in Section 6.2.

based on the type of information used as cross-media relations, collecting the contact addresses of potential callers and exchanging a *weak secret*, which is a proof of prior contact, provided by a callee. To test the concept of using cross-media relations, we conducted a user study of incoming email and SMS messages in addition to incoming calls. The user study demonstrates that using cross-media relations, especially *Web-then-email*, which is a relation between two parties who communicate using an email message preceded by a Web transaction, is potentially effective in identifying a good communication request carrying an unknown origin ID, namely, sender ID or caller ID.

7.2 Hypotheses

We first hypothesize that a significant fraction of incoming calls are good calls and originate from persons or organizations connected by weak social ties. Their contact addresses, similar to callers who make commercial bulk calls or spammers, are rarely found in the callee's address book. However, those connected by weak social ties differ from spammers in that they have had prior contact, either in one or both directions, with the callee through different communication means. They change the means of communication, typically from asynchronous to synchronous and from text to voice or video. In contrast, spammers usually make calls without any prior contact in order to make bulk calls as efficiently as possible. Thus, we hypothesize that prior contact is a helpful distinguishing feature between good and unwanted calls.

Based on these hypotheses, this thesis proposes that both parties exchange an additional piece of information which can be used in future calls as evidence of prior contact.

7.3 Cross-Media Relations

We define a piece of information that can be used for proving the existence of a prior communication as a cross-media relation. If a call is preceded by a Web transaction, the caller and callee have a relation called *Web-then-call*. If the call is preceded by an email message, the relation between the two parties is called *email-then-call*. Table 7.1 summarizes these two types of cross-media relations, Web-then-call and email-then-call, which are mainly

Type	Contact addresses of a potential caller	A weak secret provided by the callee
Web-then-call	Contact addresses either in plain text or hashed form	A customized contact address of the callee
Email-then-call	Contact addresses	Message ID of an outgoing email message

Table 7.1: Cross-media relations: Types and information exchanged in prior contact

discussed in this thesis. We distinguish two mechanisms based on the type of information for each cross-media relation. One type is the contact addresses of a potential caller and another is a weak secret, which is a proof of prior communication, provided by the callee.

In the case where both parties have established a Web-then-call relation, the callee collects the contact addresses of potential callers either in plain text or hashed form. Hashed contact addresses are useful for publishing them on the Web site while protecting the potential caller's privacy. Section 7.5 describes how the callee collects the contact addresses of potential callers in two scenarios, Web-then-call and email-then-call.

The type of information used as a weak secret varies according to the previous communication means. To prove a Web-then-call relation, a weak secret is a customized contact address since no ID for a Web transaction exists. To prove an email-then-call relation, the message ID of an outgoing email message can be used as a weak secret. Section 7.6 details the ways that both parties share a weak secret in two scenarios, Web-then-call and email-then call.

7.4 Service Architecture

Figure 7.1 provides an overview of our controlling unwanted requests (CURE) system that allows callees to collect and look up cross-media relations in order to identify good calls. Bob, a user of the CURE system, has had contact with an airline call center through a Web transaction or email message exchanges, for example. He has exchanged a weak secret as

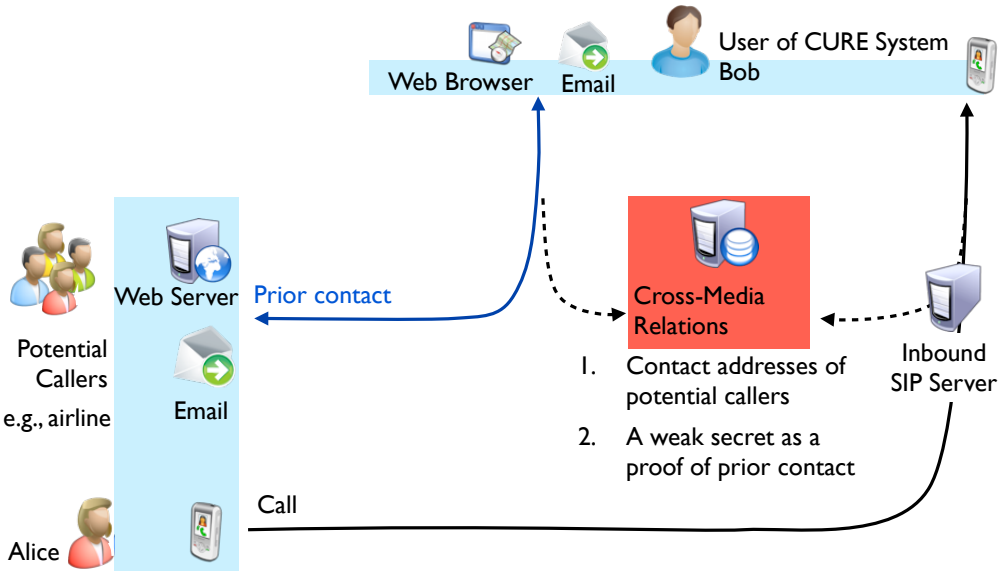


Figure 7.1: Service architecture of CURE system using cross-media relations

a proof of prior communication and stored it in a database for future calls. While a weak secret to prove a Web-then-call relation is a customized his contact address, the secret to prove a email-then-call relation is the message ID of his email message. The database where these piece of information are stored should be accessible by applications on multiple devices that Bob uses and, if need, also by an inbound SIP server that queries it on his behalf. Each application inserts a cross-media relation into the database with corresponding information, namely, the URL of the Web site or the sender ID and the destination addresses of the email message.

When Alice, a person from the call center, makes a call to Bob, the call signaling message, namely, the SIP INVITE request, conveys the information to be identified as a call originating from someone from the call center. By looking up cross-media relation data in the database, Bob correlates the incoming call with prior contact to the call center. He then determines whether or not to answer the call and adjusts his communication stance accordingly.

All messages for a Web transaction, email exchanges, SIP signaling, and database access should be protected by security mechanisms such as TLS in order to ensure their confidentiality and integrity.

7.5 Collecting Contact Addresses of Potential Callers

The more contact addresses of potential callers a callee can obtain beforehand, the more effectively existing caller-ID-based filtering systems can be used. Our first mechanism, thus, collects as many contact addresses of potential callers as possible. This mechanism is useful only when the contact addresses to be used for future calls have been determined and the number is not too large to store in a person's buddy list, unlike the list of contact addresses of all employees in a large enterprise. It is also useful in the case where either a caller or the callee connects to the PSTN or uses a conventional phone terminal, which cannot support the enhancements required for the second mechanism described in Section 7.6.

The following sections describe how potential callers provide callees with their contact addresses in Web-then-call and email-then-call scenarios, respectively.

7.5.1 Scenario 1: Web-then-Call Relations

This study considers two Web-then-call scenarios: one collecting contact addresses in plain text from a Web site, and another collecting contact addresses in hashed form, especially the contact addresses of friends from a social media Web site. The advantage of collecting contact addresses in plain text is that it can be used by existing filtering systems without adding functionality to them. Many enterprises publish their contact addresses on their Web site to allow their customers and business partners to continue identifying their calls in a conventional way.

However, many individuals are unwilling to do so, such as at user profile on a social media Web site, because of privacy concerns that publishing their contact addresses in plain text would cause more unwanted incoming calls. This is due to the fact that spammers usually harvest addresses by exploiting a Web crawler application program [Dodds, 2006]. To relieve their privacy concerns, the CURE system allows potential callers to provide their contact addresses in hashed form, such as the SHA-1 [SHA, 2002] hash of a phone number and the URL of the Web site where the hash is to be published. Such a hashed contact address can be used only for identification purposes. We hypothesize that many users prefer concealing their routable contact addresses on their profile even to their friends, but

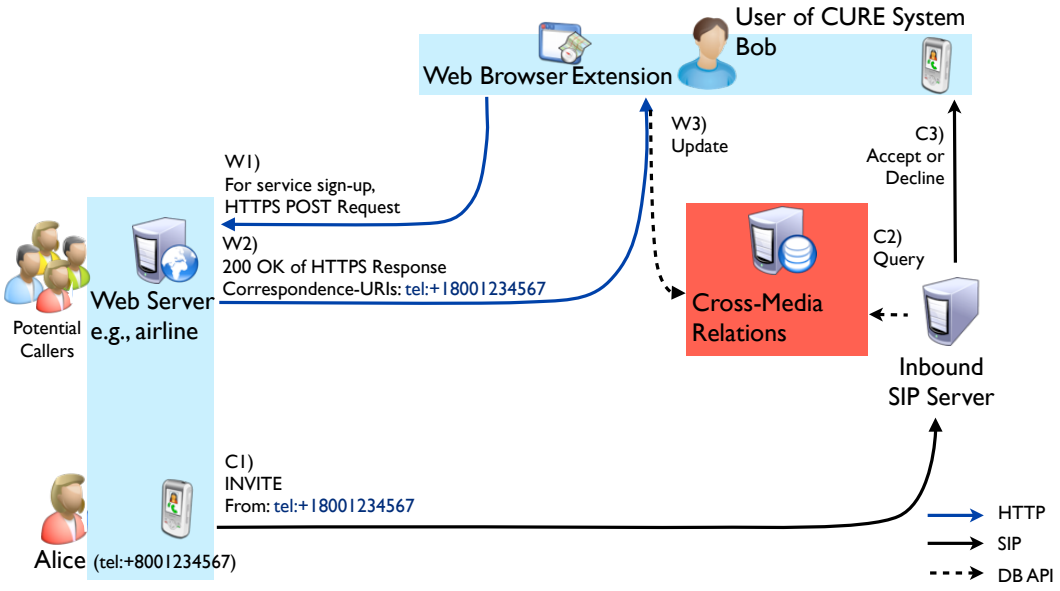


Figure 7.2: Collecting contact addresses of potential callers in plain text: Web-then-call

being still identifiable at the callee. Thus, the goal of using hashed contact addresses is to encourage users to publish their contact addresses, even on their public profile Web site.

Collecting contact addresses in plain text

Figure 7.2 illustrates an example of message exchanges in the first Web-then-call scenario between Alice, a person from an airline call center, and Bob, a user of the airline's Web server. When Bob successfully signs up for a service, the airline's Web server has two options to notify him of the airline's contact addresses. The first option is to respond with an HTTP 200 OK response carrying the contact addresses in plain text for future communication. These contact addresses are conveyed in a new HTTP header field, Correspondence-URIs [Shacham and Schulzrinne, 2007]. To do so, a Web server just needs to have an HTML meta tag, HTTP-EQUIV [Raggett *et al.*, 1999] consisting of the header field name, Correspondence-URIs, and a contact address of the Web site, in the header of an HTML page, as described in Figure 7.3. When a Web server sends the HTML page, it automatically converts the HTTP-EQUIV tag into the corresponding HTTP header.

The second option is to guide Bob to a site-wide meta information page [Hammer-Lahav

```
<html>
 <head>
   <META HTTP-EQUIV="Correspondence URIs" CONTENT="tel:+18001234567">
 </head>
</html>
```

Figure 7.3: An example of Correspondence-URIs in HTML

and Cook, 2011]. In either option to convey potential contact addresses, the data integrity of the transaction should be protected by using secure HTTP, or HTTPS (i.e., HTTP over TLS).

Upon receiving the HTTP 200 OK response, a Web client extension supporting an additional function for the CURE system extracts the contact addresses, and updates the database of cross-media relations with the URL of the Web site. To prevent misuse, the extension prompts Bob for confirmation before updating the database. Additionally, it asks Bob if the expiry date should be set for the temporary use of the service and when it will expire. If Alice needs to contact Bob afterwards regarding the service he signed up, she just needs to make a call to him from one of the contact addresses delivered in the previous Web transaction. An inbound SIP server for Bob queries the database for the caller ID in order to determine whether to accept the call.

Collecting contact addresses in hashed form

Figure 7.4 depicts the second Web-then-call scenario, where contact addresses are in hashed form. Assume that Alice has published her contact address in hashed form on her profile page on a social media Web site. The hash string is generated by an application at the Web server using SHA-1 from her contact address in plain text concatenated with the URL where the address is to be published (e.g., `H(tel:+12121234567||URL)`). This concatenation prevents hashed contact addresses from being correlated across different Web sites. Bob, a user of the social media site, retrieves Alice's hashed contact addresses through a message body in the HTTP 200 OK response to an HTTP GET request.

When Alice makes a call to Bob, she needs to specify both her contact address in plain

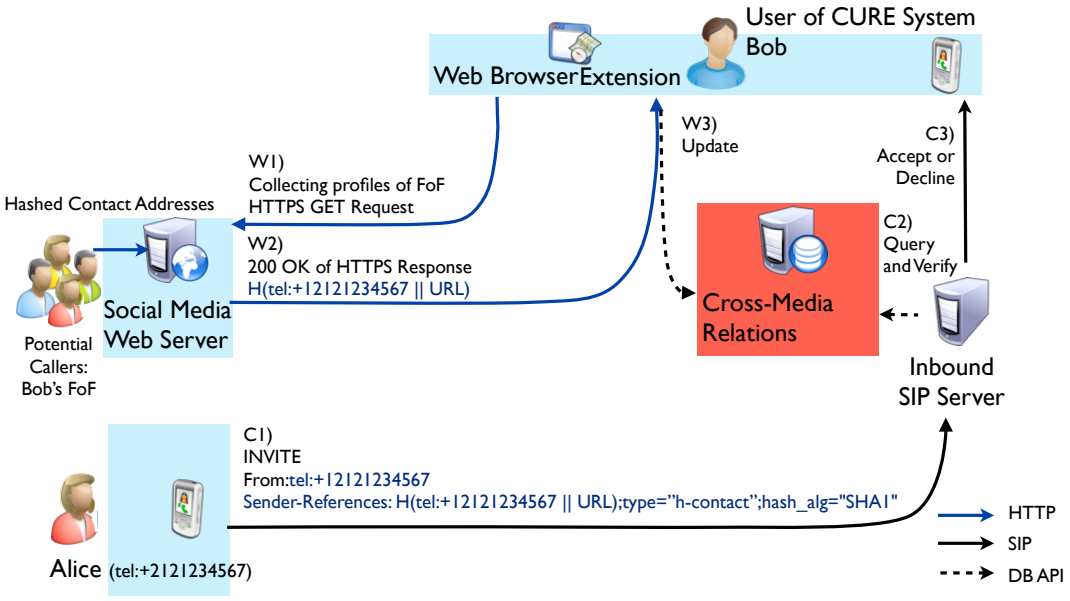


Figure 7.4: Collecting contact addresses of potential callers in hash format: Web-then-call

text and her hashed contact address published on her profile page in order to be identified as a person in Bob's social graphs. A SIP UA conveys her caller ID in the From header field and the hashed contact addresses in a new SIP header field, Sender-References [Ono and Schulzrinne, 2009b] with the h-contact type parameter and the SHA-1 hash algorithm. An inbound SIP server first on behalf of Bob queries the database for the URL corresponding to the hashed contact address in the Sender-References header field. To verify the hashed contact addresses, the SIP server computes the hash of the contact address in the From header field of the received call concatenated with the URL retrieved from the database. If it succeeds in the verification, it determines whether to accept or decline the call.

7.5.2 Scenario 2: Email-then-Call Relations

Figure 7.5 illustrates that Bob collects Alice's contact addresses in a vCard [Dawson and Howes, 1998] attachment to her email message only when he replies to that message. His reply, except from a compromised machine, is evidence that the original message from Alice was good.

An additional function on each mail delivery agent (MDA), such as an IMAP [Crispin,

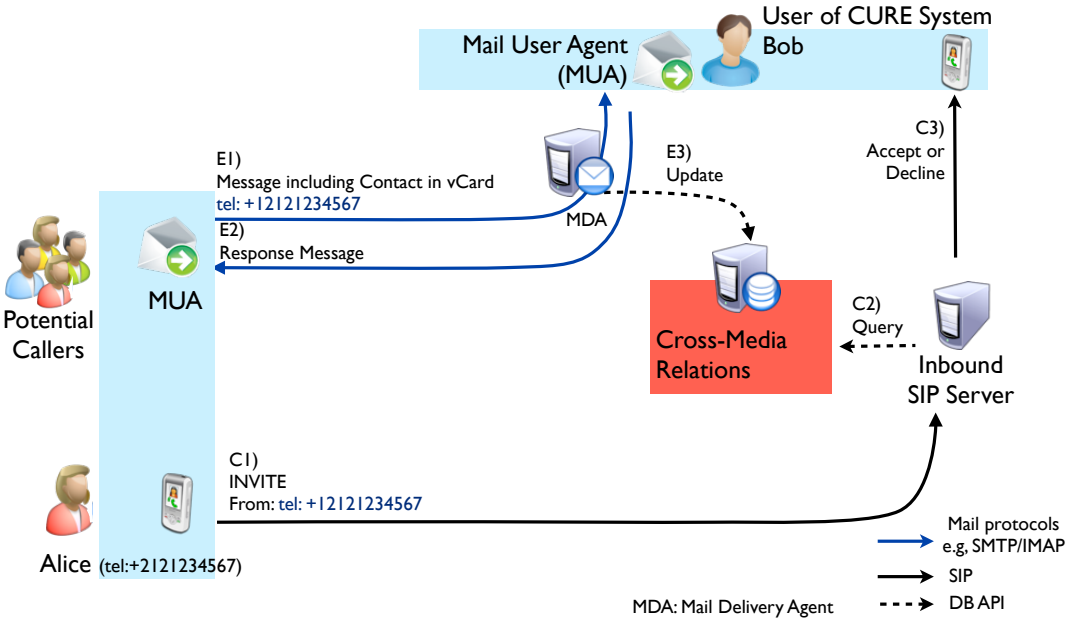


Figure 7.5: Collecting contact addresses of potential callers: Email-then-call

2003] server, that Bob uses, or a dedicated IMAP client for the CURE system parses email messages that have been replied to and extracts contact addresses (i.e., email addresses and phone numbers) in a vCard from them. To determine whether a message has been replied, checking the IMAP answered flag seems to be efficient, but does not always work. This is due to the fact that sending or storing the flag is optional to implement, according to the IMAP specification [Crispin, 2003], and popular implementations does support.[3] Thus, the additional function to the dedicated IMAP client needs to look for a reply and the original message by tracing the linkage between email messages that can be found in message IDs in the In-Reply-To and Message-ID header fields.

7.6 Sharing a Weak Secret Generated by Callee

In the second mechanism, the callee provides potential callers with a weak secret as a proof of prior communication. This weak secret, a piece of information, should be sufficiently

[3]For example, Thunderbird mailer 3.1 [Mozilla, 2010b] does not send the flag. Gmail [Google, 2011] IMAP server and its Web mail client do not store the flag.

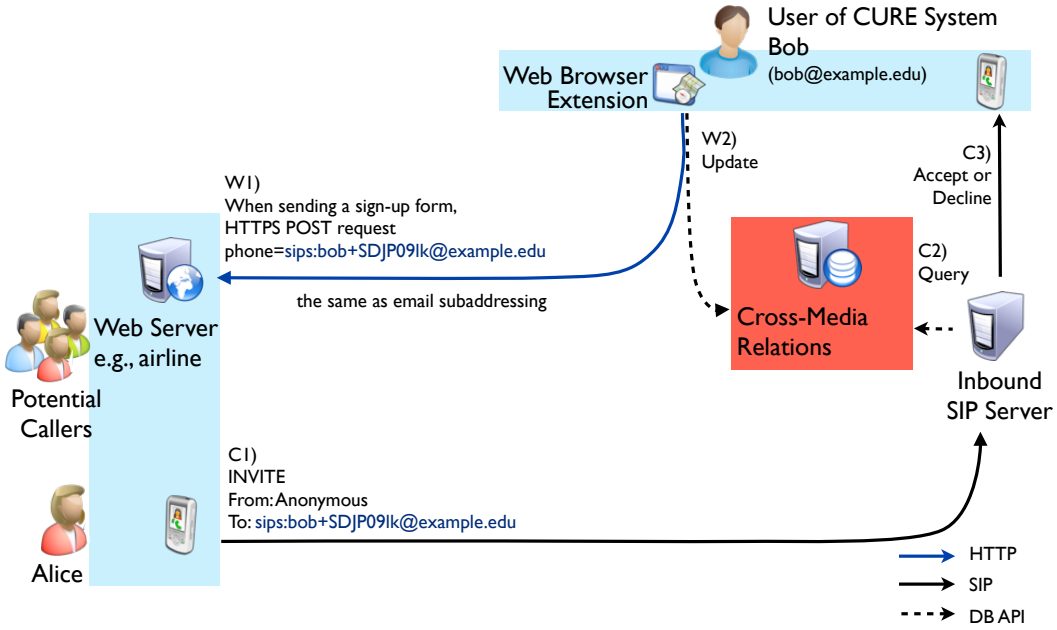


Figure 7.6: Sharing a weak secret: Web-then-call

confidential to determine whether to answer a call, which is a relatively low-risk interaction.

Sharing a weak secret between a potential caller and the callee is effective in the following cases. One is where the previous contact is one-to-many correspondence between the callee and the potential callers. The callee is usually unwilling to receive and maintain a long list of contact addresses of the potential callers, for example, when joining a club or attending a conference. Another case is where potential callers cannot assure the callee of their contact addresses, such as a call from a temporary location like a hotel.

7.6.1 Scenario 1: Web-then-Call Relations

A Web transaction, namely an HTTP transaction, does not have any transaction ID according to the HTTP specification [Fielding *et al.*, 1999]. Although an HTTP Cookie [Barth, 2011] can contain a session ID, it is generated by a Web server for its purposes, not by a Web client. Thus, rather than using an HTTP Cookie, the callee creates a customized phone number (or email address in a Web-then-email scenario) that contains a random component and uses it as a proof of a Web transaction.

As Figure 7.6 illustrates, when Bob fills out his contact information in a sign-up form, a

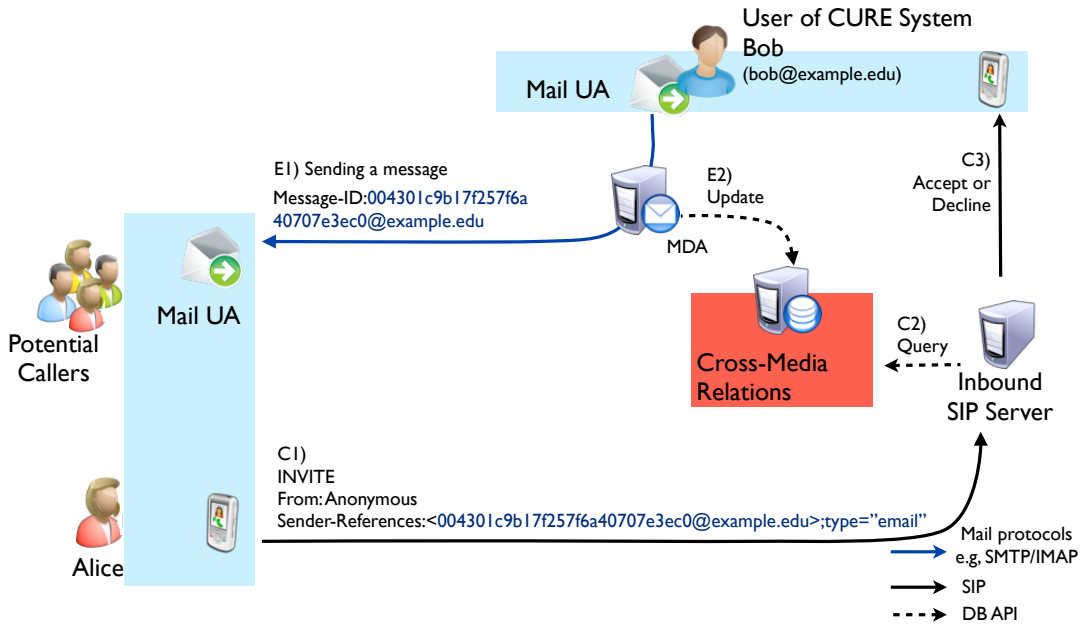


Figure 7.7: Sharing a weak secret: Email-then-call

Web client extension helps him generate a random component (e.g., `SDJP09lk`) and insert it between the user name and the domain name preceded with +, in the same way as the email addressing practice called subaddressing [Murchison, 2008]. For a phone number in the tel URI scheme [Schulzrinne, 2004], random digits follow the phone number like an extension that identifies a terminal behind a PBX (e.g., `tel:12121234567;ext=099213`). Every time the Web client extension generates a random component, the extension updates the CURE database with the customized contact address and the URL of the Web site.

Upon making a call to Bob, Alice can be identified by the destination address (e.g., `To:bob+SDJP09lk@columbia.edu`). Although no extension is needed in a SIP UA, the SIP server needs to support subaddressing of the destination address in its routing process. The SIP server then queries the CURE database for the destination address to determine whether or not to answer the call.

7.6.2 Scenario 2: Email-then-Call Relations

Unlike an HTTP transaction, an email message has a globally unique ID in the Message-ID header field [Resnick, 2008], which is generated by a mail server or client. Figure 7.7 shows

that Bob first sends an email message to a potential caller. He needs to collect the message ID of his outgoing message and keep accessible in order to prove his acceptance for future communication. When Alice makes a call to Bob, she should specify the message ID of his message so that the SIP INVITE request conveys the message ID in the Sender-References header field, specifying the `email` type. The SIP server, therefore, can determine whether or not to accept the call carrying the message ID of an email message.

To collect the message IDs of outgoing messages, the CURE system needs to add a function on each MDA that Bob uses or on a dedicated IMAP client, similar to the additional function described in Section 7.5.2.

There are privacy concerns over messages posted to a mailing list since mailing lists often publish their archives including their message IDs on the Internet. Thus, these message IDs of mailing lists available to the public should not be used as a weak secret. However, senders cannot always determine whether or not the destination of an outgoing message is a mailing list. This is because some mailing lists share their messages without the subscribers' consent. Although senders cannot distinguish a mailing list address from regular email address when sending a message, they can determine, when receiving a message from a mailing list, by checking List-* or Precedence headers. The MDA or IMAP client, therefore, needs to learn the destination address of a mailing list from a message received from the mailing list. The MDA or IMAP client then excludes an outgoing email message destined to a mailing list when extracting message-IDs from outgoing messages or finding any message IDs of email messages posted to the mailing list in the stored message IDs. Learning mailing list addresses from received messages is likely to perform without a significant delay since the subscriber of the list can usually receive a message copy immediately after posting the message. Therefore, the MDA or IMAP client avoids accepting a message ID as a weak secret that is available to the public.

7.7 Call Filtering

Figure 7.8 illustrates our new filtering process for an incoming call, adding two conditions that use cross-media relations. Upon receiving an incoming call, the filtering system parses

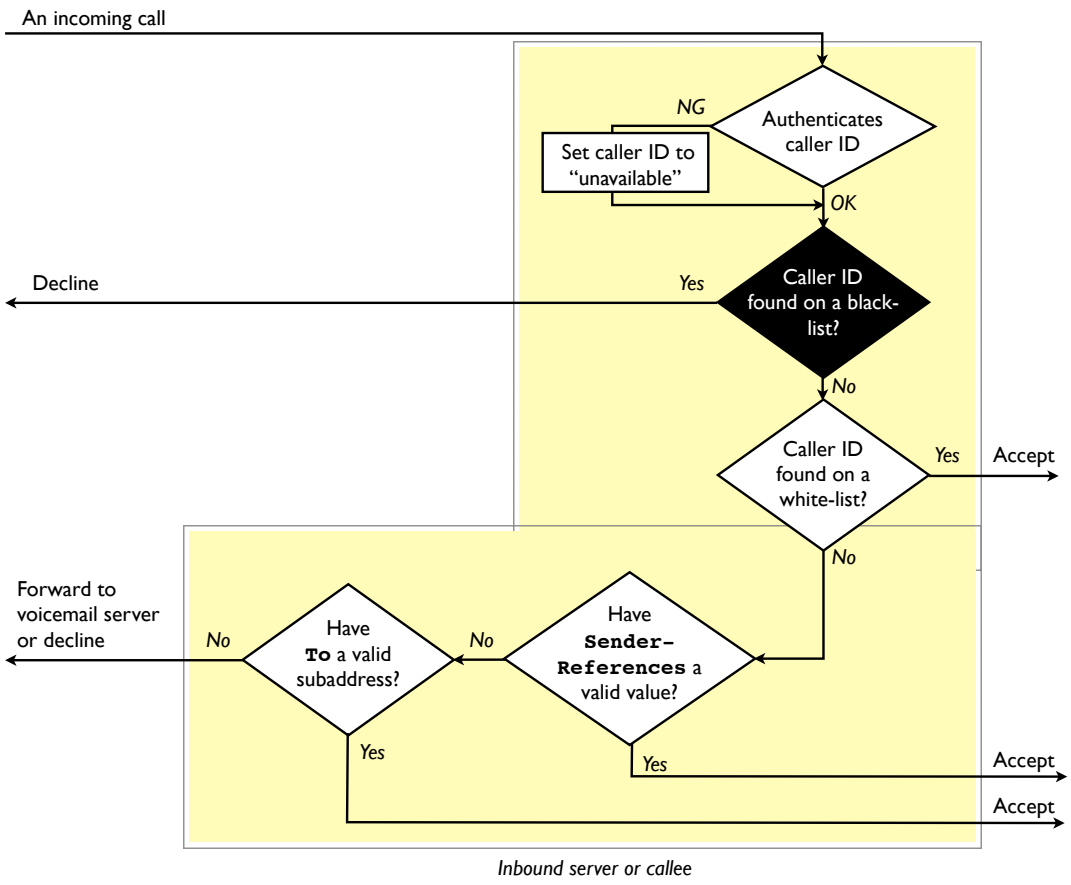


Figure 7.8: Call filtering using cross-media relations

the SIP INVITE message and extracts the authenticated caller ID, typically from the `From` header field. If the caller ID is not found either on the blacklist or white list, this filtering process proceeds to test on these two new conditions.

The first one is whether the SIP INVITE request contains a valid hashed contact address or a valid message ID in the Sender-References header field. As Section 7.5.1 described, upon starting the verification, the filtering system determines the type of the value in the Sender-References header field by parsing its type parameter. If the `h_contact` type is specified, the filtering system verifies the value of Sender-References header field as a hashed contact address. To verify the hashed contact address, the filtering system first queries the CURE database for the URL corresponding to the hashed contact address. It then calculates the SHA-1 hash of the caller ID extracted from the INVITE request concatenated with the URL retrieved from the database. The filtering system finally checks if the newly generate hash is the same with the hash extracted from the Sender-References header field. In the case where the `email` type is specified, the filtering system handles the value of Sender-References header field as the message ID of an email message, as described in Section 7.6.2. It simply queries the CURE database for the message ID and determines whether or not to accept the call.

If the call fails in being verified on the first condition, it proceeds to the test on the second condition. The filtering system tests the INVITE request whether to contain a valid subaddress in the To header field. The filtering system simply verifies the subaddress by checking is it is registered in the CURE database.

Thus, on two new conditions using cross-media relations, the validity can be determined by a database query and the information conveyed in the INVITE request. If either validity test succeeds, the call request is accepted.

7.8 Implementation

To demonstrate our concept of using cross-media relations, we have implemented the CURE system using a MySQL [MySQL, 2006] database server running Linux and RESTful JSON Web interfaces [James, 2010] to the database. To add each of our proposed mechanisms

into Web and email clients, we developed a Firefox add-on [Mozilla, 2010a] and a dedicated IMAP mail client. We also modified SIP communicator [SIP, 2010] as a SIP UA and OpenSER [Ope, 2010] as an inbound SIP proxy server.

The call filtering function has been implemented on OpenSER SIP server which queries the database for an access control list including cross-media relations, instead of using a call processing language (CPL) [Lennox *et al.*, 2004] script. Although the CPL script, which is written in XML [Bray *et al.*, 2004], is a more general mechanism, it incurs the significant storage cost of large-sized XML data and its parsing cost causing a longer setup delay. To avoid these costs, the SIP server simply uses an access control list in a database.

7.9 Applicability to Email

The mechanism for collecting contact addresses either in plain text or in hashed form is applicable to email messages, namely, Web-then-email relations. This is due to that fact that the Correspondence-URI header field can contain any types of contact addresses. It also due to that fact that email message headers are extensible. Similar to SIP, email clients can add the new Sender-Reference header field in order to convey a hashed contact address to recipients.

Similarly, the mechanism for sharing a weak secret, which is a proof of the previous Web transaction, can help recipients good email messages. However, especially in the case where the recipient's email address does not appear in the header fields in an email message, namely, being blind carbon copied (BCC'ed), the filtering based on the destination address should be installed at the inbound server, not at the recipient's mail client. Since all the destination address, not distinguishing between them contained in message header fields, To, CC, or BCC, are provided in the SMTP RCPT TO envelop, the filtering system should parse the SMTP envelop to extract the destination addresses in order to see if they include the recipient's customized address. Since the subaddressing for customizing contact addresses has been commonly used for sorting messages into mail folders, no difficulties exist in the subaddressing deploying for email.

Thus, both mechanisms help recipients identify good email messages although the in-

bound server should verify their destination addresses extracting from the SMTP envelop, not from the email header field.

7.10 Evaluation: Testing the Concept through a User Study

Ideally, we would like to evaluate our implementation by trial use, but three practical problems have made this impractical. First, people currently receive very low volume of unwanted calls compared to unwanted email messages. Second, the trial requires cooperation with Web sites which operate call center services or social media. Third, it requires end-to-end SIP connections to transfer a weak secret in a SIP message between a caller and the callee since the PSTN in certain countries, for example in the U.S., does not allow to transfer the extension in the destination phone number until the call is answered. Although SIP has been deployed for many real services, the majority of VoIP calls have interconnected with the PSTN. Thus, instead of evaluating our implementation, we tested the concept of cross-media relations by conducting a survey. In addition to incoming calls, our survey covered email and SMS messages.

7.10.1 User Survey

We conducted a survey[4] using a Web application we developed. Our survey application asked participants about incoming email, calls, and SMS messages received over the last four weeks. Our survey application fetched email headers from participants' mail boxes on their IMAP servers and parsed call detail records (CDRs) if provided by participants. If participants preferred, they could manually enter the number of incoming calls or SMS messages.

The user study was carried out from July to September in 2011, calling for participants on campus wide. Each participant had to have received at least 28 messages or calls for last four weeks. Collected answers include 3,257 messages for 19 university email accounts and 21,051 messages for 43 free email accounts. Our sample also contains 3,300 calls for 36

[4]The survey was approved by Columbia University Institutional Review Board (IRB) approved on June 20, 2011. The number was AAAI1148.

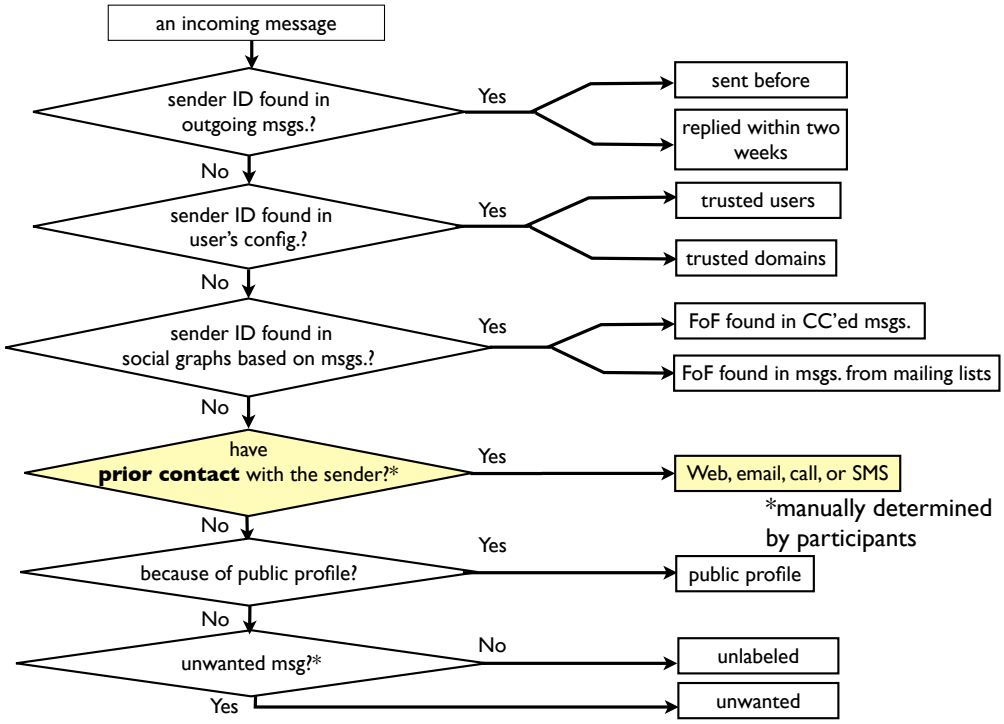


Figure 7.9: Incoming email messages categorization flow chart

accounts and 3,970 SMS messages for 34 accounts.

It is worth noting that these messages included only part of unwanted messages or calls which the participants received since many unwanted messages or calls had been removed by the spam filters on these IMAP servers or call servers, by the participant's email client, and manually by the participants.

7.10.2 Classification of Incoming Email, Calls, and SMS

Figure 7.9 illustrates how each participant categorized incoming email messages into 13 groups: 12 groups of good messages and one group of unwanted messages. The first six groups are automatically determined as good by our survey application in the following three procedures. First, the "sent before" and "replied within two weeks" groups are determined with outgoing email messages. Second, the "trusted users" and "trusted domain" groups are determined with the user configurations at our survey Web application. If participants do not modify their configurations, messages from the participant's own address and from

the default trusted domain, "columbia.edu", fall in to the "trusted users" and "trusted domain" groups, respectively. Although the survey application does not use participant's address book, these four groups are equivalent to good messages determined with their address book. Third, the survey application categorizes messages from friends of a friend (FoF) into two groups, "FoF in CC" and "FoF in ML." The application generates a list of email addresses of "FoF in CC" by collecting carbon copied (CC'ed) addresses of messages sent from trusted users. Similarly, it generates a list of email addresses of "FoF in ML" by collecting email addresses of the senders of the mailing lists (MLs) that the participants join. Using these two lists, incoming messages fall into these two groups, respectively. Thus, these two groups are determined by extended social graphs automatically calculated by email communication history.

Unlike email messages, incoming calls or SMS messages are categorized in 9 groups, excluding four out of these 13 groups for the following reasons. The "replied within two weeks" group is excluded to allow participants to easily answer by manually looking at their call history. The "trusted domain" group is eliminated since phone numbers contain no domain name. The "FoF in CC" and "FoF in ML" groups are also eliminated since a call has no CC'ed or a list of phone numbers as the destination, except a conference all.

After the survey application automatically determined good messages, the participants are asked if they have had prior contact with senders or callers, and if so, what the communication means was. Thus, the participants manually labeled the remaining messages as Web, email, a call, or SMS, according to cross-media relations. These four groups also contain good messages, which are prone to be false positives, namely, labeled as unwanted by conventional origin-ID-based filtering systems. These four groups, therefore, indicate the potential effectiveness of using cross-media relations.

Yet, good messages might remain unlabeled after the participants labeled messages using cross-media relations. Our survey application asked participants to specify a message if the message arrived because they are well-known in a specific field. Such a message is categorized into the "public profile" group. If the participants find unwanted messages, they were asked to label them as "unwanted." Finally, good messages that the recipients have no idea what triggered a message remained unlabeled. These remaining messages

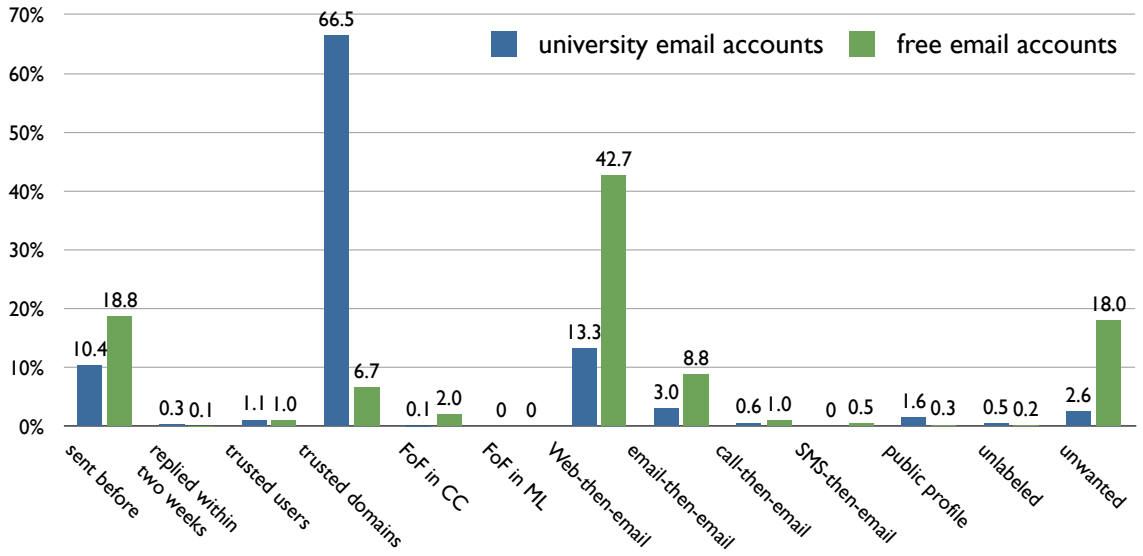


Figure 7.10: Histogram of percentages of incoming messages in 13 groups: Email (averaged across accounts)

fall into the "unlabeled" group. The "public profile" and "unlabeled" groups, therefore, indicate the limitation of our proposed filtering systems using cross-media relations in order to identifying good messages.

7.10.3 Results: Fractions of Messages

Email messages

Figure 7.10 illustrates the percentages of incoming messages in these 13 groups. These percentages are averaged across the participants of our survey. Each group has two bars: a dark (or blue) bar for university email accounts and a light (or green) bar for free email accounts, such as Gmail. The highlight of our results is that prior Web transactions triggered only 13.3 percent of messages for university email accounts, but a surprising high 42.7 percent for free email accounts. On average 33.2 percent for all email accounts had Web-then-email relations although they have wide variance across email accounts, as described in Appendix B.1. These results demonstrate that Web-then-email relations will be very effective in determining good messages whereas the other two types of cross-media relations, call-then-email and SMS-then-email, are rarely used.

Although for most groups, fractions of messages are similar between these two types of email accounts, significant differences are found in two groups: messages from trusted domains and messages triggered by Web-then-email relations. Messages from trusted domains such as the "columbia.edu" domain constitute 66.5 percent of all incoming messages for university email accounts while they are only 6.7 percent for free email accounts. This is because university email accounts, unlike free email accounts, are used for internal communication within university domains and also within professional communities. Regarding the difference in messages related to Web, we presume that this is also caused by the difference in the usage patterns of email accounts. Compared to university email accounts, the email addresses of free email accounts tend to be more easily given to the users of online services and to receive newsletters or purchase confirmations. This contrast, therefore, indicates that the effectiveness of filtering systems using cross-media relations highly depends on the usage patterns of user accounts.

Yet after using cross-media relations, a small fraction of good messages remained unlabeled. 1.6 percent of messages for university email accounts are triggered by the fact the participants are well-known (found in the public profile group) and 0.3 percent of messages remain unlabeled. This indicates the limitation of our proposed filtering mechanisms. The fraction of messages in the public profile group would be larger if the participant group included more professors, who have typically higher public profile than students. We have observed that these messages are often from other lesser-known members belonging to the same professional community. Thus, it would be helpful in identifying valid messages if we have a query mechanism about sender references, especially membership in a professional organization like IEEE. This motivates our second approach using caller attributes described in Chapter 8.

To find out a distinguishing feature between unlabeled and spam messages, we examined them how many fraction of these messages were sent to subaddressing or hidden destination addresses by using blind carbon copy (BCC), which is usually used for concealing other destinations. However, we found no significant characteristics in the usage of these destination addresses. For both unlabeled and spam groups, the recipient addresses were not explicitly listed in the destination of most messages. No subaddressing was found in the messages in

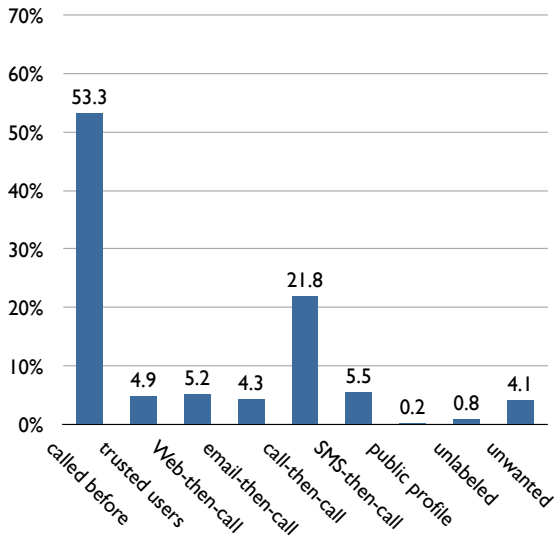


Figure 7.11: Histogram of percentages of incoming messages in 9 groups: Calls (averaged across accounts)

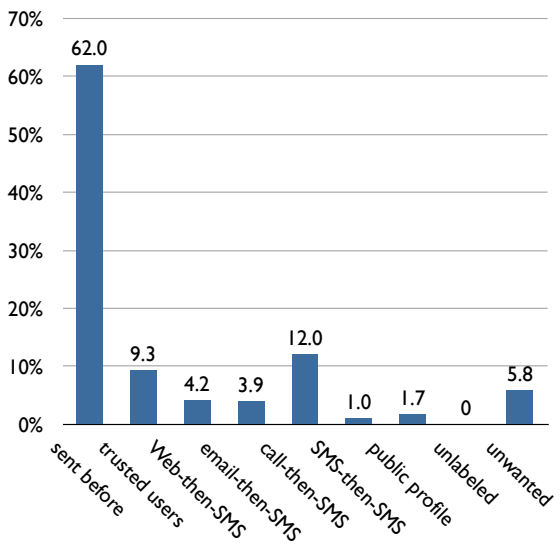


Figure 7.12: Histogram of percentages of incoming messages in 9 groups: SMS (averaged across accounts)

these two groups.

Incoming Calls

Figure 7.11 illustrates the percentages of incoming calls messages received from nine groups. These percentages are averaged across the participants. These bars show that approximately 4 - 5 percent of calls are triggered by cross-media relations, namely, Web-then-call, email-then-call, and SMS-then-call. Similar to the email survey, Appendix B.2 show wide variance across the participants. More than half the participants received no calls having these cross-media relations; thus, the medians of these groups are all zero. Their distribution in the cumulative distribution functions show that the potential effectiveness of using cross-media relations is limited to a quarter to a half of participants; thus, it highly depends on the communication patterns of participants.

Incoming SMS messages

Figure 7.12 indicates that approximately 4 percent of SMS messages are related to Web and email, respectively, whereas 12 percent are related to calls. It is understandable that more SMS messages have call-then-SMS relations than other cross-media relations since both call and SMS are provided to mobile phone users. However, we do not have reasons for higher percentage of messages having call-then-SMS relations (12 percent) compared to SMS-then-call (5.5 percent in Figure 7.11).

In summary, although there remains a small fraction of good messages unlabeled, this study can conclude that using cross-media relations, especially to prior Web transactions, would be effective in helping recipients label good email messages, calls, or SMS messages. In addition to using cross-media relations, a query mechanism about sender references like membership (See Chapter 8) could be another helpful component in identifying good messages.

7.11 Conclusion

This chapter presented our idea, design, implementation, and evaluation of using cross-media relations to identify good calls. They can be also applied to email messages. Using cross-media relations, callees can collect more contact addresses of potential callers and provide evidence of prior contact in order to identify more good calls whether or not they carry a familiar caller ID. Our user study tested the potential effectiveness of our concept using cross-media relations. It demonstrates that approximately 30 percent of email messages, 5 percent of calls, and 4 percent of SMS messages can be identified as good by using the relation to prior Web transactions. These results, thus, indicate that cross-media relations would be useful as an additional component of a call filtering system, although the degree of the usefulness varies across recipients and the usage patterns of accounts.

At the same time, our user study demonstrates the limitation of using cross-media relations. A certain fraction of good messages or calls remain that were not related to prior communication. Chapter 8 proposes a mechanism for validating a sender's or caller's attributes in order to enhance the capability of identifying good communication requests.

Chapter 8

Using User Attributes without User Identity to Identify Good Communication Requests

8.1 Introduction

Chapter 7 has introduced an approach using cross-media relations to identify good[1] communication requests, carrying an unknown origin ID (i.e., caller ID or sender ID). This chapter introduces a different approach, which uses the originator's attribute. When two people do not know each other, ascertaining a person's attributes rather than a person's name is often useful to determine the trustworthiness of the person, as described in Section 6.3. These attributes include, for example, an organizational affiliation, a role in a professional society, age, holding certificates or licenses, and being a customer of a bank. We translate the way of assessing the value of interaction with a stranger in real life to the way of assessing the value of a communication request, whether or not to answer the phone or to open the email message, carrying an unknown origin ID or for which the origin ID is blocked for privacy reasons. Such a call might be unwanted telemarketing or fraudulent, but possibly originates from an unfamiliar colleague. The callee cannot distinguish an unfamiliar colleague from a

[1] See the definition in Section 6.4.

telemarketer or a fraudster before answering the phone using caller-ID-based filtering since a caller ID is not found either a black list or white list, or even using cross-media relations since the callee has had no prior contact with both of them. However, if a caller can provide the callee with the caller's attributes that can be validated by the callee, such attributes would help the callee determine how to handle the call request by estimating whether the call is good enough to be answered. There is evidence of such a situation; one of the findings from our user survey shown in Section 7.10 – a fraction of good message originates from those have had no prior contact but obtained the contact address of the survey participants through their publications or other activities. The caller ID is solely one of the originator's attribute, but it can be helpful only in limited cases described in Section 6.8.

In this chapter, we demonstrate how callees can validate a caller's attributes without necessarily authenticating the caller ID through an attribute validation mechanism that we have designed. Our mechanism introduces an *attribute reference ID (ARID)*, which points to the location of the caller's attributes using an HTTPS URI so that the recipients of an ARID can retrieve them, establishing the validity of the ARID. Unlike existing attribute certificates, assertions, or credentials, such as an X.509 attribute certificate [Farrell *et al.*, 2010], SAML [Cantor *et al.*, 2005], this mechanism is easy deployable since it does not require the callee to authenticate a caller ID or the caller identity (i.e., caller's name), which is an obstacle to service deployment. Furthermore, unlike existing anonymous attribute credentials, such as U-Prove [Brands, 2000], this mechanism is lightweight because it relies on transport layer security, such as TLS, and transitive trust through the issuer of an ARID. Thus, our mechanism provides moderate security that is sufficient for call acceptance decision.

This chapter starts with providing an overview of the service architecture using an example of a SIP call in Section 8.2, pointing out the differences in the trust model between ours and two existing third-party authentication mechanisms such as Kerberos [Neuman *et al.*, 2005] and OAuth [Hammer *et al.*, 2012]. It then describes key design decisions based on a threat model we have mainly considered in Section 8.3. Our design choices mainly aim for moderate security and ease of both development and deployment of the mechanism. This chapter also enumerates the requirements from the perspectives of general, security, and

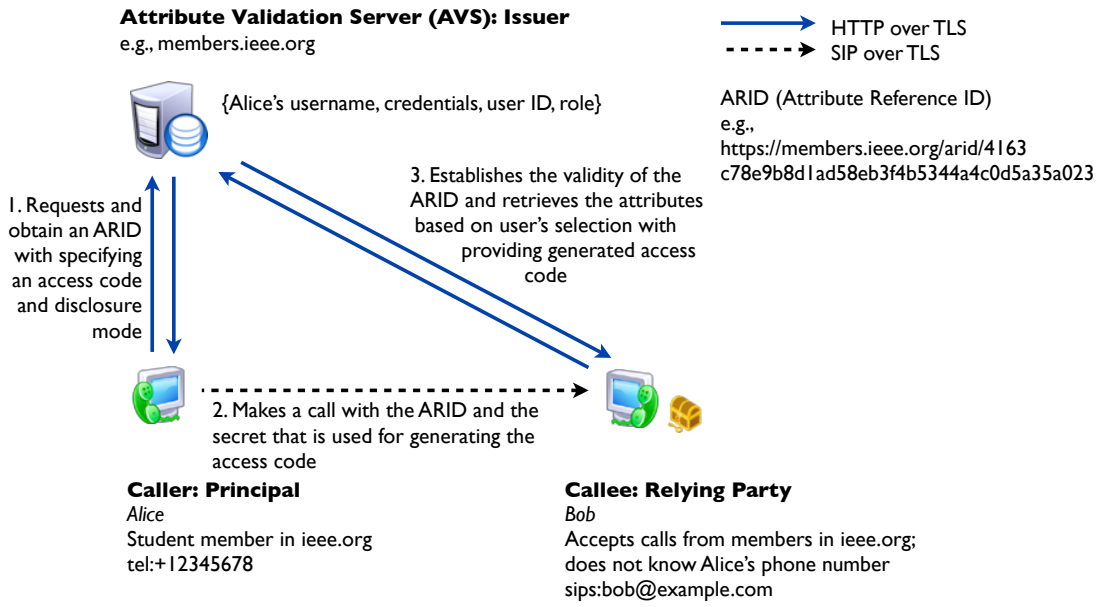


Figure 8.1: AVS service architecture applied to a SIP call

privacy in Section 8.4. It then describes the procedures for a call using SIP in Section 8.5 and security considerations in Section 8.6. Our prototype is described in Section 8.7, followed by the evaluation comparing it to U-Prove [Paquin, 2011], an existing anonymous attribute credentials, in Section 8.8. After reviewing existing attributes validation mechanisms in Section 8.9, we conclude with some remarks in Section 8.10.

8.2 Service Architecture

This section describes three entities in our mechanism of attribute validation service and the basic operations using a simple example.

Figure 8.1 shows an overview of the service architecture where Alice makes a call to Bob using SIP and provides him with her attributes through an attribute validation server (AVS). In this example, Alice is a student member in the organization, "members.ieee.org." When making a call, Alice wants to provide Bob at "sips:bob@example.com" with her attribute being a student member at ieee.org in order to raise the possibility of being answered.

To allow Alice to deliver her attributes to Bob, our mechanism introduces an attribute reference ID (ARID), which is an HTTPS URI [Berners-Lee *et al.*, 2005] that points to the

location of her attributes on the AVS.

8.2.1 Entities

We define the AVS, the originator of a communication, caller or sender (Alice), and the recipient or callee (Bob) as an issuer, a principal, and a relying party, respectively.

Attribute Validation Server (AVS): Issuer

An Attribute Validation Server (AVS) provides attribute validation services for users in an organization. The AVS performs the following three tasks: maintaining user attributes (Alice's attributes), issuing an ARID for the user (= Alice as the caller), and validating the ARID upon the callee's request (Bob). The AVS is the *issuer* of an ARID.

For each user in an organization, the AVS maintains the user attributes, in a database, that are assigned by the organization. For example, the user attribute include a user ID and a role at an organization. The AVS also maintains the username and credentials to authenticate users who attempt to access the AVS.

Upon receiving a request from a user of an organization who plays the role of a caller (Alice), the AVS issues an ARID, which is an HTTPS URI pointing to its server location where the callee (Bob) can retrieve the user attributes.

Upon receiving a query about the caller's attributes using an ARID, the AVS determines whether the querier or the callee (Bob) is someone who has been authorized by the principal of the ARID. The AVS also ensures that the ARID is not stale when receiving the query.

Once the AVS confirms the validity of the query and the ARID, it looks up the user ID of the principal on the database and determines which set of attributes the principal has authorized the relying party to retrieve from the database. The AVS then responds to the relying party with the principal's attributes.

Originator, Caller, or Sender: Principal

The originator of a communication request, specifically the caller of a call or the sender of an email message, who is a user of the services provided by an organization is called a *principal*. The principal (Alice) requests an ARID from the AVS in order to delivering her

attributes to the callee. When requesting an ARID from the AVS, the principal specifies which attributes to be disclosed and to whom. To restrict the recipients of her attributes to the callee, the principal also provides the AVS with an ARID access code that is the hash of the callee's communication endpoint ID (CEID).[2]

Recipient or Callee: Relying Party

The recipient of a communication request, specifically the callee of a call, who receives an ARID in a communication request assesses the request based on the information that is retrieved using the ARID. The recipient or callee is called a *relying party*. The relying party of an ARID (Bob) sends the AVS a query about the principal's attributes and establish its validity with the AVS. In this context, the relying party is a querier and a verifier. To be identified by the AVS as its authorized party, the relying party sends an ARID access code, the hash of his CEID found as the destination ID in the communication request.

8.2.2 Trust Relationships among Issuer, Principal, and Relying Party

The principal of an ARID and its issuer are assumed to trust each other regarding the attribute validation service for an organization. They share Alice's username and credentials, which are used for authenticating her, and her attributes, such as a user ID and a role. Alice trusts the issuer to properly maintain her information and to disclose the attributes she selects only to relying parties whom she specifies. In turn, the issuer trusts Alice as a user in the organization and trusts her attributes which it knows first-hand, such as "Alice is an IEEE student member."

It is worth noting that the issuer may store the principal's electronic contact addresses, or CEIDs, such as her email address and a SIP address of record (AoR), without ensuring the authenticity. Unless they are assigned by the organization, these CEIDs should not been treated as the users attributes. This is due to the fact that the issuer neither guarantees that the user owns these CEIDs, nor that the user can be reached by any of these CEIDs.

We also assume that a relying party, Bob, knows "members.ieee.org" as the domain name of an organization that has a user account management policy he trusts, whether

[2]See the definition in Section 6.2

or not he belongs to the organization. Bob also trusts the issuer to properly perform its attribute validation service. However, the issuer does not know anything about Bob.

It is worth noting that the relationship between the issuer and a relying party differs from other third-party authentication systems using a key shared between them such as Kerberos, which is the most common mechanism of third-party authentication, or OAuth, which has recently emerged as a third-party authentication mechanism especially across Web servers. These differences will be further described later in Section 8.9.

Alice finds Bob worth making a call or sending an email message and disclosing her attributes to establish a communication with him. In turn, Bob does not have sufficient information about Alice's trustworthiness based solely on her identity indicated by her CEID in a SIP communication request or in an email message.

8.2.3 Basic Operations using a SIP Call

We describe how this mechanism works when Alice makes a SIP call to Bob using the example described in Figure 7.1. Alice first requests an ARID from the AVS, specifying which set of her attributes she allows to disclose and what data the relying party has to provide for authorization. The request is sent using HTTP [Fielding *et al.*, 1999] over TLS [Dierks and Rescorla, 2008]. When sending the request, Alice authenticates the AVS using its X.509 Public Key Certificate (PKC) [Cooper *et al.*, 2008] which is delivered during the TLS handshake. Alice checks if the PKC is signed by a trusted certificate authority (CA) and contains the server name that she intends to connect. In addition, she checks if the PKC is neither expired nor revoked. In turn, when generating an ARID for Alice, the AVS authenticates her using any credentials supported by the AVS, such as a password or a client's X.509 PKC.

Having successfully obtained an ARID, which is an HTTPS URI, Alice makes a call to Bob specifying the ARID to be included in the Sender-References header field [Ono and Schulzrinne, 2009b] of the SIP INVITE request over TLS.

When Bob receives the SIP INVITE request containing an ARID and wants to examine the attributes of the caller in order to determine whether to answer the call, he proceeds to establish the validity of the ARID and the corresponding attributes to the AVS. To be

identified by the AVS as an authorized party by the principal, Bob generates an ARID access code by computing the hash of his contact address (found in the To header field) with the random string using the hash algorithm (found in Sender-References header field) all were included in the SIP INVITE request. He then adds the hash to the ARID, an HTTPS URI, as an additional path. He sends a validation request to the ARID by connecting to the newly created HTTPS URI. At this time, Bob verifies that he is communicating with the desired AVS using its X.509 PKC in the same way that Alice did earlier.

The AVS validates the ARID by ensuring that it has not been expired and that the query has been authorized by the principal. If the validation is successful, the AVS responds to the querier with Alice's attributes that she selected to disclose when she obtained the ARID. Based on the attributes retrieved, Bob determines whether or not to answer the call from Alice and adjusts his communication stance accordingly.

8.3 Key Design Decisions

This section focuses two key design decisions over our proposed mechanism to serve purposes – moderate security for low risk interactions and ease of service development and deployment. (Other requirements are described later in Section 8.4.) First, an ARID is loosely associated with the CEID (e.g., a SIP AoR or email address) of the principal in order to avoid practical problems, such as difficulties in service development caused by the limitations of caller ID authentication described in Section 6.6. Second, the hash of a relying party's CEID, not a cryptographically random string, is used for authorizing the relying party. Using hashed relying party CEID enables the relying party to ensure that the ARID is meant for herself, as a countermeasure to a form of replay attacks by legitimate recipients. This section starts with describing the threat we considered.

8.3.1 Threat: Impersonating Principal Forwarding a Received ARID

The primary threat model considered for our mechanism is the scenario where a legitimate recipient of an ARID sends a new communication request in which it purports to have the attributes of the original principal. For example, assume that Alice makes a SIP call to

Bob. Alice sends Bob a SIP INVITE request including a valid ARID and its access code so that Bob can retrieve her attributes from the AVS. Whether he has retrieved her attributes from the AVS or not, Bob places a call by sending another SIP INVITE request containing Alice's ARID and its access code to Carol. Consequently, Carol mistakenly accepts the call from Bob and communicates with him, assuming the holder of Alice's attributes.

When we design the countermeasures to such an attack, it is important to keep in mind that the attacker has been provided a fresh and unused ARID and the related information – the parameters used for generating an ARID access code – from the communication request sent by the valid principal. Thus, binding the ARID or its access code to the identifier of the communication request (e.g., the message ID of an email message) is ineffective. Furthermore, a simple countermeasure such as limiting the lifetime or use time of an ARID and its access code is ineffective if it is solely applied. Both or either of them is effective in enhancing security when it is applied in conjuncture with other countermeasures. In our mechanism, limiting an ARID's lifetime to a short time period is used as a precaution especially against another form of replay attacks using an eavesdropped validation request to the AVS.

There are two possible countermeasures to the attack by forwarding a received ARID. The relying party of a forwarded ARID (i.e., Carol in the example above) ensures the validity of the ARID and its access code by performing either of the following two tests (If both are available, performing both tests would provide strong security.) The first countermeasure is to test whether the sender of the ARID (Bob) is the valid principal. To do so, the relying party needs to authenticate the CEID and to ensure that the CEID is one of attributes retrieved using the ARID. However, these procedures encounter practical problems, such as no availability of mechanism for authenticating a SIP AoR in the tel URI, as described in Section 6.6. This drawback is behind our decision not to take this option, rather to design an ARID to be loosely associated with the principal's CEID, as described below in Section 8.3.2. On the other hand, the second countermeasure is to test whether the relying party (Carol) is authorized by the valid principal of a forwarded ARID (Alice) to disclose the principal's attributes. To enable this test, the principal requests an ARID with the hash of a desired principal's CEID as an ARID access code, as described below in Section 8.3.3.

8.3.2 ARID: Loosely Associated with the CEID

The first countermeasure to the attack described in Section 8.3.1, is to enable the relying party (Carol) to ensure that the sender of the ARID (Bob) is the valid principal. It requires ensure that the ARID is associated with the sender's CEID, but our mechanism avoids applying this countermeasure for the following practical reasons. Primarily, the principal's attributes usually do not include her CEID since the CEID is issued by a communication service provider, which does not always work in alliances with the issuer of the ARID. Additionally, no mechanism for authenticating a SIP AoR is available in the case where the SIP AoR is in the SIP URI issued by the domain which does not deploy the SIP identity mechanism, where the SIP AoR is in the tel URI which is sent without any other authentication mechanisms, such as a digital signature in S/MIME [Ramsdell and Turner, 2010], or where the SIP-AoR is anonymized, as described in Section 6.6. Therefore, tightening linkage between the principal's CEID and attributes makes the service deployment more difficult and limits the principal's choice of CEID. Since this does not serve our purpose, ease of service deployment, an ARID is loosely associated with the CEID, just by being sent in the same communication request over TLS.

8.3.3 ARID Access Code: Hash of Relying Party's CEID

As described in Section 8.3.1, another possible solution to detecting the attack attempting to impersonate the principal of a received ARID is to ensure that a relying party (Carol) is authorized by the principal (Alice) to disclose the principal's attributes. It is essential to design an ARID access code which is used for authorizing a relying party – which is provided to both the AVS and the relying party by the principal, and used by the AVS for validating a query from a relying party. Unlike the first countermeasure, which suffers from limited availability of caller ID authentication, this solution can employ the relying party's CEID without any difficulties because every party can authenticate his own CEID. To preserve the privacy of the communication history on the AVS, the hash of the relying party's CEID, which is typically set in the `To` header field of a communication request, is used as the ARID access code. Furthermore, To prevent re-identification based on the hash of the CEID collected on the AVS, the ARID access code is generated HMAC-SHA1, for

example, with a secret key, which is a random string of characters, for each communication request. Therefore, the relying party needs to calculate the hash of her CEID with the secret key using the hash algorithm that all are conveyed by a communication request over TLS.

8.4 Requirements

This section lays out a set of requirements for a mechanism for validating the originator's attributes in order for recipients to identify a good communication request. We simply call the originator's attributes user attributes.

Our design decisions are associated with these requirements. They are categorized into three groups: general, security, and privacy.

8.4.1 General Requirements

We have the following general requirements for our mechanism to identify a good communication request, especially a VoIP call, offering real-time communication.

GEN-REQ-1: The intension of providing user attributes must be presented in the header of a signaling message. To easily filter communication requests without needing to inspect the body of a signaling message, the intension of providing user attributes must be set in the message header.

For this purpose, our mechanism sets an ARID in an HTTPS URI pointing to the issuer of user attributes in the Sender-Reference header field.

GEN-REQ-2: A single message may contain attributes issued by multiple issuers. To allow the principal (i.e., a caller or a sender) to prove as many attributes as possible to be identified as a good communication request, the mechanism should enable user attributes issued by one or more organizations to set – by value or by reference – in a single communication request.

GEN-REQ-3: Supporting short-lived attributes. The mechanism needs to support both temporary and persistent attributes. Examples of temporary attributes include being a visitor for a business meeting or an attendant for a one-day workshop.

GEN-REQ-4: Easy validation of attributes by recipients. Given that an ARID supports short-lifetime attributes, recipients need to validate an ARID by connecting to the attribute validation server online, rather than by checking a long revocation list of ARIDs offline.

GEN-REQ-5: Easy adaptation to existing communication protocols. To minimize modifications to existing communication protocols such as SMTP and SIP, this mechanism needs to employ only the first message from the principal to a relying party.

GEN-REQ-6: Easy deployment without special security capabilities. To easily deploy this attribute validation service, this mechanism leverages the wide deployment of HTTP servers secured by TLS, requiring no additional security capabilities such as the principal's X.509 public key certificate (PKC).

GEN-REQ-7: Flexible allocation of issuer and validation functions. To support flexible deployment of validating attribute services, the issuer and validation functions should be flexibly allocated, namely, both are co-located or separately located.

8.4.2 Security Requirements

Given that our target service is to identify good communication requests, which is low-risk interactions, we analyze our security requirements as follows:

SEC-REQ-1: Data confidentiality. When transmitting user attributes and the information which is used to retrieve user attributes (e.g., an ARID access code) over networks, all entities must use a secure channel to prevent eavesdropping and replay attacks. Protection the confidentiality of user attributes

and all related information is not necessary at trusted intermediaries such as SIP or SMTP servers.

SEC-REQ-2: Data integrity. Similar to data confidentiality, when transmitting user attributes and all related information over networks, all entities must use a secure channel, such as TLS, to prevent message tampering.

If a relying party is received user attributes not directly from the issuer, but via the principal, these user attributes needs to be signed by the issuer. This application-level signature requires verification by the application, sometimes leading to complexity in parsing multipart message bodies in SIP and email. Our mechanism intends to avoid this complexity, by sending user attributes by reference so that a relying party can retrieve user attributes directly from the issuer.

SEC-REQ-3: Issuer authentication. When requesting and generating an attribute credential, the principal needs to authenticate the issuer with its X.509 PKC or pre-shared credentials. In addition, a relying party needs to authenticate the issuer by the issuer's X.509 PKC or other mechanisms.

In our mechanism, the issuer authentication is performed with its X.509 PKC during the TLS handshake by the principal when it requests an ARID and by a relying party when it establishes the validity of the ARID.

SEC-REQ-4: Limiting relying parties To reduce the effect of replaying attacks (Section 8.6.2), this mechanism needs to enable the principal to limiting relying parties who can retrieve the principal's attributes. If the issuer validates an attribute credential (e.g., AVS), given no mutual trust relationship between the issuer and a relying party, the issuer needs to authenticate a relying party as someone whom the principal has authorized. If a relying party validates an attribute credential without accessing the issuer (e.g., U-Prove), the relying party himself needs to detect replay attacks, namely, to determine whether an attribute credential is meant to himself or someone else.

For this purpose, in our mechanism, when receiving a validation request, the issuer of an ARID needs to authenticate a relying party with the access code associated with the ARID, provided by the principal. The access code is not the relying party's CEID since the issuer cannot authenticate it. However, each relying party needs to generate the access code by computing the hash of his CEID and a secret sent from the principal, as described in Section 8.3.3.

SEC-REQ-5: Proof of possession of an attribute credential. To prevent replay attacks using received or stolen attribute credentials (Section 8.6.2), the principal needs to prove a relying party that she possesses an attribute credential (an ARID). Given no requirements of the principal's CEID authentication below and the requirement of ease of deployment, this proof needs to be performed without authenticating her CEID.

This requirement contraint is relaxed by one of the requirements excluded because of the low risk of our target service. A described below, this does not have to prevent transfer or delegation of an attribute credential.

We exclude the following requirements since this mechanism is used for protecting a relatively low-risk interaction.

- **No need for user accountability to the relying party, namely, no user authentication.** A relying party needs to know user attributes as a hint to estimate the goodness of a communication request. Thus, a relying party does not need to trace uniquely to the principal, which is called the principal's accountability [Kissel, 2011]. For this reason, authenticating an origin ID of a communication request is not needed [Kent and Millett, 2003].

- **No need for non-repudiation of using an attribute credential.** According to the term definition in [Kissel, 2011], "non-repudiation" is the security service that a sending entity cannot deny having sent a message (non-repudiation with proof of origin), and the receiving entity cannot deny having received a message (non-

repudiation with proof of delivery). For our target service, neither non-repudiation with proof of origin nor delivery is needed.

- **No need to prevent transfer or delegation of an attribute credential.** Due to the low risk of our target service, the mechanism does not need to support any special features to prevent the principal from giving her ARID to others. In addition, this mechanism does not support the principal who delegates her ARID, namely, allows another party to send a communication request using her ARID, with proving the chain of authorizing delegation. If the principal wants to delegate her ARID to her assistant or friend, she can do that at her own risk. This conflicts SEC-REQ-5.

- **No need for binding an attribute credential to the signaling path of a communication.** An ARID is used end-to-end, namely between an originator and the recipient of a communication request while its signaling path is established hop-by-hop through email servers or SIP proxy servers. Since binding an ARID to the signaling path does not enhance any security features, this mechanism does not need such binding.

- **No need for binding an attribute credential to a CEID.** An email message has a global unique ID in the Message-ID header field and a SIP session can be identified by the set of values from the Call-ID header field, the From header field and its tag, and To header field and its tag. Embedding such a unique ID for a communication request into an ARID does not help to prevent replay attacks. This is due to the fact that when an ARID is stolen or received, it is likely that the communication request message itself is also stolen or received (e.g., an attack by forwarding a received ARID). Thus, this binding is useless for enhancing security. Instead, such a binding would conflict with the privacy requirement, by enabling to link between an ARID and a communication request.

8.4.3 Privacy Requirements

Among the security requirements above, data confidentiality and restricting relying parties, overlap with privacy properties. The remaining privacy requirements are as follows:

PRIV-REQ-1: Untraceability of communication history by issuer. The mechanism must allow a principal to specify relying parties without revealing their CEIDs to the issuer. The mechanism should minimize the amount of communication history collected by the issuer.

This requirement contraint is relaxed by one of the requirement excluded because we value the ease of deployment over further privacy enhancement. As described below, there is no need for untraceability of the use of attribute credentials by issuer.

PRIV-REQ-2: Selective disclosure of attributes. The mechanism must allow the principal to specify the set of her attributes that she wants to disclose for each communication session.

We intentionally omit the following requirements since we prioritize the ease of deployment over further privacy enhancement.

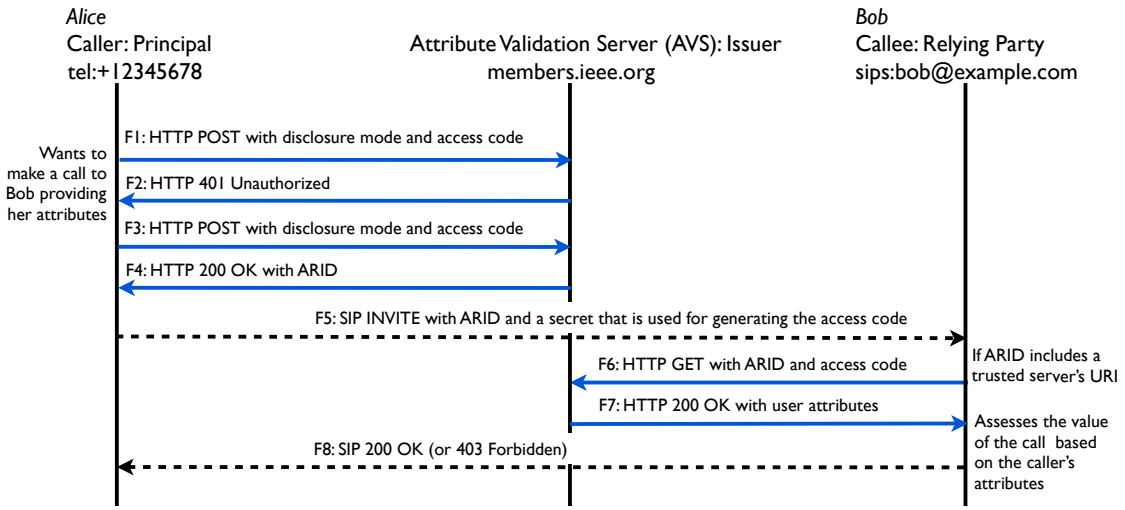
- **No need for untraceability of the use of attribute credentials by issuer.** The mechanism should minimize the principal's activity, for example, when an attribute credential is used, revealed to the issuer, but not prioritize over the ease of deployment. This relaxes the constraint of PRIV-REQ-1.

  In our service architecture using online validation of ARIDs on the issuer or the AVS, the AVS can collect and trace whether or not an ARID is used, and if so, when and from which IP address.

- **No need for unlinkability between issuance and use of an attribute credential.** This privacy property is part of untraceability of the user of attribute credentials considering the scenario where the issuer and a relying party are the same entity.

  As described above, as long as this mechanism offer online validation of ARIDs on the issuer or the AVS to the relying parties, the AVS can link the issuance of an ARID and its use for activity analysis purposes, for example.

- **No need for unlinkability between multiple uses of an attribute credential.**



Note that SIP proxy servers and messages to and from them are omitted since they are not affected by this mechanism.

Figure 8.2: Message exchanges for validating the caller's attributes using ARID

This privacy property means that a relying party cannot link among two or more uses of an attribute credential by the principal.

Given that an ARID is designed for the relying party (callee) to filter communication requests based on the attributes of the principal (caller), this privacy requirement is meaningless.

8.5 Procedures for a Call using SIP

Figure 8.2 illustrates message exchanges among a caller acting as the principal, the callee playing the role a relying party and the AVS (i.e., the issuer) for the following procedures:

- Obtaining an ARID (messages F1 - F4);
- Sending the ARID to the callee when making a call using SIP (message F5);
- Validating the ARID to retrieve user attributes (messages F6 and F7);
- Responding to the call request (message F8).

Before explaining each procedure, we describe how the AVS typically generates an ARID. We then explain the first three procedures mainly for a call using SIP since the last procedure, responding to the call request includes no ARID related information and requires no special considerations.

8.5.1 Generating an ARID

An ARID is an HTTPS URI [Berners-Lee *et al.*, 2005] pointing the location of user attributes on a server. An ARID is generated by the AVS upon the principal's request. To look up a specific set of user attributes on a database, an ARID needs to be a unique identifier within the AVS. Additionally, it needs to preserve the principal's privacy, such as a user ID of the AVS, since the ARID is transmitted remote parties. There are many ways to generate such a unique identifier, including using a random string. We give two examples in which an AVS generates an ARID using hashing and using encryption. In the first example, the AVS generates an ARID by computing the hash of a unique string of characters within the AVS using SHA1 and adding the path in the location of the server managing the attributes (e.g., `HTTPS URI path/H(counter||timestamp)`).

In the second example of generating an ARID, the AVS encrypts a string of characters with a symmetric key of the AVS using AES [AES, 2001], and the string of characters, represented by m, is a user ID concatenated with a disclosure mode, the expiry time, a salt, and an ARID access code as follows:

$$ARID = URL\ path/Encrypt(m)$$

$$m = user\ ID||disclosure\ mode||expiry\ time||salt||ARID\ access\ code$$

The disclosure mode is provided by a principal to specify an ARID access code to authenticate a relying party and which set of attributes to be disclosed to someone having the ARID access code. (PRIV-REQ-2).

Hashed relying party's SIP AoR is also provided by the principal as an ARID access code for desired relying parties. The AVS set the ARID's expiry time so that it expires shortly after its issuance, such as ten minutes later for a call, to avoid replay attacks (SEC-REQ-5). Since an approximate lifetime of an ARID depends on the communication mode

– synchronous or asynchronous –, its lifetime for email communication should be longer. Thus, the AVS needs to determine the lifetime of an ARID depending on the communication mode that is provided by the principal.

When selecting a method for generating an ARID, using hashing or encryption, there is the trade-off between the memory cost of storing ARIDs with related data in a database and the computational cost of decrypting ARIDs. When generating an ARID using hashing, the AVS needs to store the generated ARID with associated data including the expiry time, the hashed SIP AoRs of desired relying parties which the principal sent, and the disclosure mode which the principal specified. On the other hand, when generating an ARID using encryption, the AVS only needs to remember a decryption key and the salt it included in a string, but no ARIDs or related data. However, the ARID is longer and requires the computational cost of decryption for all query messages in order to determine whether they are still valid or have been expired. Our implementation described in Section 8.7 has adopted a method for generating an ARID using hashing to avoid wasting the computational cost of decrypting invalid query messages.

8.5.2 Obtaining an ARID

When Alice wants to send some of her attributes to send in a communication request to Bob, she first needs to obtain an ARID from the AVS associated with the attributes by using a SIP UA that supports this mechanism. To obtain an ARID from the AVS, Alice needs to specify the destination ID of the communication request, namely, Bob's CEID, and which set of attributes to be disclosed, called the disclosure mode. Once she specifies both information, the SIP UA sends an an HTTP POST over TLS to request an ARID to the AVS. When connecting, the SIP UA authenticates the AVS using its X.509 PKC received in the TLS handshake. In turn, the AVS authenticates Alice using her username and credentials. For user authentication, HTTP Basic or Digest authentication [Franks *et al.*, 1999], a client's PKC, or other mechanisms could be used.

To generate an HTTP POST request, the SIP UA sets the communication_mode object to the communication type (e.g.,"real-time") and the disclosure_mode object to her selected disclosure mode (e.g., "basic") as JSON [ECMA, 1999] objects in a message body, as shown

```
POST /requestARID HTTP/1.1
HOST:members.ieee.org
Content-Type:application/json

{"communication_mode":"real-time",
"destination":"af8185ae01b4c9d81e125b408f86fd4c9a0eefc5",
"disclosure_mode":"basic"}
```

Note that mandatory HTTP and SIP headers unrelated to this mechanism are not shown here and the following example messages.

Figure 8.3: Messages F1 and F3. HTTP POST sent from Alice to AVS

```
"destination_list":[
"destination":"af8185ae01b4c9d81e125b408f86fd4c9a0eefc5",
"destination":"20d9834e7477647ef4a32671e9f1cb69b0b9db01"]
```

Figure 8.4: Destinations in a destination list

in Figure 8.3. The format could be in XML [Bray *et al.*, 2004]. In addition, the SIP UA sets the destination object to an access code, which is used for limiting relying parties who can retrieve her attributes using the ARID, as described in Section 8.3.3.

To preserve Alice's communication history or plan (PRIV-REQ-1) and to prevent re-identifying Bob based on the access codes collected on the AVS, the SIP UA generates an ARID access code by calculating the HMAC [Krawczyk *et al.*, 1997] of Bob's destination ID with a secret key, which is a random string of characters that the SIP UA generates for the call. (The SIP UA sends this secret key, not the HMAC result, to Bob's destination ID in a SIP INVITE request later described in Section 8.5.3.) The SIP UA then sends the AVS an HTTP POST request including It is worth noting that the SIP UA does not include the secret key used for generating the access code into the HTTP POST request.

If a communication request can have multiple destinations, such as an email message, destination_list field should include a set of destination objects in a JSON array to contain multiple access codes, as shown in Figure 8.4. The SIP UA or an email client needs to

```
HTTP/1.1 200 OK
Content-Type:application/json

{"arid":"https://members.ieee.org/arid/4163c78e9b8d1ad58eb3f4b5344a4c0d5\
a35a023","expires":"2011-08-24T16:20:20Z" }
```

Figure 8.5: Message F4. HTTP 200 OK sent from AVS to Alice

generate each access code by hashing a destination ID.

When the AVS successfully generates an ARID for Alice, the AVS responds to her with an HTTP 200 OK response including the ARID and its expiry time in the same data format used in the HTTP request. As seen in Figure 8.5, the ARID is set in the arid JSON object. The expires object contains the expiry time of the ARID in the universal time, coordinated (UTC) format [Klyne and Newman, 2002].

8.5.3 Sending an ARID in a SIP INVITE Request

When Alice makes a call to Bob with an ARID, she needs to specify the information corresponding to the ARID in a SIP UA so that Bob retrieves her attributes using the ARID. The SIP UA generates a new SIP header field called Sender-Reference [Ono and Schulzrinne, 2011a] including the sender-ref, type, expires, secret, and hash_alg parameters to convey the ARID, the service name of this attribute validation service, the expiry time of the ARID, and the secret and the hash algorithm which were used for calculating HMAC over the relying party's AoR described in Section 8.5.2, respectively. The SIP UA then sends an SIP INVITE request including the Sender-Reference, as shown in Figure 8.6. If Alice wants to specify multiple ARIDs, the Sender-References header field includes multiple set of an ARID and related parameters concatenated by a comma separator. The INVITE request must be sent over TLS to protect message confidentiality and integrity.

Unlike email or instant messaging, a SIP INVITE request cannot contain multiple destinations for one-to-many communications, although the call request can initiate multiple party communications using a proxy server or a conference server. In these cases, relying

```
INVITE sips:bob@example.com SIP/2.0
From:Alice<tel:+12345678>
To:Bob<sips:bob@example.com>
Sender-References:<https://members.ieee.org/arid/4163c78e9b8d1ad58eb3f4b\
5344a4c0d5a35a023>;type="avs";expires="2011-08-24T16:20:20Z";
hash_alg="HMAC-SHA1";secret="tqJrcYI7"
```

Figure 8.6: Message F5. SIP INVITE from Alice to Bob

parties share a single SIP AoR, whether or not a principal knows that the AoR is shared by multiple users. Any of these relying parties can validate the ARID and retrieve user attributes before its expiry time, causing privacy concerns. This privacy problem is technically unavoidable for any attribute mechanisms where others can retrieve the attributes and distribute them.

How to convey an ARID: A new header vs. new parameters of an existing header

Other than defining a new Sender-References header field, there are two options to convey an ARID in a SIP INVITE request: new parameters of the Call-Info header field or new parameters of the From header field.

The existing Call-Info header field, available only for SIP, can contain an ARID by defining a new value of the purpose parameter, such as sender-attributes. However, to convey a secret and the hash algorithm, the header field needs two more parameters. In addition to this potential complexity of the Call-Info header filed structure, using this Call-Info header field has another drawback. This header is unavailable for the email header fields. Because of these two drawbacks, we have decided not to modify the Call-info header field to convey an ARID.

Another option is to add parameters in the URI of the From header field, similar to the SIP SAML solution (e.g., `tel:+12345678;token-info=https://example.com/assns/?ID=abcde`) [Tschofenig *et al.*, 2010]. The advantage of using the From header field is that

protecting the data integrity is guaranteed when a domain-level authentication such as the SIP identity mechanism or DKIM for email is applied. However, the tel URI for SIP cannot take advantage of this mechanism since the tel URI contains no domain name. In addition, the use of the From header field could potentially confuse recipients since user attributes are not tightly linked to the From header field.

Even though protecting a new header field is not guaranteed, DKIM, unlike the SIP Identity mechanism, allows the signer to easily add a header field for the signature. The signer can add the name of a header field into the h= tag of the DKIM-Signature header field. To deal with the use of the tel URI and to avoid potential confusion, we thus rather define a new header field.

8.5.4 Validating an ARID to Retrieve User Attributes

When Bob, the relying party of one or more ARIDs, wants to retrieve the caller's attributes, the SIP UAS needs to test the validity of the ARIDs on the corresponding AVSes based on the information found in the Sender-References header field in received SIP INVITE request. The SIP UAS first needs to check if the expiry time in the expires parameter is not stale and if it supports a hash algorithm specified in the hash_alg parameter. The SIP UAS then needs to determine whether or not it trusts each domain name of the AVS by prompting Bob or based on his preconfigured information. Only for trusted AVSes, the SIP UAS looks up the ARIDs on the corresponding AVSes to retrieve the caller's attributes by using an HTTP GET request, as shown in the following example. HTTP messages must be sent over TLS for security as well as messages between the SIP UAC and the AVS. Bob authenticates the AVS using its X.509 PKC delivered in the TLS handshake.

For this validation, the SIP UAS needs to generate a new HTTPS URI by adding an ARID access code to the path of the ARID found in the Sender-References header field and sends HTTP GET request to the newly created HTTPS URI. To do so, the SIP UAS first needs to generate the ARID access code by computing the hash of the secret parameter found in the Sender-Reference header field and its SIP AoR specified as the destination ID found in the To header field using the hash algorithm found in the hash_alg parameter of the Sender-Reference header field, `HMAC-SHA1("tqJrcYI7","sips:bob@example.com")`,

```
GET /arid/4163c78e9b8d1ad58eb3f4b5344a4c0d5a35a023/af8185ae01b4c9d81e125b\
408f86fd4c9a0eefc5 HTTP/1.1
HOST:members.ieee.org
```

Figure 8.7: Message F6. HTTP GET from Bob to AVS

```
HTTP/1.1 200 OK
Content-Type:application/json

{ "user_status":"student member" }
```

Figure 8.8: Message F7. HTTP 200 OK from AVS to Bob

getting `"af8185ae01b4c9d81e125b408f86fd4c9a0eefc5."` Once it computed the hash of the relying party's AoR, the SIP UAS finally generates the new HTTPS URI by adding it to the path of the ARID, as shown in Figure 8.7.

If Bob enables call forwarding services, the To header may not include the destination ID that Alice specified and used for generating an ARID access code. In this case, he needs to find the original destination ID in the forwarding-related header fileld or in his configuration. As long as a relying party can find the original destination ID in a communication request, this mechanism can work with any forwarding services.

If the SIP UAS finds any deficiencies in the parameters of the Sender-Reference header field Bob trusts no AVSes in these ARIDs, the SIP UAS must stop any further validation process and continue normal processing of the SIP request.

If the AVS finds that the ARID is not stale and its access code (the hash of relying party's AoR) is valid, it responds to the relying party with an HTTP 200 OK having the attributes in the message body, based on the disclosure mode which Alice has specified, as shown in Figure 8.8. The attributes are attached as a JSON object or in XML. If the ARID is expired or received access code is invalid, the AVS responds with an HTTP 404 Not Found response. Specifically the invalidity is caused by a query received after the expiry time, the AVS may respond with 410 Gone in HTTP.

If Bob receives an HTTP 200 OK response from the AVS, he is informed that the ARID is valid and the principal has the attributes set in the message body, for the example of Figure 8.8, the caller is a student member in "members.ieee.org." With any other responses, Bob can know nothing about the caller's attributes. Based on this information, he determines whether or not to answer the call and adjusts his communication stance accordingly.

8.6 Security Threats and Countermeasures

To analyze vulnerability our mechanism, this section describes security considerations including potential threats and possible countermeasures.

8.6.1 Man in the Middle Attacks

In a man in the middle attack, the attacker imposes himself in the path of signaling or message exchange between entities for impersonation, eavesdropping, and message tampering [Kissel, 2011]. This mechanism needs to protect all three parts of signaling or message path, between a principal and the AVS using HTTP, between the principal and a relying party using SIP or email protocols, and between the relying party and the AVS using HTTP.

To prevent eavesdropping and to detect message tampering, all parts of message path must be protected using TLS.

To prevent an attacker from impersonating a principal to the AVS, the AVS must authenticate the principal using HTTP Basic or Digest authentication over TLS, a client X.509 PKC or other mechanisms. In turn, to prevent impersonating the AVS to the principal, the principal must authenticate the AVS using its X.509 PKC in the TLS handshake. Similarly, to prevent impersonating the AVS to a relying party, the relying party must authenticate the AVS using its X.509 PKC in the TLS handshake. Neither user authentication between the principal and a relying party nor relying party authentication by the AVS are not needed since no trust relationships are assumed, as described in Section 8.2.2.

8.6.2 Replay Attacks

In a replay attack, the attacker obtains a valid ARID in some way and uses the ARID to spoof user attributes to a relying party, or to retrieve user attributes from the AVS. A replay attack potentially occurs from a compromised intermediary along the signaling or message path, such as an HTTP or SIP proxy server or an SMTP server. We first describe countermeasures to replay attacks from outsiders, and then analyze the vulnerability to replay attacks from compromised or malicious intermediaries. Another form of replay attacks occurs from a legitimate relying party by using a received ARID, as described in Section 8.3.1.

8.6.2.1 Replay Attacks from Outsiders

To prevent unauthorized retrieval of user attributes using a stolen ARID, the AVS must restrict relying parties who can use a specific ARID based on the information the principal specifies (an ARID access code). The information is HMAC over the SIP AoR or email address of a relying party; thus, SIP UAs or email clients must support HMAC-SHA1.

If an ARID is stolen with the information, for example, a validation request itself is captured in some way, the user attributes corresponding to the ARID can be fetched by anyone. To mitigate this unauthorized access to user attributes, the AVS must limit an ARID's lifetime to a short time period. The AVS may also limit the number of times it can be resolved. However, while limiting the use times of an ARID strengthens security, it may reduce the degree of applicability to email, especially in the following scenario. An ARID cannot be used for validating user attributes when a principal specifies a single relying party in a message, but the message is copied to multiple destinations, for example, using a forking proxy at the recipient side or using a listing service. Thus, this limitation on the use times of an ARID is recommended only for a communication service which needs the long lifetime of an ARID, such as email.

To detect posing user attributes using a stolen ARID in a communication request, a relying party may collect received ARIDs until their lifetime end. When receiving an ARID in a new communication request, a relying party sees if the same ARID exists in previously received ARIDs.

8.6.2.2 Replay Attacks from Intermediaries

An HTTP proxy server often exists between a principal and the AVS and between the relying party and the AVS. SIP proxy servers or email servers usually exist between the principal and a relying party depending on the communication means. If these intermediaries include a malicious or compromised server, the attacker can exploit the information stored on the server for replay attacks. However, a malicious server is rarely involved in the signaling path using HTTP, SIP, or email protocols, if each entity configures its application settings to connect only to a trust proxy server using TLS, and carefully authenticates the server using its X.509 PKC. Therefore, for each signaling path, we analyze the potential damage by replay attacks mainly from compromised intermediares and from malicious ones on the condition in which server authentication is not properly done.

To prevent man in the middle attacks described in Section 8.6.1, these HTTP proxy servers are supposed to transmit messages protected with an end-to-end TLS connection without terminating and restarting a secure TLS connection. If a principal neglects to authenticate an X.509 PKC of the AVS, the HTTP proxy server can intercept a response including an ARID from the AVS. However, this ARID alone is insufficient for the attacker to impersonate user attributes to a relying party or to retrieve user attributes from the AVS. If a relying party neglects to authenticate an X.509 PKC of the AVS, the HTTP proxy server can intercept a request carrying full information for retrieving user attributes. A short lifetime of an ARID and its limited use time, if enabled, can reduce the effect of replay attacks using this request for retrieving user attributes.

Similarly, SIP proxy servers or email servers also are supposed to use TLS, but they do not allow TLS tunneling to provide its communication service by dealing with the message headers of a communication request. Thus, the attacker can obtain a message including an ARID from a compromised server. However, to exploit the ARID for replay attacks on the AVS, the attacker needs the same processes as a relying party does in order to form an HTTP GET request. The processes include parsing two header fields of To and Sender-Reference and generating HMAC over the destination address with the secret found in these header fields. Although the computational costs of parsing and generating HMAC are not expensive, they are expensive than the cost of copy and replay. Combined with a short

lifetime of an ARID, this additional required process can reduce the effect of replay attacks using a captured communication request.

To detect replay attacks at a relying party, the relying party needs to test whether an ARID is new or has been used before (within the lifetime of an ARID), relying on collected received ARIDs, and also relying on the countermeasures (limiting the lifetime of an ARID) implemented on the AVS. If an intermediary wrongly modifies with message headers before forwarding a message (e.g., swapping a set of the To and Sender-References header fields of a message with others), a relying party cannot detect that. In that case, however, the communication service itself is also disrupted.

8.6.2.3 Replay Attacks from Relying Party Using a Received ARID

Any legitimate relying party of an ARID can attempt to impersonate the principal of the ARID just by sending the ARID to another user since this mechanism does not have a tight link between the username of AVS and the caller ID, as described in Section 8.3.2, nor a link between the signaling path and the ARID. As described in Section 8.3.1, to mitigate this attack, a relying party need to generate an ARID access code, the hash of his CEID (i.e., SIP AoR or email address) and the secret in the communication request. This is one of the countermeasures as preventing replay attacks from outsiders, as described in Section 8.6.2.1.

When it comes to this service applied to email and instant messaging where a message is copied and shared by multiple recipients, the attacker can easily pick another recipient of the received message and reuse the received ARID when sending a new message to the recipient.

To mitigate this form of forwarding attacks, the AVS and a relying party need to rely on the same countermeasures, as preventing replay attacks from outsiders, as described in Section 8.6.2.1. These countermeasures include, limiting relying parties to whom have been authorized by the principal, limiting the lifetime of an ARID to a short time period and limiting the use time of an ARID to one per relying party. Furthermore, they include that each relying party collects ARIDs that have been received until their lifetime ends. As the lifetime of an ARID is set longer to be adjusted to asynchronous services such as email, the countermeasures become more costly and less effective.

8.6.3 Denial of Service Attacks on the AVS

Since our service architecture requires online validation of an ARID, a flood of validation requests or queries potentially incurs overload on the AVS. To mitigate overload, the AVS can build on separate servers one for the issuer of ARIDs and another for the validator for them. A flood of issuance and validation requests originate from legitimate relying parties and also from users attempting denial of service (DoS) attacks. Thus, we need to maximize resource allocated for valid requests with minimizing it for invalid requests for both servers.

At the application level, it is important for the AVS to detect invalid requests as easily as possible. To do so, the AVS, acting as the issuer, must authenticate users. The AVS, as the validator, must use a lightweight query protocol, such as the RESTful API [Fielding and Taylor, 2002], which sets a query key in the path of an HTTPS URI.

Since these requests are sent over TLS, the AVS also needs to reduce the effect of flooding requests at TCP and TLS levels. At the TCP level, these servers must carefully configure TCP to mitigate TCP SYN flooding attacks. At the TLS level, these servers are vulnerable to DoS attacks by exceeding the maximum rate of TLS Client Key Exchange requests. To decrease the impact of flooding TLS handshake requests, these servers should disable the TLS re-negotiation since the issuance and validation requests requires one or two transaction to exchange small messages.

8.6.4 Phishing Attacks on the AVS

A Web site having a domain name confusingly similar to a well-known AVS makes it possible to steal the password of a user for remote access to the AVS. It is also possible for an evil Web site to respond to any attribute queries with an HTTP 200 OK response with forged user attributes attached to invalidate the attribute validation service. To prevent these attacks, both a user and the recipient of an ARID must use TLS when connecting to the AVS and must ensure that the server's X.509 PKC has a valid signature for the valid domain name.

8.7 Implementation

To illustrate our design concept for a lightweight and easily deployable mechanism, we have built an AVS prototype on an HTTPS server using a standard Linux, Apache HTTP Server [Apa, 2006], MySQL database [MySQL, 2006], and PHP (LAMP) package. The AVS does not need any special cryptographic capabilities other than its X.509 PKC for TLS. The AVS generates an ARID using hashing and stores it in a database. This implements the first approach to generating an ARID using hashing described in Section 8.5.1

To a MySQL database, we added RESTful JSON Web interfaces [James, 2010] so that the AVS efficiently verifies received ARID validation requests. Without parsing the message body of an HTTP request, the AVS forwards received requests as database queries to the "arid" table in a database indexed on two columns: an ARID and its access code. Each row in the "arid" table has its expiry time, which is checked and deleted, if it becomes stale, by a scheduled job. If the query finds no rows, the AVS receives and forwards an HTTP 404 Not Found response back to the relying party. Thus, a belated validation request from a legitimate relying party receives an HTTP 404 Not Found response, not conveying a specific reason of invalidity, namely, the ARID is expired or it provided with invalid access code. Even if a query successfully finds a row, it needs to check the expiry time in case the ARID ends its lifetime after the scheduled job performed. If the ARID is fresh and valid, it then needs to retrieve the corresponding disclosure mode and user ID to look up user attributes from different tables. This can be done by modified the RESTful JSON Web interfaces to support SQL SELECT with JOIN for retrieving multiple tables upon receiving a query by an ARID an its access code.

The AVS prototype is tested with a simplified SIP UAC and UAS implemented in a Web browser, instead of modifying existing SIP clients to support the AVS. Based on the settings of a UAC, the UAC written in JavaScript obtains an ARID from the AVS and generates a SIP INVITE message, as shown in Figures 8.9. When receiving the SIP INVITE message and extracting several parameters, the UAS written in JavaScript sends a validation request to the AVS and receives the attribute of the UAC, as shown in Firefox with the Firebug[3]

[3]Firebug is a Firefox add-on for web development found at http://getfirebug.com/.

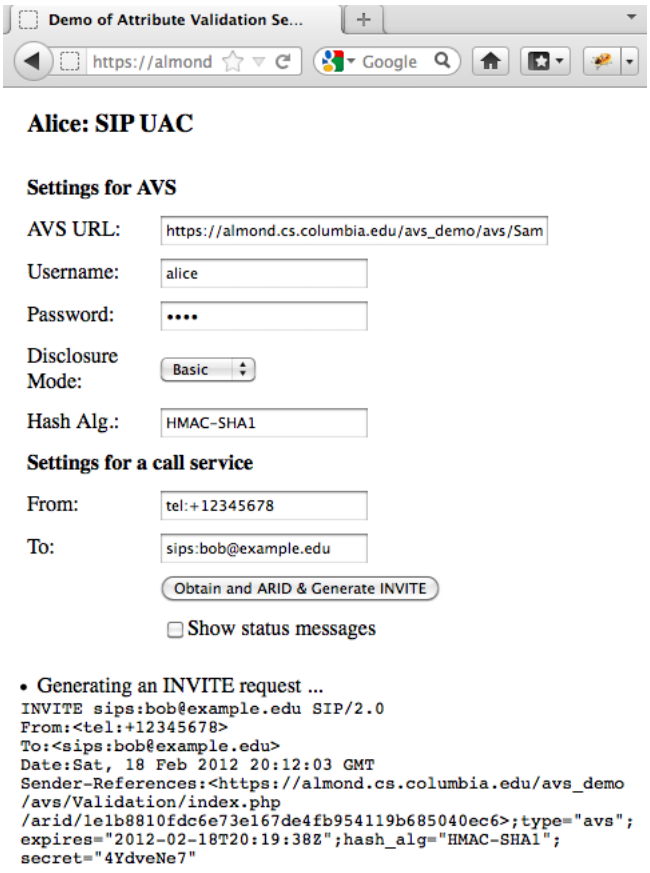


Figure 8.9: Sample implementation of UAC: Settings and the SIP INVITE request

extension in Figure 8.10.

To support selective disclosure, this implementation accepts two disclosure modes: basic or detail. Each mode has defined a set of attributes as system-wide settings. For more user-friendly selective disclosure, the AVS should allow a user to configure her preference settings of whether or not to disclose each attribute. We leave this function for a future revision of implementation.

In summary, our AVS prototype demonstrates proof of concept and the capability of easy deployment by building an HTTPS server using a standard LAMP stack and RESTful JSON Web interfaces with a minor modification.

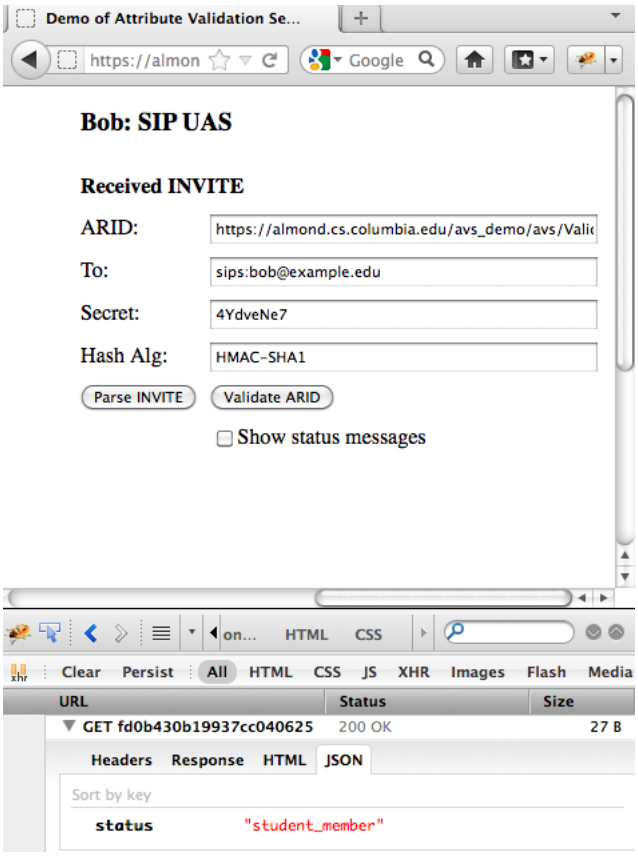


Figure 8.10: Sample implementation of UAS: Parameters extracted from a received SIP INVITE request and an HTTP response from AVS

8.8 Evaluation

We demonstrate that the issuer can be easily deployed by implementing our AVS prototype, building an HTTPS server using a standard LAMP stack without special security capabilities other than X.509 PKC for the AVS in Section 8.7. To evaluate our claim that our proposed system is simple and lightweight, ideally, we should compare our implementation with a system having the same functionality, and assuming the same trust model. We select U-Prove [Paquin, 2011] as a counterpart since it provides anonymous attribute assertions having relatively similar functionality. The U-Prove service architecture is depicted in Figure 8.11 when U-Prove is applied to a SIP call assuming that a U-Prove token is transmitted using HTTP over TLS and SIP over TLS.

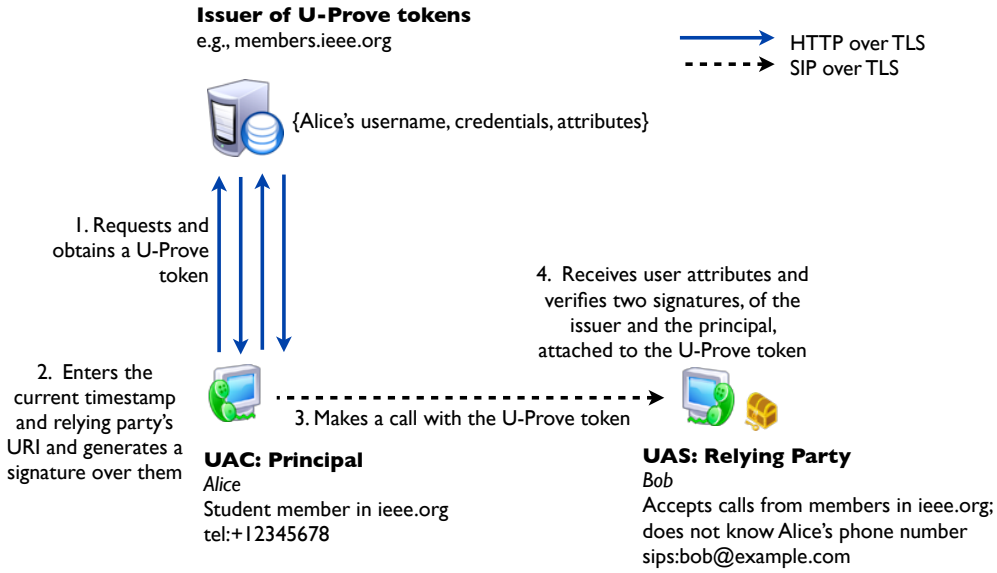


Figure 8.11: U-Prove service architecture applied to a SIP call

8.8.1 Qualitative Evaluation: Functionality

Table 8.1 compares AVS with U-Prove in required functionality listed in Section 8.4. Since AVS is designed specifically for our target service, which is to identify a good communication request in SIP and email, its functionality is optimized for the requirements. Thus, the AVS fully satisfies all the requirements. Although a security requirement, proof of possession of a token (SEC-REQ-5, and a privacy requirement, the untraceability of communication history by the issuer (PRIV-REQ-1) are not supported as the same extent as U-Prove, they are allowed to be relaxed because of the low risk of our traget service and of our priority of ease of the service deployment, as described in each requirement.

There are three differences in the general functions: the capability of sending user attributes in the header of a signaling message, easy validation of attributes by recipients of a communication request, and easy deployment without special security capabilities. First, whereas the AVS can deliver user attributes by reference, using an ARID, in the message header of a SIP request or an email message, U-Prove cannot convey its token in XML in the message header since it exceeds the maximum length of an email message header field, 998 characters [Resnick, 2008]. U-Prove therefore needs to covey its token in

	AVS	U-Prove
General functions		
User attributes delivery in the header of a signaling message	Yes: By reference, ARID in an HTTPS URI	No: By value, U-Prove token set in XML
Attributes issued by multiple issuers in a single message	Yes	Yes
Supporting short-lived attributes	Yes	Yes with on-demand option
Easy validation of attributes by recipients	Yes: Online validation from AVS using TLS	Complicated: Offline validation by relying party
Easy adaption to existing communication protocols	Yes	Yew with non-interactive option of token presence protocol
Easy deployment without special security capabilities	Yes: TLS and HMAC-SHA1	No: Adds U-Prove token crypto. functionalities to a secure channel
Flexible allocation of issuer and validation functions	Yes	Not necessary
Security		
Data confidentiality	Yes: Using TLS	Yes: Adding security, such as TLS
Data integrity	Yes: ARID generated using SHA1 and using TLS	Yes: Signature in U-Prove token
Issuer authentication	Yes: Relying on Issuer's PKC used in TLS	Yes: Relying on Issuer's PKC and signature on U-Prove token
Limiting relying party	Yes: AVS's control with ARID's lifetime and an access code generated with relying party's CEID	Yes: a relying party's check a desired relying party's ID and its expiry time in U-Prove token
Proof of possession of a token (an ARID or a U-Prove token)	Yes: By providing fresh ARID and a secret	Yes: By verifying the signature of the principal
Privacy		
Untraceability of communication history by issuer	Partly yes	Mostly yes
Selective disclosure of attributes	Yes: Supported through a trusted Issuer	Yes: User can directly control each attribute.

Table 8.1: Comparison of functionality required for our target service: AVS vs. U-Prove

the message body using a multi-part MIME message body [Freed and Borenstein, 1996a; Freed and Borenstein, 1996b], imposing the complexity of message generating and parsing on user agents. If U-Prove allows to send a U-Prove token by reference, this completely changes the service architecture of U-Prove. This change makes it unable to offer the capability of verifying the signatures and the attributes set in a U-Prove token without the relying party's accessing the issuer, called offline validation, and untraceability by the issuer.

Second, while the AVS enables recipients to validate user attributes by sending an ARID to the AVS, U-Prove requires recipients to validate offline by verifying two signatures of the issuer and the user using their public keys, the timestamp, and the specified relying party's URI, adding cryptographic computation into user agents. When a user weigh online, which requires a relying party to connect with the issuer, and offline validation, there are generally two trade-offs between the risk of server failure, such as unavailable caused by overload, and the cost of validation on the user, and between user privacy against the server and the cost of validation on the user. Offline validation is independent of the validation server availability. Offline validation also provides a privacy property for users. The validation server, which is often co-located with the issuer of attributes, cannot trace the use of attributes. However, given that our target service needs short-lived attributes, the privacy advantage of offline validations are not significant since the issuer can guess when and how many times a U-Prove token is used from when it issues the U-Prove token. The use of a U-Prove token corresponds to the issuance with one-to-one basis. The timing of these two actions is similar. Moreover, to easily deploy attribute validation services, a smaller cost of user agents overweighs the risk of server failure. Thus, we consider online validation using AVS offers easier validation of attributes than offline validation, although online validation requires countermeasures to overload and DoS attacks.

Third, while the AVS needs TLS and HMAC-SHA1, which are supported by the standard security libraries of Apache and PHP, U-Prove requires its own special cryptographic libraries in addition to a standard libraries for a secure channel, such as TLS. For example, the issuer of a U-Prove token needs to prepare a discrete-logarithm-representation public key for a U-Prove security and an X.509 PKC for TLS. Users and recipients also need to

use special cryptographic libraries. This requirement of special security capabilities hinders easy and wide deployment.

Regarding security functionality, U-Prove uses its own cryptographic functions to offer attribute integrity and issuer authentication, whereas the AVS relies on transport layer integrity and authentication, using the standard security libraries. In general, application-level attribute integrity (i.e., attributes with the signature attached) can be ensured, whether the attribute are directly transmitted between the issuer and a relying party, or through intermediaries. In contrast, transport-layer-level attribute integrity needs to rely on the trustworthiness of the intermediaries along the path. In this sense, U-Prove provides stronger security than the AVS.

However, U-Prove provides stronger security, in not all aspects, than AVS. Since U-Prove security offers no attribute confidentiality, it needs another security mechanism, such as TLS, for protecting a message over networks in addition to U-Prove security. As replay protection, both mechanisms have similar countermeasures although there is difference in the role of the issuer. Since a U-Prove token is sent by value, a principal needs to send a token In U-Prove, the issuer does nothing. Instead, a principal sends a U-Prove token including the principal's attributes and each relying party's URI to desired relying parties. Upon receiving a U-Prove token, each relying party checks if it is desired to the URI of himself. In addition, the relying party verifies a signature over the timestamp and the relying party URI using a principal's token-specific public key from the U-Prove token. The relying party also check if the timestamp is within the valid time period specified in the U-Prove token. On the other hand, the issuer of an ARID, the AVS controls the limitation with collaborating with the principal. The relying party needs to send a validity request with a newly generated access code attached to the AVS, instead of checking the valid time period or the original relying party. Thus, unless a relying party checks the relying party's URI and the U-Prove token's lifetime, it cannot detect the attack.

A U-Prove principal can prove to a relying party that she has the token and attributes on it by the signature generated by the corresponding private key. Compared to the proof of the possession of the U-Prove token, the AVS provides a relatively weak proof of the possession of an ARID, but keeps the system simple and lightweight. As described in Section 8.3.2,

when receiving an ARID, a relying party verifies the proper possession by the fact that a user sends a fresh ARID to a legitimate relying party. If a relying party can retrieve the user attributes from the trusted AVS over TLS, he considers that the user properly has the ARID. If he receives an HTTP error response from the AVS, he detects unauthorized use of the ARID. If the lifetime of an ARID is longer for asynchronous communications, such as email, the proof of possession becomes weak against replay attacks.

Regarding privacy functionality, U-Prove provides stronger privacy than the AVS since the degree of trust from a relying party to the issuer co-located with the validation server is weaker. U-Prove provides the untraceability of communication history by issuer more than the AVS. A user of the AVS can protect the relying party's ID using hashing, but reveals the number of relying parties for each communication. Online validation of an ARID reveals the IP addresses of relying parties and the time of the user of the ARID. On the other hand, offline validation of a U-Prove token can hide the number of relying parties and their IP addresses although the time of the use can be guessed by the time of the issuance for a short-lived token. Also for selective disclosure, the AVS can trace which attributes are disclosed since it gets involved in the selection. A user asks the AVS whether or not to disclose each attribute, or simply select a disclosure mode, for each communication. By contrast, U-Prove allows a user to directly control selective disclosure without any support of the issuer. A user can select attributes to disclose and encrypt undisclosed attributes with her private key.

Thus, with assuming the issuer is trusted by a user and relying party in terms of handling attributes and ARIDs, the AVS provides functions to meets all the general requirement, offers moderate security and privacy. By settling for offering moderate security and privacy, the AVS can avoid complexity of security capabilities, resulting in keeping the mechanism simple and lightweight. Compared to U-Prove, more functions are assigned to the issuer of an ARID than the issuer of a U-Prove token since we assume that a principal and relying party trust the issuer to manage access log files. By contrast, for easy deployment, fewer functions are assigned to a user and relying party of an ARID than these entities of a U-Prove token.

8.8.2 Quantitative Evaluation: Lines of Code

To support the functionality evaluation indicating the simplicity and the capability of easy development and deployment of our proposed mechanism, we would like to quantitatively evaluate our AVS prototype. For this purpose, we compare our AVS prototype with U-Prove Crypto software development kit (SDK) [Microsoft, 2012]. We measured the number of lines of code for each although the lines of code depends functionality, configuration flexibility, coding style, packaging policy, and the language. Thus, the number of lines of code is a second-order metric. The exact numbers do not directly reflect the complexity, but suggest an appropriate ordering.

Table 8.2 compares the numbers of lines of code[4] of our AVS libraries and part of U-Prove Crypto SDK. Figure 8.8.2 shows the sample AVS code using our AVS libraries. Although these implementations are written in different languages, the AVS in PHP and U-Prove in Java, both are class-based object-oriented and define setter and getter functions for most variables. The UAC and UAS for the AVS are written in JavaScript and using an external HMAC-SHA1 library. The code does not define any classes nor setters and getters of variables. However, they include functions to generate and parser a SIP INIVTE request while the U-Prove Crypto SDK includes any interfaces to messages exchanged over networks.

For a fair comparison, the part of U-Prove excludes unnecessary optional functions related to device-supported security. The part of U-Prove also excludes several files which are provided by the libraries of PHP (e.g., Base64.java), to decrease the language dependencies. It is worth noting that the U-Prove Crypto SDK provides offer cryptographic functions only. When implementing the issuer of U-Prove, developers needs to include functions of all message exchanges over networks, such as HTTP interfaces, user authentication and database interfaces. The code for these functions requiring additional implementations for U-Prove is also excluded from the subtotal and total number of the lines of code for the AVS.

For U-Prove, the interfaces of crypto and a U-Prove token are needed both on the issuer and principal. However, to avoid double counting, the number of lines of code is added

[4]The code excluding comments and blank lines is counted by a code line counter, cloc-1.55-pl at http://sourceforge.net/projects/cloc.

Entity: Functions	AVS	Part of U-Prove
Issuer: Issuing a token	235	1097
Issuer: Validating a token	199	0[1]
Issuer: Authenticating users and reading user attributes from database	192	-[2]
Issuer: HTTP interfaces	176	-[2]
Principal: Obtaining a token	98	375
Principal: User and SIP interfaces	84	-[2]
Relying party: Validating a token	33	130
Relying party: User and SIP interfaces	39	-[2]
Secure token[3]	134	246
Crypto	0[4]	636
Subtotal		
Issuer excluding user authentication, database and HTTP interfaces	568	1733
Principal: Obtaining a token	98	621
Relying party: Validating a token	33	130
Total	699	2484

[1] This is because the validation of a U-Prove token is performed on a relying party while the validation of an ARID is on the AVS upon a request from a relying party.

[2] Additional implementations or packages are needed.

[3] At subtotal, secure token-related functions are added into Issuer for AVS, but into the principal for U-Prove.

[4] The AVS security is achieved by the standard libraries such as TLS and HMAC-SHA1.

Table 8.2: Comparison of the number of lines of code: AVS vs. Part of U-Prove

only into one entity which mainly use. For example, the interfaces of a U-Prove token are added only into the principal. Principal, and the interfaces of crypto are added only into the issuer. On the other hand, all entities of AVS use cryptography provided by the standard libraries such as TLS and HMAC-SHA1. ARID-specific validation functions are centralized on the issuer. Thus, there is no overlap in three entities.

Compared to the numbers in the Subtotal of Table 8.2 of part of U-Prove, the AVS approximately requires one third of the lines of code of the part of U-Prove for issuers. This difference is caused by the AVS security relying on the standard libraries while U-Prove implements its own security in addition to the standard libraries, such as TLS. This comparison roughly indicates the simplicity of the AVS security and privacy compared to U-Prove. More remarkable, the AVS requires less than only one sixth of the lines of code of the part of U-Prove for the principal, and a quarter of the lines of code for a relying party. These differences in the lines of code roughly reflect the differences in the degree of the complexity and also in allocating the functions of token validation, as identified in Section 8.8.1.

All in all, the numbers of lines of code indicate that an AVS system, especially user and relying party, has simpler and more lightweight functionality than U-Prove although these numbers also depend on other factors such as functionality, configuration flexibility and coding style. Combined with the results of functionality evaluation, we estimate that an AVS system can be easily developed and deployed for our target service.

8.9 Related Work

This section reviews attribute certificates or assertions that have been studied and developed over two decades [ITU-T, 1993; Ellison, 1999; Brands, 2000; Cantor *et al.*, 2005; Peterson *et al.*, 2006; Farrell *et al.*, 2010; Zurich, 2011]. This section also reviews mechanisms for trusted third-party authentication, which are relatively recent work. Existing mechanisms explained below satisfy some of our requirements (Section 8.4), but not all.

Prior work on asserting user attributes has been designed for attribute-based authorization for various purposes including access control on limited resources, data origin authenti-

```
<?php
require_once("User.php");
require_once("Issuer.php");

//sets a user authentication mode
$userAuth=User::AuthByHttpBasic;

//authenticates a user by its password stored in the database
//returns a user ID (if ok), or interrupts with an error
$issr = new Issuer();
$uid = $issr->userAuth($userAuth);
if (isset($uid)) {
  //reads a message body and generates an ARID, uptating the
     database
  //then responds with HTTP 200 OK including the ARID
  $issr->generateARID($uid);
?>
```

Listing 8.1: Sample code of AVS issuer

cation for prioritizing communications, and non-repudiation for accounting and settlement among service providers [Farrell *et al.*, 2010; Peterson *et al.*, 2006], which entails different levels of risks. As a result, mechanisms offer strong security, but are over-engineered for identifying good calls or messages, which are low-risk interactions.

Most attribute credentials, such as an X.509 attribute certificate (AC) [Farrell *et al.*, 2010] and a SAML assertion for SIP [Cantor *et al.*, 2005; Tschofenig *et al.*, 2010], bind a user's attributes to her authenticated identity. More specifically, a credential binds Alice's attributes to her two authenticated IDs: one is issued by attribute service provider or an attribute authority, and another is issued by a communication service, complicating authentication relationships, as depicted in Figure 8.12. This binding enables Alice, an originator or a principal, to definitively prove to Bob, the recipient or relying party, that she possesses attributes on the credential. However, it has two drawbacks related to the service deployment. First, a credential issuer of user attributes needs to be the same, work in

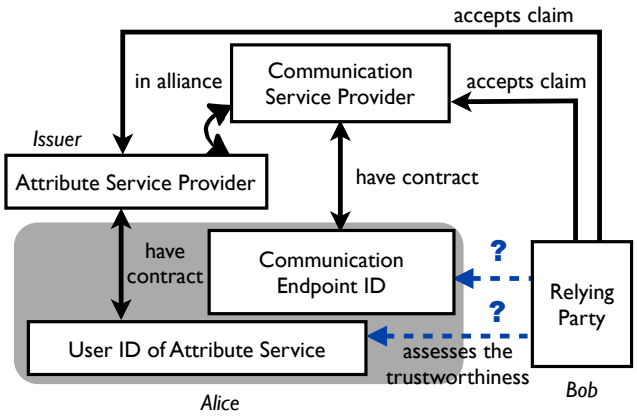


Figure 8.12: Conceptual model of authentication relationship between originator (Alice) and recipient (Bob) by validating user ID of attribute service and communication endpoint ID

alliance with a communication service provider, or confirm her communication endpoint ID (CEID) or contact address by the fact that she has performed a simple task which the issuer assigns her via her contact address (reachability). Second, a communication service needs to enable recipients to authenticate an origin ID. These two drawbacks can be overcome for email, but nor for SIP. Since caller ID authentication services have suffered from technical and practical problems, as discussed in Section 6.6. These requirements have been a serious obstacle to wide deployment of attribute certificates binding a user's attributes to the user's identity.

As people have more concerns over their privacy online, the anonymity of the holder of an attribute certificate has gained more interests. Anonymous certificates unlink the principal name from the principal's attributes by specifying the principal by its public key. The Simple Public Key Infrastructure (SPKI) [Ellison, 1999; Ellison *et al.*, 1999] offers an authorization certificate, which binds the holder's permission to the holder's key, in contrast to its attribute certificate, which binds the holder's permission to the holder's name. The permission, which is authorized to an attribute, explicitly identify authorized operations. However, for our target service, neither issuer nor principal can obtain permission from a relying party beforehand.

U-Prove [Brands, 2000] and Idemix [Zurich, 2011] provide a proof of possessing at-

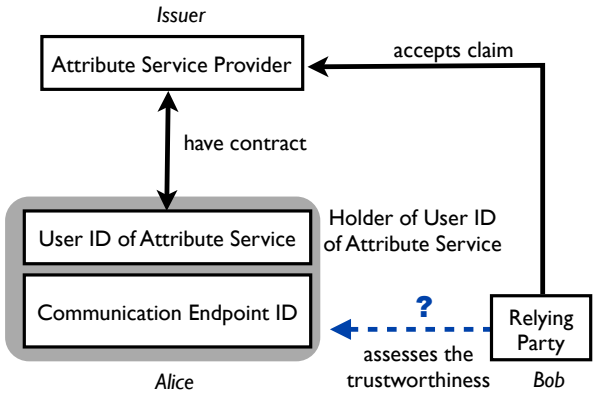


Figure 8.13: Conceptual model of authentication relationship between originator (Alice) and recipient (Bob) without validating any user IDs

tributes, which is called a token. In these mechanisms, a principal generates a token-specific public key, and signs over attributes in the token issued by the issuer. As depicted in Figure 8.13, these techniques simplify authentication relationships by removing the requirement of an authenticated CEID. These techniques also enhance privacy properties against the issuer, such as offline validation and selective disclosure without the issuer's involvement, but, at the same time, add complexity especially to user agents supporting this privacy-enhanced credential. These privacy enhancements are over-engineered for low-risk interactions.

These mechanisms use discrete-logarithm-representation-based public keys. The issuer generates a digital signature over a set of attribute values and cryptographic information with a discrete-logarithm-representation-based public key that is generated for each use. They support selective disclosure which enables a principal to directly control whether or not to disclose each attribute. They also support untraceability of the use of a certificate by the issuer and unlinkability between the issuance and the use of a certificate. Idemix enhances privacy more than U-Prove, resulting in more complexity. For example, Idemix supports unlinkability among multiple uses of a certificate by generating a random key based on the same master secret key for each use. In addition to a short-lived certificate, both support a persistent certificate of which validity is checked offline using a certificate revocation list. This rich functionality of privacy enhancement imposes the complexity of development and deployment. These privacy enhancements are beyond our requirements

listed in Section 8.4. The principal needs to rely on the issuer's policy how to handle their attributes issued and stored on the issuer's server after all. Thus, our proposed mechanism prioritizes simplicity for easy deployment over complexity for privacy enhancement.

All attribute certificates or assertions are authenticated by their issuer, a trusted third-party who is not involved in the communication in question. Kerberos [Neuman *et al.*, 2005] has been widely used an authentication system using a trusted third-party, a key distribution center (KDC). An authentication server in the KDC authenticates a principal on behalf of a relying party, a service server (SS). A ticket granting server (TGS) in the KDC issues a ticket which is effective only for a session between the principal and the SS for a limited time period. The Although Kerberos features cover all our mechanism needs, the trust model is different. Unlike our trust model (Section 8.2.2), Kerberos assumes that the issuer and the relying party, SS, share a SS's secret key to allow the SS to verify a received ticket by decrypting with the SS's key without connecting to the TGS. To resolve the difference in the trust model, sharing the SS's key with the TGS is eliminated, but it is intractable since the off-line decryption using the SS's key is a key feature in Kerberos. Thus, our target service cannot use Kerberos because of this difference in assumed trust model.

OAuth [Hammer *et al.*, 2012] has recently emerged as a third-party authentication model used across Web service providers. This mechanism allows a user of a Web service acting as a principal to delegate limited permission using an access token to another Web service in order to access part of her resource. With the OAuth terminology, a principal is the resource owner, a relying party is the client, and the issuer of an access token plays two roles of the authorization server, which is a third-party authenticator, and the resource server. The issuer of OAuth issues an access token in two steps. Upon a relying party's request through a resource owner, the issuer first provides a relying party with an authorization grant on the approval of a resource owner. Upon the relying party's request along with the authorization grant, the issuer then issues an access token to the relying party. OAuth differs from our target service in who takes the initiative in requesting an access token to give permission to retrieve attributes. In OAuth, a relying party starts to request an access token where as a principal starts in our mechanism. OAuth also differs from our trust model; the issuer and

a relying party share the relying party ID and a secret to authenticate the relying party when the issuer issues an access token, whereas our mechanism does not require them to share a key, as described in Section 8.2.2. To adopt OAuth to our target service, we need to modify the OAuth procedures for the principal, instead of a relying party, to take the initiative in requesting an access token. Correspondingly, the principal, instead of a relying party, shares a principal's ID and a secret for the principal's authentication on the issuer. This modified procedure is similar to the procedures for a principal using an application, which does not use HTTP, called non-Web application. The non-Web application model assumes that both the principal and a relying party are installed in the same entity, resulting in a simplified version of OAuth where the authorization server is no longer a third-party authenticator from the perspective of the entity. To adopt this non-Web application model to our target service, an access token should be transmitted by an email message or a SIP request over TLS. Since the client authentication information is embedded in an ARID in our proposed mechanism, the recipient of the email message of SIP request can access the resource server by solely using the ARID. Thus, this is vulnerable to replaying attacks from a legitimate recipient. OAuth needs to add a countermeasure to replaying attacks, such authenticating a relying party using an access code, similar to our mechanism. Thus, we prefer a simple mechanism using a single token rather than using a simplified option along with unnecessary complexities for other options.

8.10 Conclusion

We have designed, implemented, and evaluated a mechanism for an attribute validation service, which is simple and lightweight for validating user attributes in order to identify a good communication request. Authentication services for a caller ID in the SIP URI or the tel URI have encountered difficulties in development and deployment. This observation motivated us to introduce an anonymous attribute token, ARID, that is not associated with the origin ID or the signaling path. An ARID is an HTTPS URI, which can be conveyed in a message header of a communication request, unlike typical existing anonymous tokens. To provide the mechanism with moderate security keeping it simple and lightweight, most

security properties of an ARID relies on transport layer security such as TLS to leverage the wide deployment of HTTPS servers.

We have implemented an AVS prototype on an HTTPS server using a standard LAMP package. The AVS generates an ARID with a short lifetime and restricting desired relying parties. A legitimate relying party can easily validate an ARID by sending an HTTP GET request, which uses a RESTful API simply forwarded as a query to a database storing ARIDs. Compared to U-Prove, a relatively simple U-Prove mechanism, AVS provides weaker proof of possession of an ARID and fewer privacy enhancement. This is because AVS assumes that a relying party trusts on the issuer more than the assumption that U-Prove makes in order to minimize functions allocated to the user side for easy service deployment. This AVS mechanism aims at striking the proper balance between properties of security and privacy and the low-risk level of protected assets, whether or not to accept a communication request.

Chapter 9

Controlling Unwanted Calls Using Two Proposed Approaches

9.1 Introduction

This chapter describes a call filtering system integrating our work into the current practice. It then explains the limitations of our work and how these limitations are to be solved. It also describes potential direction where this work can be expanded.

9.2 Call Filtering Using Two Proposed Approaches

Figure 9.1 illustrates our call filtering process that integrates our two approaches – using cross-media relations (Chapter 7) and user attributes (Chapter 8) – into existing call filtering conditions. These additional filtering conditions are typically installed at the callee's user agent to make an individual decision over a call request. In the contrast, the inbound server blocks calls that are unwanted by a group of users based on IP addresses and caller IDs. The inbound proxy intends to protect their resources (e.g., the cost of processing unnecessary call requests) and to preserve the call service utility and value for their users. In addition to these two filtering conditions, the inbound server typically authenticates a caller ID on behalf of users. If any caller ID authentication mechanisms are available, but the authentication fails, the inbound server may set an asserted caller ID (e.g., the

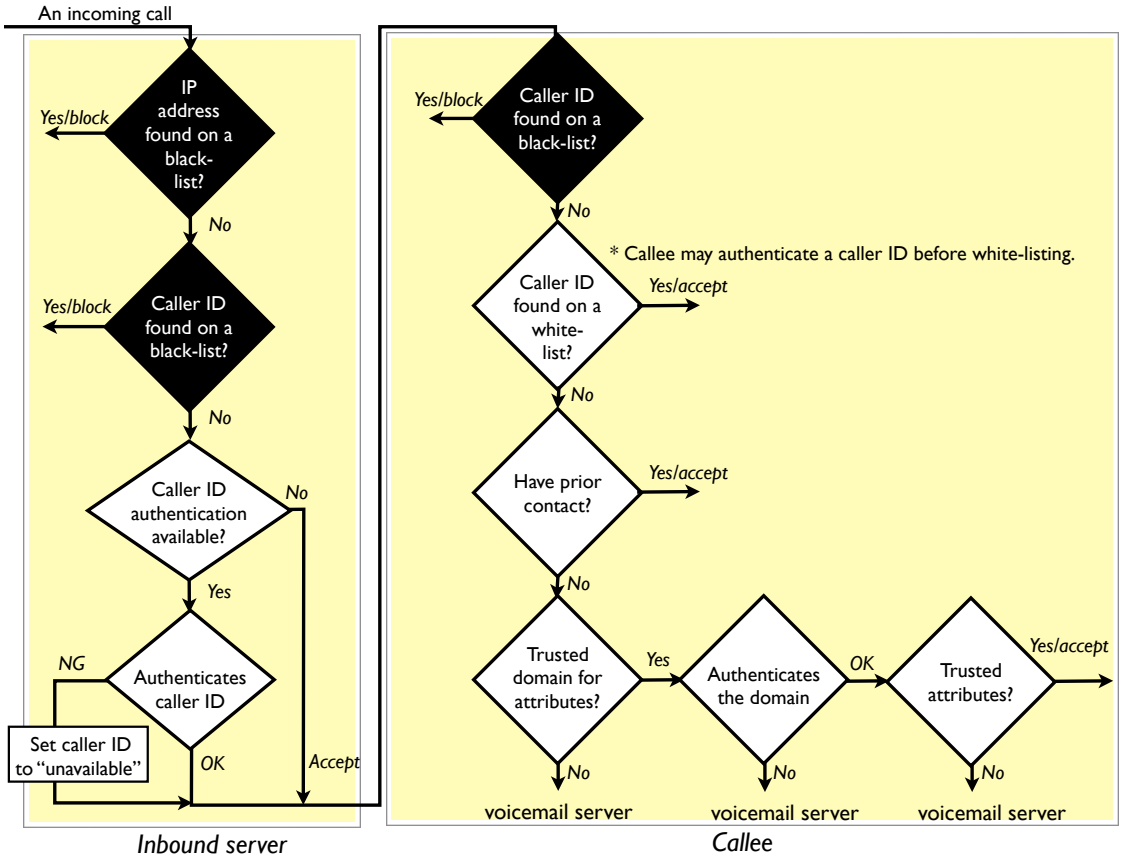


Figure 9.1: Call filtering using two approaches: Using cross-media relations and user attributes

P-Asserted-Identity header field [Jennings *et al.*, 2002]) to "unavailable."

At the callee side, if a caller ID – whether or not to be authenticated – is found on a black list, or an authenticated caller ID is found on a white list, this filtering process proceeds to our new filtering conditions based on our two approaches in the following manner. First, to determine if the callee has had prior contact with the caller, the filtering process checks SIP related header fields and accesses a database for the CURE system, as described in Section 7.7. Limitations and privacy concerns over the CURE database are discussed in Section 9.3.

Second, to determine whether the callee accepts a call from a party who holds attributes, the filtering process examines whether the callee trusts the domain server issuing the attributes, based on the callee's configuration. The filtering process then retrieves the caller's

attributes from the issuer so that it can determine whether the callee accepts a call from the holder of the attributes based on the callee's configuration, or it prompts the callee to determine that.

If the callee's configuration contains a list of all trusted domain servers and a list of all acceptable attributes, using these filtering conditions can automatically make a decision whether or not to answer the call. Thus, the inbound server on behalf of the callee can test on these filtering conditions. Enabling automated decision needs a standard framework for expressing trusted domain servers and caller's attributes. This is further described in Section 9.4.

Even if the framework is established, it is difficult for each individual to maintain a long list of trusted domain servers or acceptable attributes. When the inbound servers and callees belong to an enterprise or school, the inbound server can filter incoming calls, on behalf of a group of users. For example, the filtering is based on a white list populated with the contact addresses of business partners and based on a list of trusted domain servers, for verifying attributes, populated with all professional membership organizations

9.3 Limitations and Privacy Concerns over CURE Database

The CURE database stores cross-media relations, namely, various types of communication history including contact addresses, Web transactions, and email exchanges. Communication history is privacy-sensitive data, but there is a trade-off between privacy and service convenience. If the data is obfuscated – to make a binary decision, accept or block – in such a way that the original data cannot be restored, a user cannot retrieve any specific information about prior contact.

The CURE database can be maintained by one of three entities: a SIP inbound server, a trusted third party, and the owning individual. First, a SIP inbound server is the most likely option since it is the main user of query to the database when filtering calls on behalf of users. Except in the case of query to the database, user authentication is required for email clients and Web browser extensions to update the database. However, an issue occurs when a callee uses multiple VoIP services operated by different ITSPs since it is usually difficult to

share customer information among ITSPs. Therefore, it requires business alliances between them.

Second, in the case where a trusted third party maintains the database, user authentication is required for SIP inbound servers, email clients, and Web browser extensions that query or update the database. Thus, a user needs to set his username and credential for each entity to be authenticated by the trusted third party. To avoid this authentication issue, a third party authorization using an access token, such as OAuth [Hammer *et al.*, 2012], could be used.

Third, if the owning individual maintains the database, no privacy issues arise theoretically. The individual is responsible for storing the information about prior contact, preserving his privacy. However, when the user has multiple clients or multiple VoIP accounts, he needs to share the database with all clients or accounts he uses. As a result, the usability of the service suffers.

In summary, deploying the CURE system requires to consider how the CURE database authenticates user clients and a SIP server. In addition, an email client that collects message IDs of outgoing messages (Section 7.6.2) needs to be authenticated by the MDA (e.g., IMAP server). Therefore, to deploy the CURE system effectively, it is important for multiple clients to integrate a third party authorization with user authentication.

9.4 Platform for Attribute and Filtering Preferences

One of promising directions we can extend this work is defining the format of user preferences in trusted domains, attribute types, and attribute values. By defining the format, not only by humans but also user agents and servers can parse preferences so that they can automatically filter calls based on them. The following sub-sections describe possible ways to define the platform.

9.4.1 Attribute Preferences

As shown in the right bottom of Figure 9.1, a caller's attributes require three steps to be retrieved: to determine whether the domain is trusted, to check if it can verify domain

authentication, and to examine whether the attribute types and their values are trusted to answer the call.

To allow a SIP UA or SIP server to filter calls based on retrieved attributes, it is the key to express the callee's preferences in domain names, attribute types, and their values, similar to privacy policy preferences [Cranor, 2002]. This should be explored in future work.

9.4.2 Filtering Preferences

Call processing language (CPL) [Lennox *et al.*, 2004] has defined the data format for call filtering conditions in XML. Using the CPL script, we could expand it to filter calls based on trusted domain, attributes, and values.

Part III

Conclusions

Chapter 10

Conclusions

The growth of VoIP services requires that end users and servers process a larger amount of signaling messages, which are also becoming longer. To maintain the grade of their services, ITSPs need to redesign their server resources for their large-scale VoIP system if the transport protocol for VoIP signaling affects the server capacity and performance. To benefit by its low cost and rich features without being disturbed by unwanted calls, end users need to control unwanted calls without blocking desirable ones.

This thesis first presents quantitative analysis of the impact of using connection-oriented protocols, TCP or SCTP, on SIP server capacity and performance. To distinguish the impact of TCP or STCP implementation from SIP server implementation artifacts, this study provides the measurement results using our echo server, SIP front-end server, and full SIP server. Our measurements identified that the major impact is the memory footprint per connection and demonstrated that a 32-bit commodity server can accommodate a large number of concurrent TCP connections and process registration and call requests from users. This study also gives a rough estimate for the cost of SIP keep-alive messages on CPU utilization, but leaves the measurement for future work. Regarding SCTP, our measurements examined the effect of using the one-to-many style socket on the server and client, which reduces the number of sockets on the server and setup delay for clients. They contributed to the improvement of the Linux SCTP implementation by pointing out a few scalability problems. Since the implementation has still significant room for improvement in the efficiency of handling a large number of associations, this thesis leaves the measurement

of SIP server performance using SCTP in a user-to-server scenario for future work.

Our SIP server measurement indicates that SIP server performance is susceptible to the impact of TCP byte-streaming, which requires a longer lifetime of a thread or process for message parsing than message oriented protocols like UDP or SCTP. This impact is negligible under low loads, but become noticeable under high loads. These results can be applied to other large-scale servers that need a large number of TCP connections or SCTP associations and high throughput.

The thesis also proposes two approaches using cross-media relations and user attributes to identifying good calls, namely, legal and desirable calls. The first approach, using cross-media relations, is motivated by the observations that many good calls are preceded by a Web transaction or email messages because they attempt to mitigate the intrusive nature of call services. Our system design focusing on prior contact as a helpful distinguishing feature between good and unwanted calls, and covers Web-then-call and email-then-call. Our user study demonstrates that approximately 30 percent of email messages, 5 percent of calls, and 4 percent of SMS messages can be identified as good by using relations to prior web transactions. The user study indicates that using cross-media relations would be useful as an additional component of a call filtering system, but at the same time that the degree of the usefulness varies across recipients and the usage type of accounts. The user study also motivates us to proceed to our second mechanism, using caller attributes, by showing that a certain fraction of good messages or calls still remain unlabeled.

To design a mechanism for verifying caller attributes, this thesis focuses on easy deployment without authenticating the caller ID since existing mechanisms have suffered from difficulties in deployment. This thesis proposes an attribute validation service using a simple and lightweight attribute reference ID (ARID). Since authentication services for caller ID in the SIP URI or the tel URI have encountered difficulties in development and deployment, our design allows the ARID to loosely link with the originator ID or the signaling path. An ARID is an HTTPS URI, which can be conveyed in a message header of a communication request, unlike typical existing anonymous tokens. To provide the mechanism with moderate security, keeping it simple and lightweight, most security properties of an ARID relies on transport layer security such as TLS to leverage the wide deployment of

HTTPS servers. Compared to an existing anonymous attribute credential, our proposal provides weaker proof of possession of an ARID and fewer privacy enhancements. This is acceptable on the assumption that a relying party trusts more on the issuer of ARIDs in order to minimize functions allocated to the user side for easy service deployment. Our design aims at striking the proper balance between properties of security and privacy and the low-risk level of protected assets, whether or not to accept a communication request.

Part IV

Bibliography

Bibliography

[3GPP, 2011] 3GPP. LTE System Overviews. http://www.3gpp.org/LTE-System-Overviews, 2011. [Online; accessed May 2012].

[AES, 2001] Specifications for the Advanced Encryption Standard. Federal Information Processing Standard (FIPS) 197, National Institute of Science and Technology, November 2001.

[Alcatel-Lucent, 2002] Alcatel-Lucent. Lucent Technologies new high-capacity switch accelerates cost-effective migration to Internet Protocol networks (News Release). http://www.alcatel-lucent.com/, December 2002. [Online; accessed January 2006].

[Alvestrand, 2011] H. Alvestrand. Overview: Real Time Protocols for Brower-based Applications. Internet-draft, Internet Engineering Task Force, September 2011. Work in Progress.

[Andersson *et al.*, 2007] L. Andersson, E. Davies, and L. Zhang. Report from the IAB workshop on Unwanted Traffic March 9-10, 2006. RFC 4948, Internet Engineering Task Force, August 2007.

[Apa, 2006] Apache HTTP Server Version 2.0. http://httpd.apache.org/docs/2.0/, 2006. [Online; accessed January 2006].

[Apa, 2012] Apache HTTP Server Version 2.4. http://httpd.apache.org/docs/2.3/mod/core.html#keepalivetimeout, 2012. [Online; accessed February 2012].

[Balasubramaniyan *et al.*, 2007] V. Balasubramaniyan, M. Ahamad, and H. Park. CallRank: Combating SPIT Using Call Duration, Social Networks and Global Reputation.

In *Proceedings of CEAS2007 - Fourth Conference on Email and Anti-Spam*, Mountain View, CA, USA, August 2007.

[Barth, 2011] A. Barth. HTTP State Management Mechanism. RFC 6265, Internet Engineering Task Force, April 2011.

[Baset *et al.*, 2010] S. A. Baset, J. Reich, J. Janak, P. Kasparek, V. Misra, D. Rubenstein, and H. Schulzrinnne. How green is IP-telephony? In *Proceedings of the first ACM SIGCOMM workshop on Green networking*, Green Networking '10, pages 77–84, New Delhi, India, 2010.

[Bellamy, 2000] J.C. Bellamy. *Digital telephony*. John Wiley & Sons, New York, NY, USA, third edition, 2000.

[Berners-Lee *et al.*, 2005] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force, January 2005.

[Beverly and Sollins, 2008] R. Beverly and K. Sollins. Exploiting Transport-Level Characteristics of Spam. In *Proceedings of CEAS2008 - Fifth Conference on Email and Anti-Spam*, Mountain View, CA, USA, August 2008.

[Birke *et al.*, 2010] R. Birke, M. Mellia, M. Petracca, and D. Rossi. Experiences of VoIP traffic monitoring in a commercial ISP. *International Journal of Network Management*, 20(5):339–359, September 2010.

[Bloom, 1970] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[Boykin and Roychowdhury, 2005] P. O. Boykin and V. P. Roychowdhury. Leveraging social networks to fight spam. *Computer, IEEE Computer Society Press*, 38(4):61–68, April 2005.

[Braden, 1989] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, Internet Engineering Task Force, October 1989.

[Brands, 2000] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*. Number 0-262-02491-8. The MIT Press, August 2000.

[Bray *et al.*, 2004] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation REC-xml-20040204, World Wide Web Consortium (W3C), February 2004.

[Camarillo *et al.*, 2003] G. Camarillo, R. Kantola, and H. Schulzrinne. Evaluation of transport protocols for the session initiation protocol. *Network, IEEE*, 17(5):40 – 46, sept.-oct. 2003.

[Cantor *et al.*, 2005] S. Cantor, J. Kemp, R. Philpott, and E. Maler. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. saml-core-2.0-os, OASIS Standard, March 2005.

[Ceglowski and Schachter, 2004] M. Ceglowski and J. Schachter. LOAF. http://loaf.cantbedone.org, 2004. [Online; accessed October 2005].

[Coene and Pastor-Balbas, 2006] L. Coene and J. Pastor-Balbas. Telephony Signalling Transport over Stream Control Transmission Protocol (SCTP) Applicability Statement. RFC 4166, Internet Engineering Task Force, February 2006.

[Commission, 2005] Federal Communications Commission. Spam: Unwanted Text Messages and Email. http://www.fcc.gov/guides/spam-unwanted-text-messages-and-email, 2005. [Online; accessed February 2012].

[Cooper *et al.*, 2008] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, Internet Engineering Task Force, May 2008.

[Cormack and Lynam, 2007] G. Cormack and T Lynam. 2007 TREC Public Spam Corpus. http://plg.uwaterloo.ca/~gvcormac/treccorpus07/, July 2007. [Online; accessed November 2008].

[Cortes *et al.*, 2004] M. Cortes, J. R. Ensor, and J. O. Esteban. On SIP performance. *Bell Labs Technical Journal*, 9:155–172, 2004.

[Cranor, 2002] L.F. Cranor. *Web privacy with P3P*. O'Reilly Series. O'Reilly, 2002.

[Crispin, 2003] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501, Internet Engineering Task Force, March 2003.

[Crocker *et al.*, 2011] D. Crocker, T. Hansen, and M. Kucherawy. DomainKeys Identified Mail (DKIM) Signatures. RFC 6376, Internet Engineering Task Force, September 2011.

[CRTC, 2010] CRTC. Canada's Anti-Spam Legislation. http://fightspam.gc.ca/, December 2010. [Online; accessed February 2012].

[Dang *et al.*, 2004] T. D. Dang, B. Sonkoly, and S. Molnar. Fractal Analysis and Modeling of VoIP Traffic. In *Proceedings of Networks 2004 - 11th International Telecommunications Network Strategy and Planning Symposium*, pages 217–222, Vienna, Austria, 2004.

[Dantu and Kolan, 2005] R. Dantu and P. Kolan. Detecting Spam in VoIP Networks. In *Proceedings of Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 31–37, Cambridge, MA, USA, July 2005.

[Davids *et al.*, 2011] C. Davids, V. Gurbani, and S. Poretsky. Methodology for Benchmarking SIP Networking Devices. Internet-draft, Internet Engineering Task Force, March 2011. Work in Progress.

[Dawson and Howes, 1998] F. Dawson and T. Howes. vCard MIME Directory Profile. RFC 2426, Internet Engineering Task Force, September 1998.

[Dierks and Rescorla, 2008] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force, August 2008.

[Dodds, 2006] L. Dodds. Slug:ASemanticWebCrawler. In *2006 Jena User Conference*, Bristol, UK, May 2006.

[DSBL, 2009] DSBL. DSBL is GONE. http://dsbl.org/, March 2009. [Online; accessed February 2012].

[EarthLink, 2012] EarthLink. spamBlocker Accepting a Suspect Email with EarthLink Web Mail. http://support.earthlink.net/articles/email/spamblocker-accepting-a-suspect-email-with-earthlink-web-mail.php, February 2012. [Online; accessed May 2012].

[ECMA, 1999] Ecmascript Language Specification 3rd Edition. ECMA 262, European Computer Manufacturers Association, December 1999.

[Ellison *et al.*, 1999] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, Internet Engineering Task Force, September 1999.

[Ellison, 1999] C. Ellison. SPKI Requirements. RFC 2692, Internet Engineering Task Force, September 1999.

[Farrell *et al.*, 2010] S. Farrell, R. Housley, and S. Turner. An Internet Attribute Certificate Profile for Authorization. RFC 5755, Internet Engineering Task Force, January 2010.

[FCC Robocalls, 2012] FCC Strengthens Consumer Protections Against Telemarketing Robocalls: REPORT AND ORDER. FCC 12-21, Federal Communications Commission, February 2012.

[Fielding and Taylor, 2002] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[Fielding *et al.*, 1999] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[Franks *et al.*, 1999] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, Internet Engineering Task Force, June 1999.

[Freed and Borenstein, 1996a] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045, Internet Engineering Task Force, November 1996.

[Freed and Borenstein, 1996b] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046, Internet Engineering Task Force, November 1996.

[FTC, 2003] Telemarketing Sales Rule. 16 C.F.R. Part 310, Federal Trade Commission, June 2003.

[Gayraud and Jacques, 2010] R. Gayraud and O. Jacques. SIPp. http://sipp.sourceforge.net/, October 2010. [Online; accessed February 2012].

[Google, 2011] Google. Gmail. http://mail.google.com/, 2011. [Online; accessed September 2010].

[Graham, 2002] P. Graham. A Plan for Spam. http://www.paulgraham.com/spam.html, August 2002. [Online; accessed February 2012].

[Granovetter, 1973] M. S. Granovetter. The Strength of Weak Ties. *Amer. J. of Sociology*, 78:1360–80, May 1973.

[Group, 2011] Messaging Anti-Abuse Working Group. Email Metrics Program: The Network Operators Perspective. http://www.maawg.org/email_metrics_report, November 2011. [Online; accessed February 2012].

[Gulbrandsen *et al.*, 2009] A. Gulbrandsen, C. King, and A. Melnikov. The IMAP NOTIFY Extension. RFC 5465, Internet Engineering Task Force, February 2009.

[Hammer *et al.*, 2012] E. Hammer, D. Recordon, and D. Hardt. The OAuth 2.0 Authorization Protocol. Internet-draft, IETF, January 2012. Work in Progress.

[Hammer-Lahav and Cook, 2011] E. Hammer-Lahav and B. Cook. Web Host Metadata. RFC 6415, Internet Engineering Task Force, October 2011.

[Handley *et al.*, 1999] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. RFC 2543, Internet Engineering Task Force, March 1999.

[Handley *et al.*, 2006] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566, Internet Engineering Task Force, July 2006.

[Hansen *et al.*, 2009] T. Hansen, D. Crocker, and P. Hallam-Baker. DomainKeys Identified Mail (DKIM) Service Overview. RFC 5585, Internet Engineering Task Force, July 2009.

[Harris, 2003] E. Harris. The Next Step in the Spam Control War: Greylisting. http://projects.puremagic.com/greylisting/whitepaper.html, August 2003. [Online; accessed February 2012].

[Hatonen *et al.*, 2010] S. Hatonen, A. Nyrhinen, L. Eggert, P. Sarolahti, S. Strowes, and M Kojo. NAT Observations Regarding TCP, DCCP, SCTP and ICMP. Technical report, Department of Computer Science, University of Helsinki, May 2010.

[Hautakorpi *et al.*, 2010] J. Hautakorpi, G. Camarillo, R. Penfield, A. Hawrylyshen, and M. Bhatia. Requirements from Session Initiation Protocol (SIP) Session Border Control (SBC) Deployments. RFC 5853, Internet Engineering Task Force, April 2010.

[Hayes *et al.*, 2008] D. A. Hayes, J. But, and G. Armitage. Issues with network address translation for SCTP. *SIGCOMM Comput. Commun. Rev.*, 39:23–33, December 2008.

[Hewlett-Packard, 2010] Hewlett-Packard. HP VCX V7000 Unified Communications Series_PL: Call restrictions. h17007.www1.hp.com/docs/mark/4AA3-0828ENW.pdf, 2010. [Online; accessed February 2012].

[Hoffman *et al.*, 2009] P. Hoffman, J. Levine, and A. Hathcock. Vouch By Reference. RFC 5518, Internet Engineering Task Force, April 2009.

[Intel, 1996] Intel. Pentium Pro Family Developers Manual. http://download.intel.com/support/processors/pentiumpro/sb/24269001.pdf, 1996. [Online; accessed February 2012].

[ITU-T, 1993] ITU-T. Information technology – Open Systems Interconnection – The Directory: Models. Recommendation X.501, International Telecommunications Union, Geneva, 1993.

[ITU-T, 1995] ITU-T. Information technology – Open Systems Interconnection – Security frameworks for open systems: Authentication framework. Recommendation X.811, International Telecommunications Union, Geneva, 1995.

[ITU-T, 1998] ITU-T. Information technology – Open Systems Interconnection – Public-key and attribute certificate frameworks. Recommendation X.509, International Telecommunications Union, Geneva, August 1998.

[James, 2010] P. James. RESTful interface to MySQL using PHP. http://sourceforge.net/projects/phprestsql/, 2010. [Online; accessed November 2010].

[Janak, 2003] J. Janak. SIP server proxy effectiveness. Master's thesis, Czech Technical University Department of Computer Science, Prague, Czech Republic, May 2003.

[Jennings *et al.*, 2002] C. Jennings, J. Peterson, and M. Watson. Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks. RFC 3325, Internet Engineering Task Force, November 2002.

[Jennings *et al.*, 2009] C. Jennings, R. Mahy, and F. Audet. Managing Client-Initiated Connections in the Session Initiation Protocol (SIP). RFC 5626, Internet Engineering Task Force, October 2009.

[Jiang and Schulzrinne, 2000] W. Jiang and H. Schulzrinne. Analysis of On-Off Patterns in VoIP and Their Effect on Voice Traffic Aggregation. In *Proceedings of ICCCN 2000*, pages 82–87, Las Vegas, NV, USA, 2000.

[Jung and Sit, 2004] J. Jung and E. Sit. An empirical study of spam traffic and the use of DNS black lists. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 370–375, Taormina, Sicily, Italy, 2004.

[Kam, 2011] Kamailio SIP Server v3.2.x: Core Cookbook. http://www.kamailio.org/wiki/cookbooks/3.2.x/core, October 2011. [Online; accessed February 2012].

[Katz *et al.*, 1994] E. Katz, M. Butler, and R. McGrath. A scalable http server: The ncsa prototype. *Computer Networks and ISDN Systems*, pages 155–164, 1994.

[Kegel, 2006] D. Kegel. The C10K problem. http://www.kegel.com/c10k.html, July 2006. [Online; accessed January 2006].

[Kent and Millett, 2003] S. T. Kent and L. I. Millett. *Who Goes There?:Authentication Through the Lens of Privacy*. The National Academies Press, 2003.

[Kent and Mogul, 1987] C. A. Kent and J. C. Mogul. Fragmentation Considered Harmful. In *Proceedings of the ACM workshop on Frontiers in computer communications technology*, SIGCOMM '87, pages 390–401, Stowe, VT, USA, August 1987.

[Kissel, 2011] R. Kissel. Glossary of key information security terms. NIST IR 7298 Revision 1, National Institute of Standards and Technology, February 2011.

[Klensin, 2008] J. Klensin. Simple Mail Transfer Protocol. RFC 5321, Internet Engineering Task Force, October 2008.

[Klyne and Newman, 2002] G. Klyne and C. Newman. Date and Time on the Internet: Timestamps. RFC 3339, Internet Engineering Task Force, July 2002.

[Krawczyk *et al.*, 1997] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, Internet Engineering Task Force, February 1997.

[Leiba, 1997] B. Leiba. IMAP4 IDLE command. RFC 2177, Internet Engineering Task Force, June 1997.

[Lemon, 2001] J. Lemon. Kqueue - A Generic and Scalable Event Notification Facility. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 141–153, Berkeley, CA, USA, June 2001.

[Lennox *et al.*, 2002] J. Lennox, A. Alexiou, W. Jiang, S. Narayanan, H. Schulzrinne, P. Sebos, K. Singh, T. Kapoor, A. Voskoboynik, X. Wu, and Y. Xu. CINEMA:sipd. http://www.cs.columbia.edu/irt/cinema/doc/sipd.html, 2002. [Online; accessed January 2006].

[Lennox *et al.*, 2004] J. Lennox, X. Wu, and H. Schulzrinne. Call Processing Language (CPL): A Language for User Control of Internet Telephony Services. RFC 3880, Internet Engineering Task Force, October 2004.

[Levine, 2010] J. Levine. DNS Blacklists and Whitelists. RFC 5782, Internet Engineering Task Force, February 2010.

[Libenzi, 2002] D. Libenzi. Improving (network) I/O performance. http://www.xmailserver.org/linux-patches/nio-improve.html, October 2002. [Online; accessed January 2006].

[Lyon and Wong, 2006] J. Lyon and M. Wong. Sender ID: Authenticating E-Mail. RFC 4406, Internet Engineering Task Force, April 2006.

[Marshall, 2003] W. Marshall. Private Session Initiation Protocol (SIP) Extensions for Media Authorization. RFC 3313, Internet Engineering Task Force, January 2003.

[Mason, 2011] J. Mason. The Apache SpamAssassin Project. http://spamassassin.apache.org/, 2011. [Online; accessed February 2012].

[Mathieu *et al.*, 2008] B. Mathieu, S. Niccolini, and D. Sisalem. SDRS: A Voice-over-IP Spam Detection and Reaction System. *Security & Privacy, IEEE*, 6(6):52–59, Nov.-Dec. 2008.

[Mayer *et al.*, 1995] Roger C. Mayer, James H. Davis, and F. David Schoorman. An integrative model of organizational trust. *The Academy of Management Review*, 20(3):pp. 709–734, 1995.

[Microsoft, 2012] Microsoft. U-Prove Crypto SDK V1.1 (Java Edition). http://archive.msdn.microsoft.com/uprovesdkjava, February 2012. [Online; accessed Feburuary 2012].

[Mozilla, 2010a] Mozilla. Add-ons for Firefox. http://addons.mozilla.org/en-US/firefox/, 2010. [Online; accessed September 2010].

[Mozilla, 2010b] Mozilla. Mozilla messaging: Thunderbird. http://www.mozillamessaging.com/, 2010. [Online; accessed September 2010].

[Murchison, 2008] K. Murchison. Sieve Email Filtering: Subaddress Extension. RFC 5233, Internet Engineering Task Force, January 2008.

[MySQL, 2006] MySQL. http://www.mysql.com/, 2006. [Online; accessed January 2006].

[Nagar *et al.*, 2004] S. Nagar, P. Larson, H. Linder, and D. Stevens. epoll scalability web page. http://lse.sourceforge.net/epoll, 2004. [Online; accessed February 2012].

[Nagle, 1984] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 0896, Internet Engineering Task Force, January 1984.

[Nahum *et al.*, 2007] E. Nahum, J. Tracey, and C. Wright. Evaluating SIP Proxy Server Performance. In *Proceedings of 17th International Workshop on Networking and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Urbana-Champaign, IL, USA, June 2007.

[Narayanan *et al.*, 2002] S. Narayanan, A. Yu, T. Kapoor, and H. Schulzrinne. SIPstone test suite. http://www.cs.columbia.edu/IRT/cinema/sipstone, April 2002. [Online; accessed January 2006].

[Neuman *et al.*, 2005] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120, Internet Engineering Task Force, July 2005.

[Ono and Schulzrinne, 2008a] K. Ono and H. Schulzrinne. The Impact of SCTP on Server Scalability and Performance. In *Proceedings of IEEE GLOBECOM*, New Orleans, LA, USA, December 2008.

[Ono and Schulzrinne, 2008b] K. Ono and H. Schulzrinne. One Server Per City: Using TCP for Very Large SIP Servers. In *Proceedings of Principles, Systems and Applications of IP Telecommunications (IPTComm 2008)*, volume 5310 of *LNCS*, pages 133–148, Heidelberg, Germany, July 2008. Springer-Verlag.

[Ono and Schulzrinne, 2009a] K. Ono and H. Schulzrinne. Have I Met You Before? Using Cross-Media Relations to Reduce SPIT. In *Proceedings of Principles, Systems and Applications of IP Telecommunications*, IPTComm 2009, Atlanta, GA, USA, July 2009.

[Ono and Schulzrinne, 2009b] K. Ono and H. Schulzrinne. Referencing Earlier Communications in SIP Requests. Internet-Draft, IETF, October 2009. Work in Progress.

[Ono and Schulzrinne, 2009c] K. Ono and H. Schulzrinne. Using Cross-Media Relations to Reduce False Positives during SPIT Filtering. Internet-draft, IETF, October 2009. Work in Progress.

[Ono and Schulzrinne, 2011a] K. Ono and H. Schulzrinne. Referencing and Validating User Attributes. Internet-draft, IETF, October 2011. Work in Progress.

[Ono and Schulzrinne, 2011b] K. Ono and H. Schulzrinne. Using Cross-Media Relations to Identify Important Communication Requests: Testing the Concept and Implementation. In *Proceedings of Principles, Systems and Applications of IP Telecommunications*, IPTComm 2011, Chicago, IL, USA, August 2011.

[Ope, 2010] Kamailio (OpenSER) SIP Server. http://www.kamailio.org/w/, 2010. [Online; accessed September 2010].

[Paquin, 2011] C. Paquin. U-Prove Technology Overview V1.1. Microsoft Corporation Draft Revision 1, February 2011.

[Partridge and Hinden, 1990] C. Partridge and R.M. Hinden. Version 2 of the Reliable Data Protocol (RDP). RFC 1151, Internet Engineering Task Force, April 1990.

[Paul, 2011] P. Paul. Cultural Studies: Don't Call Me, I Won't Call You. *The New York Times*, March 2011.

[Peterson and Jennings, 2006] J. Peterson and C. Jennings. Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP). RFC 4474, Internet Engineering Task Force, August 2006.

[Peterson *et al.*, 2006] J. Peterson, J. Polk, D. Sicker, and H. Tschofenig. Trait-Based Authorization Requirements for the Session Initiation Protocol (SIP). RFC 4484, Internet Engineering Task Force, August 2006.

[Peterson, 2002] J. Peterson. A Privacy Mechanism for the Session Initiation Protocol (SIP). RFC 3323, Internet Engineering Task Force, November 2002.

[Postel, 1980] J. Postel. User Datagram Protocol. RFC 0768, Internet Engineering Task Force, August 1980.

[Postel, 1981] J. Postel. Transmission Control Protocol. RFC 0793, Internet Engineering Task Force, September 1981.

[Quittek *et al.*, 2007] J. Quittek, S. Niccolini, S. Tartarelli, M. Stiemerling, M. Brunner, and T. Ewald. Detecting SPIT Calls by Checking Human Communication Patterns. In *Proceedings of IEEE International Conference on Communications (ICC '07)*, pages 1979–1984, June 2007.

[Raggett *et al.*, 1999] D. Raggett, A. L. Hors, and I. Jacobs. HTML 4.01 Specification. http://www.w3.org/TR/REC-html40/, December 1999.

[Ram *et al.*, 2008] K. K. Ram, I. C. Fedeli, A. L. Cox, and S. Rixner. Explaining the Impact of Network Transport Protocols on SIP Proxy Performance. In *Proceedings of the ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and Software*, pages 75–84, Washington, DC, USA, 2008.

[Ramachandran and Feamster, 2006] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 291–302, Pisa, Italy, 2006.

[Ramsdell and Turner, 2010] B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751, Internet Engineering Task Force, January 2010.

[RedHat, 2007] RedHat. Red Hat Enterprise Linux. http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/ch-kernel.html, 2007. [Online; accessed February 2012].

[Resnick, 2008] P. Resnick. Internet Message Format. RFC 5322, Internet Engineering Task Force, October 2008.

[Rosenberg *et al.*, 2002] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[Rosenberg *et al.*, 2005] J. Rosenberg, H. Schulzrinne, and G. Camarillo. The Stream Control Transmission Protocol (SCTP) as a Transport for the Session Initiation Protocol (SIP). RFC 4168, Internet Engineering Task Force, October 2005.

[Salsano *et al.*, 2002] S. Salsano, L. Veltri, and D. Papalilo. SIP security issues: the SIP authentication procedure and its processing load. *Network, IEEE*, 16(6):38–44, nov/dec 2002.

[Schmidt and Lopez, 1999] F. Schmidt and F. G. Lopez. Analytical Cost Model - National Core Network, Version 1.0 . Consultative document, Wissenschaftliches Institut fur Kommunikationsdienste, April 1999.

[Schulzrinne *et al.*, 2002] H. Schulzrinne, S. Narayanan, J. Lennox, and M. Doyle. SIPstone - Benchmarking SIP Server Performance. Technical Report cucs-005-02, Columbia University, March 2002.

[Schulzrinne, 2004] H. Schulzrinne. The tel URI for Telephone Numbers. RFC 3966, Internet Engineering Task Force, December 2004.

[Sermersheim, 2006] J. Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol. RFC 4511, Internet Engineering Task Force, June 2006.

[SHA, 2002] Secure Hash Standard. Federal Information Processing Standard (FIPS) 180-2, National Institute of Science and Technology, August 2002.

[Shacham and Schulzrinne, 2007] R. Shacham and H. Schulzrinne. HTTP Header for Future Correspondence Addresses. Internet-draft, IETF, May 2007. Work in Progress.

[Shemyak and Vehmanen, 2007] K. Shemyak and K. Vehmanen. Scalability of TCP Servers, Handling Persistent Connections. In *Proceedings of Sixth International Conference on Networking (ICN'07)*, April 2007.

[Shen *et al.*, 2010] C. Shen, E. Nahum, H. Schulzrinne, and C. Wright. The impact of TLS on SIP server performance. In *Principles, Systems and Applications of IP Telecommunications*, IPTComm '10, pages 59–70, Munich, Germany, 2010.

[Siemens, 2007] Siemens. Nokia Siemens Networks to operate DNA's fixed networks in Finland. http://www.nokiasiemensnetworks.com/, December 2007. [Online; accessed February 2012].

[Singh and Schulzrinne, 2005] K. Singh and H. Schulzrinne. Failover and Load Sharing in SIP Telephony. In *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, Philadelphia, PA, USA, July 2005.

[SIP, 2010] SIP Communicator. http://sip-communicator.org/, 2010. [Online; accessed September 2010].

[spa, 2006] SpamAssassin public mail corpus. http://spamassassin.apache.org/publiccorpus/readme.html, January 2006. [Online; accessed November 2008].

[Sparks, 2006] R. Sparks. SIPit19 Summary. https://www.sipit.net/SIPit19_Summary, 2006. [Online; accessed February 2012].

[Sparks, 2008] R. Sparks. SIPit22 Summary. https://www.sipit.net/SIPit22_Summary, April 2008. [Online; accessed February 2012].

[Sparks, 2011] R. Sparks. SIPit29 Summary. https://www.sipit.net/SIPit29_summary, October 2011. [Online; accessed February 2012].

[SPEC, 2011] SPEC. SPECsip_Infrastructure2011. http://www.spec.org/sipinf2011/, September 2011. [Online; accessed February 2012].

[Stewart, 2007] R. Stewart. Stream Control Transmission Protocol. RFC 4960, Internet Engineering Task Force, September 2007.

[Stolfo *et al.*, 2006] S. J. Stolfo, S. Hershkop, C. Hu, W. Li, O. Nimeskern, and K. Wang. Behavior-based modeling and its application to Email analysis. *ACM Trans. Internet Technol.*, 6(2):187–221, May 2006.

[Tobin, 2009] F. Tobin. Pyzor. http://sourceforge.net/apps/trac/pyzor/, May 2009. [Online; accessed February 2012].

[Tschofenig *et al.*, 2010] H. Tschofenig, J. Hodges, J. Peterson, J. Polk, and D. Sicker. SIP SAML Profile and Binding. Internet-draft, IETF, October 2010. Work in Progress.

[Twining *et al.*, 2004] R. Twining, M. Williamson, M. Mowbray, and M. Rahmouni. Email Prioritization: reducing delays on legitimate mail caused by junk mail. Technical Report HPL-2004-5R1, Hewlett-Packard, 2004.

[V. Prakash and J. ODonnell, 2005] V. V. Prakash and A. J. ODonnell. Reputation-Based Approach for Efficient Filtration of Spam. http://www.cloudmark.com/en/whitepapers/reputation-based-approach-for-efficient-filtration-of-spam, September 2005. [Online; accessed February 2012].

[voice, 2012] Google voice. Call Screening. http://support.google.com/voice/bin/answer.py?hl=en&answer=115083&topic=1708125&ctx=topic, 2012. [Online; accessed May 2012].

[WIK, 2000] WIK. Analytical Cost Model - National Core Network, Version 2.0 . Consultative document, Wissenschaftliches Institut fur Kommunikationsdienste, June 2000.

[Wong and Schlitt, 2006] M. Wong and W. Schlitt. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408, Internet Engineering Task Force, April 2006.

[Zurich, 2011] IBM Research Zurich. Specification of the Identity Mixer Cryptographic Library Version 2.3.3, June 2011.

Part V

Appendices

Appendix A

Measurement Results

A.1 Slab Cache for TCP Connections

To get a detailed picture of the memory usage for TCP connections, we monitored the usage of the slab cache, where Linux implements a memory allocation mechanism for frequent allocations and deallocations of data. Figure A.1 shows that the usage of the slab cache for approximately 520,000 TCP connections. The Linux kernel, configured for 2 GB of the kernel memory in the VM split, allocates TCP socket data structures and socket buffers at 14,800 requests/second rate. This result indicates that slab cache consumes 1.2 GB for these TCP connections, which are approximately the same amount of overall memory usage shown in Figure 3.1. The slab cache was allocated mostly for the socket related data structures, a few for the data structures for the `epoll()` system calls: `eventpoll_epi` and `evenpoll_pwq`. This also illustrates that the slab cache dynamically allocated for the socket buffer heads, i.e., `skbuff_head_cache`, and user data, i.e., `size-512`, consumes only 12 MB. Therefore, we have determined that each TCP connection requires 2.27 KB of the slab cache[1] and the bottleneck of sustainable concurrent connections is the amount of allocatable kernel memory for the slab cache since this slab cache excluding for the socket buffer heads and user data is statically allocated while the TCP connection remains open.

[1]This is larger than the value in Table A.1 below since this memory footprint includes the memory consumption for partial set of data structures for a connection being created in progress, as described in Section 3.4.1.

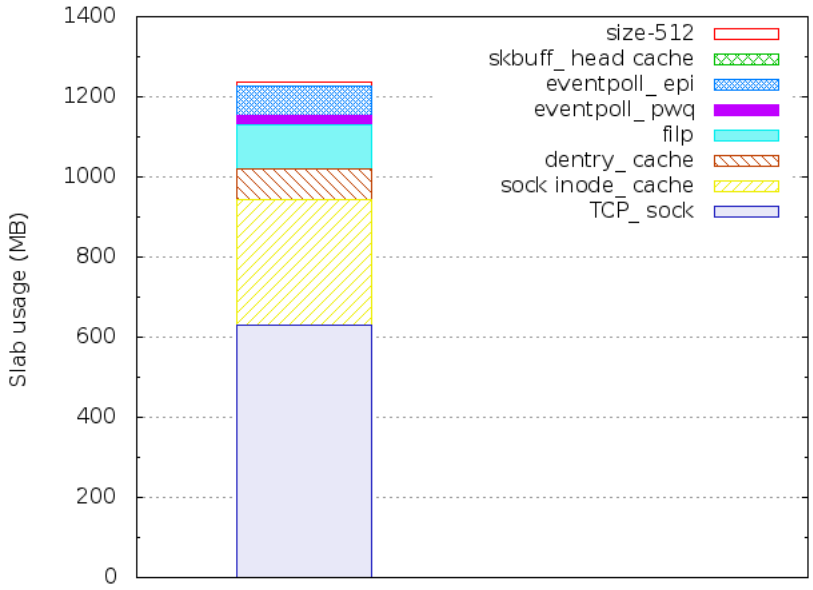


Figure A.1: Slab cache usage for 520,000 TCP connections for echo server

A.2 Data Size of TCP and SCTP Sockets

Table A.1 itemizes the sizes of socket-related data structures used by the Linux implementation. These data structures are allocated from slab cache objects to maintain an SCTP association or a TCP connection. SCTP-specific data structures requires a larger amount of memory, including the 5,120 byte `sctp_association` which stores parameters in transmission control block (TCB), while TCP requires only the 1,096 byte `tcp_socket`. Table A.1 also shows that the amount of memory using a one-to-many socket significantly increases as a function of the number of associations, while using multi-streams increases slightly with the number of streams.

Furthermore, several SCTP-specific data objects, including the `sctp_association`, are allocated from general purpose slab objects, which suffer from internal fragmentation because of power of two sized objects. As a result, maintaining an SCTP socket consumes 10,812 bytes, approximately five times of the amount needed for a TCP socket, even in the simplest case, namely, with a one-to-one socket and a single stream. The dominant data is association-related data, which consumes approximately 80 percent of the total memory

Data structure		Slab cache			
Name	Size (bytes)	Name	Size (bytes)	# of objects SCTP	TCP
dentry	128	dentry	128	1	1
file	136	filp	192	1	1
inode	328	sock_inode_cache	384	1	1
socket	40				
sock(sctp_socket)	772	SCTP	896	1	0
sock(tcp_socket)	1,096	TCP	1,152	0	1
epoll_entry	36	eventpoll_pwq	36	1	1
epitem	80	eventpoll_epi	128	1	1
sctp_endpoint	176	size-256	256	1	0
sctp_bind_addr[a]	40	size-64	64	1	0
Subtotal for a socket (bytes)				2,084	2,020
sctp_association	5,120	size-8192	8,192	1 or n[b]	0
sctp_transport	284	size-512	512	1 or n[b]	0
sctp_ssnmap	60	size-64	64	1 or m[c]	0
Subtotal for an association (bytes)				8,768	0
Total memory usage (bytes)				10,852	2,020

[a] The sctp_bind_addr structure includes the pointer to a list of IP addresses for multi-homing.

[b] one-to-many: 1-to-n

[c] multi-streaming: m streams

Table A.1: Comparison of the data structures of SCTP and TCP sockets

usage. A server needs to allocate the same number of associations for both two SCTP socket styles. Therefore, using a one-to-many socket does not contribute to a drastic reduction of memory usage contrary to our expectation described in Section 2.4.2. How to reduce association-related data size is discussed in Section 3.5.1.

Appendix B

User Study Results

This appendix provides the detailed results of our user study to test the concept of cross-media relations described in Section 7.10. The following tables provide the standard deviation of fractions for each group among participants in addition to the mean.

Our user study had 60 unique participants including 53 holders of Columbia University email accounts. Since the participants were identified by their email accounts registered as their contact address, a person might be counted as separate participants when using multiple contact addresses.

B.1 User Study of Incoming Email

For our email survey, the Web application for this survey performed as an IMAP client fetching email headers to the IMAP server where each participants had stored their received email messages. Our user study collected answers from 72 email accounts consisting of 19 university accounts (18 for Columbia University and one for other university) and 53 free email accounts (31 for gmail.com and 22 for yahoo.com or yahoo outside of the U.S.). Since 10 out of 53 free email accounts had only spam messages, the results excluded these 10 accounts used as disposal accounts, which did not fit for our study. Consequently, this study analyzes 62 email accounts.

Each participant was asked to categorize received messages into 13 groups based on sender IDs, as described in Section 7.10.2. Collected answers contains 3,257 message for

	Messages for university email accounts	Messages for free email accounts
Mean	171.4	489.6
Median	91	150
Stdev.	223.5	573.5

Stdev.: Standard deviation

Table B.1: Mean, median, and standard deviation of numbers of messages: University email accounts and free email accounts

	sent before	replied within two weeks	trusted users	trusted domains	FoF in CC	FoF in ML	Web-then-email	email-then-email	call-then-email	SMS-then-email	public profile
Mean	10.4	0.3	1.1	66.5	0.1	0.0	13.3	3.0	0.6	0.0	1.6
Median	7.4	0.0	0.1	75	0.0	0.0	6.6	0.0	0.0	0.0	0.0
Stdev.	1.4	1.1	1.8	26.9	0.4	0.0	15.8	7.2	2.1	0.0	2.8

	spam	unlabeled
Mean	2.6	0.5
Median	0.6	0.0
Stdev.	4.8	1.4

Table B.2: Mean, median, and standard deviation of percentages of messages in 13 groups: University email accounts

19 university email accounts and 21,051 messages for 43 free email accounts. Table B.1 presents the mean, median, and standard deviation of the number of messages for each type of email accounts, across participants. Tables B.2 and B.3 presents the mean, median, and standard deviation of fraction of messages received for the 13 groups for these two types of email accounts, respectively.

B.2 User Study of Incoming Calls

Our user study collected answers about 3,300 calls for 36 call accounts, including 17 free VoIP accounts, 10 landlines, and 9 mobile phones. The information about categories is

	sent before	replied within two weeks	trusted users	trusted domains	FoF in CC	FoF in ML	Web-then-email	email-then-email	call-then-email	SMS-then-email	public profile
Mean	18.8	0.1	1.0	6.7	2.0	0.0	42.7	8.8	1.0	0.5	0.3
Median	1.6	0.0	0.0	0.0	0.0	0.0	31.4	0.0	0.0	0.0	0.0
Stdev.	25.9	0.1	2.5	18.3	6.6	0.0	37.0	17.3	3.8	2.4	1.2

	spam	unlabeled
Mean	18.0	0.2
Median	3.7	0.0
Stdev.	26.4	0.7

Table B.3: Mean, median, and standard deviation of percentages of messages in 13 groups: Free email accounts

based on answers to the questionnaire described in Section 7.10.2.

Table B.4 shows the mean, median, and standard deviation of the number of incoming calls, across participants. Table B.5 presents the mean, median, and standard deviation of fraction of the number of calls received from nine groups, across participants. Compared to email, 4 out of 13 groups are eliminated. The "replied within two weeks" group is excluded to allow participants to easily answer by manually parsing call history. The "trusted domain" group is eliminated since phone numbers contain no domain name. The "FoF in CC" and "FoF in ML" groups are also eliminated since a call has no CCed or list phone numbers except a conference all.

Although the Web survey application was enabled to parse call detail records provided by two major mobile phone companies, it also allowed participants to manually enter answers.

As seen in Table B.5, the medians of the fractions of calls having web-then-call, email-then-call, and SMS-then-call are all zero although the means range from 4.3 to 5.5. To closely look at the distribution, Figures B.1 –B.4 plot the cumulative distribution function (CDF) of the percentages of calls having each relation including call-then-call. These figures show that the potential effectiveness of using cross-media relations to identify good calls is limited to less than half of the participants although it appears significant for certain users. In contrast, using call-then-call relations appears effective for more than 80 percent of the

	Calls
Mean	91.7
Median	85.5
Stdev.	46.9

Table B.4: Mean, median, and standard deviation of numbers of calls

	sent before	trusted users	Web-then-call	email-then-call	call-then-call	SMS-then-call	public profile	spam	unlabeled
Mean	53.3	4.9	5.2	4.3	21.8	5.5	0.2	4.1	0.8
Median	49.6	3.4	0.0	0.0	25.5	0.0	0.0	3.8	0.0
Stdev.	16.2	7.4	8.9	9.5	16.8	12.2	0.5	2.6	5.0

Table B.5: Mean, median, and standard deviation of percentages of calls in 9 groups

participants although the fractions of calls having the relations widely vary.

B.3 User Study of Incoming SMS Messages

Our user study collected answers about 3,970 SMS messages for 34 accounts including 27 free SMS accounts. The information about categories is based on answers to the questionnaire described in Section 7.10.2.

Table B.6 shows the mean, median, and standard deviation of the number of SMS messages, across participants. Table B.7 presents the mean and standard deviation of each fraction of the number of SMS messages in the nine groups, which are the same as the groups for calls. Similar to calls, the Web survey application was enabled to parse call detail records provided by two major mobile phone companies and also allowed participants to manually enter answers.

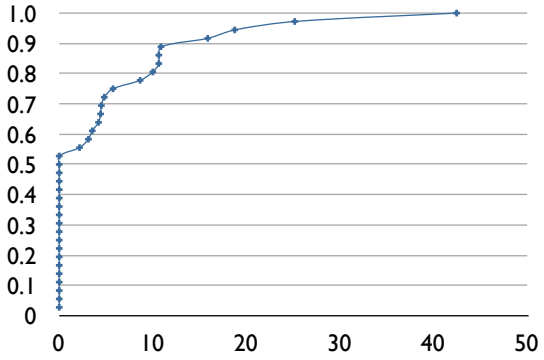


Figure B.1: CDF of percentages of calls having Web-then-call relations

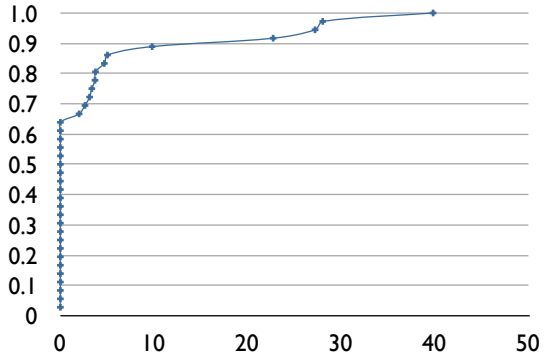


Figure B.2: CDF of percentages of calls having email-then-call relations

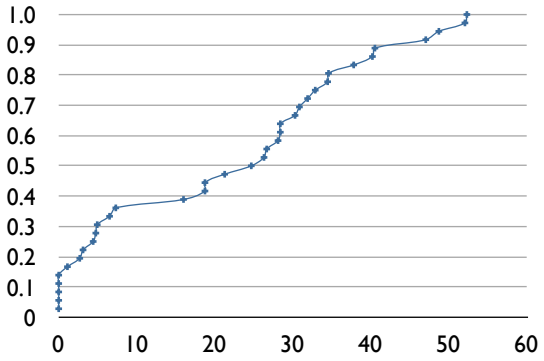


Figure B.3: CDF of percentages of calls having call-then-call relations

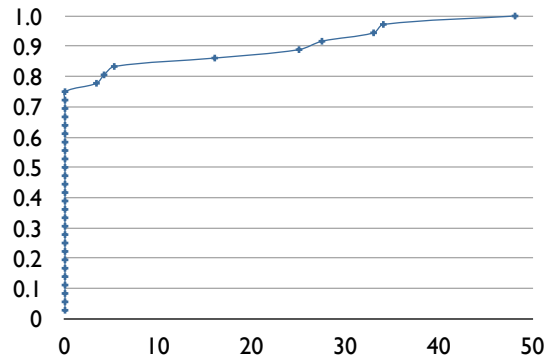


Figure B.4: CDF of percentages of calls having SMS-then-call relations

	SMS messages
Mean	117.8
Median	91
Stdev.	123.5

Table B.6: Mean, median, and standard deviation of numbers of SMS messages

	sent before	trusted users	Web-then-SMS	email-then-SMS	call-then-SMS	SMS-then-SMS	public profile	spam	unlabeled
Mean	62.0	9.3	4.2	3.9	12.0	1.0	1.7	5.8	0.0
Median	60.2	8.9	3.4	2.4	6.5	0.0	0.0	5.4	0.0
Stdev.	10.1	7.3	4.1	4.5	13.4	3.9	7.5	3.1	0.0

Table B.7: Mean, median, and standard deviation of percentages of SMS messages in 9 groups